Les principes des systèmes d'information géographique

Principes, algorithmes et architecture du système Savane

Marc Souris

Sommaire

Première partie : introduction et présentation générale du système SAVANE

0. Introduction ……..…1

1. Présentation générale des systèmes d'information géographique ……..… 9

1.1.L'apport de l'informatique pour la géographie ou la cartographie
1.2. Les SIG comme synthèse de ces domaines : historique et évolution
1.2.1. Les débuts : les années 60-70
1.2.2. La consolidation : les années 80
1.2.3. La diffusion : les années 90
1.2.4. Les évolutions actuelles
1.3. Les SIG : objectifs généraux
1.4. Conclusion

2. Présentation du système SAVANE…….…….……. 21

2.1. Le système SAVANE : principes généraux
2.1.1. Les concepts généraux de SAVANE
2.1.2. L'architecture générale de SAVANE
2.1.3. Administrer une base SAVANE
2.2. Présentation des différents modules
2.2.1. SAVATECA : administration des bases de données
2.2.1.1. Principes généraux d'administration
2.2.1.2. Configuration du système
2.2.1.3. Organisation d'une base de données SAVANE : principe généraux
2.2.1.4. Création d'une base de données
2.2.1.5. Gestion du schéma
2.2.1.6. Gestion des vues externes
2.2.1.7. Intégration d'objets
2.2.1.8. Gestion des utilisateurs
2.2.2. SAVAMER : géoréférencement d'images et constitution de mosaïques
2.2.2.1. Le géoréférencement
2.2.2.2. Le ré-échantillonage

2.2.2.3. Le mosaïquage et l'intégration
2.2.3. SAVEDIT : saisie vectorielle sur écran avec gestion des contraintes d'intégrité
2.2.3.1. Principes généraux
2.2.3.1. Format des documents SAVEDIT
2.2.3.1. Principes généraux
2.2.3.1. Principes généraux
2.2.4. SAVANE : exploitation des bases de données et cartographie
2.2.4.1. Cartes, cadres et requêtes
2.2.4.1.1. La classe CCarte
2.2.4.1.2. La classe CCadre
2.2.4.2. Principales opérations de manipulation de données
2.2.4.3. Masques et distances
2.2.4.4. Macro-commandes et méthodes
2.2.4.5. Edition cartographique et impression
2.2.4.6. Utilisateurs et partage des bases de données
2.2.4.7. Raster et vecteur
2.2.5. SAVATLAS : consultation de cartes et de données
2.2.6. SAVLIBRARY : librairie de développement VC++
2.2.7. SAVSERVEUR : serveur de requêtes via Internet
2.2.7.1. Méthodes client/serveur via Internet
2.2.7.2. L'architecture du serveur
2.2.7.3. L'architecture d'un client
2.3. La constitution et l'exploitation d'une base de données avec SAVANE
2.3.1. Généralité sur la constitution d'une base de données géographiques
2.3.1.1 Méthodes de mise en place, d'administration, et d'exploitation
2.3.1.2 Difficultés
2.3.2. De la création d'une base de données à son exploitation
2.3.2.1 La création d'une base de données : schéma et méthodes
2.3.2.2 L'intégration graphique
2.3.2.3 Modifications et unions graphiques, contraintes d'intégrités
2.3.2.4 L'intégration descriptive
2.3.2.5 Vues externes et droits d'accès
2.3.2.6 Création et gestion de méthodes

Deuxième partie : Modéliser et gérer l'information géographique

3. Modéliser le monde réel ……...… 59

3.1. Préambule
3.1.1. Quelques exemples
3.1.2. Précision et description
3.2. Pourquoi et comment modéliser le monde réel
3.2.1. Comment appréhender et représenter la réalité avec un ordinateur
3.2.2. Réalité et connaissance
3.2.3. Modéliser la connaissance
3.2.4. Collection d'objet et gestion
3.3. La schématisation du monde réel : de la réalité à la géographie
3.3.1. La localisation comme attribut : l'objet géographique
3.4. La schématisation de la localisation : de la géographie à la géométrie

3.4.1. Géographie et cartographie : une union agitée
3.4.2. Cartes de localisation et cartes thématiques
3.4.3. La représentation géométrique de la localisation géographique
3.4.4. Décrire la localisation
3.4.5. Les limites du modèle géométrique
3.5. De la géométrie à l'informatique : la représentation informatique de la localisation
3.5.1. La représentation en maille
3.5.2. La représentation vectorielle
3.5.3. Les structures de représentation de l'information localisée
3.5.4. Les méthodes de compactage
3.6. Les sources de l'information localisée : nature et validité
3.6.1. Les moyens d'acquisition des données localisées
3.6.2. La validité de l'information
3.7. Structure géométriques des objets dans le système SAVANE
3.7.1. Le modèle géométrique
3.7.2. La classe CArc
3.7.3. Les changements de représentation interne
3.8. Conclusion

4. Mesurer la localisation ……..… 95

4.1. Introduction
4.2. Formes et mesure de la Terre : géodésie et gravimétrie
4.2.1. Mesurer la forme de la Terre : la géodésie
4.2.2. L'approche géométrique
4.2.3. L'approche géophysique
4.2.3.1. Gravité et géoïde.
4.2.3.2. De nouveaux ellipsoïdes
4.2.4. Nouveaux instruments, nouvelles mesures, nouvelles précisions
4.2.4.1. Le système GPS
4.2.4.2 Les changements de datum
4.2.5. Précision des mesures
4.3. Représenter l'ellipsoïde sur une surface plane : cartographie et projections
4.3.1. Les déformations
4.3.2. Les surfaces développables
4.3.3. Coordonnées et espace de projection
4.3.4. Échelles et cartes
4.3.5. Échelle, projection, précision
4.4. Exemples de projections
4.4.1. La projection Mercator
4.4.2. La projection UTM (Universal Transverse Mercator)
4.4.3. La projection conique de Lambert
4.5. SAVANE, *datum* et projections
4.5.1. Provenance des données en entrée
4.5.2. Fonctionnement de SAVEDIT pour les projections et le datum
4.5.3. Fonctionnement de SAVANE pour les projections dans un cadre
4.6. Conclusion

5. De l'objet à la collection d'objet : les SGBD ……..... 129

5.1 Introduction
5.2 Notions classiques sur les SGDB
5.2.1. Les problèmes soulevés par l'utilisation de fichiers traditionnels : dépendance, redondance, intégrité, sécurité
5.2.2. L'objectif des SGBD
5.2.3. La modélisation de la réalité dans les SGBD
5.2.3.1. Les différents modèles de données
5.2.3.2. Les niveaux de description
5.2.4. Les modèles hiérarchique et réseau
5.2.5. Le modèle relationnel
5.2.5.1. Les principes du relationnel
5.2.5.2. La notion de clé et la théorie de la normalisation
5.2.5.3. L'algèbre relationnelle
5.2.5.4. Puissance et limite du relationnel
5.2.6. Le modèle objet
5.2.6.1. Objets, méthodes, encapsulation et héritage
5.2.6.2. Les bases de données objets
5.3. SGBDR et objets géographiques : l'extension du modèle relationnel
5.3.1 Objectifs
5.3.2 L'extension du modèle relationnel
5.3.2.1. Les principes de l'extension du modèle
5.3.2.2. Relations localisées et notion de clé géographique
5.3.3. Les extensions de l'algèbre relationnelle
5.3.3.1. La restriction spatiale
5.3.3.2. La jointure spatiale
5.3.3.3. La projection spatiale
5.3.3.4. Masques et semi-jointures géométriques
5.3.4. Objets spatiaux et opérations relationnelles classiques
5.4. Structuration et indexation de l'attribut de localisation
5.4.1. Les méthodes classiques d'indexation
5.4.2. L'indexation multidimensionnelle
5.5. SAVANE : mise en œuvre du relationnel étendu
5.5.1. Schéma : relations et attributs
5.5.2. Structuration des collections d'objet
5.5.2.1. Architecture générale et organisation des bases de données
5.5.2.2. Indexation géographique et structure des fichiers
5.5.2.2.1. Le type non localisé
5.5.2.2.2. Le type point
5.5.2.2.3. Le type ligne
5.5.2.2.4. Le type zone
5.5.2.2.5. Le type mosaïque
5.5.3. Intégration d'objets et de valeurs d'attributs
5.5.3.1. La constitution d'une relation
5.5.3.2. Codage des valeurs d'attributs descriptifs
5.5.3.3. L'intégration graphique
5.5.3.4. L'intégration descriptive

5.5.3.5. La classe CIndexAttribut
5.5.4. SAVANE : requêtes et états temporaires
5.5.4.1. Principes d'une requête dans SAVANE
5.5.4.2. Définition et gestion des macro-commandes
5.5.5. La classe CLecture
5.5.6. SAVANE : les structures internes
5.5.6.1. Vecteur et raster
5.5.6.2. Un algorithme de rasterisation
5.5.7.Les masques
5.5.7.1. La structure d'un masque
5.5.7.2. Création d'un masque : zone tampon ou définition sur écran
5.5.7.3. L'algèbre des masques
5.5.8. La réalisation des opération relationnelles classiques
5.5.8.1. La restriction
5.5.8.2. La jointure
5.5.9. La réalisation des opération relationnelles étendues
5.5.9.1. La restriction spatiale
5.5.9.1.1. La restriction spatiale par fenêtre d'étude
5.5.9.1.2. La restriction spatiale par domaine
5.5.9.2. La jointure spatiale
5.5.9.3. Les semi-jointures spatiales
5.5.10. La connexion à d'autres systèmes de gestion de base de données
5.6. Conclusion

6. Géoréférencement et intégration d'images ……. 195

6.1. Introduction
6.2. Définition d'une classe d'objet pour les données venant d'images : le pixel
6.2.1. Définition de l'objet
6.2.2. Gestion des collections de pixel
6.3. Géoréférencement des images et création des objets
6.3.1. Redressement d'image
6.3.2. Le choix des points d'appui
6.3.3. Le ré-échantillonnage
6.3.4. Le mosaïquage et l'intégration dans une relation
6.3.4.1. Les méthodes de correction géométriques pour les mosaïques
6.3.4.2. Les méthodes de correction descriptives pour les mosaïques
6.3.4.3. L'assemblage des images
6.3.5. Un cas particulier : les satellites d'observation de la Terre
6.4. SAVANE et la gestion des mosaïques
6.4.1. Structure des relations de type mosaïque
6.4.2.Opérations relationnelles
6.4.3. Création de relations de type mosaïque et opérations de changement de type
6.4.4. Utilisation des mosaïques dans le cadre objet
6.4.4.1. La classe Cmosaïque
6.4.4.2. Quelques classes dérivées
6.5. Le module SAVAMER
6.5.1. Présentation générale

6.5.2. La prise de points d'appui
6.5.2.1. Principes généraux et interface graphique
6.5.2.2. La saisie manuelle des points d'appui
6.5.2.3. La saisie semi-automatique et automatique
6.5.3. Le redressement et le ré-échantillonnage
6.5.3.1. Le redressement
6.5.3.2. Le ré-échantillonnage
6.5.3.3. Un algorithme de triangulation
6.5.3.4. Le redressement descriptif des dynamiques
6.5.4. Le mosaïquage et l'intégration dans une mosaïque
6.6. Conclusion

7. Les contraintes d'intégrités spatiales et la saisie graphique ……. 227

7.0. Introduction
7.1. La qualité dans les bases de données géographiques
7.1.1. Le problème général de la qualité dans les bases de données géographiques
7.1.2. La cohérence spatiale
7.1.3. Les contraintes d'intégrité spatiales
7.2. Les contraintes d'intégrité pour les bases de données
7.2.1. Le cadre relationnel
7.2.2. Les contraintes d'intégrité
7.2.3. Les contraintes d'intégrité pour les données localisées
7.3. Les contraintes d'intégrité spatiale
7.3.1. Une typologie des contraintes d'intégrité spatiale
7.3.2. Les contraintes géométriques
7.3.3. Les contraintes topologiques de type
7.3.3.1. Fermeture et centroïde
7.3.3.2. Connexité
7.3.3.3 Surface et périmètre
7.3.4. Les contraintes relationnelles
7.3.4.1. Contrainte d'unicité de clé
7.3.4.2. Contrainte d'appartenance à un domaine
7.3.4.3. Contraintes descriptives de voisinage
7.3.4.4. Contraintes topologiques
7.3.4.5. Contraintes métriques
7.3.5. Les contraintes de jointure
7.3.5.1. Les relations spatiales entre objets géographiques
7.3.5.2. Les contraintes topologiques
7.3.5.3. Les contraintes descriptives
7.4. Modèles géométriques et mise en œuvre des contraintes d'intégrité spatiale
7.4.1. Les différentes méthodes de stockage
7.4.1.1. Les modèles spaghetti
7.4.1.2. Les modèles topologiques
7.4.2. La vérification de la cohérence spatiale
7.5. SAVANE, saisie graphique et contraintes d'intégrité spatiale
7.5.1. Présentation du module SAVEDIT
7.5.1.1. Les principes de la saisie graphique avec SAVEDIT
7.5.1.2. Saisie et indexation géographique
7.5.1.3. La saisie de points
7.5.1.4. La saisie de lignes

7.5.1.5. La saisie de zones
7.5.1.6. Saisie, datum et projection géographique
7.5.1.7. Le format des documents SAVEDIT
7.5.2.Les contraintes sur les objets géométriques
7.5.2.1. La classe CArc
7.5.2.2. Nettoyage des arcs
7.5.2.3. Simplicité
7.5.2.4. Extra-simplicité
7.5.2.5. Fusion d'arc
7.5.2.6. Suppression d'arcs en double
7.5.2.7. Union d'arcs
7.5.3.Les contraintes sur les objets géographiques
7.5.2.1. Le nettoyage des zones
7.5.3.2. La fermeture de zones
7.5.3.3. Connexité et fermeture des arcs
7.5.3.4. Unicité de clé (zones sécantes, zones incluses)
7.5.3.5. Centroïde de zones
7.5.3.6. Valeurs adjacentes (courbes de niveaux)
7.5.3.7. Valeurs adjacentes (points cotés)
7.5.4. Quelques procédures automatiques : algorithmes
7.5.4.1. La détection automatique des contours de zone
7.5.4.2. La saisie semi-automatique d'arc par suivi de contour
7.5.4.3. Le calcul automatique de centroïde de zones
7.5.4.4. La création de taches fermées à partir d'arcs frontières
7.6. Conclusion

Troisième partie : analyse spatiale et représentation cartographique

8. Analyser : calculs et analyse spatiale ……..…. 267

8.1. Introduction
8.2. Rappels sur l'analyse spatiale
8.2.1. Structures spatiales et objets géographiques : généralisation et agrégation
8.2.2. Distribution spatiale d'un phénomène
8.2.3. Homogénéité spatiale, relations de voisinage, regroupement
8.3. SAVANE : exploration des données et statistique
8.3.1. L'exploration individuelle
8.3.2. L'exploration statistique
8.4. SAVANE et les méthodes sur les attributs descriptifs
8.4.1. Le principe de la création de nouveaux attributs descriptifs
8.4.2. Calcul numérique et logique
8.4.3. Classifications et méthodes de discrétisation
8.4.4. Agrégations descriptives
8.4.5. Comparaison d'attributs
8.4.6. Combinaison d'attributs
8.4.7. Ordre
8.5. L'utilisation de la localisation pour la création de nouveaux attributs descriptifs
8.5.1. Surface, périmètre, coordonnées
8.5.2. Orientation

8.5.3. Distance à un point, à une relation, à un masque
8.5.4. Géo-agrégations
8.5.4.1. La géo-agrégation géométrique par distance
8.5.4.2. Géo-agrégation par surface d'un masque
8.5.4.3. Géo-agrégation par valeur pondérée par la surface
8.5.5. Appartenance
8.5.6. Géo-agrégation sur les pixels : le ré-échantillonage
8.5.7. Opérations sur les modèles numériques de terrain : orientation, pente, écoulement
8.5.7.1. Pente et orientation
8.5.7.2. D'autres indices en géomorphologie et hydrologie
8.6. La création de relations temporaires
8.6.1. Le principe de la création de nouvelles relations
8.6.2. La duplication
8.6.3. La définition directe sur écran ou par masque
8.6.4. La création de mailles
8.6.5. La création par changement de type d'objet
8.7. Les opérations de changement de type d'objet
8.7.1. Vers le pixel : interpolation ou rasterisation
8.7.1.1. Mosaïque et type « Raster »
8.7.1.2. Méthodes d'interpolation
8.7.1.2.1. La méthode barycentrique
8.7.1.2.2. Barycentre sur tous les points
8.7.1.2.3. Interpolation sous contrainte
8.7.1.3. Zone vers pixel : rasterisation
8.7.2. Vers la zone : vectorisation ou tessélation
8.7.2.1. Un algorithme de vectorisation
8.7.2.2. Un algorithme de tessélation
8.7.2.3. De la ligne à la zone : créer la topologie
8.7.2.4. De la zone à la zone : regrouper, éroder, dilater
8.7.3. Vers la ligne
8.7.3.1. Squelettisation de zones
8.7.3.2. Vectorisation et courbes d'isovaleurs
8.7.4. Vers le point
8.7.4.1. Centre d'un semis de points
8.7.4.2. De la ligne aux points : l'utilisation des nœuds
8.8. Conclusion

9. Représenter …… ……... 315

9.1. Les principes de la représentation cartographique
9.1.1. Le langage cartographique
9.1.2. Les variables visuelles
9.1.2.1. La forme
9.1.2.2. La taille
9.1.2.3. La couleur
9.1.2.4. La valeurs
9.1.2.5. Grain, trame, texture
9.1.2.6. Orientation
9.1.3. La représentation cartographique

9.1.4. Légendes et habillage
9.2. La cartographique avec SAVANE : cartes et cadres
9.2.1. La carte
9.2.2. Les cadres
9.2.3. Couleurs et palettes
9.3. L'explorateur cartographique
9.3.1. Principes de l'explorateur cartographique
9.3.2. L'implantation graphique ponctuelle
9.3.3. L'implantation graphique zonale
9.3.4. L'implantation graphique linéaire
9.3.5. La représentation des images
9.3.6. Masques, fonds graphiques, documents digitalisés
9.3.7. Les procédures de tracé
9.3.8. Les légendes
9.4. La représentation en perspective
9.5. L'ouverture ou la sauvegarde d'une carte dans SAVANE

10. Conclusion ……..…..…... 343

10.1. La construction du SIG SAVANE : synthèse
10.2. Vers plus d'intelligibilité pour les bases de données géographiques

Bibliographie générale ……..……. 351

Annexe : exemples de classes et implémentation en C++ …….. 361

A.1. Savateca

A.1.1. Les classes CAttribut et CRelation
A.1.2. La classe CBase
A.1.3. La classe CDico
A.1.4. La classe CFpacc8
A.1.5. La classe CIndexAttribut
A.1.6. La classe CIntegrationObjetsLocalises

A.2. Savane

A.2.1. La manipulation de la base de données dans un cadre
A.2.1.1. Les classes CRelation et CAttribut
A.2.1.2. Les classe CSchema
A.2.1.3. La classe CLecture
A.2.1.4. La classe CMacro
A.2.1.5. La classe CCalculateur

A.2.1.6. La classe CBabel : interpolation
A.2.2. Algorithmique graphique
A.2.2.1. La classe CArc
A.2.2.2. Les classes CCellule et CYX
A.2.2.3. la classe CVectoriser
A.2.2.4. DilaterImage()
A.2.2.5. SegmentIntersection()
A.2.2.6. Les classes CZone et CTache
A.2.3. Fenêtre et projections
A.2.3.1. La classe CWind
A.2.3.2. La classe CProjection
A.2.3.3. La classe CMolodensky
A.2.4. Documents et cartographie
A.2.4.1. La classe CCarte
A.2.4.2. Les classes CODD
A.2.4.3. Les classes Cgroupe et CSelection
A.2.4.4. La classe CCadre
A.2.4.5. La classe CCartExplorateur
A.2.4.6. Les classes CDessin
A.2.4.7. La classe CLegende
A.2.4.8. La classe CPaletteSavane
A.2.4.9. La classe CPerspective
A.2.5. Exemples d'opérations dans un cadre
A.2.5.1. XQuestUnion()
A.2.5.2. XQuestJoinGeo()
A.2.5.3. XCrisCalcul()
A.2.5.4. XTri()

A.3. Savamer

A.3.1. La classe CAmers
A.3.2. La classe CSysteme1
A.3.3. La classe CTessel

A.4. Savedit

A.4.1. La classe CArc
A.4.2. La classe CSaveditDocument

Table des illustrations

fig. 2.1 : exemple de prise de points d'amers avec SAVAMER
fig. 2.2 : exemple de constitution de mosaïque d'images géo-référencées
fig. 2.3 : la saisie sur écran avec le module SAVEDIT
fig. 2.4 : l'écran principal de SAVANE avec une carte contenant un cadre
fig. 2.5 : une image satellite mise en perspective grâce à un modèle numérique de terrain
fig. 2.6 : l'écran principal d'une application de type SavAtlas
fig. 4.1 : coordonnées géodésiques, et projection d'un point sur un ellipsoïde de révolution
fig. 4.2 : un théodolite et son utilisation en géodésie
fig. 4.3 : déformation du géoïde terrestre, à l'échelle 10000
fig. 5.1 : les différents niveaux de description
fig. 5.2 : jointure entre zones et points sur la qualification d(O1,O2)=0
fig. 5.3 : jointure entre zones sur la qualification d(O1,O2)=0
fig. 5.4 : hiérarchie en feuilles et « region quadtree » associé
fig. 5.5 : découpage en k-d tree et arbre binaire associé
fig. 5.6 : le dialogue pour la gestion des relations
fig. 5.7 : le dialogue de création d'une relation
fig. 5.8 : le dialogue de gestion du schéma des attributs d'une relation
fig. 5.9 : le dialogue de création d'un attribut
fig. 5.10 : les dialogues de l'intégration graphique
fig. 5.11 : les différents dialogues de l'intégration descriptive
fig. 5.13 : structure de l'image raster associée à une relation zonale
fig. 5.14 : la création d'un masque à partir des points d'une relation et d'une distance à ces points
Fig. 5.15 : un masque résultat de la différence de deux masques
fig. 5.16 : dialogue de restriction par formule
fig. 5.17 : dialogue de restriction sur un attribut nominal
fig. 5.18 : une équi-jointure spatiale zone-zone
fig. 6.1. : le dialogue de création d'une relation mosaïque
fig. 6.2. : le dialogue de création d'un attribut d'une relation mosaïque
fig. 6.3 : structure interne des relations mosaïques
fig. 6.4 : un modèle numérique de terrain
fig. 6.5 : l'interface graphique du module SAVAMER
fig. 6.6 : exemple de prise de points d'amers dans une zone commune à deux images
fig. 6.7 : le dialogue de redressement de SAVAMER
fig. 6.8 : triangulation obtenue à partir des points d'appui
fig. 6.9 : le dialogue d'intégration d'une image dans une mosaïque
fig. 6.10 : le choix des pixels à intégrer : méthode par polynôme
fig. 6.11 : exemple de constitution de mosaïque d'images géoréférencées
fig. 7.1 : Simplicité des arcs
fig. 7.2 : Extra-simplicité des arcs
fig. 7.3 : Fermeture d'un ensemble d'arcs
fig. 7.4 : connexité d'un ensemble d'arcs
fig. 7.5 : test de simplicité sur un arc
fig. 7.6 : test de fermeture d'une zone

fig. 7.7 : visualisation de la fermeture par remplissage en couleur
fig. 8.1 : interrogation sur écran des objets présents dans le cadre géographique
fig. 8.2 : statistiques univariées sur un attribut d'une collection d'objets
fig. 8.3 : exemple de représentation des écarts par rapport à une distribution donnée
fig. 8.4 : le dialogue de création d'attribut par calcul numérique ou logique
fig. 8.5 : le dialogue de création d'un attribut d'une mosaïque
fig. 8.6 : le calcul d'un indice de végétation (NDVI) à partir des canaux 3 et 4 d'une image Thematic Mapper
fig. 8.7 : le menu de classification de SAVANE
fig. 8.8 : dialogue de définition des classes par quantiles
fig. 8.9 : le choix de l'opération d'agrégation descriptive
fig. 8.10 : dessin et histogramme de la répartition de l'orientation de failles géologiques, calculée à partir de la géométrie des objets
fig. 8.11 : exemple de géo-agrégation de points dans des zones : calcul de l'altitude moyenne par îlot à partir de points cotés (moyenne, d < 100 m)
fig. 8.12 : la page finale du dialogue de géo-agrégation par distance
fig. 8.13 : exemple de géo-agrégation de zone à zone sans ambiguité : agrégation de la population d'îlots aux quartiers (somme, d=0)
fig. 8.14 : un masque (cultures et paturages) et le pourcentage de la surface dans un découpage (cantons)
fig. 8.15 : calcul de la moyenne d'un indice radiométrique (TM, canal 1) dans des zones prédéfinies (quartiers) par géo-agrégation pondérée par la surface
fig. 8.16 : pente et orientation calculées à partir d'un modèle numérique de terrain
fig. 8.17 : Données ponctuelles, mailles, et géo-agrégation dans les mailles
fig. 8.18 : le dialogue du choix de l'opération d'interpolation
fig. 8.19 : résultat de l'interpolation barycentrique à partir de courbes de niveau
fig. 8.20 :exemple d'interpolation barycentrique sur tous les points de référence (classée par quantiles)
fig. 8.21 : vectorisation après classification d'une mosaïque (image TM)
fig. 8.22 : création de zones à partir de points : tessélation de Voronoï
fig. 8.23 : deux dilatations à partir des objets initiaux (5 m, 20 m). L'objectif est ici de diminuer ou de remplir l'espace interstitiel en ville
fig. 8.24 : création de zones par dilatation à partir de lignes ou de points
fig. 8.25 : Courbes de niveau initiales, interpolation, puis création de courbes de niveau par vectorisation (en rouge, superposées aux courbes initiales en bleu)
fig. 8.26 : Création de points centraux (en bleu) à partir d'un semis de points (provenant des îlots, en rouge) et de l'appartenance au quartier (attribut créé par géo-appartenance)
fig. 8.27 : occurrence des nœuds dans un réseau de lignes de bus
fig. 9.1 : une carte réalisée avec SAVANE
fig. 9.2 : la fenêtre principale de SAVANE est un éditeur graphique
fig. 9.3 : dialogue de propriétés d'un cadre géographique
fig. 9.4 : dialogues de définition d'une couleur et de choix dans une palette
fig. 9.5 : le dialogue principal de l'explorateur cartographique
fig. 9.6 : les trois possibilités de figurés en implantation ponctuelle : symboles, valeurs, diagrammes sectoriels
fig. 9.7 : la représentation par symbole selon deux caractères
fig. 9.8 : la représentation par diagramme sectoriel suivant trois attributs descriptifs

fig. 9.9 : les dialogues pour le tracé en implantation zonale
fig. 9.10 : propriétés de dessin pour les relations de type ligne
fig. 9.11 : représentation d'un mnt par valeur ou par illumination
fig. 9.12 : différents types de légendes générées à partir de l'explorateur cartographique et des dialogues de propriétés de légende
fig. 9.13 : dialogues pour créer une vue en perspective
fig. 9.14 : représentation en perspective cavalière des valeurs d'altitude d'un mnt

Introduction

Les systèmes d'information géographique (SIG) regroupent différentes méthodes et techniques informatiques, permettant de modéliser, de saisir sous forme numérique, de stocker, de gérer, de consulter, d'analyser, de représenter des objets ou des collections d'objets géographiques, avec la particularité essentielle de prendre en compte les caractéristiques spatiales de ces objets au même titre que les attributs descriptifs qui y sont attachés. En fait, la dénomination « SIG » recouvre une grande variété de réalisations logicielles construites suivant des choix techniques différents, aux fonctionnalités et aux performances très diverses.

Les systèmes d'information géographique ont la particularité de faire appel à de nombreux domaines scientifiques et techniques et à de nombreuses méthodes, allant de la géodésie aux systèmes de gestion de bases de données, en passant par le traitement d'images, l'algorithmique géométrique, la modélisation et l'interpolation géométrique, la statistique, la cartographie automatique, l'analyse spatiale, etc. Construire un système d'information géographique sans s'éloigner de la rigueur scientifique est une tâche complexe, aussi bien en terme de définition des concepts, d'organisation fonctionnelle, d'architecture logicielle, d'algorithmique, d'ergonomie.

Cet ouvrage présente un travail de recherche et de développement informatique visant à apporter une réponse concrète à la question suivante : « comment construire un système d'information géographique complet et opérationnel, en suivant les principes théoriques de la gestion de données et en les adaptant aux données géographiques ». Il a pour vocation de décrire l'ensemble des connaissances résultant de cette expérience, ainsi que de présenter les principes du système informatique opérationnel qui en résulte, que l'on trouvera dans le CDROM accompagnant ce livre avec des exemples de base de données permettant de mettre en œuvre les exemples présentés dans l'ouvrage.

L'objet de cet ouvrage est donc de montrer sur un exemple concret l'implication des principes théoriques dans la réalisation pratique d'un logiciel de type SIG. Il a pour objectif de présenter de façon complète et détaillée l'architecture et la

réalisation d'un système d'information géographique opérationnel – le système SAVANE - à partir des nombreux concepts, techniques et algorithmes nécessaires à cette réalisation, d'en expliquer les choix informatiques et les contraintes qui en découlent, en fonction des objectifs recherchés. Travail de recherche, de synthèse, d'architecture, d'algorithmique, et de développement logiciel, l'aspect étudié réside plus dans la synthèse des principes théoriques, dans la recherche de l'architecture d'un ensemble complet et opérationnel - à la fois système de gestion de données et programme d'application - que dans l'étude d'un thème particulier de cet ensemble.

Cet ouvrage reprend la démarche que nous avons suivie tout au long de ce travail, à savoir :

- exposer avec précision tous les aspects concernant la définition et l'utilisation de l'information géographique ;

- exposer et développer les principes de la gestion de données (modèle relationnel et objet) pour les étendre aux données localisées ;

- développer ou mettre en œuvre les algorithmes nécessaires à l'implantation de ces principes dans un système d'information ;

- construire un système opérationnel, mettant en œuvre les principes théoriques et répondant à un cahier des charges fonctionnel couvrant l'ensemble des besoins nécessaires à l'utilisation de ce système dans le cadre de projets appliqués, notamment en géographie et dans le cadre de la recherche pour le développement.

En effet, et c'est une autre caractéristique de ce travail, il a été effectué dans un cadre opérationnel et non dans un laboratoire de recherche classique. Le cadre général est celui de l'IRD (Institut de recherche pour le développement, anciennement ORSTOM), et le cadre opérationnel est celui de projets de recherche ou de développement dans divers pays ou régions de la zone tropicale. Notre objectif a donc toujours été double : concilier le développement ou l'application de principes théoriques pour la gestion de données géographiques et la construction d'un logiciel opérationnel dans le cadre de projets de recherche ou de développement. Le cahier des charges fonctionnel a été fortement influencé par cet impératif, le système à construire devant répondre à plusieurs objectifs parmi lesquels on peut notamment souligner :

- la nécessité d'une gestion efficace, et donc la construction d'un moteur BD relationnel interne étendu aux données localisées (gestion intégrée des attributs descriptifs et de localisation), avec indexation primaire sur la localisation, en plus de procédures de connexion à des moteurs BD externes ;

- la nécessité d'un système pouvant gérer un nombre important d'objets (typiquement de plusieurs millions) sans dégradation de performance, dans un cadre opérationnel ;

- la nécessité de conserver la meilleure précision possible en fonction de la modélisation de la réalité en objets géographiques, ce qui implique un système à double structure interne (vecteur, *raster*) permettant la gestion intégrée de l'imagerie aérienne et satellitale et de la troisième dimension ;

- la nécessité d'une gestion souple et de fonctionnalités évolutives, et donc la mise en œuvre de l'approche objet avec la gestion de méthodes, aussi bien au niveau du schéma des données que du développement et de l'implémentation ;

- la nécessité d'assurer la pérennité des bases de données, et donc la réflexion sur la documentation de l'information et la gestion des méta-données dans une approche orienté-objet ;

- la nécessité d'une fiabilité complète au niveau de la saisie des données, et donc la définition et la mise en œuvre concrète de multiples contraintes d'intégrité spatiale lors de la saisie graphique ;

- une ergonomie permettant l'approche exploratoire et empirique multi-utilisateur, et donc la gestion d'états temporaires par utilisateur sans modification de la base de données ;

- la nécessité de fonctionnalités intégrées permettant l'analyse, et donc la définition et la mise en œuvre d'opérations propres aux données géographiques : analyse spatiale, statistique, géostatistique ;

- des fonctionnalités de dessin et de cartographie automatique permettant d'aboutir à des produits d'édition cartographique professionnels.

Nous allons dans cet ouvrage reprendre l'ensemble des travaux effectués, en décrivant chaque aspect nécessaire à la construction du système et en expliquant les choix effectués. Nous rappellerons les principes de base, nous développerons certains domaines lorsque la théorie actuelle n'est pas adaptée aux types de données traités. L'exposé des méthodes sera souvent suivi de la présentation d'algorithmes et de leur réalisation concrète dans le système d'information géographique que nous avons développé, de manière à construire peu à peu un système opérationnel complet. Des références renverront fréquemment le lecteur à une annexe contenant l'implantation effective des structures et des algorithmes, directement en langage de programmation (C++).

Il se divise en plusieurs parties :

- première partie : Introduction. Rappels sur les SIG et présentation générale des différents modules du système SAVANE ;

- seconde partie : modélisation, mesure, gestion et SGBD, géo-référencement et intégration d'images, contraintes d'intégrité spatiale et saisie ;

- troisième partie : exploitation des données, requêtes, analyse spatiale et représentation cartographique.

La première partie donne un panorama général des SIG, puis présente le système SAVANE. Après une brève introduction sur les objectifs généraux des SIG, nous rappellerons rapidement l'histoire du développement des systèmes d'information géographiques, ainsi que les fonctionnalités actuellement rencontrées dans les systèmes du commerce. Le second chapitre présente de façon générale le système SAVANE : architecture générale, fonctionnement et possibilités des différents modules.

La seconde partie expose les caractéristiques des données géographiques : modélisation de la réalité, représentation informatique, modes d'acquisition, précision et qualité, description et documentation, méta-données. Nous détaillerons le mode de représentation utilisé dans le SIG SAVANE pour tous les objets géographiques. Nous présenterons l'aspect base de données dans les SIG, en commençant par rappeler l'ensemble de la théorie des SGBD relationnels et objets. Nous proposons ensuite des extensions du modèle de gestion relationnel et objet aux données localisées, et nous présenterons l'implémentation de ces nouvelles opérations dans SAVANE. Nous présenterons de nombreux algorithmes géométriques concernant la saisie et le contrôle de la qualité des données géométriques, le redressement d'images, les jointures géométriques, etc.

La troisième partie concerne les méthodes utilisant plus spécialement la localisation des objets dans les SGBD géographiques. Il s'agit de l'analyse spatiale dont nous donnerons de nombreux exemples et algorithmes de résolution (traitements vectoriels, traitements matriciels, distances et proximité, changements d'implantation spatiales, modèles numériques et interpolation, etc.). Nous présenterons enfin les modules de cartographie automatique de SAVANE.

L'annexe regroupe les principales structures et algorithmes du système. Ils seront données, lorsque cela est possible et facilement compréhensible par le lecteur, directement en langage de programmation.

Un système d'information géographique répondant à nos objectifs est la fois un système de gestion de données et un programme d'application, et c'est bien ce mélange des genres qui en fait la complexité, au-delà du caractère particulier dû à la dimension géométrique des données à gérer et à traiter. En tant que programme d'application, le système construit n'est pas complet, certains domaines n'étant pas ou peu abordé (l'analyse de réseaux et la recherche opérationnelle par exemple). Il n'a d'ailleurs pas vocation à l'être, mais sa structure évolutive devrait lui permettre de continuer à se développer, et c'est d'ailleurs un de nos principaux objectifs initiaux : fournir la base logicielle au développement de nouvelles fonctionnalités - aussi bien dans un cadre universitaire que dans un environnement

opérationnel - et implémenter ces fonctionnalités en tant que méthodes, soit directement dans le système, au niveau des classes de bases, soit en tant que classes d'objets dérivées qui seront à la disposition du concepteur de base de données. Nous donnerons, en conclusion, quelques éléments de réflexion sur ce thème.

Première partie : introduction et présentation du système SAVANE

Chapitre 1

Présentation générale des systèmes d'information géographique

Depuis plus de vingt ans, le développement de l'informatique a entraîné des modifications importantes pour la géographie et la cartographie. La production de données s'est accélérée, grâce à de nouvelles méthodes de collecte et d'acquisition. Le traitement des données localisées s'est largement développé, avec la saisie numérique des données graphiques, cartes et plans, avec les systèmes de gestion de bases de données et les capacités de stockage des systèmes informatiques. Enfin, de nombreux aspects de la cartographie ont été automatisés et les techniques de production complètement modifiées, avec en corollaire une accélération de la diffusion et de l'utilisation de données géographiques.

Un système d'information géographique (SIG) est avant tout un système de gestion de base de données capable de gérer des données localisées, et donc capable de les saisir, de les stocker, les extraire (et notamment sur des critères géographiques), de les interroger et analyser, et enfin de les représenter et les cartographier. L'objectif affiché est essentiellement un objectif de synthèse, permettant à la fois la gestion des données comme l'aide à la décision.

Si l'informatique a d'abord permis des progrès dans l'automatisation de la production cartographique, les SIG vont bien au-delà d'une simple fonction de stockage et de restitution graphique. Par leurs possibilités de modélisation et de gestion, par leurs fonctions d'analyse et d'interrogation, par les possibilités de mises en relation des objets les uns par rapport aux autres, par leurs capacités à stocker et traiter de gros volumes d'information, les SIG ont profondément bouleversé les méthodes traditionnelles d'analyse et de gestion de l'espace. Grâce aux possibilités

de modélisation et de calcul, l'informatique et les SIG n'ont pas seulement permis l'amélioration de techniques existantes, ils ont remis en cause bon nombre de concepts classiques de la géographie et renouvelé la dynamique de cette discipline.

1.1. L'apport de l'informatique pour la géographie et la cartographie

L'informatique intervient depuis plusieurs dizaines d'années dans beaucoup de domaines et, depuis plus de vingt ans, son développement a entraîné des modifications importantes pour la géographie et la cartographie. Parmi les secteurs qui nous intéressent, on peut citer la gestion de données, le stockage numérique, la statistique, de nouvelles formes d'expression et de communication. De plus, de nouveaux moyens d'acquisition de données se sont développés : la télédétection spatiale et le positionnement par satellite en sont les principaux exemples. La géographie et la cartographie se sont construites peu à peu sur la base de possibilités techniques qui, pendant longtemps, n'ont pas évolué. Depuis une vingtaine d'années, beaucoup de ces fondements sont les uns après les autres remis en cause ou modifiés par des possibilités techniques inédites.

Chaque domaine a d'abord été touché, mais indépendamment des autres. Par exemple, les logiciels de dessin ont remplacé peu à peu des travaux manuels longs et fastidieux, des techniques de structuration et de gestion de données et de nouveaux moyens de stockage ont permis de mettre en œuvre de nouveaux moyens de traitement et d'analyse, notamment statistiques, des logiciels de présentation de données et d'images permettent d'envisager de nouvelles formes d'expression cartographique.

L'originalité des SIG, c'est d'essayer de réunir toutes les nouvelles techniques de traitement de données localisées, tous les nouveaux moyens d'expression dans un seul et unique environnement, décuplant en cela l'efficacité de chaque domaine et permettant de nouvelles avancées conceptuelles, impossible à concevoir dans la séparation des techniques : c'est donc aux fondements de la géographie qu'il faut retourner, pour ne pas conserver des limitations conceptuelles liées à des impossibilités techniques maintenant dépassées ou en passe de l'être. Cette remise en cause, cette renaissance conceptuelle ne peut être menée que dans le cadre des SIG, et c'est bien ce qui fait la force de ce courant, qui ne doit pas être conçu ou interprété uniquement sous l'aspect de l'avancée technique qu'il apporte : il doit fournir aux géographes et aux informaticiens l'occasion de réfléchir de nouveau sur l'espace géographique, sur la manière de le concevoir, de le traiter, de le représenter.

Mais le concept de SIG n'a pas été défini directement, au début des années 60. Il s'est construit peu à peu, au fur et à mesure de l'introduction de l'informatique dans

l'ensemble des méthodes liées à l'analyse et à la représentation de données spatiales. Nous pouvons faire rapidement l'historique de ce champ d'activité.

1.2. Les SIG comme synthèse de ces domaines : historique et évolution

L'historique du développement des SIG peut être divisé en trois périodes : les années 60-70, représentant les débuts et les premières réalisation, les années 80 pour la consolidation et l'apparition des premiers logiciels commerciaux, et les années 90 pour la diffusion générale des outils et de la technologie SIG.

1.2.1. Les débuts : les années 60-70

Les applications militaires et l'intérêt croissant des gouvernements pour la gestion des ressources sont à l'origine du développement des systèmes d'information géographique. Prenons par exemple le Canada, qui a développé l'un des premiers systèmes dans les années 1960 : pour la première fois, un état avait le sentiment que ses ressources naturelles n'étaient pas sans limites, et son gouvernement a perçu la nécessité d'intervenir au niveau de la planification et de l'utilisation des ressources. L'échelle utile pour une telle planification varie entre le 1:20000 et le 1:250000 : sur un pays de la taille du Canada, il fallait utiliser entre 200 et 3000 coupures de carte, pour chaque thème d'étude. Manuellement, le travail de lecture et d'analyse de l'information cartographique aurait pris au minimum trois ans et plus de cinq cent techniciens.

Dans les années 60, le concept même de SIG n'était pas encore développé et les méthodes de stockage et d'analyse d'un nombre important de cartes étaient inexistantes. Les ordinateurs avaient de faibles mémoires et des vitesses de calculs largement dépassées par les plus petits micro-ordinateurs d'aujourd'hui. L'informatique graphique en était à ses débuts, aussi bien pour le matériel que sur la théorie et les algorithmes [PAL 75]. C'est entre 1960 et 1970 qu'ont été développés les premières tables à digitaliser, les scanners, les tables traçantes, ainsi que les techniques de base de l'informatique graphique : algorithmique graphique, topologie informatique, traitement d'image et reconnaissance des formes, modélisation bi- ou tri-dimensionnelle. L'interactivité est alors inexistante , et les possibilités de restitution graphique très limitée. Les coûts sont importants : par exemple, un écran graphique - alors basse résolution- coûte environ 10000 USD en 1967.

La plupart des systèmes d'analyse spatiale développés pendant les années soixante et soixante dix utilisent des structures de données très simples, qui sont basés sur des grilles arbitraires, maillage de l'espace qui ressemble à ce qu'un géographe peut faire manuellement sur une petite surface [DAN 81]. C'est

principalement le coût élevé des tables à digitaliser qui est à l'origine de ce choix, surtout dans les universités, où certains étudiants devaient remplir à la main les mailles en fonction de la carte d'origine… SYMAP, MIADS, GRID, MLMIS, GEOMAP sont des exemples de systèmes de ce type. D'un autre coté, les systèmes de dessin automatisé se développent, mais sans la notion de base de données. C'est ainsi que se développent les premiers systèmes d'information urbain à la fin des années 60, et ils s'apparentent plus à des systèmes de dessins automatiques qu'à des systèmes de gestion de l'information (DIME, GRDSR) [DIM 70].

Dans les années qui suivent, la perception de la nécessité d'une gestion générale des ressources et des potentialités est de plus en plus répandue. Elle implique des moyens de traitement de l'information évolué, notamment dans le domaine de l'information spatialisée. D'autre part, les progrès de la technologie informatique sont importants et rapides ; l'interactivité se développe et les coûts des matériels graphiques sont en baisse, même s'ils restent encore élevés. Les systèmes de gestion de bases de données sont en pleine évolution. Le nombre de systèmes en développement est important, mais peu d'innovations voient le jour, la plupart des développements se font dans le domaine du dessin automatique ou de la cartographie automatique. Les universités sont désormais nombreuses à s'intéresser à ce domaine, et des sociétés commerciales commencent à voir le jour alors qu'elles étaient inexistantes durant les années 60 : ESRI, GIMMS, INTERGRAPH, COMPUTERVISION…

On ne parle pas encore de SIG proprement dit, et le développement est plutôt tourné vers les systèmes de dessin et de conception assistée par ordinateur, systèmes que l'on essaye d'adapter aux besoins de la cartographie. Le développement des matériels graphiques est d'ailleurs essentiellement dû au développement de la CAO (conception assistée par ordinateur), en pleine expansion et beaucoup plus porteur à l'époque que celui de l'information géographique. Ce type de matériel reste néanmoins très coûteux.

Si le développement logiciel est important, le nombre des systèmes d'information géographique couvrant de larges territoires reste très faible. Les systèmes sont conçus soit pour stocker, soit pour gérer, soit pour traiter, mais aucun n'est encore vraiment capable non seulement d'archiver, mais aussi de gérer et de traiter un important volume d'informations spatialisées. Les techniques évoluées de gestion de données sont d'ailleurs peu développées, et les modèles utilisés conviennent mal aux données géographiques. Par contre, l'algorithmique graphique, la reconnaissance des formes, le traitement d'image sont des secteurs en plein développement en sciences de l'information.

Le début des années 70 va également voir le développement rapide de la télédétection spatiale et des méthodes de traitement d'image associées. Ces

technologies restent alors séparées des SIG, mais elles vont contribuer à favoriser l'essor général de l'infographie. Il faudra néanmoins attendre les années 80 pour voir s'esquisser un rapprochement entre la télédétection spatiale et les SIG, par une intégration réciproque d'information de sources multiples.

1.2.2. La consolidation : les années 80

Les années 80 sont une période de développement des méthodes de gestion de données. La saisie et la gestion de larges bases de données sont maintenant les principaux problèmes théoriques et pratiques. Les problèmes dus au volume des données sont souvent mal perçus ou même ignorés par les administrateurs : un système contenant une dizaine de cartes sera bien différent d'un système contenant 5000 coupures ou plus. Voici quelques chiffres donnant une idée du volume de données de SIG potentiels :

- les 359 coupures de carte de l'United States Geological Survey (utilisation des sols) contiennent approximativement 68 millions de points,

- les cartes topographiques sont beaucoup plus riches : il existe environ 40000 coupures de la carte topographique de l'USGS au 1:25000, ce qui représente environ 4000 millions de points.

Pour réaliser de telles bases de données, il faudra développer des méthodes de gestion et d'organisation de données de plus en plus efficaces. Le début des années 80 a vu une avancée importante dans les techniques de gestion de données : les systèmes de gestion à schéma relationnel sont plus performants, ils structurent l'information de façon plus rigoureuse et améliorent la résolution des problèmes d'administration de données. Dans ces systèmes, les relations spatiales entre entités géographiques sont malheureusement inexistantes, aussi bien sur le plan théorique que sur le plan pratique : la spatialisation n'est pas gérée en tant que telle, et les concepts théoriques ne sont pas adaptés aux données multidimensionnelles, si bien qu'il est impossible de les faire entrer dans le schéma des systèmes de gestion de bases de données relationnelle classique. De nombreux efforts seront donc entrepris pour étendre le modèle relationnel aux données géographiques, mais il faudra attendre la prochaine décennie pour voir les réalisations pratiques de ces développements [FRA 81] [DAN 81]

Parallèlement, l'interactivité graphique se développe. Les stations de travail apparaissent, et sont parfaitement adaptées à ce type d'application. Le matériel graphique (traceur, imprimante couleur, table à digitaliser, scanner) reste néanmoins encore coûteux, et le SIG n'est pas encore une application disponible sur micro-ordinateur et accessible du grand public.

1.2.3. La diffusion : les années 90

Les années 90 vont transformer ce panorama, et permettre une large diffusion de la technologie. Les micro-ordinateurs, de plus en plus puissants, remplacent peu à peu les stations de travail. Les accessoires graphiques deviennent également accessibles au grand public. L'offre commerciale s'étoffe, avec des logiciels plus complets et moins chers. On assiste donc à une large démocratisation de l'usage des SIG, avec de nombreux projets qui se développent sans trop de difficultés du fait du modeste volume de données traitées. Mais les méthodes n'évoluent pas beaucoup : la plupart des logiciels commerciaux sont l'émanation directe de produits conçus au début des années 80. Les logiciels commerciaux les plus répandus sont d'ailleurs parfois trop simples, et cherchent à ne pas emmener l'utilisateur non averti sur des chemins trop complexes… Mais la baisse des coûts des matériels de traitement et de restitution va peu à peu mettre le SIG à la portée de tous [WOR 95].

1.2.4. Les évolutions actuelles

Le développement de l'Internet pousse l'ensemble des produits commerciaux à offrir une solution pour l'interrogation du SIG et la conception de cartes via Internet. Mais cette offre met l'accent sur une consultation simple, au dépend de procédures d'analyse plus complexes. La gestion du temps est encore peu effective dans les SIG, même si le cadre théorique est bien posé. Le problème des multi-représentations spatiale d'un même objet doit également faire l'objet de développements, au niveau de l'implémentation des objets comme au niveau des contraintes d'intégrité. La géostatistique se démocratise peu à peu, même si elle reste encore sous-utilisée dans de nombreux domaines d'application.

L'évolution vers des SIG 3D est également sensible, avec des techniques de représentation et de visualisation qui suivent les capacités de matériels graphiques en forte évolution. L'introduction de méthodes issues de la vision par ordinateur, de la reconstruction 3D, de techniques d'animation devrait fortement pousser ce secteur en pleine évolution.

1.3. Les SIG : objectifs généraux

Avant de présenter les principes et fonctionnalités du système SAVANE que nous allons décrire tout au long de cet ouvrage, nous dressons un panorama général des principaux objectifs des systèmes d'information géographique [TOM 76] [DAN 83] [LAU 93] [SCHO 96] [SOU 86].

- Saisie et stockage numérique de plans et de cartes :

Le premier et principal objectif des SIG reste le stockage numérique de données géographiques, bi- ou tridimensionnelles. Mais il y a beaucoup de différences entre un système qui va conserver des objets, avec une description aussi bien graphique que descriptive, et un système qui va seulement conserver un dessin sans contenu sémantique.

- Structuration de l'information :

Comme tout système de gestion de bases de données, un SIG qui gère une base de données demande une modélisation du monde réel et une structuration de l'information. Cette structuration est souvent plus complexe, car elle touche des objets qui peuvent avoir de multiples représentations, aussi bien graphiques que descriptives, essentiellement en fonction de l'utilisation qui en sera faite.

- Calculs métriques (distances, surfaces), calculs techniques (visibilité, volumes, recherche opérationnelle), positionnement et projections géographiques :

Les SIG permettent de calculer facilement surfaces, distances et volumes à partir des données de localisation des objets. Les calculs et les changements de projections géographiques sont facilement accessibles. La recherche opérationnelle (essentiellement calculs de chemins dans des graphes) trouvent dans les SIG toutes les données dont elle a besoin.

- Gestion et traitement des collections d'objets :

C'est l'un des objectifs principaux des SIG. Une fois l'information structurée, elle doit être saisie et gérée par le système. Souvent, les SIG laissent la gestion des données descriptive à des SGBD relationnels classiques (comme ACCESS, ORACLE, SQL Server, DBase, etc.), et ne gèrent eux-mêmes que la localisation des objets et les liens entre graphique et description. Comme tout système de gestion de base de données, le SIG doit assurer la bonne gestion des flux d'informations, des modifications, des mises à jour, et notamment pour la partie graphique des objets.

- Gestion administrative et partage de données entre utilisateurs :

Lorsque les données sont partagées entre plusieurs utilisateurs, comme c'est souvent le cas pour les applications administratives de type cadastre, le SIG a pour objectif de gérer ce partage et d'optimiser l'accès des données entre utilisateurs.

- Gestion et analyse spatiale :

Les SIG ont vocation à gérer tout type d'objet géographique, du point au pixel, en passant par les zones, les réseaux, etc. L'objectif à atteindre est la constitution d'une base de données géo-référencées, permettant la mise en relation des différents objets de la base, quels que soient les types de ces objets. Cette mise en relation doit

permettre l'analyse spatiale, c'est-à-dire la prise en compte de la localisation dans l'analyse des données. De nombreuses procédures faisant appel à la localisation des objets sont donc implantées dans les SIG (sélections d'objets sur des critères de distances, recherche opérationnelle, agrégations spatiales et changements d'échelle, géo-jointures, interpolations, vectorisations, classifications par proximité, etc.).

- Gestion spatio-temporelle :

L'introduction du temps dans les SIG permet d'effectuer des interrogations mêlant espace et temps, de manière à pouvoir gérer à la fois l'historique d'un objet et l'état d'un ensemble d'objet à une date donnée. Les SIG ont donc également vocation à gérer les évolutions des objets géographiques. Mais les réalisations concrètes sont peu répandues, car la gestion de l'historique des modifications de la localisation d'un grand ensemble d'objets est complexe, aussi bien du point de vue informatique que de celui de la gestion des flux d'informations.

- Statistique et géostatistique :

La constitution d'une base de données géographiques a souvent pour objectif l'étude d'un territoire dans toutes ses composantes, et le SIG doit alors permettre l'accès facile au calcul statistique, qu'il soit exploratoire ou méthodologique. Certains SIG comportent un module statistique, d'autres gèrent l'interface avec un logiciel spécialisé. L'utilisation de méthodes de la géostatistique doit également être l'un des objectifs du SIG, puisqu'en gérant la localisation, il facilite considérablement l'utilisation de ces méthodes d'analyse ou d'interpolation spatiale.

- Simulation et modélisation :

L'objectif d'un SIG peut également être l'utilisation d'un modèle pour la simulation d'un processus. Le SIG doit alors faciliter l'interface entre le programme de modélisation ou de simulation et la base de données géographiques, et doit prendre en charge l'ensemble de l'accès à l'information spatiale dont a besoin le programme d'application.

- Télédétection, géo-référencement et traitement d'image :

Les SIG ont vocation à gérer tout type d'objet géographique. La télédétection aérienne ou spatiale offre une source privilégiée de données géographiques. Les SIG doivent donc également gérer et traiter de type de données, souvent volumineuses. Ils doivent en assurer le bon géo-référencement, permettre l'accès aux traitements propres à ce type de données, et permettre leur mise en relation avec l'ensemble des autres données localisées gérées par le système.

- Dessin et édition cartographique, cartographie automatique, 3D :

Comme tout système de gestion de données, les SIG ont pour objectif l'édition des données résultats d'une requête. Cette édition est souvent graphique puisque l'on traite de données localisées. Les modules de cartographie automatique à partir des données gérées par le système sont donc fondamentaux pour l'utilisateur. De plus en plus, les systèmes intègrent la troisième dimension, et permettent l'édition de données en perspective. Mais la saisie et la maintenance de la troisième dimension est plus complexe.

- Internet et accessibilité distante :

L'Internet offre depuis plusieurs années de nouvelles perspectives d'accès distant aux données. Les SIG doivent donc permettrent cet accès, en gérant la complexité de structure de l'information localisée, de manière à fournir aux utilisateurs des méthodes simples de consultation et de cartographie via Internet.

1.4. Conclusion

Commencé il y a maintenant plus de 30 ans, l'introduction de l'informatique dans le champ de la géographie a profondément renouvelé la dynamique de cette discipline. Ce renouvellement n'est pas dû au seul développement des SIG : les nouveaux moyens de mesure ou d'observation de la Terre y participent aussi largement. La dénomination « SIG » recouvre une grande variété de réalisations logicielles construites suivant des choix techniques différents, aux fonctionnalités et aux performances très diverses.

Le lien entre développement de nouvelles technologies et émergence de nouveaux paradigmes de recherche est indéniable. Et inversement, le questionnement méthodologique se nourrit en permanence de l'application thématique des méthodes et des outils qu'elle conçoit. Toutes ces technologies vont dans le même sens : celui d'une prise en compte toujours plus importante de la localisation dans la gestion ou l'analyse des phénomènes, à des précisions qui permettent de développer de nouveaux objectifs de recherche et d'améliorer considérablement les résultats obtenus dans les études qui en découlent.

Bibliographie

[DAN 81] DANGERMOND J., *Some trends in the evolution of GIS technology*, Marble Ed., 1981, Kensington Workshop, p. 25-57

[DAN 83] DANGERMOND J., *Classification des éléments logiciels utilisés habituellement dans les système d'information géographique*, fascicule n° 96 du comité français de cartographie, 1983, p. 7-20

[DIM 70] DIME, *Technical description of the DIME System*, U.S. Bureau of Census : *Census and study, the DIME Geocoding System*, Report n° 4, Washington D.C., 1970, p. 25-30

[FRA 81] FRANCK A., *Application of DBMS to Land information systems*, CH 1701-2, IEEE 81, 1981, p. 448-453

[GUP 95] GUPTILL S.C., MORRISON J.L., *Elements of Spatial Data Quality*, Pergamon/Elsevier Science, 1995

[LAU 93] LAURINI R., MILLERET-RAFFORT F., *Les bases de données en géomatique,* Paris, Ed. Hermès, 1993

[PAL 75] PALMER J.A.S., *Computer Science aspects of the mapping problem*, from Davis and Mc Cullagh, Display and analysis of spatial Data, New York, John Wiley and Sons, 1975

[SCH 96] SCHOLL M., VOISARD A., ET TOUS, *SGBD Géographiques, Spécificités*, Paris, Thomson Publishing France, 1996

[SOU 86] SOURIS M., *Systèmes d'information géographique et bases de données*, Colloques et Séminaires sur le Traitement des données localisées, Paris, Editions de l'ORSTOM, 1986, p. 29-87

[TOM 76] TOMLINSON, CALKINS AND MARBLE, *Essential part of a geographic information system*, Computer Handling of geographic Data, UNESCO Press, Paris, 1976

[WOR 95] WORBOYS M. F., *GIS, a Computer Perspective*, Taylor and Francis, 1995

Chapitre 2

Présentation du système SAVANE

Cette présentation du système SAVANE a pour objectif de décrire rapidement les principes généraux de sa conception et de situer ce logiciel dans l'ensemble des systèmes d'information géographique. Il présente les concepts généraux du système et ses spécifications fonctionnelles.

2.1. Le système SAVANE : principes généraux

Le système SAVANE présenté dans cet ouvrage est d'abord un système de gestion de base de données relationnelle étendu aux données localisées, avec des fonctionnalités de type objet. Son objet est de grouper, gérer, traiter, cartographier des données géographiques de diverses origines et différents types, comme des données d'enquêtes, des cartes thématiques, des données topographiques, des réseaux, des images satellites, des photographies aériennes, des modèles numériques de terrain, etc. Le système SAVANE possède les principales fonctionnalités des SIG : multiples sont les possibilités de croisement, de mise en relation, de regroupement, d'agrégation de données géographiques d'origines diverses. Il n'y a pas dans SAVANE de séparation entre la localisation et la description, entre le dessin et l'information : chaque objet est conservé avec tous ses attributs, qu'ils soient graphiques ou descriptifs : c'est le système qui gère l'ensemble. Le système SAVANE possède son propre système de gestion de base de données, de type relationnel étendu aux données localisées dans l'espace.

À la fois système de gestion de bases de données relationnelles, système d'analyse et d'aide à la recherche en géographie, système de cartographie et de dessin automatique, SAVANE va de la constitution d'une base de données à l'édition

: il offre ainsi la maîtrise de toute la chaîne cartographique de la conception de l'information au produit cartographique. Bien sûr, la constitution et l'exploitation d'une importante base de données localisées n'est pas chose simple et fait appel à de nombreux concepts, de la structuration et gestion de base de données à l'élaboration cartographique, en passant par la statistique, la géodésie, le traitement d'image.

Nous présentons dans ce chapitre une description rapide des principes et des fonctionnalités du système, sans rentrer dans la description de la structure interne ou des algorithmes de résolution, que nous présenterons tout au cours de cet ouvrage. Le système comprend cinq modules principaux (saisie graphique, géo-référencement d'images, administration des bases de données, exploitation des bases de données et cartographie, présentation d'atlas). Les bases de données géographiques sont gérées par un moteur de gestion de données bi-dimensionnel, avec indexation primaire basée sur la localisation. Nous présenterons dans les chapitres qui suivent l'architecture détaillée du système, la structure des bases de données, les méthodes de gestion et d'exploitation, ainsi que les principales fonctions, modules et algorithmes correspondants. Le logiciel est écrit en Visual C++ et fonctionne avec tout système d'exploitation Win32.

2.1.1. Les concepts généraux de SAVANE

Le sujet principal de notre travail, c'est de montrer comment construire un système répondant aux objectifs généraux des SIG tels que nous les avons présentés dans l'introduction de cet ouvrage, à savoir principalement : une gestion efficace d'objets localisés, une gestion évolutive, une ergonomie permettant l'approche exploratoire, des fonctionnalités permettant l'analyse spatiale, et des fonctionnalités de dessin et de cartographie automatique.

De nombreux choix peuvent être effectués pour construire un logiciel de type SIG, aussi bien pour l'architecture générale du système, pour les structures de données, pour les algorithmes de résolution, les méthodes d'indexation, de gestion, etc. Nous allons ici expliciter globalement ces choix, avant d'y revenir en détail au cours des chapitres qui suivent.

A l'origine de la construction du SIG, il y a deux objectifs majeurs : gérer des objets géographiques, en assurant la pérennisation et le partage de l'information, et avoir la possibilité de mettre ces objets en relation les uns avec les autres en utilisant leur localisation. Mettre en relation, comme pouvait le faire simplement le géographe lorsqu'il superposait deux cartes thématiques, pour vérifier ou mettre en évidence des corrélations spatiales. L'idée initiale est donc simple : gérer des objets tels qu'ils apparaissent sur des cartes, par thèmes indépendamment les uns des autres, et utiliser cette localisation, considérée comme universelle entre tous les

objets localisés, pour les relier les uns aux autres. Bien sûr, il faudra faire attention à conserver cette universalité lors de la mesure de la localisation : *datum* et projections cartographiques seront largement discutés au chapitre 4. Pour éviter tout problème lié au référentiel, nous avons choisi dans notre système de conserver les coordonnées indépendamment de la projection cartographique, et d'imposer un même *datum* pour l'ensemble des objets d'une base de données. C'est la solution la plus simple au niveau du système de gestion, mais elle requiert d'avoir les capacités de transformer toute coordonnée pour la ramener dans ce référentiel : il nous faudra donc développer changements de projection et changements de *datum*.

Gérer des objets suivant des collections thématiques, et pouvoir les mettre en relation grâce à un attribut, tout cela rappelle la gestion relationnelle pour les bases de données classiques. Les objets y sont gérés par collections d'objets de même type, et ils sont mis en relation sur un attribut commun par une qualification de jointure sur cet attribut. Le modèle relationnel permet d'assurer au mieux cohérence et intégrité d'une base de données, en laissant à des opérations de restriction ou de jointure le soin de répondre à une requête. L'idée de base de la construction du système que nous présentons, c'est d'étendre le mode de gestion relationnel aux objets localisés, avec la possibilité d'une jointure sur l'attribut donnant la localisation des objets pour les mettre en relation, entre objets de collections différentes. Dans une première étape, il faut analyser le problème de la représentation des objets géographiques dans une base de données, dans l'optique de les gérer en collections d'éléments décrits par les mêmes attributs. Car une carte représente effectivement un ensemble d'objet, à l'inverse de la description d'un paysage ou d'un quartier, qui, en général, ne représente qu'un seul objet et n'a pas pour objectif une étude comparative avec d'autres objets du même type. L'idée de la collection est donc omniprésente dans notre approche : si l'on cherche à regrouper les objets en collections décrits par les mêmes attributs, c'est pour pouvoir bien les gérer, les comparer entre eux, les comparer aux objets d'autres collections. La localisation dans l'espace doit être un élément fondamental de cette comparaison.

Partir du modèle relationnel pour la gestion de base de données permet d'utiliser un modèle qui donne satisfaction dans la gestion d'objets décrits par des attributs simples (nominaux ou numériques de dimension 1). Mais la prise en compte de l'attribut de localisation s'avère plus complexe, car des questions inédites se posent pour des attributs de dimension supérieure à 1 : comment décrire et représenter des sous-ensembles de dimension un ou deux ? Comment conserver la structure métrique de l'espace ? Est-il possible d'utiliser les systèmes existants, ou faut-il construire un nouveau moteur de gestion de données ? Faut-il stocker des cartes, ou revenir sur la schématisation et la modélisation qui a déjà été effectuée par le géographe ou le cartographe ?

Dans la conception du système que nous proposons, le point de départ de l'information reste la carte ou la mesure directe de terrain, comme d'ailleurs dans la grande majorité des SIG. Mais la carte n'intervient que comme support d'un ensemble d'objets correspondant à une même entité du monde réel, non plus cartographique mais géographique, dépouillée de toute représentation sémiologique. La modélisation ne retient que la schématisation géométrique utilisée pour rendre compte des phénomènes dans l'espace, avec, pour forte contrainte, l'unicité d'un phénomène dans l'espace et le temps : dans une collection, on ne peut avoir deux objets au même endroit et au même moment (si l'on prend en compte le temps). Le respect de cette contrainte est fondamental pour assurer la validité conceptuelle de la modélisation de la réalité. A partir de ce choix (une représentation géométrique des objets basée sur une description cartographique), plusieurs possibilités existent pour conserver la description de cette représentation géométrique dans un système informatique. Il est nécessaire de définir des types différents suivant le type de localisation, ponctuelle ou ensembliste, de dimension 1 ou de dimension 2. Déjà, on pressent qu'un système de gestion de base de données classique ne peut convenir pour stocker efficacement cette description, pour peu que l'on veuille l'utiliser dans sa structure d'espace métrique. Rien n'empêche d'utiliser des tables classiques pour stocker des points (x,y), mais un système classique sera incapable de reconstituer un contour, d'effectuer des tests d'appartenance, ou une interrogation sur la distance entre les objets : l'incapacité des SGBD classiques à gérer l'attribut de localisation dans sa structure est évidente. On peut alors séparer graphique et descriptif, utilisant un système particulier pour gérer l'attribut de localisation, et un système classique pour gérer l'ensemble des attributs descriptifs, en établissant un lien entre les deux systèmes par une clé de jointure pour reconstituer les objets. Cette solution ne nous satisfait pas, pour deux raisons essentielles : il est alors impossible d'indexer les objets sur leur localisation, alors que ce critère est de loin le plus discriminant, et la gestion séparée de deux ensembles d'attributs, correspondant à un même objet, est source d'incohérence dans les bases de données et de perte de performance dès que le nombre d'objet devient grand. Lorsque les objets sont localisés, l'attribut de localisation devient en effet le plus efficace en terme d'indexation : nous souhaitons mettre en œuvre une indexation de type séquentiel indexé basée sur la localisation, ce qui impose aux valeurs descriptives d'être stockées dans l'ordre des objets si l'on veut bénéficier des performances de l'indexation sans avoir à maintenir des index denses. Ceci ne peut se faire que dans un système gérant à la fois le descriptif et le graphique.

L'autre objectif principal du système que nous proposons de construire, c'est de pérenniser l'information, de façon centralisée, et d'en permettre la consultation et l'exploitation décentralisée, avec une démarche exploratoire. En effet, si l'interrogation d'un SGBD classique utilise habituellement un langage de requête permettant d'exprimer en une « phrase » la demande faite à la base de données, le processus d'interrogation de données géographiques et la démarche du géographe

relève bien souvent d'une approche exploratoire, sans la définition formelle du cheminement des opérations permettant de répondre à une question qui peut elle-même varier en fonction du cheminement. Il est essentiel de conserver cet aspect interactif dans le processus de requête, ce qui implique concrètement la création temporaire lors de l'interrogation de nouveaux attributs ou relations. Le système doit donc maintenir des états temporaires, par utilisateur : il ne pouvait être question de permettre la modification de la base de données par un utilisateur sous peine de remettre en question l'objectif fondamental de pérennisation et de centralisation de l'information.

Pour toutes ces raisons, nous avons choisi de construire un système possédant son propre moteur BD, permettant l'indexation primaire sur la localisation, permettant le stockage et la gestion de la géométrie représentant la localisation des objets, regroupés en relations, en fonction d'une modélisation de l'espace respectant le principe d'unicité de clé pour la localisation. Cette modélisation doit prendre en compte les différents types d'implantation spatiale : zone, ligne, points, pixels. Nous avons également choisi de séparer l'administration de l'exploitation : le module d'interrogation doit permettre l'approche exploratoire et la gestion d'états temporaires lors d'une requête, mais il ne peut pas modifier la base de données, droit réservé au module d'administration. Nous avons également choisi de séparer l'exploitation de la saisie graphique et de la vérification des contraintes d'intégrité. Pour le choix des structures internes de représentation de la localisation, nous privilégions systématiquement la simplicité des structures sur la simplicité des algorithmes de traitement : ainsi, nous choisirons de conserver des arcs frontières, quitte à avoir à reconstituer le contour des zones, plutôt que de conserver un contour fermé imposant un sens dans le stockage des arcs. Des structures plus simples pour la représentation de la localisation permettent de réduire les contraintes d'intégrité sur cet attribut.

A partir de la modélisation en collections thématiques, l'extension du modèle relationnel aux données localisées devient naturelle. Les opérations de l'algèbre relationnelle (restriction, projection, jointure) sont étendues à la localisation en utilisant des critères de distance ou d'appartenance, à la place d'une relation d'ordre ou d'égalité. Malheureusement, on s'aperçoit rapidement que cet attribut de localisation, que nous avions qualifié d'universel, ne l'est pas vraiment. Il varie considérablement, aussi bien dans sa conception, dans la manière de le représenter, que dans la précision de sa mesure. La jointure spatiale, extension de la jointure classique et opération formelle correspondant à la mise en relation par superposition, ne tient pas compte de la validité ou de l'échelle de description. Dans de nombreux cas, elle s'avère donc inutilisable en l'état : il va falloir gérer des *transferts d'échelle* pour mettre effectivement en relation des objets de validité ou d'implantation spatiale différente. La conception de notre système est-elle encore valide, si son principal objectif, à savoir mettre les objets localisés en relation grâce

à leur localisation, ne peut être atteint efficacement ? Gérer ensemble des collections d'objets en conservant l'ambiguïté sur la validité de l'attribut de localisation est-il possible ? Pour répondre positivement à cette question, il est nécessaire d'introduire de nouvelles méthodes de gestion et d'exploitation, au-delà d'une jointure dont l'objectif initial –retrouver les valeurs d'un point de l'espace à partir de collections distinctes– doit être analysé en fonction de la modélisation du monde réel. La première réponse, c'est l'introduction de procédures permettant de gérer les transferts d'échelle, à la fois par des procédures d'agrégation, par des procédures de changement de type d'implantation spatiale, par des procédures d'interpolation, par des procédures d'extrapolation. L'autre réponse, complémentaire, consiste à documenter les bases de données par l'introduction systématique de méta-données permettant à l'utilisateur de revenir à la genèse de l'information et d'éviter les écueils d'une utilisation erronée. Elle consiste également à introduire dans le schéma de la base de données des méthodes d'utilisation de ces données, de manière à proposer à l'utilisateur un ensemble de méthodes d'exploitation qui dépendent non seulement du type des objets, mais également de leur contenu sémantique et de leur précision géographique.

Enfin, l'ergonomie du logiciel est importante : elle doit permettre à la fois l'approche exploratoire et la représentation cartographique des résultats, à chaque étape de la requête. Ces deux objectifs sous-tendent la conception de l'ergonomie du module d'exploitation.

2.1.2. Architecture générale de SAVANE

Le système SAVANE se caractérise donc par une stricte application de la logique et des concepts des base de données, notamment relationnelles et objets, aux objets géographiques localisés. Il y a d'abord le concept d'objet, qui est l'entité de base gérée par le système : une zone dans une carte, un individu dans un recensement, un tronçon de réseau, etc. Chaque objet est décrit par un certain nombre d'attributs, comme un nom, des coordonnées, des valeurs numériques. Le système gère des objets et les valeurs des attributs qui les décrivent. On regroupe les objets qui sont décrits par les mêmes attributs dans des collections, que l'on appelle relations ou tables : le schéma d'une relation, c'est l'ensemble des attributs servant à décrire les objets de la relation, ainsi qu'un ensemble de méthodes pouvant être appliquées à ces objets. L'ensemble des schémas des relations donne le schéma de la base de données, qui est constituée quant à elle des objets de toutes les relations. Le système stocke et gère les objets en s'appuyant sur cette structure de relation - collections d'objets du même type - et traite les objets grâce à leur description par attributs - variables dont les valeurs décrivent l'objet : le système SAVANE est construit sur le principe des systèmes de gestion de données relationnels [SOU 87].

Lorsqu'un objet est géographique, il est souvent localisé, c'est-à-dire que l'on tient compte de sa position dans l'espace en deux ou trois dimensions, et cette position sert aussi à décrire l'objet. On parlera d'attribut de localisation comme on a parlé d'attribut de description. Le schéma d'une relation dont les objets sont localisés - on dira une relation localisée - comporte donc toujours un attribut de localisation : c'est la gestion de cet attribut qui fait la différence entre un SIG (système d'information géographique) et un simple SGBD (système de gestion de base de données). Le système SAVANE étend la gestion relationnelle à l'attribut de localisation : il utilise la localisation pour mettre les objets d'une collection en relation avec les objets d'une autre collection.

La localisation d'objets géographiques peut être zonale (l'objet est une zone; une collection de zones donne une relation zonale, ou de type zone), linéaire (l'objet est une portion de ligne, et la relation correspondante est dite linéaire), ponctuelle (la localisation est donnée par un point, la relation est dite ponctuelle), ou encore donnée sous forme d'image numérique géo-référencée (les objets sont alors les pixels qui forment l'image). La localisation peut être donnée sous forme vectorielle (des contours, des arcs) ou sous forme matricielle (des pixels) : de toutes les façons, le système SAVANE crée à partir des vecteurs une représentation matricielle lorsqu'il en a besoin, lors de l'exploitation des données. Il peut également créer des vecteurs à partir de pixels d'une image. Ces différents types de localisation correspondent à des types de base pour les objets : zone, ligne, point, pixel. A chaque type correspondent des méthodes particulières (par exemple, la surface pour les zones), méthodes accessibles directement dans les menus de SAVANE. Ce modèle de données reprend donc la schématisation cartographique de la réalité géographique.

La saisie et le stockage de la localisation des objets géographiques impliquent des procédures tout à fait spécifiques à ces objets, et l'administration d'une base de données géographiques nécessite aussi quelques connaissances en cartographie (les projections, le redressement, les échelles, la précision géographique, la généralisation, etc.).

La structuration des données dans SAVANE correspond au modèle relationnel des systèmes de gestion de base de données étendu aux données géographiques localisées. Il s'oriente également vers le modèle objet par l'introduction de classes d'objets et de méthodes sur ces classes. L'architecture générale est celle d'un calculateur base de données simplifié : il possède un dictionnaire des données, indiquant les relations, attributs, méthodes, ainsi que leurs caractéristiques (types, définition du schéma interne associé), un dictionnaire des accès permettant de gérer les niveaux externes (par l'allocation de droits et la définition de l'accès aux données), un langage de commande permettant d'interroger et de manipuler les

données suivant une structure client/serveur. Chaque opération utilise ces trois structures pour accéder au niveau interne et au système de gestion de fichier.

Le système SAVANE crée et administre ses propres bases de données, intégrant dans un même ensemble l'information descriptive et l'information de localisation, contrairement à la plupart des SIG qui utilisent un SGBD classique pour les données descriptives et créent le lien entre descriptif et graphique au moment de l'exploitation des données. L'information de localisation est conservée dans sa forme originale, vectorielle ou pixel selon la modélisation d'origine. Le système se charge de l'ensemble des opérations de changement de type (vecteur-raster ou raster-vecteur notamment) en fonction des besoins de l'utilisateur : il privilégie toujours l'aspect fonctionnel à l'aspect technique. Tous les points sont décrits par leurs coordonnées géographiques dans un *datum* unique. La projection géographique de restitution peut être choisie par l'utilisateur lors de l'interrogation des données.

Les relations localisées sont indexées sur la localisation par la notion naturelle de feuille, chaque feuille correspondant à une coupure de carte. Toute recherche dans une relation localisée passe par la recherche des feuilles concernées par le territoire d'étude. Chaque relation localisée possède son propre ensemble de feuilles, puisque cette indexation dépend essentiellement de la densité des objets propre à une relation donnée. Cette indexation s'apparente donc à un découpage en grille adaptative.

L'exploitation des bases de données est multi-utilisateur. L'interrogation se fait sous forme de requêtes, dans une démarche exploratoire. Le système gère les requêtes de chaque utilisateur en créant des états temporaires, propres à l'utilisateur, sans modifier la base de données.

L'extension du modèle relationnel sur la localisation permet de joindre les objets de la base sur la localisation, lors d'une requête, à partir d'une gestion en relation. Mais la gestion seule est insuffisante pour répondre aux besoins d'analyse qui sont omniprésents lors de l'exploitation de données géographiques. Le système comprend donc de nombreuses fonctionnalités d'analyse. De plus, les différences de précision dans la localisation, dues aux différences d'échelle de description des objets, réduisent le caractère universel de la localisation en tant qu'attribut de jointure. Nous avons donc développé dans le système SAVANE de nombreuses procédures pour répondre aux besoins de jointures spatiales lorsque la simple mise en relation sur la localisation ne peut être utilisée. L'extension du modèle relationnel se fait donc à plusieurs niveaux : d'une part par l'extension des opérations classiques de l'algèbre relationnelle à l'attribut de localisation, et d'autre part par l'introduction de fonctionnalités permettant de résoudre les problèmes de transfert d'échelle.

2.1.3. Administrer une base SAVANE

Le schéma d'une base comprend la définition des relations, des attributs, et des méthodes. Contrairement aux SGBD classiques, chaque relation a ici un type, qui définit le type des objets contenus dans la relation, et permet d'associer des méthodes de base uniquement liées au type d'implantation géographique (comme le périmètre ou la surface pour les zones, ou la longueur pour les lignes, etc.). La structuration et la gestion interne sont également propres à chaque type d'objet. Les types supportés par SAVANE correspondent aux objets habituellement rencontrés en géographie : zone, ligne, point, pixel, non localisé. Tous les objets d'une même relation sont donc du même type. La localisation des objets géographiques est conservée dans le repère d'un *datum* commun, en coordonnées sphériques et hauteur au-dessus du géoïde. Le schéma d'une base est conservé dans un dictionnaire de données ; tous les modules du système accèdent à ce dictionnaire pour lire le schéma et utiliser une base de données. Les bases de données sont créées et administrées par le module d'administration, qui est séparé des modules d'exploitation. Ce module gère également les vues externes, qui permettent aux modules d'exploitation d'avoir une vision différente de la base, par le choix ou le regroupement logique de relations, d'attributs et de méthodes. Il gère également les méta-données, pour les relations, attributs et méthodes.

L'administration d'une base de données est un travail en soi dès que la base atteint des dimensions importantes. Pour gérer une base de données géographiques avec le système SAVANE, et comme pour toute base de données, il faut réfléchir à la structuration des données, à leur organisation, à leur gestion, à la manipulation des données, ainsi qu'à la formation des utilisateurs : c'est l'administrateur de la base de données qui en aura la charge, poste fondamental pour le bon fonctionnement du système. En revanche, un utilisateur du système d'information n'a pas à se préoccuper de l'administration de la base de données, ni de celle de la machine sur laquelle il travaille. Il se consacrera uniquement à l'exploitation des données, en supposant la base fiable, mise à jour, prête à l'emploi.

D'une manière générale, l'administration d'une base SAVANE, de sa création à son exploitation, peut se résumer en plusieurs grandes étapes : décrire, saisir, intégrer, modifier :

Décrire ...

Pour créer une base de données il faut, avant tout, décrire cette base, décrire les données, indiquer comment se regroupent les objets, quels sont les attributs, leurs types, etc. C'est la première étape, celle de la réflexion, de la structuration, de la schématisation, de la description, pour créer ce que l'on appelle le schéma de la base de données.

Une fois créé le schéma de la base, l'administrateur pourra définir des utilisateurs et leur allouera des droits d'utilisation.

Saisir ...

Une fois défini le schéma d'une collection d'objet, nous savons quels sont les attributs qui décrivent chaque objet. Il faut donc saisir, pour chaque objet, la valeur de ces attributs. Si les objets sont localisés, et que l'on a décidé dans le schéma de prendre en compte cette localisation, il faut également saisir la partie graphique des objets, en fonction du type de la relation.

Cette étape, dans le cas de grandes bases de données, est contraignante, longue et lourde, et il convient de la gérer avec le plus grand soin car elle est bien sur très importante.

Intégrer ...

La saisie des données se fait, disons, par morceaux. Les données peuvent être hétérogènes : différences de formats, de codages... S'il s'agit de géographie et de localisation, les problèmes sont plus nombreux. Entre la saisie et la base de données, il y a donc une phase d'intégration des morceaux dans l'ensemble homogène que constitue une base de données SAVANE. Intégrer, c'est donc lire des données (graphiques ou descriptives) dans des fichiers pour les réécrire dans un ensemble - la base de données proprement dite - qui se constitue peu à peu. Il est important de noter que cette phase d'intégration lit des données pour créer ou modifier une base de données qui sera indépendante de ces fichiers (qui ne serviront plus dans SAVANE après l'intégration). Les fichiers contiennent soit des données descriptives (des valeurs qualitatives, des valeurs quantitatives, des dates, etc.), soit des données graphiques provenant de la saisie graphique ou d'images.

Modifier...

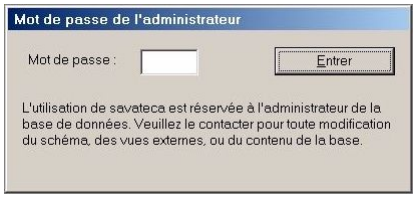
Une fois la base constituée, il faut pouvoir la modifier pour corriger d'éventuelles erreurs ou intégrer les évolutions et modifications dans l'information. Ces modifications se font grâce à des éditeurs propres à SAVANE, pour la partie graphique comme pour les données descriptives. Ces différentes opérations ne s'effectuent pas une fois pour toute. Il est courant de créer une base petit à petit, de modifier le schéma pour rajouter des relations, des attributs, etc. Il est important, pour l'administrateur de la base, de bien connaître toutes les phases indiquées plus haut, dans l'ordre comme chaque opération séparée.

2.2. Présentation fonctionnelle des différents modules

Le système est composé de plusieurs modules, séparant les trois opérations fondamentales que sont **l'administration et la gestion, la saisie, l'exploitation**.

2.2.1. SAVATECA : administration des bases de données

Le module SAVATECA regroupe l'ensemble des opérations de gestion et d'administration des bases de données, indépendamment de leur exploitation. Il gère la configuration du système global (emplacement des bases de données, emplacement des utilisateurs), les dictionnaires des bases de données, les vues externes. Son utilisation est en principe réservé à l'administrateur de la base de données : un mot de passe permet d'en restreindre l'accès.



2.2.1.1. Principes généraux d'administration

Nous décrirons plus tard l'organisation précise des données graphiques et descriptives d'une base de données gérée par SAVANE. Nous pouvons dès à présent indiquer l'organisation générale du système :

- **Bases de données** : tous les fichiers internes d'une même base de données SAVANE (données, méthodes, accès, dictionnaire) sont regroupés dans un seul répertoire, à l'exception des relations de type mosaïque dont l'emplacement peut être distinct (pour pouvoir les partager entre plusieurs bases de données). Le répertoire de la base peut se trouver n'importe où sur le réseau local. Le système SAVANE permet de gérer plusieurs bases de données. Le schéma d'une base de données est décrit dans un fichier spécial (le fichier base), les vues externes (description d'accès aux relations, aux attributs et aux méthodes) sont décrites dans le fichier nommé *fpacc*.

- **Utilisateurs** : pour exploiter une base de données SAVANE, il faut être un utilisateur déclaré du système SAVANE. Lors de l'exploitation, de nombreux fichiers temporaires sont créés : ils seront stockés dans un répertoire propre à l'utilisateur. De même, ce répertoire de l'utilisateur permet de stocker les cartes et méthodes créées par cet utilisateur. Le répertoire de l'utilisateur peut se trouver n'importe où sur le réseau.

- **Exécutables** : les exécutables peuvent être installés n'importe où sur le réseau local. Lors de la première utilisation de SAVATECA, un répertoire *admin* est créé. Ce répertoire contient les fichiers de configuration du système.

Toute utilisation de l'un des modules d'exploitation que nous décrirons plus loin (SAVEDIT, SAVAMER, SAVANE) nécessite les trois paramètres suivants : nom de la base, nom de l'utilisateur, vue externe.

2.2.1.2. Configuration du système

Le système conserve dans le répertoire *admin* les fichiers contenant les informations relatives à la configuration générale :

- Le fichier *fpconf* conserve le nom des bases de données déclarées, avec l'emplacement physique du répertoire contenant les fichiers de la base de données.
- Le fichier *fpuser* conserve le nom des utilisateurs avec l'emplacement du répertoire propre à chaque utilisateur.

2.2.1.3. Organisation d'une base de données SAVANE : principe généraux

Les objets sont regroupés en collections, ou relations, suivant le principe des bases de données relationnelles. Chaque individu d'une collection est décrit par un certain nombre de critères, ou *attributs*. Bien sûr, ces critères sont les mêmes pour tous les objets d'une même relation. Il existe dans Savane cinq types de relation, suivant la définition géographique des objets : zones, lignes, points, pixels, non localisés. Un objet est décrit par des attributs descriptifs, plus un attribut graphique dans le cas de relations localisées.

On rencontre de nombreux types d'attributs descriptifs, parmi lesquels les plus courant sont :

• *Nominal* ou *qualitatif* : ceux qui prennent des valeurs nominales (une chaîne de caractères). Certains attributs nominaux jouent un rôle particulier : une clé est un attribut dont chaque valeur correspond à un seul objet (par exemple le numéro d'immatriculation est une clé de la relation voiture; le nom du propriétaire n'est pas une clé, car un même propriétaire peut avoir deux voitures).

• *Ordinal* : ceux qui prennent des valeurs nominales que l'on veut pouvoir ordonner,

• *Entier* : ceux qui prennent des valeurs numériques entières,

• *Réel* : ceux qui prennent des valeurs numériques quelconques (comme un prix).

• *Couleur* 8 bits, 16 bits, RVB : la valeur de l'attribut correspond au codage d'une couleur. Avec 8 bits, on peut coder 256 couleurs, avec 16 bits 64000. Une couleur RVB comprend un niveau de rouge, un niveau de vert, un niveau de bleu, chaque niveau étant codé sur 8 bits (256 valeurs possibles pour chaque niveau, soit 16 millions de couleurs possibles).

• *Date* : ceux qui prennent la valeur d'une date,

• *Son* : la valeur de l'attribut correspond à la description d'un son,

• *image* : la valeur de l'attribut correspond à une image ou à une collection d'images (album de photographies, documents scannés,…),

• *vidéo* : la valeur de l'attribut correspond à une séquence vidéo,

• *graphe* : la valeur de l'attribut correspond à la description d'un graphe en deux dimensions.

Chaque objet d'une relation localisée est donc décrit par un ensemble de valeurs d'attributs descriptifs ainsi que par la valeur de l'attribut de localisation. L'attribut de localisation n'apparaît pas dans la liste des attributs d'une relation, il est sous-jacent à la définition même de la relation et est géré directement par le système. En fait, on définit implicitement l'attribut de localisation lors de la création de la relation, en indiquant le type des objets (zone, ligne, point, pixel, non localisé).

Définir des types d'objets en fonction de leur type de localisation revient en fait à se rapprocher du modèle objet : nous avons cinq grandes classes d'objets, les zones, les lignes, les points, les pixels, les objets non localisés, et à chaque classe correspond différentes fonctions de description, de stockage, d'intégration, d'exploitation. Le système SAVANE propose également quelques types dérivés des types de base : ces types proposent déjà la définition d'un certain nombre d'attributs descriptifs.

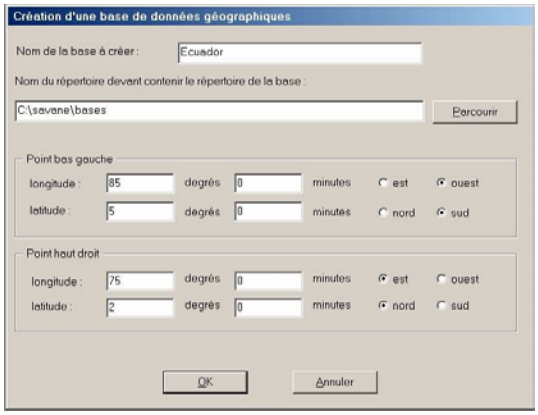
A chaque relation sont associés des fichiers qui contiennent les valeurs des objets. La description de la localisation est différente suivant le type de la relation. Pour les relations de type zone, la localisation des objets est donnée par les arcs constituant la frontière de chaque objet : un fichier conserve la description de tous les arcs de la relation, un autre fichier conserve les valeurs des attributs descriptifs, et un troisième fichier donne l'indexation primaire suivant la localisation. Pour les objets de type ligne, la structure est presque identique, mais la localisation ne fait pas référence à la notion de frontière, et la localisation d'un objet est décrite par des arcs qui n'appartiennent qu'à un seul objet. Pour les objets de type points, la localisation est donnée directement par les coordonnées, et est conservé dans le fichier des valeurs descriptives : ce type de relation ne comprend donc que deux fichiers, l'un pour la localisation et les données descriptives, l'autre pour l'indexation géographique. Nous reviendrons en détail sur l'organisation interne d'une base SAVANE dans le chapitre 3.

Le module SAVATECA permet de gérer l'ensemble de l'organisation d'une base, que ce soit pour la description des relations, pour la gestion des fichiers associés à chaque relation, pour la gestion des objets de chaque relation, pour la

gestion des vues externes, ou pour la gestion des utilisateurs et l'utilisation en réseau.

2.2.1.4. Création d'une base de données

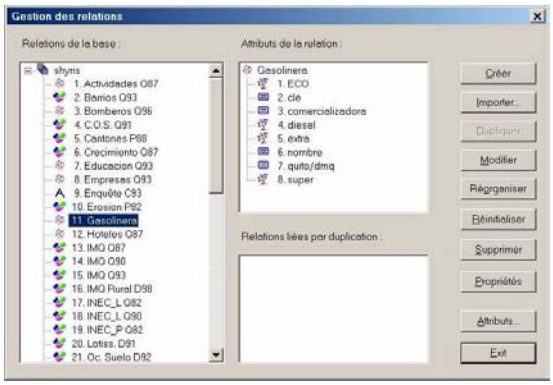
Le module SAVATECA permet de créer une base de données *SAVANE*. Lors de la création, l'utilisateur doit indiquer le nom de la base, son emplacement sur le réseau, l'espace géographique concerné. Cette déclaration va créer l'ensemble de la structure nécessaire à une base de données : un répertoire à l'emplacement indiqué, et dans ce répertoire, le dictionnaire de la base (fichier *base*), le fichier des vues externes (fichier *fpacc*). La description de la fenêtre géographique de la base (espace géographique concerné par cette base) est stockée dans le dictionnaire de la base.



2.2.1.5. Gestion du schéma

Le schéma d'une base de données contient la description des relations, des attributs, et des méthodes. Cette description est conservée dans le fichier *base* : c'est le dictionnaire de la base. Pour chaque relation, ce dictionnaire comprend :

- le nom de la relation
- le type de la relation (zone, ligne, point, image, non localisé)
- le nombre d'attributs descriptifs
- le nom des fichiers associés (objets et valeurs descriptives, objets graphiques associés)
- la description des attributs



Pour chaque attribut, la description comprend :
- le nom de l'attribut
- le type de l'attribut (nominal, ordinal, entier, numérique, date, RVB, etc.)
- le nombre de valeurs nominales (dans le cas d'un attribut nominal)
- l'adresse de la première valeur nominale dans le fichier des valeurs d'attributs (dans le cas d'un attribut nominal).

La lecture du dictionnaire permet au système de charger le schéma des données et l'ensemble des paramètres permettant d'accéder aux objets (données graphiques et descriptives). La classe `CDico` permet d'encapsuler l'ensemble des opérations de lecture et d'écriture du dictionnaire de la base de données (A.1.3.).

2.2.1.6. Gestion des vues externes

Une vue externe permet de donner à l'utilisateur une autre vision de la base de données. Il est possible d'exclure un certain nombre de relations, d'attributs ou de méthodes, de modifier l'ordre des relations ou des attributs, et de présenter relations, attributs et méthodes en groupes. La vue externe est donc un ensemble d'indirection entre le schéma interne de la base et le schéma externe présenté à l'utilisateur.

Les vues externes sont conservées dans le fichier nommé *fpacc*, qui se trouve dans le répertoire de la base de données. Le programme SAVATECA permet de gérer les vues externes : création, modification (ajout ou suppression de relations, d'attributs ou de méthodes, ordre, regroupement), suppression. Une classe `CFpaccV8` encapsule l'ensemble des manipulations de ce fichier (A.1.4.).

2.2.1.7. Intégration d'objets

La constitution d'une base de données se fait par intégrations successives d'objets dans la base. Pour les objets localisés, l'intégration se divise en deux phases distinctes : la première phase consiste à créer les objets en intégrant l'attribut de localisation (saisi ou importé dans le module SAVEDIT) et un identifiant pour

chaque objet. La seconde phase permet d'intégrer les valeurs descriptives des objets en utilisant l'identifiant comme attribut de jointure. Ces valeurs descriptives peuvent provenir d'un fichier classique ou d'une base de données relationnelle classique. Les valeurs des attributs des objets sont alors lues dans ces fichiers et recopiés dans la base Savane.

Pour les relations localisées, chaque phase de création crée un groupe d'objets dont l'adresse et la position géographique en deux dimensions sont conservées par le système et qu'il utilise lors de l'exploitation de la base comme indexation primaire, dans une structure séquentielle indexée en deux dimensions.

Une fois des objets intégrés dans la relation, il est bien sûr possible de modifier le schéma de la relation en ajoutant ou supprimant des attributs. Il est également possible de modifier les valeurs des objets. Ces opérations modifient le contenu de la base de données.

Les attributs sont gérés différemment suivant leur type. Par exemple, les valeurs des attributs nominaux sont codées et les modalités sont conservées dans un fichier (*fpvaleurs*) séparé des fichiers conservant les valeurs des objets.

Nous verrons en détail dans le chapitre 5 l'organisation du stockage des objets et les principes de leur indexation dans une base de données Savane.

2.2.1.8. Gestion des utilisateurs

Le système SAVANE est conçu pour être multi-utilisateur et pouvoir être utilisé dans un réseau local d'ordinateur. Comme de nombreux fichiers temporaires sont créés lors de l'utilisation du système, chaque utilisateur doit avoir un espace de travail distinct pour stocker ses propres fichiers temporaires. Le module SAVATECA permet donc d'administrer ces utilisateurs (création, modification, suppression). La description des utilisateurs (nom, répertoire de stockage) est conservée dans le fichier *fpuser*, qui se trouve dans le répertoire spécial *admin*. Le répertoire de stockage d'un utilisateur peut se trouver n'importe où sur le réseau local.

2.2.2. SAVAMER : géo-référencement d'images et constitution de mosaïques

Le système comporte deux modules de saisie graphique, l'un pour les images, l'autre pour les données vectorielles. Le module SAVAMER permet de géo-référencer des images *raster* et de les intégrer dans une relation de type image.

Pour être intégrés dans une relation de la base de données, les pixels d'une images doivent tous être géo-référencés : la position de chaque pixel doit être

connue. Bien souvent, l'image que l'on souhaite intégrer dans une base n'est pas en conformité géographique avec la réalité, et il faut la redresser, c'est-à-dire modifier la position d'un certain nombre de ses pixels pour les remettre à leur position géographique correcte. L'image d'origine doit donc être déformée.

Les objets d'une relation de type image sont des pixels définis sans ambiguïté, pour leur position comme pour leur taille. Pour affecter une valeur à un objet pixel de la relation à partir d'une image, il faut donc savoir quels sont les pixels de l'image qui participent à la définition de la valeur de l'objet pixel de la relation. L'opération de géo-référencement consiste donc en une opération de redressement (repositionnement des pixels dans l'espace) et une opération de ré-échantillonage (modification de la taille des pixels de l'image).

A partir d'une image non géo-référencée, le module SAVAMER crée donc une nouvelle image, par redressement et ré-échantillonage. Les paramètres de cette nouvelle image sont alors compatibles avec les paramètres de la relation qui doit recevoir les valeurs (projection géographique, taille de pixel). Une fois cette opération effectuée, le module SAVAMER permet d'intégrer les valeurs de l'image à un attribut de la relation.

2.2.2.1. Le géo-référencement

Le géo-référencement a pour objet la mise en conformité géographique de l'image. L'opération a donc pour objet de placer chaque pixel de l'image d'origine à sa position géographique exacte. Comme il n'est pas possible d'indiquer manuellement la position exacte de chaque pixel de l'image, on utilise des fonctions de déformation globales ou semi-locales. Le choix de la fonction de déformation dépend de la déformation supposée de l'image. Certains paramètres de déformation sont intrinsèques au type de capteur, comme la position et le type de capteur d'un satellite ou l'objectif d'un appareil de prise de vues aériennes. Les autres paramètres doivent être calculés à partir de points d'appui entre image et position dans le plan de projection, et par l'utilisation d'un modèle numérique pour la prise en compte de la déformation due aux différences d'altitude.

Le module SAVAMER permet la saisie de points d'appui (fig. 2.1) et le redressement, en donnant le choix de la fonction de redressement : similitude, projection centrale, fonction polynomiale de degré 1 ou 3, tessellation, etc. Plusieurs fonctions de redressement peuvent être combinées : il est courant d'effectuer une transformation globale (de type projective ou de degré1) suivie de transformations locales (par exemple, redressement de degré 1 dans les triangles issues d'une tessellation à partir de points d'appui).

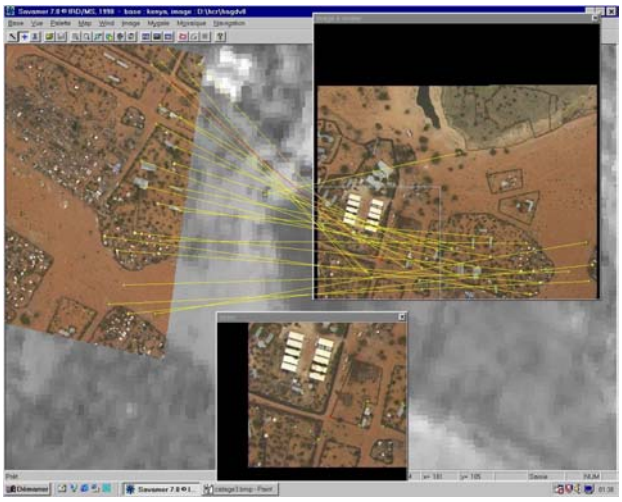


fig. 2.1 : exemple de prise de points d'amers avec SAVAMER

2.2.2.2. Le ré-échantillonage

Les pixels d'une relation sont définis sans ambiguïté, taille et position. La taille de ce pixel peut être différente de celle -supposée- des pixels de l'image d'origine. Pour savoir quelle valeur affecter à un pixel de la relation à partir des pixels d'une image, il faut calculer, par géo-référencement, quels pixels de l'image d'origine participent par leur position à la définition de la valeur du pixel de la relation, puis, par une opération de ré-échantillonage, il faut affecter une nouvelle valeur à partir de ces pixels.

Plusieurs méthodes peuvent être employées pour effectuer cette opération. La plus simple consiste à prendre la valeur du pixel de l'image d'origine le plus proche (plus proche voisin), mais il est également courant de faire une moyenne sur les pixels voisins, par un calcul bilinéaire ou bicubique. Ces trois opérations de ré-échantillonage sont implantés dans le module de redressement de SAVAMER.

2.2.2.3. Le mosaïquage et l'intégration

L'opération de mosaïquage permet d'intégrer une image géo-référencée à un attribut d'une relation de type pixel. Le système SAVANE gère ces attributs de manière à permettre des ensembles de pixels plus complexes que des images rectangulaires. L'opération d'intégration consiste donc, à partir d'une image géo-référencée, à créer des objets pixels dans une relation (si ces objets n'existent pas déjà), et à affecter les valeurs des pixels de l'image à un attribut dans cette relation. On constitue ainsi une « mosaïque » (fig. 2.2) dont la gestion est entièrement à la

charge du système, et dont la structure externe est identique à celle des autres types de relation.

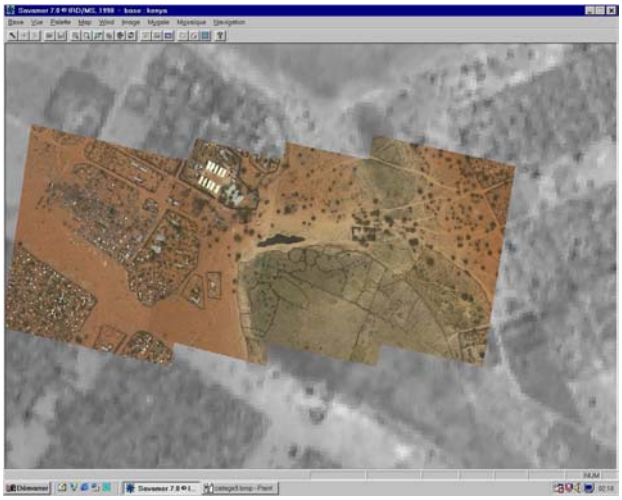


fig. 2.2 : exemple de constitution de mosaïque d'images géo-référencées

Nous verrons en détail dans le chapitre 6 l'organisation du stockage des objets pixels, les principes de leur gestion dans une base de données Savane, ainsi que les algorithmes utilisés pour le redressement et le ré-échantillonage.

2.2.3. SAVEDIT : saisie vectorielle sur écran avec gestion des contraintes d'intégrité

Le module SAVEDIT permet la saisie et la correction de l'attribut de localisation pour les objets dont la localisation est donnée sous forme de zone, de ligne, ou de point, à partir de plans ou de cartes. Il permet également l'importation de documents provenant d'autres logiciels (format ShapeFile ou DXF). La saisie est dite *vectorielle* car le résultat est stocké sous forme discrète d'ensembles de points, permettant de schématiser les contours des zones ou les arcs des lignes.

2.2.3.1. Principes généraux

La saisie se fait sur écran avec une souris. Le support de la digitalisation peut être une carte ou un plan, géo-référencé (grâce au module SAVAMER) et visualisé sur l'écran. L'utilisateur peut également visualiser l'ensemble des objets déjà intégrés dans une base de données. De nombreuses options permettent de choisir l'espace géographique affiché à l'écran.

Contrairement aux logiciels de saisie graphique (de type AUTOCAD) qui permettent la saisie, dans un même document, d'objets de différents types (des zones, les lignes, du texte, des symboles, etc.), un document SAVEDIT ne doit contenir que les objets d'une même collection, destinés à être intégrés dans une même relation. La saisie se limite, pour chaque objet, à la description de la localisation, et à la saisie d'une valeur propre à chaque objet (un identifiant, ou clé) qui sera utilisé par la suite comme clé de jointure pour l'intégration des valeurs descriptives de l'objet. Aucun attribut de dessin (couleur, type de trait, symbole, etc.) n'est indiqué lors de la saisie (fig. 2.3). Ces attributs de dessin seront affectés par la suite à chaque objet en fonction de leurs attributs descriptifs, dans le processus d'interrogation et d'exploitation de la base de données effectués grâce au module SAVANE.

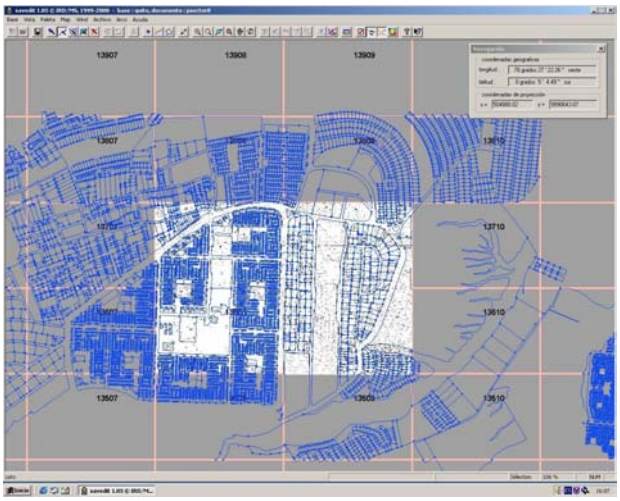


fig. 2.3 : la saisie sur écran avec le module SAVEDIT

Puisque la saisie suit la schématisation du réel en collections, cette saisie doit satisfaire de nombreux contrôles de cohérence liées à cette description. Par exemple, les zones doivent être fermées ; les zones d'une même relation ne doivent pas avoir d'intersection ; les arcs ne doivent pas se croiser ; etc. Nous verrons en détail dans le chapitre 6 les contraintes d'intégrité qui pèsent sur l'attribut de localisation, et les algorithmes employés dans le module SAVEDIT pour vérifier ces contraintes et produire des document exempts d'erreurs.

Le module SAVEDIT présente également de nombreuses fonctionnalités propres à améliorer les performances de l'opération de saisie graphique. Par exemple : gestion de la précision et de la connexité entre arcs ; jointure automatique d'arcs ; division automatique d'arcs en cours de saisie sur l'intersection ; vérification

permanente de la fermeture des zones ; suppression automatique d'arcs en double ; etc. Nous présenterons en détail dans le chapitre 7 (contraintes d'intégrité spatiale) la mise en œuvre et les algorithmes utilisés par SAVEDIT.

Chaque point saisi sur l'écran est transformé en coordonnées géographiques (longitude, colatitude) selon le *datum* et la projection géographique utilisés. Tous les points sont donc conservés en coordonnées sphériques. Le *datum* d'un document peut être modifié : dans ce cas, l'utilisateur a le choix de transformer ou non les coordonnées des points saisis. Ceci peut être nécessaire car tous les objets d'une base de données SAVANE doivent être exprimés dans le même *datum*.

2.2.4. SAVANE : exploitation des bases de données et cartographie

Le module d'exploitation SAVANE est le principal module du système. Il intègre un ensemble étendu de fonctionnalités pour l'interrogation, le traitement et la représentation des informations contenues dans une base de données géographiques. L'utilisation de SAVANE ne réclame pas l'apprentissage d'un langage de commande spécifique. L'interface rassemble dans un environnement ergonomique une grande variété d'outils de consultation, de cartographie, de recherches multicritères, d'analyse statistique, de superpositions, géo-jointures, lissages, d'analyses d'itinéraires et de voisinage, de modélisation. Le logiciel est un outil polyvalent et intégré qui permet d'appréhender rapidement toutes les facettes du travail d'exploitation d'un SIG.

2.2.4.1. Cartes, cadres et requêtes

Le document de base du logiciel SAVANE est une carte, c'est-à-dire une feuille de papier destinée à contenir une représentation graphique de données géographiques. A l'entrée dans SAVANE – après avoir indiqué le nom de la base, l'utilisateur et la vue externe - apparaît donc une feuille de papier sur l'écran. Cette feuille contient par défaut un *cadre géographique* (fig. 2.4).

Un cadre géographique correspond au résultat d'une requête sur la base de données : SAVANE utilise le principe de la requête pour l'interrogation d'une base de données. Il faut enchaîner les opérations, le résultat de l'une pouvant servir d'entrée à la suivante. Une interrogation, une requête, est donc composée d'un ensemble d'opérations que l'on déclenche grâce au menu principal, et que l'on peut conserver sous forme de macro-commande.

Un cadre peut donc être vu comme une fenêtre sur la base de données contenant, à un moment donné, un état temporaire de la base correspondant à une interrogation et aux opérations effectuées (calculs, création de nouvelles variables, croisement et jointure d'objets, etc.). L'état temporaire de la base lié au cadre est conservé avec la

carte qui contient le cadre, ainsi que la macro-commande correspondant à toutes les opérations effectuées dans ce cadre. En plus de la requête et de l'état temporaire de la base, au cadre sont attachés des paramètres qui lui sont propres (espace visualisé, projection géographique, échelle de tracé, visualisation des amorces géographiques ou de projection, type de dessin pour le bord, position dans la carte).

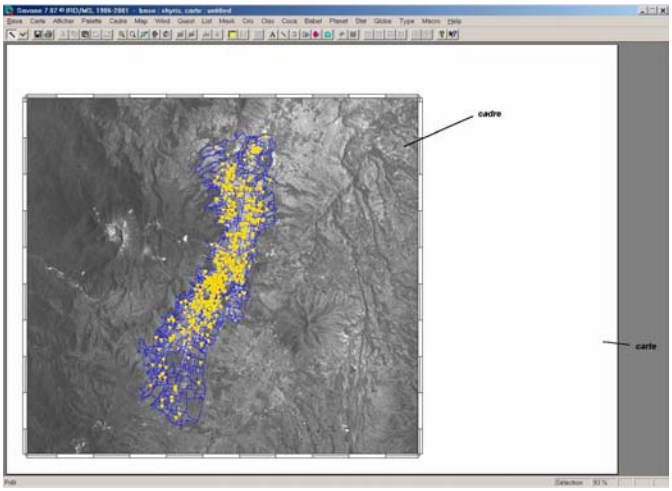


fig. 2.4 : l'écran principal de SAVANE avec une carte contenant un cadre

Une carte peut contenir simultanément jusqu'à dix cadres différents. Elle doit en contenir un au minimum. Un cadre doit être sélectionné dans la carte pour avoir accès aux menus des commandes d'interrogation de la base de données. Le cadre correspond à un espace géographique, délimité par le bord du cadre. La requête liée au cadre utilise par défaut cet espace géographique pour sélectionner les objets de la base, mais l'utilisateur peut spécifier une fenêtre géographique différente pour la requête, ou modifier l'espace de visualisation du cadre sans modifier la fenêtre géographique de la requête. Dans un cas, la fenêtre de la requête correspond à une opération de sélection des objets de la base de données, dans l'autre cas, l'espace de visualisation du cadre correspond à un espace de dessin dans la carte, à une échelle donnée par l'utilisateur.

Un certain nombre de classes correspondent à ces objets : la classe `CCarte` (A.2.4.1.) pour encapsuler l'ensemble des variables et des opérations correspondant à la carte dessinée sur l'écran, la classe `CCadre` (A.2.4.4.) pour les cadres géographiques, les classes `CWind` (A.2.3.1.) et `CProjection` (A.2.3.2.) pour décrire la fenêtre géographique de travail et la projection géographique utilisée dans un cadre, et la classe `CSchema` (A.2.1.2.) pour décrire l'état de la base de donnée correspondant à la requête effectuée dans un cadre.

Nous allons ici décrire rapidement les classes `CCarte` et `CCadre`. Les autres classes essentielles seront décrites en détail dans les chapitres 4 (`CProjection` et `CWind`) et 5 (`CSchema, CRelation, CAttribut`).

2.2.4.2.1. La classe `CCarte`

Comme nous l'avons déjà dit plus haut, le document de base du logiciel SAVANE est une carte : une feuille de papier destinée à contenir une représentation graphique de données géographique. Il n'existe qu'un seul objet de type `CCarte` par session de travail dans SAVANE (on ne peut avoir plusieurs cartes ouvertes en même temps, il faut fermer une carte pour en ouvrir une autre). La classe `CCarte` encapsule toutes les informations et opérations propre à une carte (A.2.4.1.).

Les seuls membres liés à la base de données sont les pointeurs des objets de type `CCadre` (`CCadre* m_pCadre[NB_MAX_CADRES]`). Tous les autres membres et méthodes de cette classe sont liées à la représentation graphique de la carte, sur l'écran ou sur l'imprimante. Outre des cadres, une carte peut contenir des objets graphiques provenant du dessin direct effectué à l'écran. Ces objets sont gérés par la classe `CODD` (A.2.4.2.). Les ODD représentent des classes propres à chaque type d'objet graphique (texte, ligne, polyligne, polygone, rectangle, rectangle dégradé, cercle, ellipse, symbole, échelle graphique, bitmap). Par exemple, la classe `CODD_Symbole` encapsule toutes les informations nécessaires à la manipulation des objets graphiques de type symbole (A.2.4.2.). Les objets graphiques sont conservés dans la carte par le tableau des pointeurs des objets de type `CODD` correspondants.

Toutes les coordonnées des objets sont conservées en mm/100 dans le repère de la carte (origine : point bas-gauche de la feuille, unité mm/100). Plusieurs procédures permettent de passer des coordonnées « carte » aux autres repères présents dans le système : vue (écran ou imprimante), cadres, coordonnées géographiques, projection géographique, coordonnées dans la base de données (A.2.4.).

2.2.4.2.2. La classe `CCadre`

La classe `CCadre` (A.2.4.4.) est essentielle : elle encapsule toutes les données et méthodes liées à une interrogation de la base de données et à la représentation graphique du résultat obtenu. La classe regroupe toutes les données et méthodes permettant de décrire, stocker et dessiner le résultat d'une requête. La description et le stockage de l'état de la base de données utilise les classes `CSchema` (description de l'état de la base de données, voir chapitre 3), `CWind` et `CProjection` (description de l'espace d'étude et de la projection géographique utilisée dans le cadre, voir chapitre 4), `CMacro` (gestion des opérations relationnelles effectuées dans le cadre, A.2.1.4.). La description des paramètres de représentation graphique

du résultat de la requête du cadre est conservée dans un objet de la classe `CCartExplorateur` (A.2.4.5.), que nous présenterons au chapitre 9.

2.2.4.2. Principales opérations de manipulation de données

Les principales opérations concernent l'interrogation relationnelle des données géographiques dans un cadre. Toutes ces opérations sont accessibles par les menus et les dialogues :

- restriction, par formule, par valeur, ou sur un espace géographique,
- suppression de relations ou d'attributs, correspondant à l'opération relationnelle

de projection,

- jointures (classique, semi-jointure, jointures géométriques, semi-jointures géométriques),
- agrégations,
- regroupement thématique d'objets d'une même relation,
- géo-jointures suivies d'agrégations.

Outre les opérations relationnelles utilisant la localisation comme clé de jointure, des opérations sont spécifiques à la localisation des objets :

- calculs métriques (longueurs, périmètres, surfaces),
- création de zones tampons,
- calculs de distance entre objets,
- calculs de proximité ou faisant intervenir les voisins (comme par exemple les calculs de texture dans une relation de type pixel),

Plusieurs opérations permettent de changer de type d'objet géographique, comme par exemple :

- procédures d'interpolation, permettant de passer d'un type ligne ou point à un type pixel,
- procédures de rasterisation, permettant de passer d'un type zone à un type pixel,
- procédures de vectorisation, permettant de passer d'un type image à un type zone ou ligne.

De nombreuses autres opérations sont disponibles. Elles concernent essentiellement la création de nouveaux attributs à partir des attributs d'une relation, et l'exploration statistique des attributs. Par exemple, on peut créer de nouveaux attributs dans une relation :

- par calcul numérique ou logique entre les autres attributs de la relation. Il suffit d'indiquer la formule, qui peut contenir des opérateurs logiques (and, or, xor, not) comme des opérateurs numériques.

- par agrégation statistique (moyenne, somme, écart-type, etc.) d'un attribut numérique par rapport à un attribut nominal,

- par combinaison, comparaison, tri, etc.,

- par classification, pour créer un attribut nominal à partir d'un attribut numérique (classifications par intervalles, par quantiles, par progression, par moyennes emboîtées, etc.).

Nous présenterons toutes ces opérations à partir du chapitre 5, après avoir décrit en détail la structure et l'organisation des bases de données SAVANE.

2.2.4.3. Masques et distances

La création de zones tampon, ou masques, permet de gérer agréablement les opérations de semi-jointures géométriques. Un masque est défini comme une zone géographique sans attribut dont l'objectif est l'utilisation ultérieure dans des opérations de sélection sur la localisation (appartenance ou non d'un objet au masque, lors d'une opération impliquant l'objet). Les masques peuvent être créés directement en dessinant sur l'écran, ou par rapport aux objets d'une relation (avec une distance permettant de créer un *tampon* autour de ces objets). Le système SAVANE permet de combiner des masques entre eux grâce à une algèbre de masques : intersection, union, union exclusive, différence, inversion. Il permet également les opérations morphologiques d'érosion et de dilatation. Les masques sont conservés par le système à la fois sous forme vectorielle et sous forme matricielle. La forme matricielle est recalculée à chaque changement d'espace géographique de travail.

2.2.4.4. Macro-commandes et méthodes

Toutes les opérations réalisées pendant une session de travail dans un cadre sont conservées dans une structure de macro-commande, liée au cadre.

Cette macro-commande peut être exécutée de nouveau pour actualiser l'état de la base de données lié au cadre, et donc lié à la requête associée.

Une macro-commande peut également être créée directement par l'utilisateur, en enregistrant une partie d'une requête. Cette macro-commande peut ensuite être exécutée de nouveau, et être intégrée dans la structure de la base comme une méthode sur les relations de la base. La base de données initiale peut donc s'enrichir peu à peu de méthodes d'utilisation des données. Une méthode est soit liée à une relation, si les procédures utilisées ne font appels qu'aux seuls attributs de cette

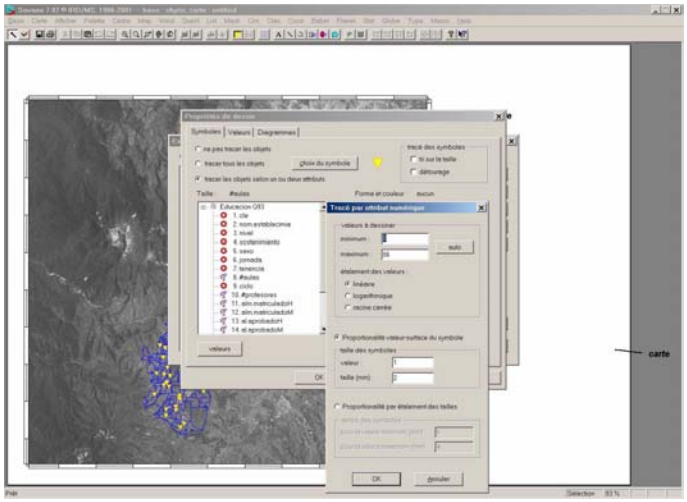
relation, soit liée directement à la base, si elle fait appel aux attributs de plusieurs relations.

Le schéma des données, initialement relationnel (mis à part les types d'objets liées à la localisation), se rapproche donc du modèle objet, en encapsulant données et méthodes dans une même structure. On bénéficie ainsi à la fois de la gestion de la localisation (SIG) et de la gestion objet (BDOO) dans un même système.

2.2.4.5. Edition cartographique et impression

Puisque le système gère et manipule des données localisées, il faut pouvoir représenter le résultat d'une requête, exactement comme l'on peut choisir les attributs graphiques d'une liste de valeurs (police, taille des caractères, largeur des colonnes, etc.) pour les imprimer sous forme de tableau. Pour des données bi-dimensionnelles, le processus est plus complexe puisque le résultat d'une requête peut être lui-même bi-dimensionnel, et le processus de représentation s'apparente alors à une cartographie.

Le système comporte donc naturellement un certain nombre de possibilités de représentation visant à associer valeurs (nominales ou numériques) ou types d'objet à des attributs graphiques : couleur, trame, symbole, type de trait, épaisseur, etc. Evidemment, les possibilités sont différentes en fonction du type d'objet géographique : par exemple, on peut représenter une zone par son contour, ou remplir l'intérieur de la zone par une couleur et une trame, mais on ne peut représenter un point que par un symbole.



Cette association ne se fait qu'au moment de représenter sur une carte le résultat d'une requête dans un cadre. Pour cela, l'utilisateur dispose d'un explorateur cartographique qui lui permet de choisir les relations à dessiner dans le cadre, ainsi que tous les paramètres permettant d'associer les valeurs d'un attribut à une représentation graphique. Les légendes sont un moyen d'afficher ces paramètres : le système permet également de dessiner les légendes sous des formes classiques en cartographie (caissons, symboles superposés ou contigus, etc.). Bien sûr, la requête peut être vide : l'utilisateur dessine alors les objets en fonction de leurs attributs d'origine. C'est la démarche habituelle lorsque l'on utilise le SIG uniquement à des fins d'exploration des objets.

L'explorateur cartographique permet également de dessiner des masques (en indiquant couleur et trame), des images géoréférencées, ainsi que des fonds graphiques. Les fonds graphiques sont des fichiers contenant exclusivement du dessin, sans la notion d'objet ou d'attribut descriptif. Les fichiers au format DXF sont un exemple de ce type de document. Le système SAVANE permet à l'utilisateur d'afficher ce type de dessin dans un cadre, à condition bien sûr que les coordonnées soient compatibles : le repère doit être connu. Le dessin n'a pas forcément besoin d'être dans la même projection géographique que le cadre, le système se charge de la transformation de projection si un fichier décrivant la projection du document a été créé.

Le système permet également la représentation en trois dimensions si les données s'y prêtent (fig. 2.5). Dans ce cas, l'image calculée en perspective peut être collée dans la carte, ou sauvegardée comme un fichier image à part.

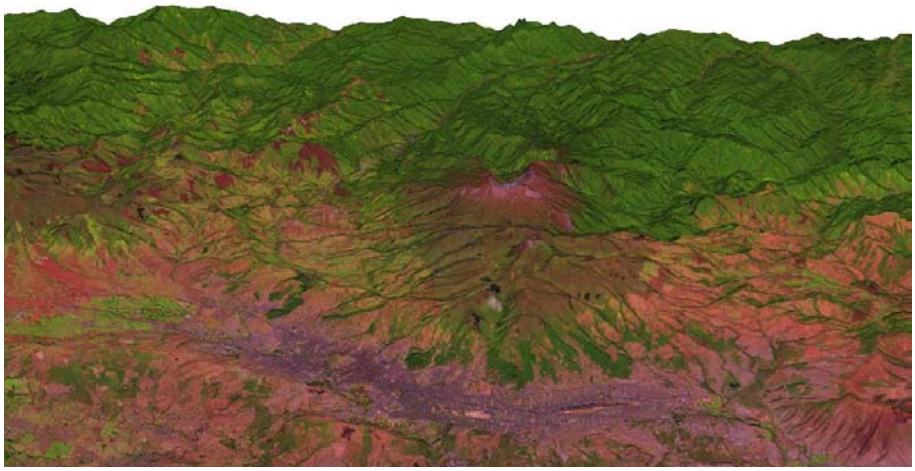


fig. 2.5 : une image satellite mise en perspective grâce à un modèle numérique de terrain

2.2.4.6. Utilisateurs et partage des bases de données

Chaque utilisateur doit avoir un espace de travail réservé car de nombreux traitements créent des fichiers temporaires qui ne doivent pas être mélangés entre différents utilisateurs. De même, les cartes et tous les objets graphiques qu'elles contiennent sont stockés dans un répertoire de l'utilisateur qui les a produites. Cet espace de travail est créé lors de la création de l'utilisateur avec le module SAVATECA.

Les bases de données peuvent être partagées par tous les utilisateurs du système. Chaque module actualise le schéma de la base dès qu'une modification a été effectuée par l'administrateur avec le module SAVATECA. Les états temporaires des bases de données sont stockées dans les cartes correspondantes, et se trouvent donc dans le répertoire de l'utilisateur.

2.2.4.7. Vecteur et raster

C'est une question qui était posée pendant de nombreuses années aux concepteurs de systèmes d'information géographique : *vecteur* ou *raster* ? La question concernait le mode de stockage de l'information de localisation (par arc ou par pixel) et, donc, implicitement, le mode de résolution et la précision de certaines procédures impliquant la localisation, notamment pour les mises en relation spatiale.

Dans le système SAVANE, le stockage des données géographiques est résolument vecteur, pour tout ce qui est point, ligne, ou zone. Le type pixel s'apparente à un stockage *raster*, mais la résolution (la taille du pixel) est variable et est directement liée à la précision de la donnée : il n'y a pas de dégradation de l'information due au stockage. Par contre, le système crée une matrice *raster* pour la résolution rapide de certaines opérations impliquant des relations zonales ou des masques, lorsque la précision de cette matrice est suffisante pour l'opération en question, ainsi que pour améliorer les temps d'affichage sur écran. Cette opération de rasterisation est transparente pour l'utilisateur.

2.2.5. SavATLAS : présentation d'atlas et de données

Ce petit module permet de présenter une série de cartes et de notices associées, sous forme d'un atlas électronique. Plusieurs chemins de lecture peuvent être associés à la série de cartes (fig. 2.6). L'utilisateur peut consulter les données correspondantes aux objets représentés dans une carte en cliquant directement sur l'écran. Les valeurs sont alors recherchées dans la base de données.

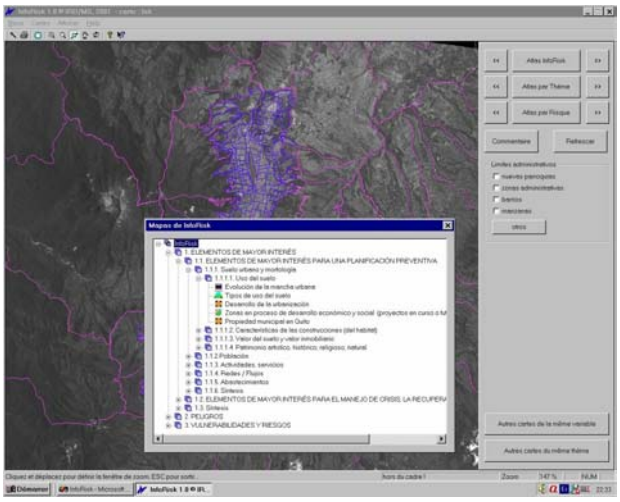


fig. 2.6 : l'écran principal d'une application de type SavAtlas

2.2.6. SavLIBRARY : librairie de développement C++

Cette librairie de classes C++ permet l'intégration de fonctions de consultation ou d'interrogation d'une base de données SAVANE dans un programme C++. Elle s'inspire des objets de type DAO (DataBase Access Object) de la MFC (Microsoft Fundation Class). Elle comprend notamment les classes suivantes :

- CSavaneDatabase : pour ouvrir une base de données Savane,
- CSavaneRelation : pour décrire et utiliser une relation de la base de données ouverte,
- CSavaneAttribut : pour décrire et utiliser un attribut d'une relation de la base de données ouverte,
- CSavaneWind : pour décrire et utiliser l'espace géographique d'interrogation, et pour gérer les projections géographiques,
- CSavaneLecture, CSavaneRecordset, CSavaneArc, etc.

2.2.7. SavSERVEUR : serveur de requêtes via Internet

Cette application permet d'implémenter une architecture client/serveur en recevant les requêtes de type SAVANE et en renvoyant sur le client le résultat obtenu. L'interrogation se fait via une liaison TCP/IP, et peut donc être effectuée via Internet.

2.3. La constitution et l'exploitation d'une base de données avec SAVANE

2.3.1. La constitution d'une base de données géographiques

2.3.1.1 Méthodes de mise en place, d'administration, et d'exploitation

Comme pour tout système d'information, la mise en place d'un SIG important est une opération délicate qui demande une approche rigoureuse et l'évaluation précise des besoins et des moyens disponibles. L'analyse générale décrite ci-dessous doit aboutir à la décision de faisabilité et au calendrier de mise en place. Elle se compose de trois étapes principales :

- la rédaction d'un cahier des charges décrivant les objectifs et les besoins de l'application, l'évaluation des données nécessaires au fonctionnement ainsi que les flux d'acquisition ;

- l'évaluation des spécifications du système et de ses objectifs, en fonction du cahier des charges, puis l'étude des systèmes existants sur le marché ; cette étape doit déboucher sur une évaluation de la faisabilité de l'opération et des coûts qu'elle implique (matériel, fonctionnement) ;

- l'évaluation finale des différents choix possibles en termes de bénéfice et de coûts, en prenant en compte l'ensemble des conclusions des deux premières étapes.

A partir des décisions d'acquisition, l'organisation logique de la mise en place et de l'exploitation sera décomposée en sous-systèmes suivant une analyse organique précise :

- un organe de mise en place et d'administration générale du système, qui aura la charge de répondre aux besoins humains et financiers tout au long de la période de développement et de mise en place, d'établir les plans de formation et d'assistance aux utilisateurs, de gérer l'évolution future en fonction des résultats d'exploitation. Il aura également la charge de rassembler et gérer la documentation technique de l'ensemble des sous-systèmes ;

- un organe d'acquisition de données qui va permettre de gérer les divers flux d'informations. Les flux peuvent être réguliers ou propres à une application précise. Il aura la charge d'évaluer et de décrire les sources d'informations, les modalités d'accès, les procédures d'acquisition, et de construire et gérer l'ensemble des méta-données. Cet organe est le plus important pour le fonctionnement régulier d'un système d'information. Il devra également guider et renseigner les utilisateurs en insistant sur les contraintes imposées par un système d'information géographique : conception uniforme des données, coût et durée de la saisie, coût de l'exploitation, etc. ;

- un organe de saisie et d'intégration de données : structuration, homogénéisation, validation, codage, saisie, contrôle, correction et intégration des données suivant les techniques requises par le système d'information et de manière à

produire des informations directement exploitables. Les différents problèmes de la saisie graphique seront traités par cet organe ;

- un organe d'exploitation et d'analyse de données assurant la réponse aux demandes des utilisateurs. Cet organe pourra être conçu de manière à faire participer les usagers à l'utilisation interactive des logiciels en fournissant assistance, conseils et documentation. Il pourra également être conçu comme un bureau d 'étude et fournir les réponses aux questions posées en se chargeant de l'ensemble de la procédure d'exploitation et de mise en forme définitive des résultats.

La nomination d'un responsable ayant la direction de l'ensemble de ces organes est essentielle : l'administration d'une base de données est un poste de haute responsabilité.

2.3.1.2 Difficultés

Si les apports et possibilités d'un système bien géré sont très importantes, les contraintes d'utilisation sont également nombreuses. Pour peu qu'une mauvaise analyse initiale ait oublié de spécifier tous les moyens nécessaires à une bonne mise en place, notamment en ce qui concerne les flux de données, alors commencent retards et difficultés d'exploitation qui peuvent facilement mener à des bilans financiers désastreux.

Les besoins doivent également être analysés avec le plus grand soin, car il faut éviter de mettre en place un *système complexe* lorsque cela ne s'impose pas. Dans de nombreux cas, la consultation de documents cartographiques ou graphiques, sans notion de base de données, peut être suffisante : les moyens à mettre en œuvre seront alors bien différents, et beaucoup plus légers et faciles à gérer.

2.3.2. SAVANE : de la création d'une base de données à son exploitation

Nous allons décrire ici rapidement les principes de mise en place d'une base de données géographiques avec le SIG SAVANE.

2.3.2.1 La création d'une base de données : schéma et méthodes

La base de données va être créée à partir de l'analyse des besoins et des flux d'informations. L'administrateur du système va donc créer le schéma de la base, en spécifiant la structure de l'information en suivant les principes du modèle relationnel étendu à la localisation. Le schéma comprend la définition des relations, des attributs, des regroupements de relations, des regroupements d'attributs, des méthodes, des vues externes.

Le schéma peut évoluer dans le temps : c'est toujours l'administrateur qui aura la charge de modifier ce schéma, en ajoutant, supprimant, ou modifiant des éléments de ce schéma, en principe toujours en fonction des prévisions établies dans le cahier des charges et dans l'analyse fonctionnelle du système.

L'ensemble de ces opérations sont effectuées avec le module d'administration SAVATECA.

2.3.2.2 La digitalisation et l'intégration dans la base de données

La digitalisation des documents graphiques s'effectue avec le module SAVEDIT après préparation des documents suivant le schéma des données, ou par importation à partir de documents numériques digitalisés sous un autre format. La digitalisation requiert plusieurs étapes :

- une homogénéisation des différents documents,
- leur transformation en image scannée, puis le redressement de ces images avec le module SAVAMER,
- la saisie des éléments correspondant à la schématisation,
- le contrôle et la correction des erreurs.

La saisie graphique peut bien sûr être répartie en plusieurs postes. Il faut alors gérer la qualité et la vitesse des différents opérateurs pour satisfaire le chronogramme établi pour la phase de saisie graphique et assurer la qualité des résultats.

Le contrôle et la correction des erreurs doivent être effectués au fur et à mesure et avec le plus grand soin. Le module SAVEDIT contient de nombreuses fonctions permettant d'assurer la cohérence graphique, topologique, sémantique des objets saisis.

Les documents saisis sont ensuite intégrés dans la base de données. L'intégration crée les objets dans la base et les indexe en fonction de leur localisation. Les données descriptives sont intégrées dans une seconde phase, par jointure avec les objets grâce à une clé de jointure descriptive. Toutes ces opérations se font sous le contrôle de l'administrateur de la base de données.

2.3.2.3 Création et gestion de méthodes

Une fois défini le schéma des relations et attributs, l'administrateur peut également définir des méthodes d'utilisation qui vont se placer dans le schéma de la base de données. La définition de ces méthodes peut intervenir à tout moment : certaines méthodes dérivent directement de la définition des relations et attributs, d'autres peuvent être constituées au fur et à mesure de l'utilisation de la base de

données et être ajoutées dans le schéma par la suite, de manière à enrichir la connaissance sur la base de données par l'utilisation qui en est faite. Un utilisateur peut donc proposer à l'administrateur d'intégrer une méthode dans la base, si cette méthode correspond à un besoin général ou permet d'enrichir la connaissance. Sinon, l'utilisateur conserve ses méthodes pour son propre usage, mais elles n'apparaissent pas dans le schéma de la base de données.

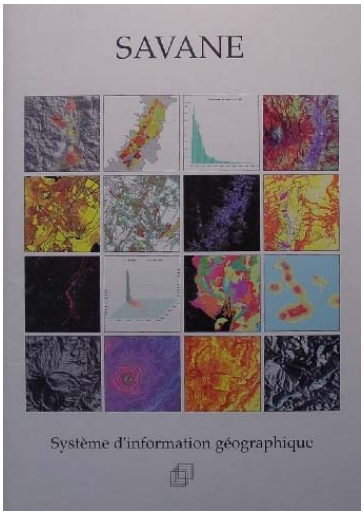
2.3.2.5 Vues externes et droits d'accès

Une fois le schéma de la base de données constitué, l'administrateur peut définir des droits d'accès en constituant des vues externes sur la base de données. Ces vues externes contrôlent l'accès aux relations, à leurs attributs, ainsi qu'aux méthodes définies dans le schéma. Elles permettent également d'avoir une vision hiérarchique de la base de données par la création de dossiers.

Bibliographie

[SOU 87] SOURIS M., *A Geographic Information System with relational architecture : principles and example of use*, in Primera conferencia sobre informatica y geografia, 1987, San Jose, Costa Rica, pp 703-728

[SOU 90] SOURIS M., *Le système SAVANE, Manuels de référence*, 1990-2002, IRD



Deuxième partie : Modéliser et gérer l'information géographique

Chapitre 3

Modéliser la réalité : cartographie, géographie, géométrie, et informatique

L'extension du modèle relationnel utilisée dans le SIG SAVANE est construite sur la prise en compte de types de données provenant de la modélisation du monde réel en deux ou trois dimensions, types qui ne sont pas ou mal gérés par les systèmes de gestion de base de données classiques. Dans ce chapitre, nous allons donc tout d'abord rappeler les principes de cette schématisation, avant de présenter dans le chapitre 4 les méthodes de mesure et dans le chapitre 5 les méthodes de représentation et de gestion de ces types de données dans le cadre des systèmes de gestion de bases de données étendus aux données localisées. Cette réflexion, très générale, nous servira de base dans l'ensemble de cet ouvrage. Elle nous permet de revenir sur les concepts de base de la cartographie et de la géographie, et de montrer comment l'élaboration de l'information géographique influence de manière décisive l'utilisation qui en est faite dans les systèmes d'information géographique et dans les procédures d'analyse spatiale.

3.1. Préambule

Une des plus grandes difficultés qu'ait toujours eu à affronter la géographie est celui de la précision de son observation et donc de la précision de la description du phénomène observé. La nouveauté tient dans le fait que nous disposons d'une avalanche d'information de plus en plus abondante et variée qui modifie considérablement notre perception du réel. Dans le même temps, les limites dans la gestion et le traitement de ce volume d'information ont été repoussées grâce à l'évolution de la technologie. Tout cela concourt à faire de la géographie une

discipline en profonde mutation ; mutation dans sa perception du réel, mutation dans ses concepts et méthodes. La mise en oeuvre et l'utilisation des SIG oblige à une complète remise à plat de nos certitudes et à une réflexion approfondie sur l'information géographique, sa schématisation, son traitement et sa représentation. La validité et la fonction de la carte, support privilégié à la fois de la schématisation et de la représentation, s'en trouvent profondément modifiées.

Au fond, le souci de pouvoir profiter d'une information sans cesse plus abondante procède d'une certaine suspicion à l'égard de celle-ci. Une méfiance qui se justifie non pas tant parce qu'elle serait suspecte de manipulation mais plus simplement parce qu'il n'est pas d'information qui ne soit élaborée en fonction d'une vision schématique du réel. D'où l'idée, parfaitement justifiée, de multiplier et confronter les points de vue par le traitement simultanée d'informations d'origines très diverses. Mais, c'est aussi tout le paradoxe de ce type d'approche, le regard critique que nous portons sur l'information géographique et sur la validité de sa localisation exige de brasser toujours plus d'information. Et plus l'information est abondante et diversifiée, plus la réalité géographique nous apparaît complexe et inaccessible.

3.1.1. Quelques exemples

Nous sommes entourés d'innombrables objets, des quantités d'événements se produisent autour de nous, nous-mêmes faisons partie et sommes acteurs de divers mouvements et comportements. Pour mieux comprendre le monde qui nous entoure et sur lequel nous agissons, nous avons donc depuis longtemps entrepris d'en faire l'état des lieux, ou si l'on préfère, l'inventaire. A la manière d'une comptabilité analytique nécessaire à la bonne gestion de toute entreprise, cet inventaire implique des rubriques et des critères. Si on introduit la localisation de ces objets dans leur description, c'est bien sûr parce que l'on suppose que cette localisation intervient de façon essentielle dans la définition ou dans l'interprétation de l'objet. Il faut donc maintenant envisager la manière dont on décrit celle-ci. A travers quelques exemples nous montrerons que la description de la localisation et son niveau de précision sont indissociables de l'objet que l'on souhaite décrire, et donc, de la question posée. Nous allons successivement envisager la description de plusieurs objets géographiques : une rivière, un terroir villageois, un milieu, une région, une ville.

Rivière ...

Plaçons-nous au bord d'une rivière avec l'irrépressible envie de vouloir la décrire. Que peut-on faire ? Sans doute commencer par situer l'endroit où l'on se

trouve ; puis, mesurer la largeur du lit, établir un profil transversal au moyen de mesures régulières de la profondeur. Il sera sans doute utile de mesurer la vitesse du courant afin de calculer un débit. Une légitime curiosité sur l'activité biologique de cette rivière justifiera la mesure de la turbidité de l'eau ; des prélèvements d'eau, analysés en laboratoire, livreront des indications précieuses sur l'abondance du plancton...

Toutes ces observations faites, sommes-nous réellement en mesure de proposer une bonne description de la rivière ? Evidemment non. Tout au plus, une bonne description d'un lieu sur la rivière en un lieu et un instant précis. On ne sait rien de sa longueur et de son tracé, on ignore tout de sa source, de son embouchure, du nombre d'affluents qui l'alimentent, des affleurements rocheux qu'elle franchit tout au long de son cours et de la végétation qui la borde... Bref, on sait fort peu de choses de l'objet que l'on prétendait décrire. Autrement dit, l'objet géographique ainsi appréhendé ne peut en aucun cas être la rivière en tant que totalité.

Mais la démarche était-elle la bonne ? ou est-ce au contraire la question qui n'avait que peu de rapport avec la démarche adoptée ? Dire que je veux décrire La rivière c'est déjà percevoir celle-ci dans sa globalité et c'est poser cette globalité comme un objet en soi. On voit donc que le problème apparemment simple de description d'une rivière s'avère être dans la pratique d'une immense complexité. Ainsi, pour appréhender cet objet complexe qu'est une rivière, sommes-nous dans l'obligation de mieux sérier les questions, chacune d'elle impliquant une méthode adaptée à la localisation de l'objet et réciproquement. Vouloir décrire le tracé de cette rivière fait appel à des descripteurs de localisation différents de ceux qui seront employés pour localiser son embouchure ou tel ou tel confluent. Donner une évaluation du débit total de la rivière implique la mise en oeuvre de protocoles de mesures particuliers tout en sachant qu'on ne connaîtra jamais son débit réel. Ainsi pour tout objet géographique devons-nous nous poser les questions suivantes : quel est l'objet que je veux vraiment décrire ? de quel point de vue dois-je l'aborder pour appréhender sa totalité ? Puis-je séparer cette totalité en éléments simples ? Quel degré d'approximation du réel me parait acceptable ?

Terroir ...

Changeons de continent et imaginons un de ces beaux villages de l'Afrique soudanienne, niché dans l'immensité et la monotonie de la savane. Autour des cases, harmonieusement regroupées dans des enclos, s'étendent les auréoles des champs. Du village jusqu'à ces derniers, de nombreux sentiers dessinent une sorte de toile d'araignée compliquée et irrégulière dont les fils finissent par se perdre à mesure que l'on s'éloigne du centre et que l'on pénètre dans la savane. Les champs eux-mêmes se distinguent nettement en fonction de la proximité du village. Aux

parcelles soigneusement délimitées et parfaitement entretenues succèdent, à mesure que l'on se dirige vers la périphérie du terroir, des parcelles aux formes beaucoup plus irrégulières et aux limites d'autant plus floues que les plantes adventices colonisent les bordures des champs.

D'ores et déjà, un terroir villageois s'annonce donc comme un objet singulièrement complexe, et ce d'autant plus que l'on soupçonne aisément qu'il existe un rapport étroit entre les habitations (et donc ses habitants), les champs, les sentiers et la savane qui a été défrichée pour permettre l'installation de ce village. Tous les descripteurs du terroir villageois sont localisables. On peut évidemment décrire séparément chacun des éléments constitutifs du terroir et l'on sent bien que chacun d'eux renvoie à une classe d'objets de nature différente mais également géographiques. Ainsi, un objet géographique localisable peut être lui même le produit, ou le résultat, d'une combinaison de plusieurs objets géographiques de nature différente mais également localisables.

Poursuivons la réflexion et tenons-nous en à la localisation de cette combinaison, le terroir villageois. Vu sur une photographie aérienne, le village, les sentiers qui s'en éloignent et les parcelles gagnées sur la savane trancheront sur la relative homogénéité de cette formation végétale. A ce niveau, un terroir villageois peut donc assez aisément être décrit comme une tache, de forme plus ou moins circulaire, localisée *quelque part* dans la savane soudanienne. Mais préciser la

localisation de cette tache implique nécessairement de préciser la description de l'objet. Quelle référence prendra-t-on ? le point sombre du grand manguier qui constitue le centre du village - et, par extension du terroir ? le détail du tracé de chacune des parcelles ? ou seulement la limite extérieure du terroir ?

En fait, la localisation n'ayant de sens que par rapport à l'objet que l'on analyse, il va de soi que ces trois localisations sont admissibles, soit séparément soit simultanément, mais que chacune, renvoyant à une collection d'objets différents, dépend de l'objectif visé et donc, de l'information descriptive qu'il s'agira de recueillir puis de traiter ;

- considérer le terroir comme un point sur une carte ne permet d'utiliser la localisation comme un descripteur supplémentaire que si l'on rapporte ce terroir à d'autres terroirs également considérés comme des points. En effet, dans l'hypothèse où l'analyse ne porte que sur un seul terroir on ne voit pas ce que peut apporter une localisation aussi imprécise de l'objet. Dans ce cas de figure, la mention de la localisation se résumera à l'énoncé, sans véritable intérêt, des coordonnées géographiques de ce point. Comme pour mieux assommer le lecteur, combien d'ouvrages ne commencent-ils pas ainsi : « le village de ... se trouve situé par 4 ° 38' 45" de latitude Nord et 3° 27' 56" de longitude ouest » ? Dès lors le village ne peut plus être considéré comme un objet géographique. Il ne le redeviendra que si l'analyste ne considère plus un seul terroir pris isolément sinon l'ensemble des terroirs compris dans un espace donné. Chaque terroir, et donc chaque point, peut alors être comparé à tous les autres points. Des mesures de distance pourront être effectuées, et des regroupements significatifs de villages ne manqueront pas de se révéler.

- à l'inverse nous pouvons considérer le terroir comme un objet en soi, en dehors de tous les autres terroirs proches ou lointains. Nous pourrions croire que les choses devraient s'en trouver simplifiées par la réduction du nombre d'objets (un terroir au lieu de plusieurs). C'est pourtant le contraire qui se produit puisque la localisation de cet objet unique ne peut plus être assimilée à un point. Avec ses cases, ses parcelles et ses sentiers, le terroir villageois est un objet complexe. L'observateur souhaitera sans doute établir la relation entre les familles vivant dans chacun des enclos et les parcelles cultivées ; ces dernières, de forme et de surface différentes, seront plus ou moins fertiles selon l'exposition, la nature des sols et la proximité d'un cours d'eau. Il sera enfin tout aussi essentiel de mesurer le temps et la distance parcourue par les femmes pour aller puiser de l'eau ou ramasser du bois et il sera probablement justifié d'en mesurer les effets sur la gestion des activités agricoles collectives. Enfin, la forme et l'aspect même des parcelles éloignées justifiera certainement une question subsidiaire : toutes les parcelles appartiennent-elles à la même classe d'objets ? Ou, au contraire, faut-il distinguer les champs cultivés en

permanence de ceux qui, du fait de leur éloignement, sont périodiquement laissés en jachère ?

A chaque entité, enclos familial, parcelle, sentier, seront donc associées un certain nombre d'observations localisées dans l'espace et dans le temps. Les déplacements seront associés à des objets géographiques linéaires, les sentiers ; la fertilité des sols et la production agricole sera ramenée aux parcelles. Enfin, les revenus tirés de cette production sera rapportée au chef de famille habitant la case principale.

Région ...

S'il existe un objet géographique à la fois aussi « évident » mais en même temps aussi incertain et controversé, c'est bien celui de région. Nous voulons parler ici d'une forme plus complexe d'organisation de l'espace qui se construit d'elle-même et connaît de constantes modifications au gré des dynamismes locaux et des concurrences interrégionales.

En première analyse la région fait partie de ces espaces intermédiaires entre le niveau local d'une ville ou d'une commune et le niveau plus englobant de la nation toute entière. Nous disons bien, en première analyse, car, comme pour compliquer les choses, bon nombre de régions économiques et culturelles n'existent que par leur position frontalière, et dans ce cas, c'est bien souvent la limite du territoire qui donne vie à la région ...

Dans les pays de vieille tradition urbaine, chaque région intègre une ou plusieurs grandes villes et tout un semis de bourgs et de petites localités. Il y a bien longtemps que les géographes ne s'arrêtent plus à la question de l'homogénéité de la région. A moins d'évoquer des *régions qualifiées* (région agricole, région de savane, région de montagne, ...), la principale caractéristique de la Région serait au contraire d'associer une grande diversité d'espaces -de micro régions si l'on préfère- plus ou moins complémentaires en terme d'activités agricoles, industrielles et commerciales. Toute région est enfin traversée par un grand nombre de flux, les uns centripètes et orientés vers la ou les villes principales, les autres centrifuges, à destination d'autres régions, proches ou lointaines.

A une toute autre échelle que le terroir villageois, la région s'apparente elle aussi à un objet géographique complexe composé d'objets géographiques eux-mêmes « simples » (parcelles, usines, équipements, ...) ou complexes (villes). C'est la complexité même de l'objet qui rend difficile la description de sa localisation. La région, telle que nous prétendons l'appréhender ici, serait « plutôt » à ranger dans la catégorie des objets zonaux. Pourtant la difficulté qu'il y a à identifier des frontières

précises et incontestables montre bien que nous ne sommes pas dupes de notre propre schématisation, même si celle-ci est le plus souvent passée sous silence. C'est tout le problème de la plupart des objets géographiques complexes que nous assimilons -faute de mieux- à des zones. Or, toute délimitation zonale introduit l'idée d'une continuité interne et d'une discontinuité externe.

Milieu ...

La notion de « milieu naturel », suppose d'abord qu'il en existe une infinie variété. C'est la première discontinuité que nous introduisons dans le monde qui nous entoure car il va de soi que ce vocable ne serait pas employé si, pour décrire l'environnement, nous n'étions pas dans l'obligation de sélectionner et séparer les objets qui le composent. La notion de milieu naturel est l'exemple même de la réduction du réel continu à des objets géographiques discontinus beaucoup plus simples.

Dans son acception la plus large, le milieu naturel correspond chez les géographes et l'ensemble des naturalistes à une entité géographique considérée comme homogène, au niveau d'analyse choisi, et résultant d'une combinaison particulière dans un espace donné ; combinaison entre certains caractères climatiques, topographiques, pédologiques, géologiques, hydrologiques et botaniques. Cette approche intègre fort mal la dimension temporelle et pose au contraire l'hypothèse d'une grande permanence dans le temps. Un milieu naturel se caractérise donc plutôt par sa moyenne que par ses écarts. D'où son nom : mi-lieu. On le définit par un régime climatique moyen, par des sols qui « dans l'ensemble » sont « plutôt » de tel type, une végétation « dominante », etc.

L'information la plus précise dont disposent les chercheurs pour mettre en évidence la diversité des milieux provient pour l'essentiel des relevés et observations de terrain qu'ils réalisent ; profils pédologiques, transects, stations météorologiques, sondages, etc. Cette information ponctuelle, donc facile à localiser, est ensuite étendue par des méthodes d'interpolation ou d'extrapolation. On attribue donc les résultats d'une observation localisée à un objet géographique zonal beaucoup plus vaste. Dans la mesure du possible le chercheur aura à cœur d'effectuer de nouvelles mesures sur le terrain afin de valider cette extension de l'observation localisée. Il n'en demeure pas moins, comme souvent pour les objets géographiques zonaux et malgré l'incertitude des limites, que l'on est amené à créer des objets géographiques supposés continus à partir d'une information que la précision de la localisation ne devrait pas autoriser, sans s'entourer de nombreuses précautions.

Ville

La ville est un objet formidablement complexe. Un système, qui pour être décrit en terme d'objet géographique, doit être décomposé en sous ensembles d'objets plus simples : immeubles, monuments, sièges sociaux; quartiers, arrondissements, commune ; réseaux de transport, d'eau de gaz, d'électricité, d'égouts ; commerces, industries, banques, services ; rocades, boulevards, avenues, rues, impasses ; ouvriers, fonctionnaires, cadres, ...

Bien sûr une ville c'est cela, mais c'est beaucoup plus que cela. C'est surtout l'interrelation et l'interdépendance de l'ensemble de ces objets et de bien d'autres encore. Dès lors, la description d'une agglomération urbaine ne peut se réduire à la seule analyse de ses éléments matériels stables et visibles.

3.1.2. Précision et description

De la grande diversité des objets géographiques évoqués à titre d'exemple tentons maintenant de dégager quelques conclusions. Certaines paraîtront parfaitement évidentes, mais il n'est pas inutile de les rappeler tant elles sont lourdes de conséquences, qui, elles, n'apparaissent que lorsque l'on cherche à modéliser ou représenter la réalité avec un ordinateur.

La perception visuelle d'un objet dépend à la fois de sa dimension et du point de vue de l'observateur. Un observateur placé au bord d'une rivière se trouve dans l'impossibilité de voir la totalité de cette rivière. Pour contourner cette difficulté, la

seule solution consiste à s'éloigner de cet objet. Ainsi, plus la dimension des objets géographique est importante plus il faut s'en éloigner pour en appréhender la totalité. Ainsi, on sait qu'on appréhende mieux les formes, les proportions et l'altitude d'une montagne vue d'une certaine distance que lorsqu'on a le nez collé à la paroi rocheuse.

Quel que soit le point de vue de l'observateur, la perception d'un objet géographique reste toujours fragmentaire et partielle. En effet, plus on s'éloigne de la montagne pour l'appréhender dans sa totalité plus s'estompe le détail de sa végétation, de ses vallées et ravines, mais plus les principaux éléments structurants de cette montagne apparaissent clairement. Ainsi, à toute distance par rapport à l'objet correspond un niveau particulier de schématisation de cet objet.

Toute description d'un objet géographique se traduit par une schématisation de sa dimension, de sa forme, de sa distance, de sa localisation, de son comportement. La perception de tout objet est fonction de l'objectif poursuivi par l'observateur. Il n'existe *en soi* ni précision absolue, ni « bonne » ni « mauvaise » précision. Dans la mesure où l'objet géographique n'existe que par rapport à l'observateur, une montagne ou une rivière n'ont ni plus ni moins de réalité qu'une région, une nation ou toute autre abstraction créée pour les besoins de telle ou telle cause. Ce que bien souvent on nomme « échelle » correspond à une certaine description d'un objet géographique. Contrairement à la notion d'échelle en cartographie (c'est-à-dire le rapport entre un objet réel et sa représentation sur la carte), l'échelle de description correspond à une modélisation du réel, à un niveau de description qui lie étroitement la précision de la localisation et les autres critères de la modélisation.

3.2. Pourquoi et comment modéliser le monde réel

La réalité géographique se révèle donc être bien difficile à modéliser : une multitude d'objets, des regards et des points de vue différents, une imbrication hiérarchique de diverses descriptions, des détails qu'il convient d'oublier pour avoir une vision globale, des limites floues, des interactions multiples entre différents éléments, une représentation multiforme. Loin d'une schématisation naturelle ou universelle, voici le lecteur prévenu : il entre dans le territoire des monstres.

3.2.1. Comment appréhender et représenter la réalité pour la traiter avec un ordinateur ?

La cartographie et la géographie se sont construites peu à peu sur la base de possibilités techniques qui, pendant longtemps, n'ont pas beaucoup évoluées.

Depuis une trentaine d'années, tous ces fondements sont les uns après les autres remis en cause par des possibilités techniques inédites apportées par le développement de l'informatique et le pouvoir de modélisation et de calcul des ordinateurs. Il est donc utile de revenir sur des concepts de bases et notamment sur la question de la schématisation et de la modélisation du monde réel, l'objectif étant de répondre à un problème donné en utilisant un ordinateur, c'est-à-dire fondamentalement une machine capable de représenter et de traiter des connaissances. L'aspect calcul de l'ordinateur est ici secondaire : il doit d'abord être considéré comme une machine permettant de conserver et de traiter une représentation formelle d'une réalité.

Face à une réalité et un désir d'étudier ou de gérer cette réalité dans un certain but, il faut schématiser cette réalité et exprimer cette schématisation de manière à pouvoir gérer les objets qui en découlent. L'information géographique n'existe pas en tant que telle : ce n'est que par rapport à une *vision du monde* et face à un *objectif donné* que l'on peut modéliser le monde réel, en fonction de cet objectif, et que l'on va passer de la réalité à l'information géographique. La description ne peut être universelle : elle s'effectue toujours par rapport à un objectif, avec une précision donnée et des descripteurs qui sont définis par rapport à cet objectif et cette précision. Et c'est bien souvent la difficulté première à laquelle est confronté l'utilisateur d'un SIG : comment les données qu'il va utiliser ont-elle été conçues, comment ont-elles été structurées, dans quel but, avec quelle précision, quelle validité, etc. Pour ne pas rendre un SIG inefficace, ou même dangereux à utiliser, il faut pouvoir répondre à ces questions. Il faut pouvoir affirmer que la réalité est devenue information ou connaissance sans que cette transformation ne la vide du sens qui va lui être donné. Car les transformations sont nombreuses, de la réalité à l'écran : une interprétation par rapport à un contexte, soit culturel, soit procédural, une modélisation par rapport à un objectif, une structuration par rapport à un modèle, une représentation en rapport avec des possibilités techniques, une utilisation par rapport à un problème donné.

Bien sûr, de nombreuses représentations du réel nous paraissent *naturelles*, tellement elles font parties de notre société ou de notre culture. Nous essayerons même plus tard d'en répertorier un certain nombre, de manière à les proposer, comme un canevas initial, au concepteur d'une base de données adhérant à cette vision du monde ou devant répondre à un problème *relativement* universel. Ces classes d'objets, comme nous les appellerons, correspondent à la vision habituelle d'une structuration de l'espace, en fonction de la précision des données initiales et de l'échelle des cartes qui servent habituellement à les représenter graphiquement.

3.2.2. Réalité et connaissance

Nous sommes à tout instant confrontés à une réalité dont notre vision est fortement contextuelle. Nous ne voyons ce qui nous entoure qu'avec nos possibilités techniques et que par rapport à nos besoins, ou plutôt nous ne retenons de ce que nous voyons que ce qui sert nos besoins, suivant une schématisation, un apprentissage, une représentation interne « naturelle » qu'il est bien difficile de connaître. L'homme se sert de son fort pouvoir d'abstraction et de sa faculté d'oublier, d'une façon quasi inconsciente et automatique mais extrêmement efficace.

Lorsque l'on cherche à appréhender une réalité, non plus de façon inconsciente, mais suivant des critères et des objectifs donnés, nous sommes amenés à essayer de faire ce même travail de modélisation et de représentation d'une réalité par le raisonnement et pour le raisonnement. Par le raisonnement, car il nous faut choisir un nombre fini de critères, de relations, de schémas nécessaires à la description et oublier le reste. De plus, on cherche à représenter et schématiser non pas seulement des faits, mais des connaissances, puisque un fait réel (incertain) devient un objet de connaissance à travers la vision de l'homme (vision qui n'est bien sûr pas unique, et qui dépend à la fois de l'homme et de son contexte). Pour le raisonnement, puisque le but de toute représentation du réel est l'accès au raisonnement, non plus instinctif, mais structuré. Si l'on cherche ainsi à représenter les connaissances, c'est essentiellement pour établir un lien entre des concepts du monde réel ainsi exprimés et des modèles théoriques permettant d'avoir accès au raisonnement.

3.2.3. Modéliser la connaissance

La connaissance comprend dès lors des faits (des objets) et des procédures pour les interpréter (des méthodes). Le problème de la définition d'un modèle de représentation de la connaissance est donc fondamental. On distingue l'aspect déclaratif (les concepts ou les objets et leurs descriptions), et l'aspect procédural (les procédures permettant d'interpréter et d'utiliser les concepts et les objets). Un schéma conceptuel de représentation de la connaissance comprend nécessairement les deux aspects. Les connaissances déclaratives semblent naturellement plus faciles à exprimer que les connaissances procédurales, mais elles n'ont en soi aucune valeur sémantique. C'est un interpréteur procédural (un homme ou un programme, plus généralement une méthode) qui exprime la façon dont elles vont être utilisées, car toute connaissance est contextuelle, même si de nombreux efforts ont de tout temps été entrepris pour rendre la connaissance universelle.

Plusieurs modèles de description de la connaissance ont été proposés [GAR 83] [SCH 89] [DAV 91]. Voici un rapide résumé des principaux :

- le modèle sémantique représente la connaissance sous forme d'un réseau, ensemble de nœuds (concepts) et d'arcs exprimant les relations qui peuvent exister entre ces nœuds. L'arc est orienté et représente l'action dont l'acteur est le nœud de départ, agissant sur le nœud d'arrivée. L'interprétation du réseau permet de décrire des hiérarchies d'actions entre les concepts. Les concepts peuvent représenter des objets réels, comme des situations ou des actions. Mais les concepts ne sont pas décrits par des attributs ou des variables : ce sont uniquement les nœuds et arcs du réseau qui décrivent la connaissance. Le modèle sémantique sert surtout à définir et organiser les concepts et les relations entre concepts.

- le modèle entité-association décrit les également des concepts et des associations entre concepts, mais chaque concept comme chaque association est décrit par des attributs, qui peuvent être considérés comme des concepts atomiques. Le concept devient l'objet.

- le modèle à objets complexes est très proche du modèle entité-association. La représentation des connaissance est basée sur des structures d'informations bien organisées, les schémas, en essayant de se rapprocher au maximum du fonctionnement supposé de la mémoire humaine. On distingue alors le prototype (description du schéma), des objets eux-mêmes, réalisations particulières des prototypes. Un prototype est caractérisé par un certain nombre d'attributs, qui caractérisent les objets et définissent ses relations sémantiques avec les autres objets.

Le modèle orienté objet est une extension du modèle à objets complexes enrichi dans l'aspect procédural. Un objet, en terme d'information, sera donc par définition l'aspect d'un concept décrit par un nombre fini de critères. Le modèle orienté objet introduit des relations procédurales entre les objets, qui conversent donc entre eux par des procédures.

Les critères retenus pour décrire un objet peuvent s'apparenter à de simples données mesurables, peuvent prendre leur valeurs dans des espaces plus complexes (un espace métrique par exemple), comme être eux-mêmes d'autres objets, des associations entre objets, ou des évènements qui s'appliquent aux objets sous certaines conditions. En fait, il faudra la plupart du temps définir de nombreux objets et relations entre ces objets pour décrire la connaissance que l'on cherche à représenter.

3.2.4. Collection d'objets et gestion

Lors de l'étude ou de la gestion d'un problème, on est amené à identifier ou à mieux définir des objets composant ou décrivant ce problème. Souvent, la définition porte non pas sur un ou plusieurs objets, mais sur des ensembles d'objets, tous les objets d'un même ensemble étant par définition décrits par les mêmes critères. On a alors à étudier ou gérer non pas des objets isolés, mais des collections d'objets.

C'est même ce que l'on s'efforce de faire la plupart du temps : sachant que l'on ne peut décrire et étudier tous les éléments un à un, on cherche une description minimale suffisante pour l'étude envisagée et avec laquelle on pourra décrire tous les éléments composant le problème à étudier. On constitue donc des ensembles d'objets qui sont décrits par un nombre fini de critères identiques.

On cherche également à décrire et modéliser la réalité non pas seulement pour décrire et caractériser des objets, mais surtout pour comparer et différencier plusieurs objets ayant le même schéma de description. D'ailleurs, on modélisera souvent la connaissance de manière à représenter avec un même schéma un ensemble d'objets distincts, s'il semble plus simple de regrouper les objets qui se ressemblent en catégories (mais est-ce bien la démarche naturelle de la mémoire ou de l'intelligence ?). On passe donc de la notion d'objet à celle d'ensemble d'objets, en introduisant de nouvelles contraintes liées à la gestion de ces objets devenus éléments. Il devient alors souhaitable de ne considérer que des ensembles d'objets homogènes, afin de pouvoir contrôler la cohérence des objets les uns par rapport aux autres et afin d'effectuer des opérations ensemblistes de manière efficace. Au problème de la représentation des connaissances et de description des objets, s'ajoute donc le problème d'une gestion efficace d'un ensemble d'objets.

Les systèmes de gestion de base de données sont chargés de la gestion d'un ensemble d'objets, et les méthodes employées sont bien sûr fonction du modèle de représentation des objets. Suivant les contraintes que l'on s'impose pour la gestion, on sera amené à choisir tel ou tel modèle de représentation de la connaissance. Ce sont souvent les contraintes de gestion qui amènent à choisir un modèle de représentation, et non l'inverse : il est donc fondamental de connaître ces contraintes avant de choisir un modèle de représentation.

Plus la description des objets est complexe, plus une collection d'objets sera difficile à gérer : toutes les théorie des systèmes de gestion de base de données, que nous verrons au chapitre 5, reposent sur cette idée. Soit on impose de simplifier la description des objets, et on pourra alors assez facilement les gérer, mais au risque de trop simplifier la modélisation du réel ou d'avoir à définir de nombreux objets distincts et d'avoir à effectuer de nombreuses opérations entre ces objets pour retrouver une description acceptable de la réalité. Soit on essaye de conserver une modélisation plus complexe, quitte à avoir des difficultés de gestion et une faible marge de manœuvre dans les traitements. Modèle de description et modèle de gestion sont donc intimement liés. C'est encore une fois essentiellement l'appréhension de la réalité et l'utilisation de cette modélisation qui va nous amener à utiliser un modèle de description et de gestion plutôt qu'un autre.

3.3. La schématisation du monde réel : de la réalité à la géographie

La prise en compte de la localisation répond à deux préoccupations majeures : repérer pour se repérer, repérer pour décrire et interpréter. Pourquoi là et pas ailleurs ? Comment expliquer la distribution spatiale d'un objet ? Comment expliquer la distribution des objets les uns par rapport aux autres ? Ces questions sont, implicitement, au cœur de toute question *géographique*.

3.3.1. La localisation comme attribut : l'objet géographique

Nous avons à plusieurs reprises employé le terme « objet géographique », sans en avoir encore donné une définition formelle. Mais de ce qui précède, nous pouvons dégager un besoin général : modéliser la réalité pour étudier des situations ou des phénomènes ayant une composante spatiale. Implicitement, on suppose même que la localisation dans l'espace est l'une des *causes* du phénomène à étudier, et non une *conséquence* de ce phénomène. Nous appellerons donc dans cet ouvrage « objet géographique » un objet modélisant un phénomène du monde réel pour lequel la localisation dans l'espace a valeur de causalité : autrement dit, on ne pourra avoir au même endroit deux objets géographiques correspondant à deux phénomènes de la même catégorie. On assure ainsi une dépendance fonctionnelle de l'espace et du temps sur le phénomène. Cette schématisation ne peut être appliquée aux objets localisés que s'ils sont regroupés en catégories, que nous appellerons plutôt collections. Un objet ne devient donc « géographique » au sens où nous venons de le définir que s'il appartient à une collection. Cette définition de se veut pas générale au sens de la géographie, car elle serait alors beaucoup trop simpliste : elle est directement liée à notre besoin de modélisation en vue d'une gestion relationnelle par collections d'objets du même type, et est guidée par l'obligation du respect de la contrainte d'unicité : on ne peut avoir deux objets d'une même collection au même endroit et au même moment.

Le terme collection utilisé ici recouvre le concept d'ensemble d'objets d'un même type. Ces objets sont décrits, en fonction de l'interprétation de l'observateur, par des variables élémentaires, appelés attributs. Un objet géographique a la particularité d'associer une localisation géographique et quelquefois temporelle à l'ensemble des attributs qui constituent la description non localisée de cet objet. L'attribut donnant la localisation est appelé attribut de localisation, pour le distinguer des autres attributs qui constituent l'information descriptive.

Il est important de noter qu'il n'y a pas de différence théorique entre information graphique et information descriptive, toutes deux constituent l'ensemble des attributs qui servent à décrire un objet de la collection déterminée par l'observateur ou le modélisateur. Par contre, la modélisation impose une dépendance fonctionnelle de l'une vers l'autre qui motive cette distinction sémantique et qui se traduit souvent dans les SIG par la séparation physique et

conceptuelle du graphique et du descriptif. En effet, la localisation géographique et temporelle d'un objet va déterminer l'ensemble des valeurs des autres attributs de cet objet : implicitement, on ne peut avoir, au même moment et au même endroit, des valeurs descriptives différentes pour un même objet d'une collection.

Nous verrons un peu plus loin dans ce chapitre comment décrire cet attribut de localisation. Nous pouvons néanmoins déjà souligner son caractère relativement universel, puisqu'il correspond toujours à la description d'un lieu, dans un espace qui est unique pour tous les objets rencontrés. La difficulté, pour modéliser le monde réel, se trouve bien dans les rapports entre la description, la schématisation, la précision de la localisation d'une part, et le contenu sémantique de l'objet et les attributs qui servent à décrire ce contenu d'autre part.

3.4. La schématisation de la localisation : de la géographie à la géométrie

A partir d'une réalité géographique complexe, le besoin de représentation graphique rend nécessaire la schématisation de la localisation. : la transition devient frontière, le lieu devient point... La géométrie remplace la description géographique, dans un processus de schématisation très radical.

3.4.1. Géographie et cartographie : une union agitée

La réalité géographique est donc complexe, souvent diffuse, et elle est le plus souvent décrite en utilisant la cartographie. Jusqu'à présent, en effet, la carte reste encore le procédé le plus efficace de représentation puisqu'il permet de présenter sur un même document visuel une représentation de l'objet géographique qui combine à la fois sa localisation et son contenu. Cependant, comme toute modélisation, la cartographie consiste à réduire le réel à des faits simplifiés et à un nombre fini de critères. Le problème qui se pose donc est celui des hypothèses et des techniques qui aboutissent à cette double schématisation de la localisation de l'objet et de l'information descriptive qui s'y rapporte. Aussi, on ne peut oublier que l'objet géographique est beaucoup plus riche que la carte qui est censée le schématiser. Il est donc absolument essentiel de comprendre et connaître pas à pas les étapes de cette schématisation, puisque le degré de validité du modèle en dépend.

La géographie n'a pas toujours su se faire comprendre des cartographes. Cette incompréhension relève bien d'une contradiction fondamentale. Au delà de la description du réel qui, chez le géographe, s'exprime le plus souvent par des cartes, celui-ci recherche aussi des modèles. Le cartographe quant à lui recherche la précision de la mesure, et ce souci le conduit toujours à un choix douloureux entre l'irrépressible désir d'une mesure parfaite et l'inévitable altération de la réalité

induite par la représentation cartographique de cette réalité. Dans le même temps, et c'est pour cette raison que les disciplines sont séparées, le géographe, sans que son collègue s'en soit véritablement rendu compte, est passé à tout autre chose. Sauf cas rares, il ne cherche plus à localiser de façon très précise et très exacte un phénomène, mais à en donner une vision simplifiée, ou plutôt schématisée, qui permette que soit compris sa nature et son fonctionnement dans un espace donné. Les SIG, comme nous le verrons tout au cours de cet ouvrage, permettrons en partie de réunir ces deux disciplines, et c'est bien ce qui fait l'une des forces de ces nouveaux outils.

3.4.2. Cartes de localisation et cartes thématiques

La vertu de toute carte de localisation bien faite est de ne prêter le flanc à aucune critique en terme d'interprétation. Elle réunit sur un plan des objets facilement repérables, donc visibles, localisés les uns par rapport aux autres en fonction de leur distance relative et de leur position par rapport au nord géographique. Ce type de carte introduit une première *relation* entre deux informations : la nature de l'objet et sa position dans l'espace.

La plupart des cartes sont porteuses d'autres enseignements que la seule localisation. Il est apparu de plus en plus utile de dresser des cartes dont la finalité première n'est plus de se repérer pour mieux suivre un itinéraire, mais plutôt de repérer un phénomène dans l'espace pour mieux l'appréhender. Nous en arrivons ainsi à la carte thématique.

L'abondance et la variété des cartes thématiques est telle aujourd'hui qu'il semble qu'on passe sans distinction claire de la carte descriptive à la carte interprétative qui prétend avoir valeur de modèle. Les premières seraient à rapprocher des cartes de localisation dans la mesure où la principale préoccupation semble être de reporter scrupuleusement les observations faites sur le terrain. Les cartes géologiques, celles des sols ou de la végétation en fournissent de très beaux exemples. Bien qu'élaborées par des spécialistes, le choix d'une représentation cartographique n'implique pas systématiquement la recherche de phénomènes complexes qui seraient à relier à leur position particulière dans l'espace. En effet, le premier objectif, mais aussi la principale qualité de ce type d'entreprise semble être de restituer au lecteur toute l'information recueillie, qui, à l'état brut, sous forme de carnets de notes et d'échantillons, reste évidemment impossible à interpréter par un non spécialiste.

3.4.3. La représentation géométrique de la localisation géographique

Il va de soi que nous n'aurions nullement besoin de nous interroger sur la représentation géométrique de la réalité géographique si nous n'avions pas fait auparavant le choix de représenter cette modélisation par une carte. Mais dès que ce choix est fait nous sommes bien obligés de nous demander comment nous allons figurer et représenter les différents objets géographiques que l'on souhaite décrire. Et ce choix va de nouveau fortement influencer notre modélisation du monde réel : par rapport à la diversité et la complexité des formes de localisation que prennent les différents objets (ville, région, terroir, paysage, ...) nous nous trouvons extraordinairement limités dans nos choix puisque selon l'objet, et donc selon l'objectif poursuivi, nous devrons nous résoudre à assimiler celui-ci à un élément ou un ensemble d'éléments géométriques : un point, une ligne, une zone.

Prenons l'exemple des villes. Pour simplifier l'explication on peut, très schématiquement, prendre trois orientations différentes :

- donner une image très précise de chaque ville en mettant en évidence sa diversité interne, la spécificité socio-économique de chaque quartier (1) ;

- comparer les attributs descriptifs de l'ensemble des villes d'un territoire (population, activité, etc.) (2) ;

- étudier les rapports et les flux (de transports de biens, de personnes, d'information) qui s'établissent entre l'ensemble des villes d'un territoire (3).

(1) Dans le premier cas, ce qui nous intéresse est ce qui se passe *dans* la ville. Cette notion contient deux implications fortes. La première est qu'on suppose l'existence de fortes différences entre ce qui se passe à l'intérieur de la ville et ce qui se passe au dehors. Dedans, dehors, la notion de fermeture est implicite et introduit *ipso facto* la notion de zone. La ville aurait donc une enveloppe, une frontière et cet objet se distingue fondamentalement de ce qui l'entoure. A ce niveau, on pose l'hypothèse d'une certaine homogénéité interne qui contraste avec la périphérie de la ville, supposée également homogène. Cependant, si nous nous arrêtions là, il est évident que le problème posé ne serait pas tant celui de la description de la ville que ce qui la différencie de la campagne. On pourrait donc énumérer, voire représenter graphiquement les différences qui opposent ville et campagne, sa population, ses comportements démographiques et sociaux, etc.

Mais, et nous en arrivons à la seconde implication, la précision de la localisation des contours de la ville aurait-elle véritablement un sens ? Ne pourrions-nous pas tout aussi bien représenter cette enveloppe par un cercle au centre d'un grand rectangle qui lui, représenterait la campagne ? En fait la réponse est plus complexe qu'il y parait car elle est en grande partie fonction de l'intérêt porté à l'objet campagne. Mais, si nous nous en tenons à la ville, la précision choisie pour le tracé

de ses limites n'a de sens que si on considère la ville comme un ensemble lui-même constitué de sous-ensembles, les quartiers, les rues, ... Comme tout objet auquel nous attribuons une représentation zonale, la ville est un ensemble hétérogène qui interdit de le considérer comme une zone ; mais *parce que c'est aussi un ensemble homogène par rapport à ce qui l'entoure* on est également autorisé à considérer la ville comme une zone.

Autrement dit, ou bien je considère la ville dans son contexte *englobant* et mon objet mérite d'être assimilé à une zone, ou bien je ne considère que la ville, son contexte m'indiffère, et je n'appréhenderai celle-ci que comme un ensemble d'objets zonaux, linéaires ou ponctuels. La question des limites de la ville ne sera pas posée mais la réponse me sera malgré tout donnée par la densité et la proximité des objets que j'aurai représentées sur la carte.

(2) Si, dans une approche comparative, mais déjà très proche de la notion de « réseau urbain », je souhaite représenter sur une carte les divers attributs socio-économiques associés aux villes d'un territoire national la question de la localisation change radicalement. La seule question qui mérite d'être posée est : où se situent les villes dans l'espace national et comment se situent-elles les unes par rapport aux autres ?

La forme de chaque ville, sa surface et ses limites ne nous intéressent guère, et, à supposer que ces attributs descriptifs nous soient d'une quelconque utilité pour cette approche comparative il serait tout à fait possible de les ajouter à la liste des descripteurs socio-économiques et de les représenter pareillement. La localisation de ces villes sera rapportée sans difficulté à une liste de points repérés par leurs coordonnées. Pourquoi sans difficulté ? Tout simplement parce que la question posée est d'une double nature géographique puisqu'elle s'interroge à la fois sur l'ensemble des villes (objets ponctuels) *dans* le territoire national, objet zonal.

(3) Supposons maintenant que cette approche soit jugée insuffisante pour décrire et représenter la multiplicité des relations qui unissent les villes entre elles et font de cet ensemble un ensemble hiérarchique, un réseau... La nature des liens unissant chaque ville à ses voisines est largement fonction de son rang (capitale, métropole régionale, villes secondaires, bourgs, ...) et de sa localisation. L'évocation de ces liens fait immédiatement référence à de nouveaux objets qui, en la circonstance, pourront être des routes, des lignes de chemin de fer, des flux d'énergie et d'information. Aux objets ponctuels que sont les villes dans l'espace national seront alors associées des lignes (arcs, segments) représentant ces liens. Points et lignes, lieux et liens seront donc les deux représentations d'un nouvel objet, le réseau. Notons que dans ce cas de figure le point est aussi un nœud. Une même localisation peut donc renvoyer à des objets différents ; un point peut être la

représentation d'une ville, mais ce peut être aussi la représentation d'un carrefour routier d'un réseau urbain.

Comme ces exemples l'ont montré, la représentation cartographique de la réalité géographique introduit d'énormes contraintes et, parfois, d'excessives réductions au regard de notre perception, intuitive ou non, de l'objet. Si bon nombre d'objets géographiques sont modélisés en utilisant la cartographie, l'avènement de l'informatique et en particulier le pouvoir de modélisation des ordinateurs permettent de remettre en question cette schématisation excessive. Faut-il employer la cartographie pour modéliser la réalité ? N'a-t-on pas maintenant les moyens techniques, grâce à l'informatique, de revenir sur une schématisation excessivement réductrice de la réalité ?

Malheureusement, les SIG n'ont pas vraiment remis en cause la modélisation de la réalité par la cartographie. Ils se sont contentés de reprendre les bases de cette schématisation pour l'améliorer, mais sans vraiment remettre en cause le passage de la *géographie* à la *géométrie,* en conservant comme base de la schématisation de la localisation absolue la représentation des objets en zones, lignes, ou points [BOU 82] [SCH 89] [ROU 91] [VOI 92]. Le SIG Savane n'échappe pas à cette règle.

La représentation multi-échelle est la seule avancée tangible [SCH 92] [WOR 95]. Elle permet d'associer plusieurs descriptions de la localisation pour un même objet, sans en changer les attributs descriptifs, et de choisir la description en fonction de la précision recherchée. Elle permet de résoudre un certain nombre de problèmes d'utilisation et de représentation cartographique, mais ne change pas fondamentalement la description de la localisation des objets en zone, ligne, ou points. Dans le SIG Savane, nous ne conservons qu'une seule représentation de la localisation pour un objet, quitte à multiplier les collections d'objet. Nous tentons plutôt d'apporter des solutions procédurales en développant de nombreuses procédures de transfert d'échelle et de changement de type d'objet, lors de l'utilisation des objets. Nous verrons ces procédures dans la troisième partie de cet ouvrage.

3.4.4. Décrire la localisation

La localisation peut être décrite de manière absolue (par exemple de façon analytique, par des coordonnées dans un espace vectoriel), de manière relative (un objet par rapport à un autre, en exprimant des relations spatiales en les objets grâce à des propriétés topologiques ou ensemblistes, comme l'adjacence, l'intersection), ou de manière indirecte (comme des adresses) [SCH 96]. Nous ne nous intéresserons ici qu'à la description absolue de la localisation. La description relative ou indirecte n'est utilisée que pour améliorer l'efficacité de la représentation, et retrouver

facilement la localisation absolue d'un objet à partir de la localisation absolue d'un autre objet.

L'utilisation d'une schématisation cartographique de la localisation absolue nous amène à représenter cette localisation par des zones, des lignes, ou des points. Nous laissons ici volontairement de coté l'information représentée des pixels, c'est-à-dire des éléments d'image : nous consacrerons un chapitre spécial à ce type d'objet (chapitre 6).

Détaillons le passage d'une description formelle à une représentation géométrique de la localisation. L'élément de base de l'espace réel est le point (au sens mathématique), de R^nxR, (n=2 ou 3 pour l'espace, plus une composante pour le temps) et chaque objet géographique devrait donc logiquement être un point de R^nxR. Cette description formelle est bien sûr impossible, car nous sommes confrontés à des objets physiques, et il est nécessaire de changer d'espace de représentation et d'introduire la notion de précision. Comme nous l'avons vu plus haut, ce passage est à la base des problèmes de validité et des difficultés de traitement de l'information localisée : la description d'un objet, si ce n'est même sa définition, va être en partie déterminée par les méthodes de représentation utilisées.

C'est en fait exactement ce que fait implicitement le géographe lorsqu'il décrit un objet à une certaine échelle. Il y a là bien sûr confusion avec l'échelle de représentation cartographique des objets étudiés, mais l'on voit bien que le choix de la description de la localisation, comme la précision de cette localisation, est lié aux choix des attributs descriptifs, et que l'objet est lui-même défini par l'ensemble de ces attributs. Comme dans tout problème informatique, le choix de l'espace de représentation et le codage de l'information influencent de manière déterminante la nature même de l'information représentée et donc l'utilisation qui en sera faite [BEK 92] [GUP 95].

Tout le problème de la représentation de l'espace est de définir un nouvel espace de représentation qui permette de réduire la description des objets à un nombre fini d'éléments et de pouvoir gérer la précision géographique de cette représentation : il faut passer d'une description mathématique (les éléments ou ensembles de l'espace mathématique) à des sous-ensembles de cet espace et décrire la localisation de ces sous-ensembles avec un nombre fini de paramètres. Les entités auront alors comme objets non plus des points, mais des ensembles de points, et les valeurs des attributs descriptifs se rapporteront a ces ensembles. Dans ces conditions, il est évident que la méthode de définition de ces sous-ensembles va influencer, sinon déterminer, la définition des attributs descriptifs.

Comme nous l'avons déjà largement souligné, le cheminement de l'information est plus complexe que le passage direct du point mathématique à la description

informatique. L'information est rarement donnée directement en terme d'espace mathématique et de données brutes, mais seulement après l'étape schématisation-extrapolation-cartographie : à partir de l'information de terrain, la donnée est interprétée, schématisée pour correspondre au modèle du monde réel, puis elle est représentée sur une carte. Cette carte, représentation géométrique, fait alors l'objet d'une représentation informatique. Mais si l'on veut raisonner en terme de globalité géographique, il est nécessaire de revenir à un espace de référence qui ne peut être que l'espace mathématique.

Deux méthodes différentes sont utilisées : la définition a priori des sous-ensembles, et la définition des sous-ensembles en fonction des objets de l'entité.

Avec la première méthode, la description de la localisation des sous-ensembles est très simple, car ils sont justement définis pour faciliter représentation et description, en utilisant le moins de paramètres possible. Par contre, on perd une partie importante de l'information descriptive, les objets n'étant plus définis par les valeurs descriptives des points mais uniquement par les paramètres de leur localisation dans l'espace. Par exemple, si l'on prend comme définition des sous-ensembles les cellules d'un maillage de l'espace, cela revient à dire que tous les points (mathématiques) d'une même cellule auront les mêmes valeurs descriptives. Cette méthode utilise principalement le découpage de l'espace en polygones réguliers : carrés, triangles, hexagones… Ces polygones sont en général repérés dans l'espace par un système de coordonnées propres à leur définition (ligne-colonne pour les carrés, etc.). L'attribution des valeurs descriptives au maillage prédéfini de l'espace se fait en général par superposition d'une grille à une carte classique, ou par saisie automatique grâce à un scanner.

Outre la facilité de description de la localisation, les avantages de cette méthode sont essentiellement dus à la facilité de traitement qui en découle : les opérations ensemblistes se font sur des cellules dont la localisation est fixée pour tous les objets. Les inconvénients sont par contre très importants : perte d'information descriptive, taille des objets fixée a priori, important volume de données, même s'il existe des méthodes permettant de réduire ce volume, par compactage ou structure arborescente.

La seconde méthode de description géométrique utilisée est la définition des objets en fonction des valeurs descriptives de l'entité. On va ainsi créer des sous-ensembles que l'on classe en zones, lignes, points [BOU 81] [DAV 91]. Les points correspondent au point mathématique : on assimile donc la localisation de l'objet à un point de l'espace mathématique. Les lignes sont des sous-ensembles de dimension 1, les zones des sous-ensembles de dimension 2, fermés et bornés. La connexité n'est pas requise dans cette définition. De même, nous ne parlons pas ici de polygone : cet élément n'intervient en fait que dans la représentation

informatique de ces éléments géométriques, alors que nous ne sommes ici que dans une représentation géométrique de la localisation.

3.4.5. Les limites du modèle géométrique

La description géométrique en zone, ligne, point est-elle suffisante pour décrire de manière satisfaisante les objets géographiques ? Oui et non. Si l'on décide de modéliser la réalité avec des cartes, oui. Mais l'on a vu combien cette modélisation peut être réductrice et combien elle simplifie la réalité. La géométrie introduit des discontinuités dans la réalité, et la précision ou l'incertitude n'est pas traitée par ce modèle de description. Dès lors, les traitements appliqués aux objets géographiques dans les SIG ne sont-ils pas trop sophistiqués par rapport à la validité de la schématisation de la réalité ?

3.5. De la géométrie à l'informatique : la représentation informatique de la localisation

La description informatique se base sur la description géométrique. Il s'agit maintenant de représenter cette description géométrique par un ordinateur, et donc par un nombre fini de descripteurs. C'est en effet ce qui caractérise cette nouvelle étape : là où en mathématique on peut nommer, en informatique il faut représenter. Nous allons rappeler rapidement comment représenter des éléments géométriques avec un nombre fini de paramètres, et quelles sont les structures informatiques qui permettent de conserver efficacement cette représentation.

3.5.1. La représentation en maille

La représentation en maille est très proche d'une représentation informatique. C'est d'ailleurs essentiellement pour cela qu'elle est employée. Comme le nombre de maille peut être élevé, le seul vrai problème de la représentation par maille consiste à trouver des structures de représentation interne permettant de réduire l'espace de stockage. Nous aborderons rapidement cet aspect un peu plus loin.

3.5.2. La représentation vectorielle

On nomme représentation vectorielle la représentation informatique des zones, lignes, et points qui conserve cette structure géométrique.

La représentation de points est immédiate : ils sont décrits tout simplement par leurs coordonnées dans un système de référence choisi (nous consacrerons

néanmoins le chapitre suivant au *simplement*…). Les coordonnées sphériques, qui correspondent à la longitude et latitude, sont souvent utilisées. La précision informatique des coordonnées est un facteur important du codage, car il peut influencer la validité des données graphiques.

Les lignes sont décrites par un ensemble de points, appelés sommets : la ligne est alors représentée graphiquement par l'ensemble des segments définis par ces sommets. Bien sûr, il y a perte d'information entre la représentation mathématique de la courbe et sa représentation informatique. Ici, le nombre de sommets en fonction du rayon de courbure détermine la précision du codage : elle dépend à la fois de la méthode de saisie (l'opération qui permet de passer de la géométrie à l'informatique), des matériels utilisés et de l'opérateur, s'il existe.

Les zones sont repérées par leur contour, qui est considéré soit comme propre à chaque zone, soit comme un ensemble de frontière. Dans le premier cas, la zone est décrite graphiquement par un ensemble de lignes fermées, appelées boucles. Si l'on conserve ces lignes telles quelles, il y a duplication d'une partie de l'information graphique, un contour étant en général commun à deux zones : globalement, ces lignes sont alors stockées deux fois. Le problème des trous et des zones incluses imposent habituellement à ce type de représentation de conserver le sens des arcs.

Dans le second cas, les contours sont considérés comme un ensemble de frontières, chaque frontière étant décrite par une ligne. Les extrémités de chaque ligne sont appelées nœuds, les lignes elles-même étant nommées arcs. Chaque nœud appartient au moins à deux arcs, et l'ensemble des arcs d'une zone doit constituer une courbe fermée. Notons qu'il est important de pouvoir reconstituer facilement le contour d'une zone à partir de ses frontières. Nous présenterons plus loin un algorithme permettant d'effectuer cette opération (A.2.2.6.).

Des entités plus complexes peuvent être définies à partir d'entités dont les objets sont des points, des lignes, ou des zones. Par exemple, on peut définir des entités dont les objets sont des couples d'éléments d'autres entités. La représentation de ces objets est donc un peu plus complexe, car elle fait appel non seulement à la description graphique des éléments, mais également à la description des relations qui définissent les objets. L'exemple le plus courant est celui d'une collection dont les objets sont des couples ordonnés de points (description de flux, migrations, réseaux linéaires, etc.). La description de la collection s'apparente alors à celle d'un graphe.

Les structures de représentation de l'information localisée ne sont donc pas simples : elles dépendent du type de l'objet, et pour chaque type, elles comprennent à la fois une *description de la localisation absolue* utilisant un nombre fini de points de l'espace et des *relations* entre ces différents ensemble de points [EGE 90].

3.5.3. Les structures de représentation de l'information localisée

Nous reviendrons en détail au chapitre 7 sur les modèles de représentation de la géométrie. En effet, ces modèles sont directement liés aux méthodes de saisie graphique, ainsi qu'aux contraintes que l'on impose pour assurer l'intégrité de la représentation. Ils sont même pour la plupart construits pour permettre les contrôles de cohérence et d'intégrité. Mais nous pouvons dès à présent décrire les principes de cette structuration.

Le choix de la structure de représentation se pose pour tout type de donnée : il doit être effectué en fonction des besoins ultérieurs et leur être adapté. Pour la partie graphique de l'information géographique, cette structure est surtout fonction de trois critères : facilité des manipulations (ensemblistes, métriques et topologiques), compacité, structuration.

Plusieurs niveaux de représentation peuvent ainsi être nécessaires suivant les traitements à effectuer. Des procédures d'identification et de reconnaissance permettent de passer d'un niveau de représentation à un autre, suivant une hiérarchie d'opérateurs de reconnaissance (de l'ensemble des points au contour, de la courbe à la zone, de la courbe à la forme caractéristique, etc.), qui permettent également de passer d'un ensemble de représentation infini à un ensemble fini.

Le premier niveau de représentation que nous avons vu au paragraphe précédent correspond aux coordonnées des points de l'espace décrivant les objets (par exemple, pour une ligne, l'ensemble des points qui permettent l'approximation des éléments graphiques à une précision donnée). A un niveau supérieur, la représentation des éléments fait appel à des relations entre ces éléments, relations qui doivent, en principe, donner l'ensemble de l'information nécessaire aux problèmes à résoudre ultérieurement [EGE 89]. On distingue ainsi :

- les relations unaires, ou formes primitives, qui donnent le type de l'élément et ses paramètres (trait et longueur ou orientation, par exemple) ;

- les relations binaires, qui peuvent être soit numériques (basées sur des indices de ressemblance ou de dissemblance, ou sur des relations métriques), soit topologiques (basées sur les notions de voisinage, de position, etc.). La représentation est souvent donnée sous forme de graphe dont les sommets sont les éléments, et les arcs entre sommets créés si la relation est vraie. Dans le cas numérique, les arêtes sont valuées par les distances entre les sommets. Si le nombre de forme primitives est faible, le nombre de relations binaires ou n-aires possibles pour décrire les éléments graphiques est très grand : tout le problème consiste alors à choisir les bonnes relations permettant une description minimum, en fonction du

but poursuivi. La description d'éléments à caractère géographique utilise de préférence des relations invariantes par transformation isométrique.

Voici par exemple un certain nombre d'opérateurs permettant d'exprimer des caractéristiques ou des relations entre objets :

- des opérateurs ensemblistes : égalité, appartenance, inclusion, intersection, union, différence, cardinal ;
- des opérateurs topologiques : fermeture, frontière, voisinage, recouvrement, compacité, connexité ;
- des opérateurs métriques : distance, angle, longueur, périmètre, surface, centroïde.

Le problème de la représentation se pose pour tous les types d'objets graphiques, mais plus spécialement pour les lignes et pour les zones. Un objet de type ligne peut en effet être constitué de plusieurs arcs ; les extrémités des arcs peuvent jouer un rôle particulier dans la description de l'objet ; la position de chaque arc par rapport aux autres également. Un objet de type zone peut être constitué de plusieurs « taches » non connexes (composantes connexes) ; chaque tache peut comporter des « trous » (genre). Ainsi, de nombreux systèmes emploient la terminologie de « polygone », et il est courant de confondre « zone » et « polygone » dans un souci de simplicité. Mais c'est alors confondre la description d'un modèle (la zone représente un objet), et la description de la représentation interne de la géométrie des objets. Et c'est une très forte limitation que d'imposer aux zones d'être des ensembles connexes de genre 1, limitation qui engendre souvent des problèmes concrets (les zones incluses disparaissent, par exemple).

3.5.4. Les méthodes de compactage

Un SIG peut contenir une grande quantité d'informations graphiques. Il est donc important que la structure de représentation choisie permette de compacter cette information pour pouvoir la stocker sur le moins de place possible, sans néanmoins que l'opération de transformation n'altère les performances du système [BOU 81]. Les méthodes de compactage sont évidemment fonction des méthodes de représentation de l'information graphique (maille ou vecteur), et doivent si possible permettre dans la forme compactée des manipulations graphiques ou des calculs métriques élémentaires. D'autre part, l'interrogation des données dans un SIG consiste essentiellement en une opération de recherche et de sélection sur la localisation : sélection d'éléments dans une fenêtre, dans le voisinage d'un objet, etc. Il est donc nécessaire d'organiser et de structurer les objets par rapport à leur localisation, de manière à ce que ces opérations soient exécutables en un temps acceptable par l'utilisateur. Ce sont ces trois critères (facilité de manipulation

graphique, compacité, structuration) qui vont guider le choix des méthodes de représentation des éléments graphiques.

Dans la représentation des objets en points, lignes, zones, nous avons vu que, si les points ne posent pas de problèmes particuliers (leurs coordonnées sont conservées directement), pour deux les autres types d'objet les éléments graphiques à prendre en compte sont des courbes, habituellement représentées par des suites de points dont on connaît les coordonnées. Les méthodes de compactage consistent à réduire l'espace nécessaire au stockage direct des coordonnées de l'ensemble de points. Dans la plupart des cas, le stockage direct sans compactage est néanmoins maintenant une solution satisfaisante. Nous ne rappellerons ces méthodes de compactage que par souci historique.

Les méthodes de compactage de représentation de courbes sont basées pour la plupart sur un codage incrémental, appelé encore codage en chaîne : les courbes sont approchées par un ensemble de points où chaque point est déduit du précédent par la donnée d'un paramètre (angle, distance, courbure, etc.). le plus connu de ce type de codage est celui de Freeman, basé sur un ensemble de huit vecteurs codés de 0 à 7 [FRE 61]. Chaque point est déduit du précédent par translation d'un de ces huit vecteurs. Le codage d'une courbe comprend donc les coordonnées du premier point puis la suite des codes de Freeman pour la meilleure approximation. Des algorithmes très usités ont été développés pour calculer ces approximations (par exemple l'algorithme de Bresenham pour les lignes droites). D'autres codages en chaîne ont également été employés : codage à quatre directions, par incrément de courbures, par vecteurs de type Freeman mais de norme variable. Ces codes induisent des grammaires qui peuvent servir à la génération et la manipulation des éléments codés.

Le principal inconvénient du codage en chaîne provient de son caractère relatif : les coordonnées d'un point doivent être calculées à partir du début de la chaîne. Ceci complique ou rend impossible a priori les manipulations graphiques sur les éléments codés (intersection, inclusion, calculs topologiques et métriques, etc.). Pour remédier à cette difficulté, on rajoute des informations, notamment sur les caractéristiques topologiques, le plus souvent sous forme de relations binaires entre primitives graphiques (la courbe est frontière de telles zones, deux zones sont connexes, etc.) [EGE 89]. Ainsi, dans le cas des courbes fermées, on peut indiquer par pointeur les relations entre les arcs, ou représenter les inclusions de zones par un graphe, ainsi que chaîner les arcs pour retrouver rapidement la structure de courbe fermée. Le système DIME a été l'un des premiers à avoir incorporé des structures topologiques [DIM 70].

Dans le cas de courbes fermées, d'autres méthodes ont été développées, consistant à ne conserver que l'intersection du contour avec un balayage horizontal : certaines opérations métriques sont rendues plus simples avec cette représentation [AMI 71] [AOK 79].

La représentation de l'espace géographique par un découpage arbitraire, qui est encore utilisée dans de nombreux systèmes (dits *raster*), produit un important volume d'information car le nombre de cellules créées est grand, dès que l'espace d'étude devient significatif. Le compactage consiste ici à essayer de regrouper les cellules de même valeur. La méthode élémentaire de compactage d'une grille de cellules consiste à balayer cette grille ligne par ligne, et pour chaque ligne de ne conserver que les séquences de cellules de même valeur. La méthode peut être améliorée en testant plusieurs orientations de balayage, car les données peuvent être agencées suivant une direction privilégiée qui produira un compactage beaucoup plus performant. La qualité du compactage dépend bien sûr de la complexité de l'information graphique, à savoir le nombre de composantes connexes. De nombreux autres algorithmes de compactage par compression ont été développés ces dernières années pour améliorer le volume de stockage des images. On peut également compacter les données *raster* en utilisant des méthodes arborescentes par subdivision successive de l'espace (*quadtree*) [ABE 84] [SAM 80], ou en essayant de représenter le contour de mailles identiques (méthodes ensemblistes) [CAP 84] [COO 67].

3.6. Les sources d'information géographique : nature et validité

3.6.1. Les moyens d'acquisition pour les données localisées

A la base de toute donnée géographique ou de toute schématisation cartographique existent des moyens d'acquisition. Rappelons très rapidement :

- le relevé de terrain ou levé topographique. Le levé topographique permet de mesurer la position d'un point sur la surface terrestre. Les mesures sont effectuées soit par triangulation, soit grâce à des systèmes de positionnement par satellite. Le relevé de terrain permet d'associer des caractéristiques à un lieu de la surface terrestre ;

- enquêtes et recensements, registres administratifs, état civil, archives. Les enquêtes et recensements sont une source de données très importante. Bien sûr, ces données ne sont pas objectives et relèvent déjà d'une modélisation. Le schéma en est d'ailleurs toujours accessible, il est constitué de l'ensemble des attributs. Il est néanmoins toujours important de connaître les contraintes et limites de validité de ce type d'information ;

- photographies aériennes et télédétection spatiale. Les images fournies par des capteurs aériens ou spatiaux sont une source importante d'information. L'interprétation initiale est en quelle que sorte due aux limites techniques du capteur employé. De nombreux traitements spécifiques permettent de dériver d'autres informations à partir des données brutes.

3.6.2. La validité de l'information

L'information géographique est extraordinairement riche et variée, tant par sa nature que par sa précision et sa validité géographique. Les sources sont innombrables et l'énergie et l'ingéniosité déployées pour dénicher l'information recherchée mérite d'être soulignée. On trouvera des géographes passionnés par leurs sujets dans tous les domaines : archives, journaux, cadastre, services techniques, mairies, syndicats, entreprises, enquêtes, recensements, photographies et images aériennes, relevés de terrain.

L'information, sa précision, sa validité sémantique, spatiale et temporelle est fonction de son émetteur, donc de sa source. Quel que soit le protocole adopté pour construire ou recueillir l'information dont on a besoin, le résultat final prend toujours la forme d'une liste d'objets décrits par leur quantité ou leur qualité. Cette liste prend diverses formes ; il peut s'agir d'enquêtes, de recensements, de notes d'archives, d'échantillons, de mesures... En fonction du but recherché, des difficultés inhérentes à toute description et d'un certain nombre de difficultés techniques, on emploiera soit des méthodes visant à l'exhaustivité, soit des méthodes par sondage.

En fait, la mesure exhaustive, parfaite et indiscutable, sera toujours un leurre. Il reste que c'est encore le devoir et la charge principale de travail de nombreuses disciplines et de nombreux services techniques que de tendre à cette « perfection ». Dans de nombreux cas, la question d'une mesure exhaustive ne se pose même pas dès lors qu'il est matériellement impossible de prétendre parvenir à un résultat satisfaisant et fiable par cette voie. La mesure du débit du Nil ou du Rhône, phénomène continu par excellence, restera toujours une *mesure approchée* par des méthodes de mesures ponctuelles. Dans la pratique, l'élaboration de l'information procède le plus souvent de mesures ou d'observations par sondage: stations météorologiques ou hydrologiques, profils pédologiques, ballons, sondes, carottes, enquêtes d'opinion, etc.

Parce que l'information est très souvent insuffisante par rapport à l'objectif qu'on se fixe, il est essentiel de pouvoir accéder à une information dont la forme et la présentation permettent d'établir les relations que nous recherchons ou dont nous supposons l'existence. Cette exigence suppose d'avoir accès à une information aussi détaillée et aussi désagrégée que possible. C'est pourquoi une information brute est toujours plus riche de sens que sa présentation agrégée, et donc, schématisée.

3.7. Structure géométrique des objets dans le système SAVANE

Nous allons présenter dans ce paragraphe comment les objets géographiques sont représentés dans le système SAVANE.

3.7.1. Le modèle géométrique

Le système SAVANE utilise le modèle géométrique pour représenter les objets localisés. On utilise donc la classification habituelle en zone, ligne, point. La localisation d'un point utilise directement les coordonnées du point. Les lignes et les zones sont décrites par des arcs. La topologie, c'est-à-dire les relations entre les arcs pour former un objet géométrique, est créée lors de la saisie des objets par le module SAVEDIT, que nous verrons en détail au chapitre 7.

Les données provenant de la télédétection (photographie aérienne scannée, télédétection spatiale) peuvent être assimilées à des zones, mais elles se présentent toujours comme des images *raster*, car la forme de ces zones est élémentaire et identique pour tous les objets. La représentation interne de ce type de données est donc différente des autres types de zones : nous lui consacrerons un chapitre spécial (chapitre 6).

Le système SAVANE gère non pas des objets isolés, mais des collections d'objets. La structure de représentation géométrique des objets est construite de manière à améliorer la gestion de ces collections, et nous la verrons en détail au chapitre 5. Néanmoins, nous pouvons déjà en donner une description rapide.

Les objets sont regroupés en collections suivant une schématisation qui correspond à la fois à la modélisation du monde réel en entités et au modèle de données imposé par le système de gestion des collections, à savoir le modèle relationnel. Quatre types de collections peuvent être définies, suivant la localisation des objets : les trois types géométriques classiques (zone, ligne, point), et le type pixel, dont la représentation interne diffère du type zone. La représentation interne de la localisation n'utilise que le premier niveau de description pour les points (leurs coordonnées). Elle relie les objets lignes avec un arc, sans utiliser de relation sur les nœuds. Nous avons en effet privilégié la simplicité de la structure : les relations entre éléments seront soit vérifiées lors de la constitution des collections (par des contraintes d'intégrité), soit recalculées par un traitement procédural. Ainsi, la représentation des zones utilise des arcs (premier niveau de description), et les relations des arcs avec les zones dont ils sont frontières (second niveau). Cette description est suffisante pour retrouver le contour d'une zone, connaître les zones incluses, les trous, les relations d'adjacence entre zones, entre arcs, etc. Mais ces propriétés ne sont pas inscrites dans la structure de représentation, comme c'est le

cas pour d'autres systèmes, qui maintiennent plus d'information (le sens des arcs par exemple). Elles peuvent toutes être retrouvées par des procédures (nous en donnerons quelques exemples dans la suite), à condition que la validité topologique de l'ensemble soit assurée : le respect des contraintes d'intégrité est essentiel pour une bonne utilisation de ce modèle de représentation.

3.7.2. La classe CArc

Nous pouvons dès à présent présenter la classe `CArc` (A.2.2.1.) qui encapsule données et méthodes propres aux arcs géométriques, ainsi qu'un certain nombre d'algorithmes graphiques utilisés pour résoudre des problèmes géométriques : intersections d'arcs, distance d'un point à un arc, intersection ou union de zones, chaînage des arcs pour passer d'une représentation interne par frontières à une représentation interne par boucles, appartenance d'un point à une zone, etc. (A.2.2.).

Les coordonnées sont lues et stockées dans un tableau (maximum 2500 points pour un arc). La structure comprend le plus petit rectangle englobant l'arc, et des méthodes pour le calculer (`CalculBBox()`). Ce rectangle permet d'améliorer considérablement la vitesse de certains algorithmes utilisant des arcs, en éliminant les arcs qui ne peuvent, de par leur position, être candidats à la propriété choisie.

Les procédures de calcul d'intersection d'arcs ou de distance d'un point à un arc (A.2.2.1, A.2.2.5) interviennent dans de nombreuses fonctions. Toutes ces procédures utilisent les segments des arcs, en écartant les segments qui ne peuvent répondre à la question. Comparer deux arcs pour trouver les intersections peut être long : pour améliorer les performances, il est utile de trier les segments de chaque arc suivant l'un des axes.

3.7.3. Les changements de représentation interne

Pour les objets de type zone, il est parfois nécessaire de changer de représentation interne, et de passer de la représentation par arcs frontière à la représentation par contour fermé. Le chaînage des arcs pour obtenir des courbes fermées est simple, d'autant plus que l'on suppose la topologie sans faille (pas de d'arc dit « pendant », c'est-à-dire sans un autre arc qui s'y rattache). Pour créer le contour d'une zone, il suffit de sélectionner les arcs de la zone, puis de chaîner les arcs les uns aux autres. Bien sûr, une attention particulière doit être apportée aux bouts multiples (plusieurs arcs se joignent sur un même bout). Une zone peut être constituée de plusieurs « taches », chaque tache pouvant avoir des « trous ». On retrouve ici la notion de polygone employée dans la représentation interne de

nombreux systèmes. Le polygone ne correspond en fait qu'à la notion de courbe fermée.

Le changement de représentation peut parfois se limiter à la création de courbes fermées, sans se préoccuper du problème de l'*intérieur* de la zone. Par exemple, le langage PostScript utilise ce modèle et permet de dessiner les zones sans problème, car il prend lui-même en charge la détermination des zones incluses à partir du contour en courbes fermées.

D'autres systèmes sont plus exigeants. Ainsi, le format Shapefile d'ARCVIEW impose une description des zones en courbes fermées avec indication de l'intérieur de la zone. Le sens des arcs est utilisé à cet effet (à gauche l'intérieur, à droite l'extérieur). Le changement de représentation doit donc reconstituer la zone en donnant un sens à chaque arc. Voici les principes de la méthode employée dans le système SAVANE :

- pour chaque zone, chaînage des arcs pour former des taches fermées (polygones) ;

- On cherche ensuite l'ordre des taches si la zone est formée de plusieurs polygones. Pour les zones formées de deux polygones, on calcule la surface de chaque tache puis, s'il y a possibilité d'intersection entre les deux taches, on teste l'appartenance d'un point de la plus petite tache dans la plus grande, pour tester s'il y a inclusion de l'une dans l'autre. Le point intérieur est trouvé en cherchant le centre du premier triangle formé de trois points du contour et inclus dans le polygone . Pour les zones formées de plus de deux polygones, le traitement est un peu plus complexe : on calcule la surface de toutes les taches, on trie ces surfaces, et, en partant de la plus petite tache, on cherche le premier incluant, ceci pour chaque tache. Enfin, on change le sens des taches en partant de la plus grande, et en descendant dans le chaînage ainsi construit.

Pour ces procédures, nous avons construit deux classes, l'une pour représenter les zones (`CZone`, A.2.2.6.), l'autre pour représenter les taches (`CTache`, A.2.2.6).

3.8. Conclusion

Ce chapitre présente les bases conceptuelles de la schématisation du monde réel utilisée dans le système SAVANE. Il est, à ce titre, fondamental, car cette schématisation détermine à la fois la structure logique et la structure interne du SIG. Elle est également déterminante sur les fonctionnalités du SIG en terme de programme d'application : au-delà de la modélisation du monde réel et de la gestion de données qui en découle, l'efficacité « conceptuelle » des fonctionnalités du SIG dépend directement de la manière dont le monde réel a été modélisé et schématisé pour être manipulé par le système d'information. Les méthodes d'exploitation qui

utilisent la localisation des objets doivent prendre en compte les limitations dues à cette schématisation.

La modélisation utilisée dans le système SAVANE (modélisation géométrique en zone, ligne, point, pixel) privilégie la simplicité du modèle sur la simplicité des algorithmes de traitements ultérieurs. Ce modèle est simple, dans sa schématisation du réel comme dans sa structure interne. Il ne peut être utilisé efficacement qu'à condition de vérifier certaines contraintes d'intégrité, auxquelles nous consacrerons le chapitre 7.

Bibliographie

[ABE 84] ABEL D.J., *A B+-Tree Structure for Quadtrees*, Computer Vision, Graphics, and Image Processing, 1984, vol. 27, n° 1, p. 19-31

[AOK 79] AOKI M., *Rectangular region coding for image data compression*, Pattern Recognition, 1979, vol. 11, p. 297-312

[AMI 71] AMIDON E.L. AND ATKIN G.S., *Algorithmic selection of the best method for compressing map data string* : Communications, ACM, 1971, vol. 14, n° 12, p. 769-774

[BEK 92] BEKKER J.H., *Semantic Data Modeling*, Prentice-Hall, 1992, New-York

[BOU 81], BOURSIER P., *Représentation compacte des cartes dans les systèmes d'information géographique*, Thèse de 3-ième cycle, Université Paris VI, 1981,

[BOU 82] BOURSIER P. ET SCHOLL M., *Performance Analysis of compaction techniques for map representation in geographic data cases*, Computer and Graphics, 1982, vol. 6, p. 73-81

[BRE 65] BRESENHAM J.E., *Algorithm for computer control of a digital plotter*, IBM System Journal, 1965, vol. 14, n° 1, p.44-50

[CAP 84] CAPSON D.W., *An improved algorithm for the sequential extraction of boundaries from a raster scan*, Computer Vision, Graphics, and Image Processing, 1984, vol. 28, n°1, p. 109-125

[COO 67] COOK B.G., *A Computer Representation of plane Region Boundaries*, Autralian Computer Journal, 1967, vol. 1, n°1, p.44-50

[DAV 91] DAVID B., *Modélisation, représentation et gestion d'information géographique, un approche en relationnel étendu*. Thèse de doctorat, Université Paris VI, 1991

[DIM 70] DIME, *Technical Description of the DIME System*, US Bureau of Census : Census use study, the DIME Geocoding System, Report n°4, 1970, Washington DC, p.25-30

[EGE 89] EGENHOFER M.J., *A formal definition of binary topological relationships*, Proceedings of the third International Conference FODO, Paris, Lecture Notes in Computer Science 367, 1989, Sprinter Verlag, Berlin, p.457-472

[EGE 90] EGENHOFER M.J. AND HERRING J.R., *A mathematical framework for the definition of topological relationships*, Proceedings of the 4th International symposium on Spatial Data Handling, 1990, Zurich p. 803-813

[FRE 61] FREEMAN H., *On the Encoding of arbitrary geometric Configuration*, Inst. Radio Engineers Trans. Elec. Computers, 1961, vol. EG 10, p. 260-268

[FRE 75] FREEMAN J., *The modelling of spatial relations*, Computer Graphics and Image Processing, 1975, vol. 4, p. 156-171

[GUP 95] GUPPTILL S.C. AND MORISSON J.L., *Elements of Spatial Data Quality*, Elsevier Science, 1995

[MAT 84] MATSUYAMA T., LE VIET H., NAGAO M., *A File Organisation for Geographic Information System based on Spatial Proximity*, Computer Vision, Graphics, and Image Processing, 1984, vol. 26, n°3, p.303-318

[MER 73] MERRILL R.D., *Representation of contours and region for efficient computer search*, Communication ACM, 1973, vol. 16, n°2, p. 69-82

[ROS 80] ROSENFELD A. AND SAMET H., *Tree structure for region representation*, Aangeenburg, Autocarto 4, 1980, vol. 1, p. 108-118

[ROU 91] ROUET P., *Les données dans les systèmes d'information Géographique.* Traité des nouvelles technologies, série Géomatique, 1991, Hermès, Paris

[SAM 80] SAMET H., *Region Representation : quadtrees from boundaries codes*, Communication ACM, 1980, vol. 23, n°3,p. 163-170

[SAM 90] SAMET H., *Applications of Spatial Data Structures*, Addison-Wesley, 1990

[SCH 89] SCHOLL M., VOISARD A., *Thematic map Modeling*, Int. Symposium on Large Spatial Databases, 1989, LNCS n°409, p. 167-192

[SCH 96] SCHOLL M., VOISARD A., PELOUX J.P., RAYNAL L., RIGAUX P., *SGBD Géographiques, Spécificités*, Thomson Publishing, 1996, Paris

[SHA 78] SHAMOS M., *Computational Geometry*, PHD, 1978, Yale University

[SHL 88] SHLAER S, MELLOR S-J., *Objet Oriented System Analysis : Modelling the World in Data*, Englewood Cliffs, 1988, Prentice-Hall

[VOI 92] VOISARD A., *Bases de données géographiques : du modèle de données à l'interface utilisateur*. Thèse de doctorat, Université d'Orsay, 1992

[WOR 95] WORBOYS M.F., *GIS, a Computer Perspective*, 1995, Taylor&Francis

Chapitre 4

Mesurer la localisation

Dans ce chapitre, nous allons décrire les principes de mesure et de représentation de la localisation d'un lieu, du globe terrestre à la carte. La précision de la mesure est essentielle car la notion de précision est omniprésente dans la définition conceptuelle d'un objet géographique.

4.1. Introduction

Pour localiser un point sur la Terre, pour mesurer un élément ou un morceau de la surface du globe terrestre et pour le représenter sur une feuille ou un écran (dans tous les cas sur une surface plane), nous sommes confrontés à deux problèmes distincts : d'une part connaître et mesurer la forme de la Terre pour pouvoir facilement localiser un point à sa surface, et d'autre part représenter cette surface curviligne sur une surface plane. La géodésie est la discipline dont l'objet est la mesure de la forme de la Terre. La géodésie utilise presque constamment la notion de verticale, et donc nécessite la connaissance de la gravité : on ne peut parler de géodésie sans introduire le géoïde gravimétrique terrestre, surface équipotentielle pour la gravité, et nous parlerons donc également de gravimétrie (discipline qui traite de la gravité terrestre). Puisque ces deux surfaces, Terre et géoïde, sont complexes, il faut définir une surface mathématique de référence dont on connaît l'équation (ce sera en l'occurrence un ellipsoïde de révolution) et se rapprochant au mieux de l'une ou de l'autre pour pouvoir décrire facilement une position sur le globe terrestre et la représenter sur une feuille, par une opération de projection cartographique. La projection cartographique elle-même va introduire des déformations, mais elles seront connues et quantifiables puisque l'on est ramené ici à un problème de calcul et non plus de mesure.

Toutes les méthodes de positionnement et de mesure en géodésie ont été profondément modifiées par l'avènement des systèmes de positionnement par satellite. Paradoxalement, avec la possibilité actuelle de mesurer facilement et avec grande précision toute position dans les trois dimensions, grâce aux systèmes de positionnement globaux de type GPS, il devient nécessaire de connaître les principes généraux de la géodésie et de la gravimétrie pour estimer la précision des mesures. Les méthodes de projection géographique n'ont pas été modifiées par cette révolution technologique, mais elles utilisent les résultats de la géodésie. De plus, les moyens de calcul offerts par les ordinateurs en banalisent l'utilisation.

Dans ce chapitre, nous aborderons donc les principes de la mesure de la forme de la Terre, de la mesure d'un lieu, puis les principes de la représentation de ces lieux sur une surface plane.

4.2. Forme et mesure de la Terre : géodésie et gravimétrie

4.2.1. Mesurer la forme de la Terre : la géodésie

On sait depuis l'antiquité que la Terre est un corps à peu près sphérique, tournant sur lui-même et tournant autour du soleil, avec à sa surface quelques rugosités à peine sensibles par rapport au rayon. Bien sûr, on ne peut se contenter d'une telle approximation. La surface réelle de la Terre comporte des vallées et des montagnes, des creux et des bosses, que l'on ne connaît pas et que l'on essaye de mesurer. Les vallées et les montagnes sont néanmoins négligeables par rapport à la taille de la Terre. Le problème majeur est donc de trouver une surface de forme géométrique simple, représentant la forme globale de la Terre, et qui permette le plus simplement possible de situer un lieu de la surface de la Terre dans deux dimensions par sa projection orthogonale sur cette surface (fig 4.1). On pourra exprimer ensuite la troisième dimension (l'altitude par rapport à cette surface) sur la normale au lieu projeté.

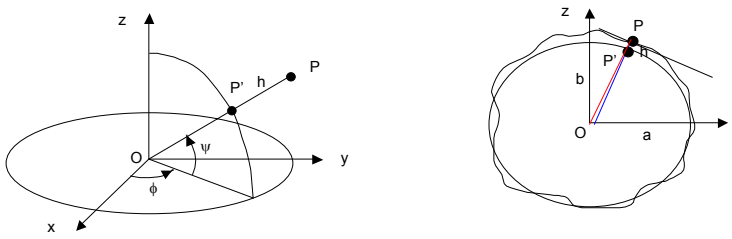


fig. 4.1 : coordonnées géodésiques, et projection d'un point sur un ellipsoïde de révolution

On chercha donc depuis très longtemps à décrire et mesurer la surface du globe terrestre, pour en trouver la forme géométrique la plus proche [LEV 88]. Les mesures sont difficiles, non seulement parce que l'on cherche à mesurer un corps sur lequel on se trouve, mais aussi parce que ce corps est entouré d'une atmosphère qui trouble les mesures optiques quand on vise les étoiles. Ces étoiles, utilisées depuis l'antiquité, sont censées être des corps immobiles : elles sont trop éloignées de la Terre pour que leurs mouvements propres soient sensibles - le rapport [mouvements propres/distance par rapport au soleil] est négligeable -, et aient une quelconque influence par rapport au mouvement de la Terre. Elles sont donc considérées comme des corps fixes et permettent de définir des repères absolus.

On se heurtait, avant l'avènement des satellites artificiels, à un problème majeur : ne connaissant pas la forme géométrique exacte de la Terre (puisque c'est justement ce que l'on cherche), on ne pouvait effectuer de mesure *absolue* liée à cette forme. Le seul moyen était donc, à partir d'un point de référence, de mesurer de proche en proche le lieu d'autres points. En théorie, la mesure de la distance entre deux points et de la direction par rapport aux étoiles fixes permet, à partir d'un point connu, de décrire n'importe quel autre point du globe par ses trois coordonnées dans un repère fixe lié aux étoiles. Mais voilà, l'atmosphère fausse les mesures optiques car la présence d'air courbe les rayons lumineux (par réfraction), et ce phénomène est quasiment impossible à mesurer car il dépend des conditions météorologiques le long du trajet suivi par la lumière, conditions qui changent le long du parcours et que l'on ne peut prévoir dans le temps.

Cette méthode directe n'étant pas utilisable, il fut nécessaire de trouver autre chose. Tout naturellement, une direction s'impose en chaque point du globe : c'est la verticale, toujours mesurable, qui peut facilement être mise en évidence par un fil à plomb ou un niveau à bulle, ces objets matérialisant la direction du champ de force lié à la gravitation universelle. Si l'on définit (ou plutôt si l'on connaît) une surface mathématique simple dont les normales se confondent avec les verticales au globe terrestre, on peut projeter chaque point du globe sur cette surface de façon biunivoque. Si l'on peut situer sur le globe terrestre une verticale, on peut alors situer le point correspondant sur cette surface : le problème est résolu pour deux des trois coordonnées. De plus, si l'on arrive à mesurer sur l'axe des verticales les différences de niveaux entre deux points du globe (en suivant la courbure de la surface de référence), on a la coordonnée manquante, et on peut donc décrire complètement la position d'un point par rapport à un autre. De proche en proche, on peut ainsi connaître la forme géométrique réelle de la surface terrestre.

Mais voilà, on ne connaît pas non plus cette surface –le géoïde gravitationnel-, qui s'est avérée complexe car dépendant de multiples facteurs, comme par exemple de la position interne des masses dans la Terre [GEO] [CAZ 94]. La verticale est la normale à cette surface de forme complexe. En étant sur un point du globe terrestre,

on n'a donc pas de moyen simple de trouver, par projection orthogonale, le point correspondant sur une surface mathématique simple servant de référence et approchant au mieux la surface réelle. On a un moyen simple de trouver sa position sur une autre surface, le géoïde, mais malheureusement on ne connaît pas cette surface a priori, puisqu'elle dépend de la valeur de la gravité, qui n'est pas constante sur le globe.

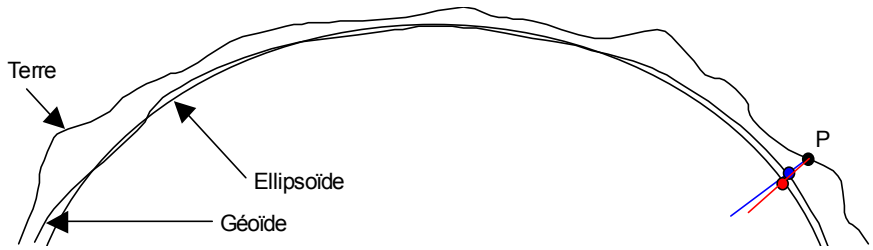
La *triangulation* est une méthode simple pour trouver, à partir de deux points, la mesure du lieu de la verticale à un troisième. Application directe de la trigonométrie, elle a été inventée par Snellius au XVIIe siècle et a servi de base à toutes les mesures géodésiques, de la mesure de l'arc de méridien par Picard et Cassini au XVIIe siècle jusqu'à l'avènement des satellites... On se sert en chaque point de la verticale réelle, alors que les mesures d'angles devraient être effectuées en utilisant la normale à la surface de référence pour projeter le point sur cette surface. Mais si les différences de direction entre ces deux verticales sont suffisamment faibles, les corrections à apporter aux mesures sont négligeables par rapport à la précision de ces mesures.

fig. 4.2 : un théodolite et son utilisation en géodésie

Supposons connue la position de deux points (un clocher et un château d'eau). On a placé une mire sur le point dont on cherche la position. On va sur le château d'eau, et à l'aide d'un théodolite (lunette tournant autour de la verticale et permettant de mesurer des angles, inventée par Galilée), on vise le clocher, puis la mire; on répète l'opération à partir du clocher, en visant le château d'eau puis la mire (fig. 4.2). On définit ainsi deux plans, l'un passant par la verticale au château d'eau et par la mire, l'autre passant par la verticale au clocher et par la mire. L'intersection de ces deux plans donne donc une droite passant par la mire. L'intersection de cette droite avec la surface de référence donne le point recherché sur cette surface. La mesure est assez simple puisqu'elle ne fait intervenir que les angles des

visées, et la position des deux points d'origine, le château d'eau et le clocher, sur la surface de référence. Connaissant la position de la mire, on peut répéter l'opération en utilisant ce nouveau point et le clocher pour déterminer la position d'un quatrième point, et de proche en proche, on définit sur la surface de référence une triangulation dont la position de chaque point est connue. La précision de la mesure de la position des deux premiers points est fondamentale puisqu'elle détermine la précision de toutes les mesures qui suivent.

Le nivellement cherche quant à lui à mesurer les différences d'altitude entre deux points, mais le problème est plus complexe qu'il n'y paraît : utilisant dans toutes les mesures la notion d'horizontale, le nivellement ne permet que de mesurer la cote d'un point au-dessus du géoïde, surface équipotentielle pour la gravité. Pour rester indépendant du relief lui-même (ce que l'on cherche à mesurer), on calcule des différences de travail à partir des différences de niveau. En effet, la mesure par nivellement met en évidence deux surfaces équipotentielles pour la gravité, et la distance (suivant la normale) entre ces deux surfaces n'est pas constante : les mesures de niveaux dépendent donc du chemin suivi. Par contre, le travail effectué par la chute d'une masse d'une surface à l'autre est le même, quel que soit l'endroit (par définition même des deux surfaces). Il faut donc connaître, en chaque point mesuré, la valeur de la gravité, appelée *cote orthométrique*, altitude du point au-dessus d'une surface équipotentielle de référence pour la gravité, le géoïde.



Pour mesurer la différence d'altitude entre deux points éloignés, on est obligé d'effectuer des mesures successives entre points plus rapprochés pour éviter les problèmes de réfraction. Entre deux points rapprochés (moins de cent mètres), on lit directement les dénivelés grâce à deux mires verticales graduées, chacune placée en un des deux points. La lecture se fait par une lunette dont l'horizontalité est assurée par un niveau placé à égale distance des deux mires. On chemine ainsi le long d'un parcours. Bien sûr, la mesure finale doit être indépendante du chemin suivi : on ramène donc la valeur du dénivelé à une différence de travail (produit du dénivelé par la valeur de la gravité) en mesurant la cote orthométrique en chaque point du cheminement.

On a donc déplacé et élargi le problème : il s'agit de trouver une surface mathématique simple approchant au mieux la forme de la Terre et dont les normales coïncident au mieux avec la direction de la pesanteur à la surface terrestre, et il

s'agit également de déterminer une surface approchant au mieux une surface équipotentielle pour la pesanteur (le géoïde terrestre). On a maintenant deux approches pour résoudre le problème de la mesure de la surface terrestre, deux approches intimement liées et complémentaires : l'approche géométrique et l'approche gravimétrique.

4.2.2. L'approche géométrique

La première approche est purement géométrique : dès le XVIIIe siècle, des mesures à la surface de la Terre ont permis de choisir l'ellipsoïde de révolution aplati aux pôles comme une surface mathématique répondant à peu près à la question de la forme de la Terre.

On a cherché depuis l'antiquité à mesurer le rayon de la Terre (la première mesure connue est due à Erathostène vers 250 avant J.C. en Égypte), mais c'est au XVIIe siècle que les avancées se précipitent. De la mesure du rayon, on est passé à la mesure de l'arc intercepté à la surface de la Terre par un angle au centre de un degré, ce qui permet bien sûr de calculer le rayon correspondant. La Terre était alors considérée comme un solide de révolution tournant autour de l'axe des pôles, et assimilée à une sphère. En 1617, Snellius, abbé hollandais, invente la triangulation et mesure un arc de méridien. Son résultat, 107 km par degré, est mauvais. En 1669, un autre abbé, Picard, réalise un théodolite (grâce à une lunette de Galilée) et obtient une valeur excellente pour la mesure du degré qu'il calcule par triangulation près de Paris (111,212 km, soit une erreur de 1 pour mille). Mais en 1672, on constate qu'une horloge bien réglée à Paris retarde de 2 mn 30 par jour à Cayenne. Huygens et surtout Newton, qui avait déjà élaboré sa loi de gravitation mais qui ne l'avait pas encore publiée, attribuent ce retard à une diminution de l'attraction terrestre, et donc à une augmentation de la force centrifuge : la Terre ne doit pas être une sphère mais un ellipsoïde de révolution aplati aux pôles. Les astronomes Cassini (père puis fils) avaient été chargés par la France de réaliser une carte précise, et ils obtinrent des résultats contraires aux intuitions de Newton. Louis XV ordonna donc deux mesures de l'arc de méridien : l'une près du pôle (expédition de Clairaut et Maupertuis en Laponie), l'autre près de l'équateur (expédition de La Condamine et Bouguer en Équateur, alors le Pérou). Les résultats confirmèrent bien sûr la thèse de Newton. A partir de cette date, de nombreuses mesures ont été effectuées et de nombreux ellipsoïdes de révolution ont été proposés pour approcher au mieux la forme de la Terre. Le dernier date d'une dizaine d'années.

L'ellipsoïde de révolution est une surface mathématique simple. Elle est caractérisée dans un repère cartésien par la position de son centre, par son grand coté (habituellement désigné par a), et par son petit coté (habituellement désigné par b). Souvent, un ellipsoïde est caractérisé par son grand coté et par son aplatissement ($f=(a-b)/a$) ou par le carré de son excentricité ($e^2=(a^2-b^2)/a^2$).

Tout point d'un ellipsoïde de révolution peut être caractérisé par deux des coordonnées sphériques (la longitude et la colatitude), la troisième coordonnée étant donnée implicitement par l'appartenance du point à l'ellipsoïde.

Profitons de ce paragraphe pour définir avec exactitude la terminologie employée pour positionner un point dans l'espace.

Le repère naturel est un repère cartésien, de centre O et d'axes orthonormés i,j,k. Tout point de l'espace peut être défini par ses coordonnées cartésiennes dans ce repère, habituellement nommées x,y,z. Le point peut également être défini par ses coordonnées sphériques : λ (angle iOP', où P' représente la projection orthogonale du point P dans le plan (i,j)), ψ (angle kOP), R (distance OP).

Les coordonnées géodésiques, ou géographiques, ne sont autres que les coordonnées sphériques, en supposant que l'ellipsoïde de révolution qui représente la Terre soit centré sur l'origine de ce repère. Le rayon est omis, puisque le point se trouve par définition sur l'ellipsoïde : on peut donc le calculer en connaissant la forme de l'ellipsoïde. La longitude correspond à λ, la colatitude à ψ.

Le nord géométrique correspond au point de coordonnées (0,0) sur l'ellipsoïde. Le nord magnétique correspond au champ magnétique et non au champ gravitationnel. Le nord indiqué par une boussole n'est donc pas le nord géographique. La déviation est fonction du champ magnétique, elle varie en fonction du temps et du lieu.

En géodésie classique, on opère en fait de façon inverse : la forme de l'ellipsoïde de référence est déterminée par la mesure géodésique, la position du centre de l'ellipsoïde par rapport au géoïde est déterminée par une condition de tangence en un point (le point fondamental). Une fois l'ellipsoïde ainsi placé, on définit un repère cartésien : l'origine du repère est le centre de l'ellipsoïde, l'axe des z passe par les pôles de l'ellipsoïde, l'axe des x est placé de façon à fixer la longitude d'un point donné (par exemple, 0 pour Greenwich ou Paris).

De nombreux ellipsoïdes de révolution ont été définis pour approcher au mieux la forme de la Terre à une latitude donnée. Ainsi, chaque service géodésique a défini une forme d'ellipsoïde se rapprochant au mieux de la courbure de la Terre à la latitude de son pays, pour minimiser les déformations entre géoïde et ellipsoïde et pouvoir utiliser la verticale sans induire trop d'erreur.

Une fois la forme de l'ellipsoïde déterminée, encore faut-il positionner cette surface par rapport à la Terre. Quel que soit l'ellipsoïde de révolution considéré, des différences existent entre la normale au géoïde (c'est-à-dire la verticale au globe terrestre) et la normale à l'ellipsoïde, sauf peut-être en un point. Comme toute

triangulation part d'un point fondamental dont la position est calculée avec beaucoup de précision, chaque service géodésique a choisi de positionner l'ellipsoïde en le rapprochant au mieux de la verticale corrigée au point fondamental (verticale observée corrigée de l'influence des mouvements de la Terre et de l'attraction des autres astres) en imposant donc qu'en ce point, ellipsoïde et géoïde soient tangents au même plan. La position de l'ellipsoïde par rapport à la Terre est déterminé par cette condition. L'ensemble des paramètres - forme et position de l'ellipsoïde - est appelé *datum*. Toute mesure d'un point sur la Terre (donnée par exemple en longitude-latitude) dépend donc du *datum* auquel elle se rapporte. Nous verrons plus loin comment passer d'un *datum* à un autre.

De nombreux ellipsoïdes ont été ainsi définis. Par exemple, la triangulation qui couvre la France entière a été rapportée à l'ellipsoïde de Clarke 1880 (a=6 378 249 m, $(a-b)/a$ =1/293,5) avec comme point fondamental la croix du Panthéon à Paris. L'Association Internationale de Géodésie a défini en 1909 puis en 1924 un ellipsoïde particulier pour toutes les études internationales, appelé pour cela « ellipsoïde international » (Hayford, 1909 ou 1924 : a=6 378 388, $(a-b)/a$ =1/297).

Un ellipsoïde de révolution est caractérisé par son grand coté -a- et son petit coté -b-. En géodésie, on considère plutôt le grand coté et l'aplatissement (a-b/a) ou le carré de l'excentricité ($e^2=(a^2-b^2)/a^2$). Nous verrons plus loin comment les problèmes liés à la « bonne » projection d'un ellipsoïde sur un plan ont influencé le développement des mathématiques dans beaucoup de ses domaines modernes (calcul différentiel, géométrie analytique).

Voici les ellipsoïdes les plus couramment utilisés [MAG 97] :

Nom	***a (mètres)***	***Aplatissement (a-b)/a***
Airy 1830	6377563.396	1/299.3249646
Australian National	6378160	1/298.25
Bessel 1841 (Ethiopie, Indonesie, Japon, Corée)	6377397.155	1/299.1528128
Bessel 1841 (Namibie)	6377483.865	1/299.1528128
Clarke 1866	6378206.4	1/294.9786982
Clarke 1880	6378249.145	1/293.465
Everest Brunei, Malaisie orientale (Sabah, Sarawak)	6377298.556	1/300.8017
Everest India 1830	6377276.345	1/300.8017
Everest India 1956	6377301.243	1/300.8017
Everest Malaisie occidentale, Singapour	6377304.063	1/300.8017

Everest, Malaisie orientale 1969	6377295.664	1/300.8017
Geodetic Reference System 1980	6378137	1/298.257222101
Helmert 1906	6378200	1/298.3
Hough 1960	6378270	1/297
International 1924	6378388	1/297
Krassowsky 1940	6378245	1/298.3
Modified Airy	6377340.189	1/299.3249646
Modified Ficher 1960	6378155	1/298.3
South American 1969	6378160	1/298.25
WGS 72	6378135	1/298.26
WGS 84	6378137	1/298.257223563

Les ellipsoïdes de révolution utilisés en pratique n'ont malheureusement pas le même centre, puisque leur position est définie par un point de tangence entre le géoïde et l'ellipsoïde (le point fondamental). Si l'on ne connaît pas la position relative d'un point de tangence par rapport à un autre, on ne pourra passer d'un système géodésique à un autre. Pour comparer la localisation de deux points, il faut que cette localisation ait été mesurée dans le même *datum*. Des coordonnées d'un point de la Terre exprimées en longitude et latitude ne sont donc pas absolues : elles dépendent du *datum* utilisé. L'influence du *datum* peut atteindre plusieurs secondes d'arc, soit plusieurs centaines de mètres après projection dans un plan. La nécessité d'avoir un système global est rapidement apparue pour les applications intercontinentales, comme la balistique civile ou militaire. Plusieurs *datum* globaux ont ainsi été définis (WGS 60, WGS 66, WGS 72, WGS 84), mais les positions ne peuvent être déterminées dans ces *datum* que par calcul (changement de *datum* à partir d'une position donnée dans un *datum* local) ou positionnement global par GPS. Nous verrons plus loin méthodes et programmes pour changer de *datum*. Notons également que l'on parle de *datum* vertical car, si la condition de tangence est imposée, l'altitude du point de tangence n'est pas fixée, et l'origine sur l'axe des *z* est choisie par chaque service cartographique. Ainsi, on a souvent une différence dans le calcul de l'altitude entre cartographie terrestre et cartographie marine.

Avant l'apparition des satellites artificiels et des méthodes de positionnement global, il était difficile de relier deux triangulations ne partant pas du même point fondamental si ces deux triangulations ne pouvaient se rejoindre. C'est bien sûr le cas des îles et des continents, puisque l'on ne peut trianguler sur la mer qu'avec des techniques particulières. Pour cette raison, la cartographie des îles a toujours été difficile à relier à la cartographie des continents, car chaque île éloignée avait besoin d'un *datum* propre. Les problèmes de ce type ont été résolus grâce aux satellites artificiels et à la gravimétrie qui permet d'avoir une approche globale.

Chaque point d'un ellipsoïde de révolution (et donc le point de la surface terrestre auquel il correspond) peut être repéré par rapport au centre de l'ellipsoïde par des coordonnées sphériques : la longitude et la latitude. L'origine des longitudes est déterminée de façon arbitraire : on prend le plus souvent le méridien passant par la projection sur l'ellipsoïde de la ville de Greenwich (Angleterre), quelquefois celui passant par celle de Paris (Observatoire de MontSouris). L'origine des colatitudes est le pôle nord. Le plus souvent, on parle de latitude *nord* ou *sud*, en prenant comme origine l'équateur (ligne de colatitude 90°), et de longitude *est* ou *ouest*, suivant le sens dans lequel on tourne, en mettant le nord en haut. Toutes ces conventions nous viennent bien sûr de l'antiquité. L'unité est le degré ($2\pi/360$) ou le grade ($2\pi/400$). La différence entre le méridien de Greenwich et celui de Paris est de 2° 20' 14,02500''.

4.2.3. L'approche géophysique

L'autre approche, pour déterminer la forme de l'ellipsoïde, est géophysique. Elle cherche à déterminer la forme du géoïde et à en déduire directement la forme de l'ellipsoïde. Elle fait donc appel à l'étude du champ de pesanteur à la surface du globe. On dispose donc ici d'un outillage mathématique beaucoup plus important qui laisse espérer une résolution théorique globale du problème : déterminer l'équation complète de la surface équipotentielle recherchée, le géoïde. Mais comme on le verra, subsiste une certaine ambiguïté : la mesure de l'altitude au-dessus de l'ellipsoïde nécessite la connaissance du géoïde, et la détermination exacte du géoïde ne peut se faire qu'avec la connaissance du relief. Mais heureusement, les deux disciplines se complètent car les échelles de précision requises ne sont pas du même ordre dans chaque domaine, et les avancées de l'une servent à améliorer les mesures de l'autre. Le développement des techniques (notamment spatiales) a considérablement enrichi les possibilités de mesures, en arrachant enfin le géodésien de la surface qu'il doit mesurer.

4.2.3.1. Gravité et géoïde.

On ne peut donc établir de mesure de la forme de la surface terrestre sans connaître la gravité. Grâce à la loi d'attraction universelle, on sait que la force de pesanteur dérive d'une fonction de force, et donc qu'il existe une famille de surfaces (toutes homothétiques par rapport au centre de gravité des masses) qui sont perpendiculaires en tous leurs points à la direction de la pesanteur. Ces surfaces sont appelées équipotentielles (La valeur de la gravité n'est pas la même en tout point d'une surface équipotentielle : ce n'est vrai que pour le potentiel dont elle dérive).

On montre également que la surface d'un fluide homogène en équilibre coïncide avec une équipotentielle de pesanteur. Le niveau moyen des mers et océans suit donc une telle équipotentielle : cette surface particulière est appelée géoïde [CAZ

94] [BAL 98]. En dehors des mers et océans, elle passe sous la surface terrestre. D'autre part, on montre facilement que la forme de la surface d'un fluide homogène en rotation uniforme autour d'un axe et soumis à l'attraction universelle est, en équilibre, un ellipsoïde aplati aux pôles. On retrouve la forme approchée de la Terre connue depuis le XVIIIe siècle.

Il aurait été bien agréable de ramener toutes les mesures géodésiques et gravimétriques au géoïde, si celui-ci avait présenté une forme simple. Mais cette forme n'est pas connue et n'a aucune raison d'être une forme simple puisque la Terre et la répartition des masses qui la composent sont complexes. On ramène donc les mesures à un ellipsoïde de révolution qui sera pris comme surface théorique de référence pour la gravité.

Pour un géodésien, le géoïde correspond à la déviation de la verticale, c'est-à-dire la différence entre la verticale observée à la surface et la verticale à l'ellipsoïde de référence. On a donc mesuré cette déviation sur un certain nombre de points (en s'appuyant sur les astres). En interpolant de façon linéaire dans les triangles obtenus en reliant tous ces points, on obtient une surface appelée géoïde astrogéodésique, qui permet d'évaluer la cote du géoïde par rapport à l'ellipsoïde de référence.

On démontre que la forme d'une surface équipotentielle, à l'intérieur de laquelle se trouvent toutes les masses, est déterminée si on connaît en tous ses points l'intensité de la pesanteur. D'où une seconde méthode, géophysique et non géométrique, pour la détermination du géoïde : la prospection gravimétrique, consistant à mesurer la pesanteur en un réseau dense de points de la surface. On calcule en fait les anomalies de la gravité, différence entre la gravité mesurée puis corrigée et la gravité théorique sur l'ellipsoïde de référence. A partir du géoïde, surface équipotentielle, on pourra théoriquement déduire le potentiel de gravité dans tout l'espace extérieur. La prospection sur la mer est difficile, mais le lancement de satellites capables de mesurer directement par radar l'altitude relative sur la mer avec une précision du centimètre (SEASAT, Poseïdon, ERS1) a résolu le problème.

On calcule facilement la gravité théorique à la surface d'un ellipsoïde en rotation à partir de ses caractéristiques géométriques :

$$g=ge(1+\gamma sin^2\theta)$$

Quelle que soit la densité, on a toujours $applatissement+\gamma=5/2(w^2a/ge)$, w étant la rotation de l'ellipsoïde. L'association Internationale de Géodésie a donc fixé la valeur de la gravité à la surface de l'ellipsoïde de Hayford :

$$g=978.049(1 + 0.005\ 288\ 4sin\theta - 0.0000059sin^2\theta).$$

Pour trouver la forme du géoïde, on peut utiliser la formule de Stockes si l'on connaît la différence entre la gravité théorique sur l'ellipsoïde et la gravité sur le géoïde, à condition de supposer que le géoïde contient toutes les masses. Or on part d'une mesure effectuée au sol, au-dessus du géoïde. On doit donc, pour estimer la valeur de la gravité sur le géoïde, effectuer plusieurs corrections :

- une correction d'altitude, puisque le point mesuré est plus haut que le géoïde. On obtient une diminution de 30.8 mgals pour 100 m d'élévation (la correction se calcule bien sûr par rapport à la pesanteur théorique).

- une correction de plateau, consistant à corriger l'existence de matière extérieure au géoïde alors qu'on l'a supposé contenir toutes les masses : on comble l'espace entre le point de mesure et le géoïde par de la matière et on en calcule l'influence gravitationnelle.

- une correction topographique, pour tenir compte du modelé du relief au voisinage du point de mesure.

La somme de ces trois corrections est appelée correction de Bouguer. Les anomalies de Bouguer sont données par la différence entre gravité mesurée et gravité théorique moins la correction de Bouguer. La forme systématique de ces anomalies par rapport au relief a donné lieu à l'introduction d'une force de compensation dépendant vraisemblablement de la densité de la croûte terrestre et de sa réaction séculaire et élastique à la surcharge, appelée isostasie.

La troisième méthode est plus générale, et consiste, à l'inverse, à rechercher directement l'expression mathématique du potentiel dans tout l'espace et d'en déduire la forme du géoïde. La fonction de force définissant la gravité est définie par l'équation différentielle indiquant l'annulation du Laplacien en dehors des masses : la forme des solutions de cette équation est connue et peut être exprimée en série de fonctions sphériques, mais les coefficients sont à déterminer par l'expérience. Pour cela, on suit le mouvement des satellites artificiels par laser, à partir d'un certain nombre de stations au sol. On approche la valeur des coefficients de chaque terme de la série grâce aux caractéristiques de ces trajectoires, corrigées de l'influence des autres astres et de l'atmosphère. Les résultats obtenus par gravimétrie sont également utilisés pour affiner le calcul des coefficients dont les effets sont très faibles.

Les solutions de l'équation différentielle indiquant l'annulation du Laplacien sont constituées des fonctions harmoniques, qui elles-mêmes peuvent être développées en série de fonctions sphériques, c'est-à-dire fonctions des puissances de *1/r*. A la distance *r* du centre de gravité des masses, dans la direction τ,ϕ, le potentiel de pesanteur est donné par une expression de la forme :

$$v=GM/r(1+J_1P_1(\tau,\phi)(a/r)^2 + J_2P_2(\tau,\phi)(a/r)^3 + ...)$$

où a représente le rayon équatorial de la Terre, M la masse de la Terre, G la constante de gravitation universelle, $P_n(\tau,\phi)$ des fonctions de la direction (τ,ϕ) du point d'observation par rapport au centre de gravité. τ représente l'orientation, ϕ l'inclinaison.

Ce développement a une interprétation physique immédiate : l'influence de chaque terme apparaît au fur et à mesure que l'on se rapproche de la Terre. En négligeant tous les coefficients, le potentiel est celui d'une sphère, c'est ce qui se passe si r est grand. En ne négligeant plus J_1, on obtient l'équation d'un ellipsoïde aplati : le terme est proportionnel à $(1/r^3)(3\cos 2\tau - 1)/2$, J_1 est de l'ordre de 1/1000, puisque l'aplatissement de la Terre est de l'ordre de 1/300. Grâce aux autres termes (qui sont de l'ordre de 1/1000000) on obtient des écarts locaux avec l'ellipsoïde pouvant aller jusqu'à 100 mètres.

En 1982, la NASA a calculé un modèle de géoïde en déterminant 20 termes, exclusivement basé sur des données de poursuite de satellites. La précision est de l'ordre du centimètre pour les coefficients d'ordre deux, et de l'ordre du mètre à l'ordre 20. Depuis, d'autres modèles ont utilisé plus de trente termes, en intégrant des données d'altimétrie sur les océans et de gravimétrie au sol.

La connaissance de la valeur de la pesanteur en tout point de l'espace est d'une importance primordiale en science spatiale et en balistique : elle permet, entre autre, de prévoir la trajectoire exacte d'un satellite artificiel ou d'un missile (fig. 4.3).

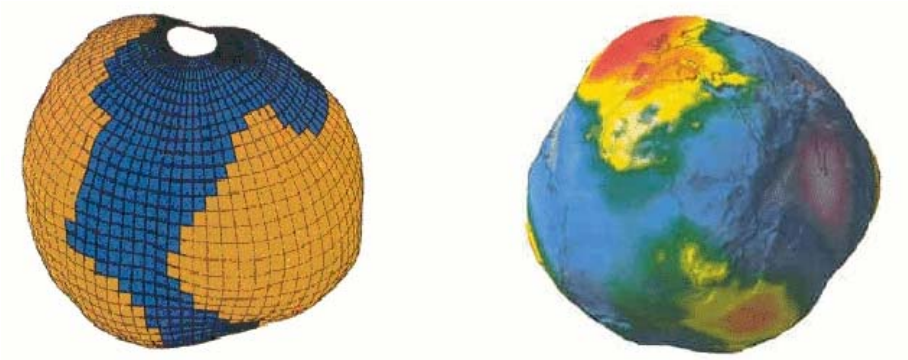


fig. 4.3 : déformation du géoïde terrestre, à l'échelle 10000 (d'après [CAZ 94])

4.2.3.2. De nouveaux ellipsoïdes

L'ellipsoïde de référence, depuis toujours déterminé par des considérations géométriques, peut maintenant être défini par une simplification de l'expression du

potentiel, en se limitant au terme en $(1/r)^3$ dans l'équation fondamentale du géoïde (on obtient des valeurs proches de 6 378 160 m, avec un aplatissement de 1/298,25).

Ainsi ont été définis successivement les systèmes géodésiques mondiaux WGS 60, WGS 66, WGS 72, WGS 84. Les modifications successives correspondent aux avancées dans la recherche de la forme du géoïde gravitationnel, notamment par les mesures de poursuite de satellites artificiels. La position de l'ellipsoïde est en général donnée en fixant le centre de l'ellipsoïde au centre de gravité des masses de la Terre (indépendamment d'une condition de tangence en un point).

4.2.4. Nouveaux instruments, nouvelles mesures, nouvelles précisions

Si nous avons présenté triangulation, nivellement et autre méthodes de mesure, c'est que la plupart des travaux en cartographie et représentation de la Terre sont encore attachés à ces techniques, malgré l'émergence d'instruments nouveaux qui ont révolutionné la géodésie et la gravimétrie.

Dans le domaine de mesures de distances, on est passé du fil étalonné au telluromètre (mesure du temps d'aller et retour d'une onde électromagnétique), puis au géodimètre à laser (même chose mais avec une onde lumineuse cohérente). Les mesures d'angles, à la base de la triangulation, peuvent être remplacées par des mesures de distances. Les angles étaient mesurés au théodolite avec une précision relative de 1/200000, celle des distances est maintenant supérieure à 1/500000.

Dans le domaine du positionnement, la géodésie spatiale permet maintenant de connaître le lieu d'un point indépendamment de la notion de verticale. Les mesures prennent appui sur un certain nombre de stations au sol dont la position est connue de façon très précise par des mesures sur des signaux intergalactiques. Le principe de base est simple : à partir de deux points connus, on peut déterminer la position exacte du satellite à l'instant t. Si au même instant t, on repère à partir d'un point inconnu la position du satellite par rapport aux étoiles, on peut en déduire la position de ce point. Actuellement, on calcule directement la position du point inconnu par rapport aux satellites.

4.2.4.1. Le système GPS

Le système GPS fonctionne sur ce principe, mais avec un réseau de plusieurs satellites en ne prenant en compte que la distance et non la direction. Chaque satellite émet en continu des signaux qui permettent à un point récepteur d'évaluer la distance qui le sépare du satellite, en mesurant le temps de réception du signal (pour cela, chaque satellite est équipé d'une horloge atomique très précise) [BOT 98]. La position de chaque satellite est repérée grâce à un réseau de stations au sol; cette position est également transmise au point récepteur via le satellite, qui émet de

plus des informations sur sa trajectoire et sur les conditions atmosphériques de transport des signaux. La réception de la position de trois satellites permet à un point récepteur de calculer sa position sur un ellipsoïde. Avec quatre, on peut également avoir l'altitude. La précision dépend principalement de la géométrie du polyèdre formé par les satellites et le point récepteur, de la position des arêtes par rapport à l'atmosphère et l'ionosphère (les erreurs sont d'autant plus fortes que l'élévation du satellite par rapport à l'horizon est faible), de la qualité de la synchronisation des horloges, de la densité des signaux réfléchis, du type de récepteur employé. Pour les coordonnées horizontales, la précision, en mesure absolue, varie de quelques dizaines de mètres à un mètre; en mesure relative (on mesure alors la phase des signaux), elle peut être supérieure au centimètre. La mesure de l'altitude est bien sûr plus délicate, et d'une moindre précision.

Le réseau comprend, en 2000, 28 satellites, dont 6 de réserve, ce qui permet d'atteindre chaque point du globe à tout moment. La connexion des satellites russes au réseau américain permet d'obtenir des configurations géométriques encore meilleures, donc une amélioration du temps et de la précision des mesures.

La plus grande précision est obtenue par les mesures différentielles : plutôt que de mesurer une position absolue, on mesure la différence entre un point connu et le point à déterminer. La simultanéité des mesures permet d'éliminer la plupart des sources d'imprécision, et d'atteindre ainsi en relatif une précision inférieure au centimètre.

Lors d'une mesure par GPS, la position du point est donnée par rapport à une surface de référence, géoïde ou ellipsoïde. Lorsque l'on utilise un ellipsoïde, le *datum* est important, puisque le GPS peut donner la mesure suivant plusieurs *datum*. La connaissance de toutes ces notions est donc fondamentale pour une bonne utilisation de ces instruments, surtout lorsque les mesures effectuées doivent être intégrées dans un système d'information.

Le système GPS (Global Positioning System) est un système de navigation et de positionnement mondial utilisant une constellation de 28 satellites. Son étude, son financement et son entretien sont entièrement assurés par le Département de la Défense des Etats-Unis, qui se réserve le droit de dégrader le signal pour des raisons stratégiques et militaires. Les satellites GPS évoluent sur des orbites circulaires à une distance de l'ordre de 20200 km de la Terre, ce qui correspond à une période de rotation de l'ordre de 12 heures. Les récepteurs GPS permettent de capter et traiter les signaux envoyés par les satellites. La disponibilité du système dépend du nombre de satellites observables durant les mesures et de leur positionnement géométrique qui influe sur la qualité des résultats.

Le mode absolu

Dans un mode d'utilisation où le récepteur GPS fournit des coordonnées de manière instantanée et de façon autonome (mode absolu), le GPS permet de fournir une position à quelques mètres près. Cette précision assez modeste est la conséquence de plusieurs facteurs :

- erreurs du segment spatial : erreur sur les paramètres orbitaux des satellites (20 m), erreur sur l'horloge propre du satellite par rapport au temps GPS (quelques mètres).
- erreur de propagation : le signal GPS se propage depuis le satellite vers l'antenne du récepteur. Il traverse ainsi la totalité de l'atmosphère terrestre et subit les influences des différentes couches. L'ionosphère retarde le signal en fonction de l'activité solaire et de la situation géographique (erreur entre 0 et 50 m). La troposphère influence la propagation du signal par l'intermédiaire de phénomènes de réfraction. Cette erreur est très sensible aux basses élévations du satellite (erreur entre 2 et 5 m).
- erreur de multi-trajets : le signal GPS peut subir, à l'approche de toute surface proche de l'antenne de réception, une réflexion qui rallonge le chemin optique parcouru. Cet effet est connu sous le terme de multi-trajets. On peut l'éliminer en éloignant l'antenne de toute surface métallique proche et en l'équipant d'un plan de masse absorbant les signaux réfléchis par le sol.
- erreur due à la précision de l'horloge du récepteur (l'horloge du récepteur sert à mesurer des différences de temps). Cette erreur est plus importante que celle induite par l'horloge du satellite car la qualité de l'horloge détermine directement le coût du récepteur (environ 20 m).

Le mode différentiel

Comme la précision de la localisation GPS absolue est parfois insuffisante, il est possible de contourner le problème et d'effectuer la localisation relative d'un point par rapport à une référence connue : deux récepteurs GPS vont faire des mesures simultanées sur les mêmes satellites, pour permettre de déterminer les différences de coordonnées entre les deux stations. On parle alors de GPS différentiel. Le calcul des coordonnées se fait en post-traitement après déchargement sur un ordinateur des mesures enregistrées par les deux récepteurs. Il s'agit d'un positionnement relatif par rapport à une station de référence placée sur un point connu. On doit donc disposer de deux récepteurs qui effectuent des mesures simultanées, l'un sur le point à déterminer, l'autre sur la station de référence. Le principe du différentiel consiste à retirer les erreurs systématiques corrélées entre la station de référence et la station mobile. Le bilan d'erreur va être considérablement réduit :

- l'erreur d'horloge du satellite va s'annuler lors du traitement des observations simultanées.
- les erreurs d'orbite vont être résiduelles ou négligeables,
- l'erreur de propagation ionosphérique sera considérablement réduite si les deux récepteurs ne sont pas éloignés de plus de 20 km environ,
- l'erreur de propagation troposphérique sera également diminuée, surtout s'il n'existe pas une trop grande différence entre les conditions météo à chaque récepteur (attention aux grandes différences d'altitude),
- en revanche, les erreurs multi-trajets s'ajoutent les unes aux autres,
- l'erreur due à l'horloge du récepteur sera du même ordre que pour le positionnement absolu, mais peut être réduite par les méthodes de double différence.

La précision du positionnement relatif varie entre 0.1 et 5 m environ, et dépend essentiellement de la qualité des récepteurs et du nombre de mesures effectuées sur le point à

mesurer. Dans le cadre d'un GPS différentiel post-traité, la station de référence sera équipée d'une mémoire lui permettant d'enregistrer les mesures réalisées. Les mesures seront récupérées ultérieurement par l'utilisateur sur un ordinateur par connexion directe ou par modem. La station mobile enregistre également les mesures propres à chaque point. Le logiciel de post-traitement permet de traiter les mesures et de calculer les positions des points avec la précision requise. Plus le nombre de mesures augmente meilleure est la précision. Il faut au minimum une centaine de mesures pour permettre un calcul différentiel sur un point fixe.

4.2.4.2 Les changements de datum

Avec les techniques géométriques, toute nouvelle mesure s'appuyait sur un point connu de la triangulation, et utilisait donc le *datum* local. Avec l'avènement du GPS, des mesures peuvent être effectuées indépendamment du *datum* normalement utilisé localement. Lorsque toutes ces mesures doivent être intégrées dans un même système, il devient nécessaire de savoir comment transformer les coordonnées sphériques (longitude, latitude) d'un *datum* à un autre. Les coordonnées dans des *datum* différents d'un même point physique peuvent donner des différences de plusieurs dizaines de mètre après projection.

Les ellipsoïdes définis au cours du temps pour approcher la Terre et positionner un point diffèrent par leur forme ou leur centre, ou les deux. Pour passer des coordonnées géographiques d'un point par rapport à un *datum* aux coordonnées du même point par rapport à un autre *datum*, il faut donc connaître les formes des deux ellipsoïdes de révolution et les écarts de position entre les deux centres. C'est encore grâce à l'étude du mouvement de satellites artificiels que l'on a pu mesurer les écarts de position du centre de la plupart des *datum* avec le centre des masses de la Terre, qui est par définition le centre du *datum* WGS84. Ces écarts de position ont une précision qui varie selon les *datum*, de quelques mètres à quelques dizaine de mètres. Beaucoup de *datum* utilisent l'ellipsoïde international défini en 1924.

Exemples de *datum* et des valeurs DX,DY,DZ par rapport à WGS84 :

PROVISIONAL SOUTH CHILEAN 1963 (Southern Chile):
Ellipsoïde : International 1924;
DeltaX=16.; DeltaY=196.; DeltaZ=93.;
PUERTO RICO:
Ellipsoïde : Clarke 1866;
DeltaX=11.; DeltaY=72.; DeltaZ=-101.;
QATAR NATIONAL:
Ellipsoïde : International 1924;
DeltaX=-128.; DeltaY=-283.; DeltaZ=22.;
SCHWARZECK (Namibie) :
Ellipsoïde : Bessel 1841 Japan;
DeltaX=616.; DeltaY=97.; DeltaZ=-251.;

Deux principales méthodes permettent d'effectuer cette transformation. La plus connue utilise la formule de Molodensky. Cette formule correspond à un calcul approché, si bien que la transformation a une précision limitée (environ 3 mètres), et qu'elle n'est pas utilisable près des pôles. L 'implémentation en C++, utilisée dans le module SAVEDIT (voir chapitre 7), est donnée dans la classe `CMolodensky` (A.2.3.3.). La seconde méthode est basée sur la recherche des coefficients par suivi de la trajectoire de satellites.

4.2.5. Précision des mesures

Il est essentiel de parler de la précision des techniques et des mesures que nous avons mentionnées, d'autant plus que la représentation cartographique va induire d'autres déformations. On a souvent tendance à perdre de vue le domaine de validité d'une mesure, à mélanger des échelles de précision diverses et souvent incompatibles. Par exemple, un changement d'ellipsoïde de référence induit des différences de latitude largement supérieures aux erreurs d'observation (sur une distance de mille kilomètres, du nord au sud de la France, la différence représente un écart de l'ordre de quatre décimilligrades). Bien sûr, cela concernait plus le géomètre que le géographe, mais le développement du stockage informatique et des systèmes d'information géographique rend cette question importante, car le système facilite l'accessibilité de mêmes données à plusieurs groupes d'utilisateur qui n'ont pas forcément les mêmes besoins de précision, ou qui ne savent plus évaluer, ni ce dont ils ont ou auront besoin, ni ce qu'ils utilisent vraiment.

4.3. Représenter l'ellipsoïde sur une surface plane : cartographie et projections

On sait maintenant ramener la position d'un point dans l'espace géographique (dans les trois dimensions) à sa position par rapport à un ellipsoïde de révolution. Pour représenter un bout de cette surface sur une feuille (c'est à dire une surface plane) ou sur un écran d'ordinateur (ou plutôt sur la mémoire graphique correspondante, qui peut être également considérée comme une surface plane, alors que certains écrans sont encore un peu bombés…), il faut passer de la surface curviligne d'un corps dans un espace à trois dimensions à une surface plane à deux dimensions. Comme les ellipsoïdes ne sont pas développables sur un plan (un cylindre ou un cône peuvent être développés sur un plan; la surface d'une sphère, d'un ellipsoïde de révolution ne peut pas l'être sans déchirement), il faut utiliser une fonction mathématique, que l'on veut bijective, pour effectuer cette transformation : cette fonction s'appelle une projection cartographique. Elle s'applique aux points comme aux éléments linéaires et surfaciques.

Passer d'une surface curviligne non développable à une surface plane implique des déformations, dont on cherche au mieux à maîtriser les caractéristiques suivant le but recherché de la représentation cartographique : conserver les formes, les distances, les surfaces, les angles... Choisir la fonction de projection permet de choisir les caractéristiques de la déformation.

La projection n'est donc plus un problème de mesure mais seulement de calcul, puisque l'on part de l'ellipsoïde et que toutes les mesures ont été effectuées auparavant. La précision est donc liée au calcul, alors qu'elle était principalement liée aux mesures dans les problèmes précédents, en géodésie comme en gravimétrie. De même, connaître la déformation induite par la projection n'est qu'un problème de calcul, et la précision en est donnée par la précision du calcul utilisé.

4.3.1. Les déformations

Revenons au problème de la projection : représenter une partie de l'ellipsoïde de référence sur une surface plane en maîtrisant les paramètres de la déformation. Notons tout de suite que le terme projection est mal choisi, car il peut faire croire à une transformation surjective, alors que dans son domaine de définition, la transformation sera toujours une application bijective : on pourra toujours définir la projection inverse d'une projection donnée. Les déformations introduites par la projection sont définies par les relations entre éléments correspondants sur les deux surfaces, l'ellipsoïde et le plan, et essentiellement par l'étude différentielle de la déformation, c'est à dire l'étude de la déformation en un point et au voisinage (infinitésimal) de ce point suivant une direction donnée. On a l'habitude de classer les déformations suivant trois groupes :

- les déformations linéaires : au voisinage d'un point, on compare la longueur d'un segment sur l'ellipsoïde et la longueur du segment projeté sur le plan. En général, le rapport des longueurs (appelé module de déformation linéaire ou échelle locale) varie avec le point et avec la direction du segment. En étudiant la distribution du module de déformation linéaire dans le plan de projection, on peut y définir des lignes isométriques (tous les points de la ligne ont même échelle locale). Si le module linéaire est égal à 1, il n'y a aucune déformation linéaire quelque soit la direction considérée. On définit également les lignes automécoïques : le module de déformation linéaire est égal à 1 mais seulement dans la direction de la ligne. Aucune projection ne conserve les distances en tout point : une telle propriété impliquerait l'absence totale de déformation. Les lignes automécoïques sont importantes, car sur ces lignes, les distances (planes) dans le plan de projection correspondent aux distances (curvilignes) sur l'ellipsoïde. Elles permettent donc de fixer une unité de longueur dans le plan de projection.

- les déformations surfaciques : on considère au voisinage d'un point un élément de surface et la projection de cet élément dans le plan. Si le module de déformation des superficies est égal à 1 en tout point, la projection sera dite équivalente.

- les déformations angulaires : au voisinage d'un point, on considère deux éléments linéaires formant un angle α. Ces deux éléments se transforment en deux éléments dans le plan, formant un angle α'. On appelle déformation angulaire la différence des angles α-α'. Les projections conformes ont comme propriété de conserver les angles (la déformation angulaire est nulle en tout point).

La connaissance des déformations induites par la projection est importante lorsque l'on veut retrouver une valeur curviligne (sur l'ellipsoïde, et donc sur la surface terrestre) à partir d'une mesure sur une carte : on souhaite minimiser les calculs à effectuer entre la mesure sur la carte et la correspondance sur l'ellipsoïde. Par exemple, une projection conforme conserve les angles : on peut donc mesurer directement un angle sur une carte utilisant une projection de ce type, cet angle sera le même sur l'ellipsoïde (les projections conformes sont très utilisées en balistique et en navigation). Le calcul d'une distance est plus difficile, car aucune projection ne permet de conserver les distances entre deux points quelconques. Avant le développement des ordinateurs, on utilisait des tables d'interpolation pour ces calculs, et la précision de l'interpolation donnait la précision du calcul. L'automatisation des calculs grâce aux ordinateurs et la cartographie numérique ont rendu ces tables d'interpolation quelque peu obsolètes.

D'autres propriétés peuvent être importantes. Par exemple, la projection Mercator est non seulement conforme, elle a également l'avantage de transformer les loxodromies (routes à cap constant) en des droites, ce qui facilite la navigation sur de courtes distances.

Beaucoup de concepts mathématiques proviennent de problèmes cartographiques, ou plus généralement de problèmes de représentation de la Terre. Ces problèmes ont été depuis l'antiquité un moteur du développement théorique : fonctions analytiques, calcul différentiel, etc. Par exemple, la condition de conformité est identique à la condition d'holomorphisme pour les fonctions analytiques (équations de Cauchy-Riemann).

4.3.2. Les surfaces développables

Les projections peuvent également être classées d'une autre façon, en utilisant une surface intermédiaire développable sur laquelle on effectue une première projection. La forme de cette surface intermédiaire détermine :

- les projection coniques, tangentes ou sécantes : on choisit un cône tangent ou sécant à l'ellipsoïde. La projection Lambert est la plus célèbre des projections coniques conformes ;

- les projections cylindriques : c'est un cylindre qui est tangent à l'ellipsoïde ;

- les projections planes (ou perspectives), en projetant directement sur un plan tangent à la surface de l'ellipsoïde (par exemple dans les projections utilisées aux pôles).

Suivant la position de l'axe de la surface par rapport à l'axe de l'ellipsoïde, on parle également de :

- projections directes : l'axe de la surface développable est identique à l'axe de rotation de l'ellipsoïde (Lambert, Mercator,...) ;

- projections transverses : l'axe de la surface est perpendiculaire à l'axe de rotation (par exemple, la projection Universal Transverse Mercator) ;

- projections obliques ou horizontales, si l'axe est un diamètre quelconque de l'ellipsoïde.

4.3.3. Coordonnées et espace de projection

La projection transforme un point de coordonnées (λ,ψ) de l'ellipsoïde en un point d'un plan dans un repère orthonormé (O,*i*,*j*), d'origine O. On a le choix de l'origine et de l'unité dans le plan. L'origine est le plus souvent choisie en prenant un point *Po* de coordonnées $(\lambda 0,\psi 0)$ sur l'ellipsoïde, et en fixant des coordonnées *(X0,Y0)* du point projeté dans le plan (O,*i*,*j*). On s'arrange en fait pour choisir *(X0,Y0)* de manière à ne pas avoir de coordonnées négatives pour le domaine projeté, pour la partie de l'ellipsoïde à laquelle on s'intéresse. $\lambda 0$ s'appelle alors le méridien central, *X0* le faux nord (FALSE NORTHING), *Y0* le faux est (FALSE EASTING). Par exemple, pour la projection UTM, on fixe à 500000 la valeur de *X0*, abscisse de la valeur projeté du point *P0* de coordonnées sphériques $(\lambda 0,\psi 0)$. L'unité est toujours le mètre. On essaye là aussi d'avoir la meilleure correspondance entre la distance curviligne (distance mesurée sur l'ellipsoïde entre deux points, c'est à dire longueur de l'arc) et la distance entre les points projetés dans le plan. Les projection ne conservent pas les distances, comme nous l'avons vu plus haut, sauf sur les lignes isométriques ou automécoïques où les distances sont conservées. La définition de l'unité utilise l'une de ces lignes, en prenant en compte la valeur du module linéaire correspondant à la ligne (pour cela appelé facteur d'échelle locale). Si on utilise une ligne automécoïque pour définir l'unité de la projection, on aura sur cette ligne conservation des distances, dans la même unité. Nous verrons dans les exemples de projections, plus loin dans ce chapitre, quelles sont les lignes automécoïques utilisées pour définir l'unité de la projection.

Par exemple, les ordonnées d'une projection UTM, exprimées en mètres, vont de 0 à 10 000 000 dans chaque hémisphère. On peut se demander par quel miracle la distance curviligne entre l'équateur et le pôle d'un ellipsoïde de révolution approchant la Terre mesure exactement dix millions de mètres (en approchant l'ellipsoïde par un cercle, on a $d=\theta*R$, ou R est le rayon du cercle, $\theta=90°=\pi/2$ radian. 6 400 000*3.14159/2=10 000 000 !). La réponse est toute simple : c'est ainsi qu'a été défini le mètre au XVIIIe siècle. Alors que les unités utilisées variaient entre la lieue, la verge, et un grand nombre de valeurs du pied, le développement de la cartographie et des mesures de l'arc ont incité les scientifiques de l'époque à définir une unité unique facile à utiliser pour les calculs de projections et de distances. On a donc défini le mètre comme la 10 000 000-ième partie de l'arc allant de l'équateur au pôle sur l'ellipsoïde de Picard (loi du 19 Frimaire an VIII - 10 décembre 1799), tout en définissant le grade comme la 100-ième partie de l'angle au centre correspondant (90°). L'UTM ayant un méridien comme ligne automécoïque (le méridien central), les coordonnées dans la projection vont également de 0 à 10 000 000, exprimées dans cette nouvelle unité, le mètre.

Notons encore que les rosaces indiquant le nord géographique indiquent en fait la direction du nord sur le méridien central de la projection : les projections des méridiens ne sont parallèles que dans certaines projections.

4.3.4. Échelles et cartes

Nous n'avons pas encore parlé d'échelle, car cette notion n'intervient pas dans la propre action de projeter sur un plan. Dans le plan de projection, les distances sont sensiblement les mêmes que sur l'ellipsoïde, comme nous l'avons vu plus haut. Impossible bien sûr de représenter ce plan de projection sur une feuille de papier, il faut réduire la taille des éléments. On va donc effectuer une homothétie de rapport inférieur à 1 par rapport à l'origine du plan de projection. D'un domaine très grand, on arrivera donc à une surface plus petite, permettant d'attendre la taille d'une feuille de papier : la carte. L'échelle indique le rapport utilisé pour l'homothétie. Lorsqu'un point est projeté puis mis à l'échelle par homothétie, les coordonnées indiquées sur la carte sur laquelle on l'a dessiné sont soit des coordonnées après projection mais avant homothétie (les coordonnées du point dans le plan de projection), soit des coordonnées géographiques (on indique donc la longitude et la latitude du point sur l'ellipsoïde, avant sa projection puis sa mise à l'échelle ; on a d'ailleurs l'habitude de représenter sur une carte des points correspondant à des valeurs entières des coordonnées sphériques : ce sont les amorces de la projection). Bien sûr, ces coordonnées ne correspondent pas à l'unité de la carte après mise à l'échelle, mais les coordonnées ne sont jamais exprimées dans cette unité.

L'échelle de restitution cartographique est donc une pure transformation mathématique, elle n'est pas directement liée à la précision. Ainsi, quelle que soit l'échelle, la déformation liée à la projection est la même (puisque la projection ne

dépend pas de l'échelle). Si l'on mesure cette déformation sur une carte, sa valeur absolue sera bien sûr divisée par l'échelle de la carte, comme tous les éléments représentés. La valeur relative entre un objet et la déformation liée à la projection reste donc la même quelque soit l'échelle utilisée. Par exemple, on dit souvent qu'à partir d'une certaine échelle (pour le cadastre par exemple), la projection géographique peut être négligée car la déformation due à la projection devient très faible, et l'espace étudié assimilable à un plan. En fait, c'est la faiblesse de la déformation par rapport à la taille ou la précision de mesure des objets représentés qui importe. Et puis il ne faut pas oublier non plus tout ce que l'on a dit auparavant sur le passage de la Terre à l'ellipsoïde : les grandeurs relatives des erreurs doivent être comparées pour pouvoir négliger l'une ou l'autre.

Si l'on a pas besoin de réduire la taille du document (virtuel bien sûr), on peut conserver des coordonnées projetées sans facteur de réduction - ou plutôt avec un facteur 1. C'est ce que l'on fait dans tous les systèmes numériques, où la description de la localisation est conservée soit en coordonnées sphériques sur un ellipsoïde, soit en coordonnées projetées, et où la mise à l'échelle ne se fait que lorsque l'on en a besoin (pour visualiser des résultats).

4.3.5. Échelle, projection, précision

Au cours de ce qui précède, nous avons parlé à plusieurs reprises de précision, et il est intéressant de revenir sur ce sujet car il est d'une importance capitale dans la conception et dans l'utilisation des bases de données localisées.

La précision n'est jamais indiquée sur une carte. Elle est, dans l'esprit de tous, implicitement donnée par l'échelle : on entend souvent dire qu'une carte au 1:1000000 est moins précise qu'une carte au 1:1000. Qu'étant donné que le micron n'est pas visible à l'œil nu, il ne sert à rien de chercher à représenter une précision d'un mètre terrain à l'échelle 1:1000000. Que si c'est pour les représenter à une petite échelle, il est inutile de chercher une trop grande précision. Et qu'implicitement, c'est en fonction des possibilités de représentation de la carte que l'on va fixer la précision des objets... Mais ce serait bien sûr prendre le problème fondamental de la précision à l'envers.

Comme dans toute science physique, la définition d'un objet ne peut être séparé de la précision de la mesure des attributs servant à le décrire. Les objets traités en géographie et en cartographie n'existent pas a priori, comme nous l'avons souligné dans le chapitre précédent : entre autre, la précision apportée à la description de la localisation est fondamentale dans la définition sémantique de l'objet et donc dans l'utilisation géographique ou cartographique qui peut en être faite. La confusion vient de ce qu'implicitement on s'arrangeait, avant l'avènement du numérique, pour

que toutes les cartes aient *grosso modo* le même degré de précision mesuré sur la carte. La valeur *absolue* de cette précision dépendait donc de l'échelle de la carte. Comme toutes les cartes en papier ont à peu près les mêmes dimensions, quelque soit l'échelle qui, elle, varie considérablement, les objets que l'on y représente sont finalement à peu près de la même taille - sur la carte - et le rapport *taille de l'objet/précision de la localisation* reste à peu près le même. Mais - et c'est fondamental quand il faut définir l'objet géographique - la précision géographique absolue de la localisation est la seule notion vraiment importante, la seule intimement liée à la définition des attributs de l'objet, et la seule qui puisse être indiquée dans un système informatique où la localisation est conservée indépendamment de la notion d'échelle cartographique.

La projection apporte une erreur systématique et connue (ou plutôt une déformation contrôlée), et la précision du calcul de la déformation peut être très grande et adaptée aux besoins. La réduction ou mise à l'échelle n'apporte aucune erreur. Ces opérations ne jouent donc aucun rôle dans la précision de données géographiques.

La seule précision vient donc de la mesure, et cette précision doit être définie lorsque l'objet géographique et son champ d'application sont eux-mêmes définis. Les instruments de mesure doivent être compatibles avec cette précision (par exemple, le capteur d'un satellite, l'objectif d'un appareil de photographie aérienne, les mesures des géomètres, les GPS, la restitution photogrammétrique, etc.). Lorsque les objets sont thématiques (pédologie, végétation, ...), la précision de leur localisation dépend de deux critères : un fond de carte, dont on peut estimer la précision avec les critères précédents, et la définition même des objets, qui dépend en général de « l'échelle » à laquelle travaille celui qui élabore l'information, et que nous appelons plutôt *définition sémantique* ou *échelle de description*. Plus qu'une notion d'échelle, cette notion correspond bien à un niveau de précision et à une vision de la réalité propre à cette précision. Elle est directement liée à la vision de la réalité et à sa modélisation.

La généralisation est un processus cartographique qui permet de modifier la précision de la localisation, par simplification, après avoir modifié la schématisation conceptuelle des objets lorsque l'on effectue un changement d'échelle de description. Cette opération, présentée souvent comme purement cartographique, permet d'adapter le dessin d'une carte à son échelle, pour ne pas avoir une précision excessive qui n'apporte plus d'information à l'échelle à laquelle on représente les objets. En fait, elle montre bien l'exigence d'une bonne cohérence entre précision de la localisation et validité des descripteurs des objets que l'on veut représenter. La généralisation des contours est une opération difficile à automatiser, puisqu'elle fait partie d'un processus de redéfinition conceptuelle des objets. Il ne faut pas la confondre avec les opérations de filtrage, qui sont seulement destinées à réduire le nombre de

points nécessaires à la description d'arcs, sans en modifier la forme ou l'apparence. Nous verrons au chapitre 9 la description de l'algorithme de généralisation de Douglas-Peucker.

Les SIG remettent en cause les méthodes de travail classiques pour l'élaboration de l'information localisée : le critère fondamental pour la précision des objets n'est plus l'échelle d'une carte, considérée comme document final, et où l'on mélange précision de la mesure, précision sémantique, et précision du dessin. C'est uniquement la définition sémantique des objets qui donne la précision à laquelle on doit envisager de décrire la localisation de ces objets, puisque les SIG permettent de conserver les objets sans la notion d'échelle de restitution. Et on retrouve bien sûr ici tous les problèmes liés à la modélisation de l'espace en zones homogènes : rien ne sert de décrire avec trop de précision des zones alors que la validité de l'information n'y est pas assurée à cette précision. Il serait beaucoup plus efficace de définir un modèle permettant de prendre en compte et de quantifier cette imprécision. Pour le moment, les méthodes classiques de représentation de l'espace en zone, ligne, point, méthodes héritées de l'ère pré-numérique et des exigences de la cartographie d'édition, n'ont pas vraiment été remises en cause par les SIG. Il ne faut jamais perdre de vue ces limitations lorsque l'on utilise un SIG, et bon nombre de traitements possibles sont souvent inutilisables pour des problèmes de précision et de modèle de représentation de la réalité.

4.4. Exemples de projections

Nous allons présenter trois exemples de projections géographiques qui sont parmi les plus utilisées : la projection Mercator, la projection UTM, la projection conique Lambert. Toutes trois sont conformes (elles conservent les angles), cette propriété étant de loin la plus intéressante pour la navigation ou la balistique. Nous avons défini une classe `CProjection` qui contient comme membres tous les paramètres nécessaires au calcul, quelque soit la projection utilisée (A.2.3.2.).

4.4.1. La projection Mercator

La projection Mercator est une projection cylindrique directe, contrairement aux projections TM et UTM qui sont transverses. Cette projection est très utilisée en navigation, car, outre la conformité, elle a l'avantage de transformer les routes à cap constant en des droites sur la carte. Elle a longtemps été utilisée pour les planisphères, malgré les considérables déformations qu'elle introduit dès que l'on s'approche des latitudes élevées. De plus, les projections conformes ne sont pas équivalentes : elles ne conservent pas les surfaces. La projection Mercator augmente les surfaces dès que l'on s'éloigne de l'équateur : le poids visuel de l'Europe, de

l'Asie du nord, de l'Amérique du nord, de l'Océanie augmente au détriment de l'Afrique, de l'Asie centrale, de l'Asie du sud et de l'Amérique du sud.

4.4.2. La projection UTM (Universal Transverse Mercator)

La projection UTM est une projection cylindrique transverse : le cylindre de projection est tangent à l'ellipsoïde suivant une ellipse passant par les pôles. La position de cette ellipse correspond au méridien central de la projection. La déformation devient importante si l'on s'éloigne trop de ce méridien : la projection UTM n'est utilisée que pour représenter des régions proches du méridien central (au maximum trois à quatre degrés de part et d'autre). Des méridiens centraux ont donc été définis une fois pour toute pour couvrir l'ensemble du globe, découpé en zones : le premier est à 3° et les suivants se suivent à 6°. L'indication de la zone UTM sur une carte permet de retrouver le méridien central correspondant :

zone 1 est : de 0° à 6° est, méridien central 3° est;
zone 2 est : de 6° à 12° est, méridien central 9° est;
zone 3 est : de 12° à 18° est, méridien central 15° est;
...
zone 30 est : de 174° à 180° est, méridien central 177° est;
zone 1 ouest : de 174° à 180° ouest, méridien central 177° ouest;
zone 2 ouest : de 168° à 174° ouest, méridien central 171° ouest;
...
zone 30 ouest : de 6° ouest à 0° ouest, méridien central 3° ouest;

La projection est dite universelle car elle utilise un même point de tangence par continent. Lorsque cette condition n'est pas respectée, la projection devient TM (transverse Mercator). La seule ligne automécoïque est une droite qui correspond à la projection du méridien central. Cette ligne permet de définir l'unité de la projection. Comme le mètre a été défini comme la 10 000000-ième partie de la projection du méridien central (de l'équateur au pôle), l'unité naturelle de la projection UTM est le mètre. On applique néanmoins un facteur d'échelle pour rattraper les modifications dues à l'ellipsoïde utilisé (le mètre a été défini avec l'ellipsoïde de Picard) et à l'altitude : un mètre mesuré au sol sur le méridien central correspond alors exactement à un mètre après calcul de projection. Ce facteur d'échelle est toujours très proche de 1.

Comme pour beaucoup de projections, le calcul inverse nécessite le calcul d'un arc de méridien sur l'ellipsoïde. Ce calcul est effectué par approximations successives (`CProjection::SetEllipse()`), ce qui influence la précision de la transformation. Cette précision est de l'ordre du centimètre.

Les coordonnées UTM ont le défaut de ne pas différencier l'hémisphère : il y a donc une rupture au niveau de l'équateur, où l'on passe de 9 999 999 à 0. Pour éviter cette rupture, nous avons choisi d'ajouter 10 000 000 dans l'hémisphère nord. Les méthodes `UTMToGeo()` et `GeoToUtm()` donne l'implantation du calcul de la projection UTM dans la classe `CProjection`.

4.4.3. La projection conique de Lambert

La projection Lambert est une conique conforme qui permet de représenter des surfaces plus importantes que la projection UTM et convient particulièrement aux latitudes moyennes. Elle a été particulièrement utilisée pour la France, l'Afrique du Nord, l'Amérique du Nord. La projection Lambert admet de nombreux paramètres : le cône peut être tangent ou sécant; le choix du méridien central permet de fixer la ligne automécoïque et l'origine de la projection. Certaines Lambert portent un nom, et tous les paramètres sont alors fixés. Par exemple :

Lambert I Nord (tangente) :
```
Méridien central : 2°20' 14.025",EST;
Parallèle tangent : 49° 30' 0.",NORD;
Ellipsoide(6378249.2,0.00680348765);
Facteur d'échelle : 0.99987734;
Coordonnées au point d'origine : 600000.,200000.;
```
Lambert II Centre (tangente) :
```
Méridien central : 2° 20' 14.025",EST;
Parallèle tangent : 46° 48' 0.",NORD;
Ellipsoide(6378249.2,0.00680348765);
Facteur d'échelle : 0.99987742;
Coordonnées au point d'origine : 600000.,200000.;
```
Lambert III Sud (tangente) :
```
Méridien central : 2° 20' 14.025",EST;
Parallèle tangent : 44° 6' 0.",NORD;
Ellipsoide(6378249.2,0.00680348765);
Facteur d'échelle : 0.99987750;
Coordonnées au point d'origine : 600000.,200000.;
```
Lambert IV Corse (tangente) :
```
Méridien central : 2° 20' 14.025",EST;
Parallèle tangent : 42° 9' 54.",NORD;
Ellipsoide(6378249.2,0.00680348765);
Facteur d'échelle : 0.99987771;
Coordonnées au point d'origine : 234.358,185361.369;
```
Eurolambert (tangente) :
```
Méridien central : 2° 20' 14.025",EST;
Parallèle tangent : 46° 48' 0.",NORD;
Ellipsoide(6378388.2000,0.00672267002);
Facteur d'échelle : 0.99987750;
Coordonnées au point d'origine : 600000.,2200000.;
```
Lambert Etendu (tangente) :
```
Méridien central : 2° 20' 14.025",EST;
Parallèle tangent : 46° 48' 0.",NORD;
Ellipsoide(6378249.0000,0.00680348765);
Facteur d'échelle : 0.99987742;
Coordonnées au point d'origine : 600000.,2200000.;
```
Lambert Mexique (sécante) :
```
Méridien central : 102° 0' 0.,OUEST;
Parallèle du point d'origine : 24° 0' 0.",NORD;
Parallèle 1 : 17° 30' 0.",NORD;
Parallèle 2 : 29° 30' 0.",NORD;
Ellipsoide(6378206.4000,0.006768657997);
Coordonnées au point d'origine : 2000000.,2000000.;
```

L'initialisation des paramètres est différente selon les types de projection Lambert (tangente ou sécante). Le calcul de projection est ensuite identique. Les diverses méthodes de la classe `CProjection` correspondant à la projection Lambert peuvent être consultées dans l'annexe (A.2.3.2.).

4.5. SAVANE, datum et projections

Le système SAVANE conserve l'ensemble des coordonnées géographiques sous la forme longitude-colatitude dans un *datum* donné. Le *datum* doit être identique pour l'ensemble des objets d'une même base de données. Nous avons donc deux besoins fondamentaux pour intégrer des objets dans une base de données SAVANE : pouvoir passer de coordonnées données suivant une projection aux coordonnées géographiques du même point, par projection inverse, et pouvoir changer de *datum*. L'ensemble de ces procédures est disponible dans la classe `CProjection`. Cette classe est également utilisée dans l'utilitaire GLOBE qui permet de faire des calculs de changement de projection :

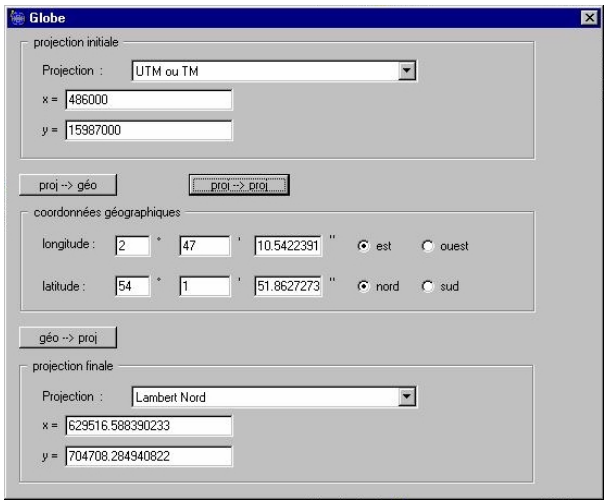


fig. 4.4 : le programme Globe

4.5.1. Provenance des données en entrée

Il n'y a en fait que peu de sources de données pour la localisation : relevés de terrain, images aériennes ou spatiales, cartes topographiques ou toponymiques en forment l'immense majorité. Les images aériennes ou spatiales sont redressées et mises en conformité géographique, suivant une projection, grâce notamment à des

mesures de localisation et d'altitude. Les cartes sont réalisées à partir d'images aériennes et de mesures de terrain. Les mesures de terrain, longtemps effectuées par triangulation, sont maintenant le plus souvent effectuées par GPS.

Les données de localisation servant à constituer une base de données localisées proviennent de ces documents. Le système doit donc être en mesure de fournir les outils permettant la saisie graphique de ces documents. La source principale de données est donc constituée de cartes, réalisées avec une certaine précision, selon une projection géographique et sur un *datum* de référence. Le système SAVANE permet la saisie sur écran à partir de ces cartes scannées et géoréférencées grâce au module SAVEDIT. Il permet également l'importation directe de données de positionnement provenant de relevés de terrain.

4.5.2. Fonctionnement de SAVEDIT pour les projections inverses et le datum

La saisie graphique sur écran avec SAVEDIT s'effectue en utilisant la fenêtre du programme comme espace de digitalisation. Cette espace correspond à une fenêtre ouverte sur un espace géographique. Il est important de bien maîtriser les paramètres de cet espace géographique. En voici les principes.

La surface de l'écran de saisie de SAVEDIT correspond à un espace muni d'un repère orthonormé : on peut donc passer des coordonnées écran aux coordonnées d'une projection géographique, par une simple translation-homothétie dépendant seulement de l'espace visualisé sur l'écran. On peut ensuite retrouver les coordonnées sphériques (longitude, latitude), dans un *datum* donné, par projection inverse. Pour faire cette transformation, SAVEDIT doit seulement connaître la projection géographique et l'espace de travail correspondant à la fenêtre de visualisation : l'utilisateur du programme doit préciser ces paramètres lorsqu'il rentre dans le système, de manière à initialiser l'espace de travail. Lorsque la saisie se fait à partir d'une carte scannée ou d'une image géoréférencée par SAVAMER, le chargement de la projection correspondant au géoréférencement de la carte ou de l'image est tout à fait naturel. Lors de l'importation d'un fichier externe, les coordonnées du document s'adaptent automatiquement à la projection de visualisation. L'utilisateur doit seulement préciser la projection initiale du document, de manière à convertir les coordonnées de projection en coordonnées sphériques. L'espace de travail est choisi tout aussi naturellement. Au départ, l'utilisateur le choisit en fonction du support de la digitalisation (espace d'une carte ou d'une image géo-référencée, espace complet d'un document, visualisation d'éléments de la base de données), ensuite, il peut le modifier en fonction des besoins de visualisation ou de saisie.

Tout document digitalisé est conservé en coordonnées géographiques, dans un *datum* donné. Lors du processus de digitalisation, SAVEDIT convertit chaque point saisi sur l'écran en coordonnées sphériques, en fonction de la fenêtre de visualisation et de la projection géographique inverse. S'il est ouvert avec une autre projection, il sera tracé dans la fenêtre de visualisation selon cette autre projection, sans aucun problème.

Le fonctionnement de SAVEDIT est donc différent de la plupart des logiciels de saisie graphique. Dans SAVEDIT, l'espace de travail, celui qui est visualisé sur l'écran, correspond toujours à un espace géographique tracé selon une projection géographique connue (par défaut, le système utilise la projection dite géographique, avec un méridien central centré sur la base de donnée ouverte). Chaque point de l'écran peut être repéré par ses coordonnées dans les divers systèmes (projection, ellipsoïde, écran) grâce à la fenêtre de navigation. Lors de la saisie, la précision sera d'autant plus grande que la fenêtre de visualisation sera étroite, puisqu'à la taille du pixel correspondra alors un espace plus petit dans la réalité.

Lorsque l'on crée un document avec SAVEDIT, on peut indiquer son *datum*. Cette indication n'est pas toujours indispensable, et elle ne change en rien les coordonnées des points à saisir. Par contre, on peut changer le *datum* d'un document par la suite, et si les coordonnées des points saisis doivent être modifiés pour prendre en compte le déplacement de l'ellipsoïde de référence, chaque coordonnée est modifiée pour refléter ce changement (le programme utilise la formule de Molodensky). Ce problème est courant lorsque des données saisies avec un GPS dans un *datum* doivent être intégrées avec des données conservées selon un autre *datum*. Il est nécessaire, après avoir importé les points dans SAVEDIT, de spécifier le *datum* initial des points GPS puis de modifier ce *datum* en modifiant les coordonnées des points. Nombreux sont ceux qui ont eu le désagrément, en reportant sur une carte leurs points saisis avec précision sur le rivage, de les retrouver loin dans la mer ou dans un lac… Question de *datum* !

La classe `CProjection` est utilisée pour les calculs de transformation de projection dans SAVEDIT (A.2.3.2.).

La classe CWind de SAVEDIT contient la description de la fenêtre de travail, et quelques fonctions permettant de modifier cette fenêtre. Par exemple, la fonction `NouvelleFenetreProj()` permet de modifier la fenêtre de travail correspondant à l'écran en donnant les coordonnées de projection de cette fenêtre. Quelques fonctions globales permettent d'effectuer les transformations écran-espace de projection-datum.

La classe `CMolodensky`, que nous avons décrite au début de ce chapitre, permet d'effectuer les transformations de *datum* en utilisant la formule de Molodensky (A.2.3.3.).

4.5.3. Fonctionnement de SAVANE pour les projections dans un cadre

Une requête dans SAVANE correspond à un espace de travail dans une carte. Pour la visualisation des résultats de cette requête, SAVANE utilise un cadre et une opération de cartographie. Chaque cadre a donc des paramètres de projection géographique, de coordonnées correspondant à l'espace à visualiser, d'échelle de restitution.

Toutes les données graphiques sont conservées en coordonnées géographiques dans SAVANE, selon un *datum* identique pour tous les objets d'une même base de données. La visualisation d'un cadre utilise donc :

- la position du cadre dans la carte
- l'échelle du cadre
- la fenêtre géographique du cadre (le cadre est comme une fenêtre ouverte sur la base de données géographique)
- la projection géographique choisie
- lorsque la visualisation se fait sur un écran, il faut également prendre en compte la position de la carte dans l'écran.

Lorsque l'on démarre une requête dans SAVANE, tous les paramètres du cadre par défaut sont définis par défaut. Il faut agir sur les boutons CADRE, MAP et WIND pour modifier ces paramètres. La classe `CCadre` (A.2.4.4.) contient un certain nombre de variables membres correspondant à ces paramètres. Les classes `CProjection` (A.2.3.2.) et `CWind` (A.2.3.1.) sont identiques aux classes du même nom dans SAVEDIT.

4.6. Conclusion

Mesurer la localisation s'avère être une opération complexe si l'on veut en maîtriser la précision et l'incertitude. Tous les paramètres de la mesure interviennent lors de la réalisation d'un SIG. Dans le système SAVANE, nous avons choisi de conserver les coordonnées des points décrivant la localisation des objets géographiques en coordonnées sphériques (longitude, colatitude) dans un même datum. Il est donc important de pouvoir passer d'un datum à un autre, et de pouvoir passer de coordonnées projetées à des coordonnées sphériques. La projection géographique est utilisée lors de la constitution de la base de données, lors de la

saisie graphique, et lors de l'exploitation des données et de la cartographie des résultats, de façon interactive.

Des objets géographiques ne peut être intégrés à une base de données que s'ils sont géoréférencés avec la précision correspondant à leur définition sémantique. Si tel n'est pas le cas, la définition elle-même des objets doit être remise en cause. Dans le cas contraire, certains traitements ou mise en relation d'objets pourraient amener à des résultats non valides, notamment en analyse spatiale.

Bibliographie

[BAL 98] BALMINO G., *Champ de pesanteur terrestre et géoïde. Principes, progrès et connaissance actuelle*, 1998, Bureau Gravimétrique International, Toulouse, France

[BEG 94] BEGUIN M., PUMAIN D., *La représentation des données géographiques*, 1994, CURSUS-Armand Colin, Paris

[BOT 98] BOTTON S ., DUQUENNE F., EGELS Y., EVEN M., WILLIS P., *GPS, localisation et navigation*, 1998, Hermès, Paris

[CAZ 94] CAZENAVE A., FEIGL K., *Formes et mouvements de la Terre*, 1994, Ed.Belin-CNRS

[DUF 98] DUFOUR J.P., *Cours d'introduction à la géodésie*, Ecole nationale des sciences géographiques,1998, IGN-ENSG

[GEO 71] *Géophysique*, Encyclopédie de la Pléiade, sous la direction de Jean Goguel, Gallimard, 1971, Paris

[LEV 88] LEVALLOIS J.J., BOUCHER C., BOURGOIN J., COMOLET-TIRMAN A., ROBERTOU A., *Mesurer la Terre : 300 ans de géodésie française, De la toise du Châtelet au satellite*, 1988, Association française de topographie, Presse de l'Ecole Nationale des Ponts et Chaussées, AFT, Paris

[MAG 97] MAGELLAN SYSTEMS CORPORATION, *User guide for the Magellan GPS ProMARK X-CM*, 1997

Chapitre 5

De l'objet à la collection d'objet : les SGBD

5.1. Introduction

Jusqu'à présent, nous avons étudié la manière de schématiser et représenter la réalité par des objets, sans tenir compte du problème de la *collection*. Dans beaucoup de situations, et notamment en géographie, la réalité à étudier ne peut se schématiser par quelques objets distincts, ou alors ces objets seraient tellement complexes qu'ils en deviendraient difficilement utilisables pour un objectif autre qu'une pure description. En particulier, on ne pourrait pas les comparer entre eux et utiliser tout l'arsenal de la statistique pour les étudier, les analyser, et en extraire des caractéristiques. On cherche donc plutôt à définir des objets assez simples, et à représenter la réalité par des collections de ces objets. L'ensemble des collections schématisant une réalité à étudier est appelé une base de données.

La gestion efficace de ces collections d'objets est nécessaire, car les objets peuvent être très nombreux, même si leur description individuelle est assez simple. Pour gérer efficacement ces collections d'objets indépendamment des programmes qui vont les utiliser, il faut des systèmes de gestion de bases de données. Plusieurs modèles de description ont été définis pour passer d'une schématisation en objets à une schématisation en collections d'objets, non plus seulement en fonction de critères de description, mais également en fonction de critères de gestion et de la difficulté à gérer ces collections d'objets. Nous reviendrons donc également sur les problèmes de schématisation et de représentation de la réalité.

La mise en base de données n'est pas toujours nécessaire, notamment lorsque l'objectif est la description de quelques objets distincts : ainsi, la description de paysages rentre difficilement dans le cadre contraignant d'une schématisation de la

réalité, car l'objectif est alors la description détaillée d'un seul ou de quelques objets, chaque objet ayant des attributs qui lui sont propres. La difficulté vient bien de l'objectif recherché, et non pas de la technologie des bases de données.

5.2. Notions classiques sur les SGBD

Le problème de la gestion des données se pose lorsque les applications ont besoin de grands volumes d'information, c'est-à-dire de grandes collections d'objets décrits de façon complexe par de nombreux attributs. La création, la disponibilité, l'utilisation de ces grandes collections est très coûteuse pour une seule application. On a donc cherché à mettre en commun ces objets, en les définissant de façon intrinsèque et en les mettant dans des bases de données entièrement gérées par des programmes spécialisés, qui ne font que ça : les systèmes de gestion de bases de données (SGBD) [GAR 83] [ULL 88]. Un SGBD, c'est une interface entre l'usager (un utilisateur, un programme) et les mémoires de stockage, lui donnant l'illusion d'avoir à sa disposition des données stockées et assemblées comme il le souhaite, d'être le seul à les utiliser, et de pouvoir les manipuler à sa guise grâce à un langage spécial. C'est un outil de gestion efficace permettant de rechercher, modifier, ajouter des données dans une grande masse d'information, partagée par tous les usagers suivant leurs droits d'accès, chacun travaillant sur sa vision et sa propre structuration de l'information.

5.2.1. Les problèmes soulevés par l'utilisation de fichiers traditionnels : dépendance, redondance, intégrité, sécurité

A l'origine de l'informatique, les données utilisées par une application étaient toujours liées à cette application, et utilisées sous forme de fichiers traditionnels. Ces fichiers représentaient des collections d'objets décrites par des attributs, mais ces objets et leur description étaient en général définis uniquement pour l'application. Lorsqu'un programme utilise des données sous forme de fichier traditionnel, données et programme sont dépendants : le fichier est conçu pour le programme ou le programme pour le fichier, et les structures logiques (description pour les objets, exécution pour le programme) sont dépendantes l'une de l'autre. Nous n'avions donc pas de problèmes de description et de représentation du monde réel, et la définition, description et représentation des objets découlaient naturellement de l'analyse du problème étudié et de sa résolution algorithmique et informatique. Cette organisation, optimale dans certains cas, est désastreuse dans d'autres, notamment quand les mêmes données sont utilisées par plusieurs applications différentes, car on est alors rapidement confronté à des problèmes de redondance et de cohérence entre fichiers : si plusieurs programmes utilisent les mêmes données dans des fichiers différents, rien ne garantit en effet que ces

données soient équivalentes et que les mêmes contrôles soient effectués. On est confronté à de nombreux problèmes lorsque plusieurs programmes veulent utiliser le même fichier : problèmes de redondance ou de stockage, problèmes d'accès pour unifier les modifications et mises à jour. La création de fichiers réservés à une seule application est très coûteuse : en volume, en coût de développement informatique, en disponibilité des données, en problèmes de cohérence et de sécurité, en fiabilité, en accès aux données par des non-informaticiens.

L'idée toute simple et qui est à l'origine des SGBD, c'est de séparer données et applications en créant des ensembles cohérents de données indépendants des applications et en utilisant des programmes spécialisés pour les gérer et les manipuler, et d'offrir aux programmes d'applications ces données déjà structurées et ces utilitaires pour y accéder le plus facilement possible. Les données classiques deviennent donc peu à peu des collections d'objets (c'est-à-dire des données qui ont une définition sémantique propre, indépendante de l'utilisation qui en est faite) définies de manière intrinsèque par schématisation du monde réel suivant un modèle de description et des règles bien précises. Définis de manière statique dans la plupart des cas, les objets peuvent devenir plus dynamiques par l'introduction dans leur définition d'autres objets ou de procédures et de fonctions que l'on peut leur appliquer et qui permettent de modifier leur description. Le travail d'un programme d'application utilisant de tels objets est fortement modifié : on passe de la définition des données et des procédures (programmation classique) à l'utilisation d'objets et au choix des procédures offertes par le schéma de données sur ces objets (programmation objet).

5.2.2. L'objectif des SGBD

Tous les SGBD ne présentent pas les mêmes possibilités, et les modèles ont évolués depuis leur apparition il y a quelques dizaines d'années. Mais l'objectif général de tous les SGBD peut se résumer en plusieurs points [GAR 83] :

- garantir l'indépendance physique et logique entre les données et les programmes d'application,
- assurer la persistance des données,
- permettre une administration centralisée des données,
- gérer la mémoire de façon optimale et assurer l'efficacité de l'accès aux données,
- gérer le partage des données entre utilisateurs et les accès concurrents,
- assurer la fiabilité, l'intégrité et la cohérence des données, éviter les redondances,
- assurer la sécurité des données,

- assurer les interrogations interactives, la consultation déclarative, et l'accès à de non-informaticiens.

L'indépendance entre les données et les programmes d'application : les données et les programmes étant séparés, ce ne sont plus les applications qui sont chargées de structurer, d'organiser et de vérifier les données et de les stocker dans des fichiers. Les données sont définies et structurées par le SGBD qui offre cette structure aux applications. L'indépendance est à la fois logique (au niveau de la définition des données) et physique (au niveau du stockage et de la gestion de la mémoire).

Le SGBD assure la persistance des données : alors que les résultats d'une application sont liés à l'exécution de l'application, l'existence des données d'une base est assurée indépendamment de leur utilisation. Les données d'une base ont donc une durée de vie supérieure à la durée de vie du programme qui les crée ou qui les utilise.

L'administration centralisée des données : une des fonctions essentielles des SGBD est la définition des structures de données, la définition et l'organisation des structures de stockage, les modifications de ces structures, les contrôles de validité et de cohérence. Il est essentiel de centraliser ces fonctions : c'est l'administrateur du système qui gère l'ensemble des données, indépendamment de leur utilisation et des applications. Les usagers ou les applications sont déchargés de toute tâche d'administration, et peuvent utiliser les données de la base sans se préoccuper de l'administration de ces données tout en étant assurés de leur validité et leur cohérence. Un SGBD permet d'optimiser l'administration des données en centralisant ces tâches dans l'entreprise.

Gérer la mémoire de façon optimale : chaque SGBD optimise le stockage et l'accès aux données, en utilisant des techniques complexes inaccessibles aux programmes d'application eux-mêmes. En évitant les redondances, il optimise également le volume des données stockées. Avec un SGBD, on est donc sûr d'avoir accès à de volumineux ensembles de données de manière efficace.

Gérer le partage des données entre usagers : c'est le SGBD qui va se charger de gérer les accès concurrents, les mises à jour, le tout en fonction des droits de chacun. Chaque usager aura l'illusion d'être seul à utiliser les données de la base, qu'il verra assemblées et structurées comme il le souhaite.

Assurer la fiabilité, l'intégrité, la cohérence : la description du schéma des données contient un certain nombre de règles qui permettent de vérifier la fiabilité, l'intégrité, la cohérence de tous les objets présents dans la base (par exemple, l'âge d'une personne doit être compris entre 0 et 120; l'âge d'un enfant ne peut être

supérieur à celui de ses parents, etc.). Le SGBD contient des procédures spéciales permettant d'effectuer l'ensemble de ces tests. Par exemple, les contraintes d'intégrité sont des règles qui précisent la cohérence sémantique des données entre elles.

Sécurité des données : tous les utilisateurs d'une même base n'ont pas nécessairement les mêmes droits d'accès aux données. Il est utile de pouvoir définir de tels droits pour assurer la sécurité de certaines données confidentielles et réservées à certains groupes d'utilisateurs.

Assurer les interrogations interactives et l'accès aux non-informaticiens : les données de la base sont bien gérées et bien organisées. Il ne reste plus qu'à les consulter, objectif initial de toute la démarche, sans avoir à se préoccuper du stockage ou de l'organisation interne. Le SGBD offre un certain nombre de fonctions pour accéder aux données de façon déclarative, aussi bien pour les programmes d'applications que pour l'usager, sans avoir à effectuer le moindre développement informatique particulier.

L'architecture client-serveur est la plus efficace pour répondre aux objectifs d'indépendance physique et logique, de partage entre usagers, d'interrogation interactive, dans le cadre des réseaux et de l'informatique décentralisée. Le serveur gère toutes les transactions avec les clients et répond à leurs requêtes, et est dédié à ces tâches. Cette architecture permet à de nombreux clients d'accéder à partir de petites machines à de grandes bases de données, de manière très efficace.

5.2.3. La modélisation de la réalité dans les SGBD

Puisqu'il y a indépendance entre données et application, le SGBD doit également proposer un modèle pour créer et décrire des objets à partir de la vision de la réalité en collections d'objets, indépendamment de leur utilisation par les programmes ou les applications. Ce formalisme permet de définir la structure des données et offre un ensemble d'opérateurs destinés à manipuler ces données. De nombreux modèles ont été définis : hiérarchique, réseau, relationnel, objet.

5.2.3.1. Les différents modèles de données

Séparer données et applications nous oblige à passer du concept de données à celui d'objet, entité schématisant la réalité et ayant une définition sémantique intrinsèque. Comme nous l'avons déjà vu, il n'y a pas de vision universelle de la réalité lorsqu'il s'agit de définir des objets par leur contenu sémantique. C'est chaque domaine d'application qui détermine la modélisation du réel en objets de façon contextuelle, mais malgré tout d'une manière beaucoup plus globale que lorsqu'il s'agit d'utiliser des données pour une seule application. Le SGBD offre un formalisme de description et des utilitaires permettant d'aboutir à un ensemble

cohérent d'objets bien structurés, et propose un ensemble d'opérateurs destinés à manipuler ces objets. Chaque application du domaine concerné par cette vision du réel devra choisir ses données parmi les objets que lui propose le SGBD. C'est la différence de ces formalismes, ou modèles de données, qui fait la différence entre les SGBD.

Il ne s'agit pas de décrire des objets en particulier, mais les propriétés d'ensembles d'objets, puisque ce sont les collections d'objets qui nous intéressent et non plus les objets isolés. On appelle type d'objet un ensemble d'objets possédant des caractéristiques similaires et manipulables par des opérations identiques. Un objet sera donc vu ici comme un élément d'un ensemble d'objets.

Un modèle de description, c'est un ensemble de concepts et de règles de composition de ces concepts permettant de décrire des ensembles d'objets. Toute description s'effectue au niveau du type, à l'aide d'un ensemble d'éléments descriptifs permettant d'exprimer les propriétés d'ensembles d'objets et composant un modèle de description de données. Un modèle de description de données est mis en œuvre à l'aide d'un langage de description de données.

Si le modèle de données est simple, il sera facile à utiliser mais risque de trop simplifier la description du réel, ou de ne pas permettre cette description dans certains domaines, lorsque les structures adéquates et les opérateurs associés n'ont pas été prévus. S'il est complexe, les objectifs des SGBD seront difficiles à assurer, notamment en ce qui concerne la fiabilité, la cohérence, et la facilité d'utilisation par de non-informaticiens. De nombreux modèles de description ont été définis. Les principaux sont le modèle hiérarchique, le modèle réseau, le modèle relationnel et le modèle objet.

5.2.3.2. Les niveaux de description

Quelque soit le modèle de données, on distingue trois niveaux de description (fig. 5.1) :

- le niveau conceptuel, dont nous avons déjà longuement parlé au chapitre précédent, et qui correspond à la conception des objets, c'est-à-dire à la structure sémantique des données, sans aucun souci d'implantation en machine. On distingue plusieurs types de données pour décrire des objets : les types élémentaires, prenant leur valeurs dans un ensemble bien défini (N, Z, R, R2, R3, C, ...), les méthodes (fonctions ou plus généralement relation sur les types précédents), et les types complexes (prenant leurs valeurs dans des ensembles d'objets).

- le niveau interne, qui correspond à la structure de stockage supportant les données (fichiers, structures, organisation et chemin d'accès), en général inaccessible aux usagers, uniquement gérés par le SGBD,

- le niveau externe, correspondant aux descriptions du réel perçues par les applications ou groupes d'utilisateurs du domaine, et qui correspondent à des vues partielles des objets déduites du niveau conceptuel.

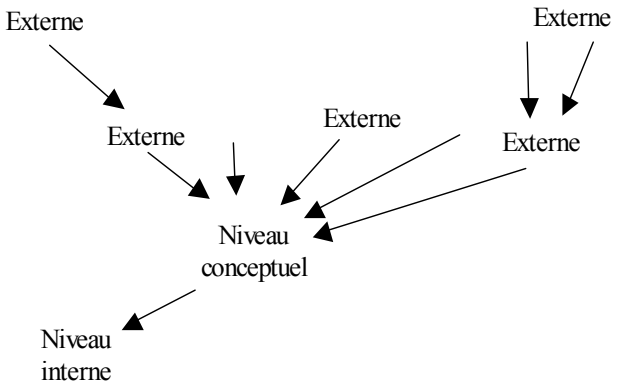


fig. 5.1 : les différents niveaux de description

C'est le SGBD qui fait le lien entre les différents niveaux de description : il a besoin pour cela de toutes les descriptions et des règles de correspondance entre ces descriptions, qui sont conservées dans ce que l'on appelle le dictionnaire des données. La définition des différents schémas est théoriquement assurée par un administrateur de données. Les schémas, les règles associées, la description sémantique sont stockées dans le dictionnaires des données, et qui peut lui-même être organisé comme une base de données.

Si l'on prend l'exemple du cadastre, les entités clairement identifiables sont : les parcelles, les constructions, les propriétés, les propriétaires, les transactions. Les transactions sont des associations entre propriétaires et propriétés, avec des attributs comme la date d'achat, le prix, etc.

Nous allons décrire rapidement les modèles hiérarchique et réseau, et plus en détail le modèle relationnel et le modèle objet. Nous verrons ensuite comment utiliser ces deux derniers modèles pour gérer et manipuler des objets géographiques.

5.2.4. Les modèles hiérarchique et réseau

Ces deux modèles datent des années soixante. Le modèle hiérarchique permet de décrire des liens hiérarchiques entre objets : le schéma entre objets est construit

uniquement grâce à ces liens hiérarchiques. Dans le modèle réseau, on peut établir des liens entre objets, dans tous les sens. Dans ces deux modèles, l'indépendance entre structure logique et structure physique n'est pas vraiment respectée, et les possibilités des structures logiques sont calquées sur les problèmes d'implantation physique et d'accès aux données. Les structures logiques et les relations entre objets ne sont que la représentation logique de pointeurs entre objets (physiquement des tables d'index reliant les enregistrements de différents fichiers), et tout le formalisme des modèles repose sur la navigation dans la base grâce à ces pointeurs : le formalisme d'interrogation (d'ailleurs souvent complexe) repose sur la connaissance de l'existence de ces pointeurs et donc sur des ordres élémentaires de navigation, et non sur un langage de plus haut niveau permettant de s'affranchir des liens physiques entre objets. Les objectifs des SGBD, au niveau de l'intégrité et de la cohérence des données, et de l'indépendance physique et logique entre données et programmes d'applications, ne sont pas assurés par ce modèle.

5.2.5. Le modèle relationnel

5.2.5.1. Les principes du relationnel

Le modèle relationnel a été introduit dans les années soixante-dix pour avoir une totale indépendance entre les structures logiques et physiques, ce qui n'était pas le cas avec les modèles antérieurs (hiérarchique et réseau). Il utilise un modèle de description très simple, mais rigoureusement défini et permettant ainsi de remplir un bon nombre des objectifs des SGBD. Les objets sont décrits par des attributs *simples*, qui prennent chacun leurs valeurs dans un ensemble de valeurs appelé domaine. Les attributs sont simples : les domaines sont soit des ensembles finis, soit des valeurs entières ou réelles (N, Z ou R). Un type d'objet peut être décrit par des attributs, et est appelé une relation.

Puisque chaque attribut est de type simple, les objets d'une relation peuvent être représentés par une table donnant les valeurs des attributs pour chaque objet (appelé alors *tuple*). Les opérateurs de manipulation des objets n'utilisent que des opérations liées aux domaines des attributs : soit des opérations ensemblistes (pour les domaines finis), soit des opérations liées à l'ordre naturel dans N, Z ou R.

Le terme de relation vient de la définition mathématique du modèle : les objets d'un type d'objets correspondent à un sous-ensemble du produit cartésien d'un ensemble de domaines, ce qui peut être vu comme une relation entre ces ensembles.

Le modèle de description est donc très simple : pas de types complexes, pas de définition récursive, pas de méthodes ou fonctions. Tous les attributs prennent leurs valeurs dans des ensembles finis ou ordonnés de dimension 1. Les opérateurs associés pour valider ou manipuler les *tuples* vont tous être développés sur cette

simplicité de description. Le succès du relationnel vient en grande partie de cette simplicité, qui a permis la mise en œuvre effective de la plupart des objectifs des SGBD.

Les opérateurs liés au formalisme de description sont également simples. Ils se divisent en deux grandes familles : les opérateurs de description (théorie de la normalisation), et les opérateurs de manipulation (algèbre relationnelle). L'inconvénient majeur du relationnel, c'est qu'il est difficile sinon impossible de modéliser des objets complexes : les objets géographiques en fournissent de nombreux exemples (par exemple, aucun formalisme ne permet d'utiliser une distance pour mettre en relation deux objets).

5.2.5.2. La notion de clé et la théorie de la normalisation

Une mauvaise perception de la réalité, une mauvaise conception des entités et associations conduit à des relations qui souffrent d'anomalies et présentent le plus souvent des redondances. Ces redondances conduisent à des risques d'incohérences lors des mises à jour. Pour éviter cette mauvaise perception, le modèle relationnel propose un certain nombre de notions qui permettent d'aboutir, par *décomposition* à partir de l'ensemble des attributs, à des relations ne souffrant pas d'anomalies. Il est nécessaire pour bien définir les relations d'étudier les propriétés sémantiques des attributs et de définir les dépendances entre attributs qui résultent de ces propriétés.

Ces dépendances sont classées en trois types :

- les dépendances fonctionnelles : un attribut b dépend fonctionnellement d'un attribut a si à toute valeur de a correspond une valeur unique de b :

$A \rightarrow B \Leftrightarrow ((xy \in R \text{ et } xy' \in R) \rightarrow y = y')$

On introduit la notion fondamentale de clé à partir de la notion de dépendance fonctionnelle : une clé est un ensemble minimum d'attributs qui détermine tous les autres (tous les attributs sont en dépendance fonctionnelle avec la clé).

- les dépendances multivaluées qui caractérisent l'indépendance de deux ensembles d'attributs corrélés à un même troisième : b dépend de a si toute valeur de a détermine un ensemble de valeurs de b indépendamment des autres attributs de la relation :

$A \rightarrow\rightarrow B \Leftrightarrow ((xyz \in R \text{ et } xy'z' \in R) \rightarrow (xy'z \in R \text{ et } xyz' \in R))$

- les dépendances de jointures qui permettent de décrire les relations entre sous-ensembles d'attributs d'une relation.

A partir de l'ensemble des attributs et de leurs dépendances, des algorithmes pourront déterminer les entités et les associations canoniques, et fournir des relations qui ne souffrent pas d'anomalies : cette opération se fera par décomposition successive jusqu'à l'obtention de relations dites normalisées.

La théorie de la normalisation définit cinq formes normales pour les relations :

- une relation est en première forme normale si tout attribut contient une valeur atomique.

- une relation est en deuxième forme normale si elle est en première forme et si tout attribut n'appartenant pas à une clé ne dépend pas que d'une partie de cette clé.

- une relation est en troisième forme normale si elle est en deuxième forme et si tout attribut n'appartenant pas à une clé ne dépend pas d'un attribut non clé.

- une relation est en quatrième forme normale si et seulement si les seules dépendances multivaluées élémentaires sont celles dans lesquelles une clé détermine un attribut.

- une relation est en cinquième forme normale si et seulement si toute dépendance de jointure est impliquée par les clés candidates de la relation.

La théorie de la normalisation est peu appliquée en pratique dans les SIG, alors que la complexité des objets engendre souvent une mauvaise perception du réel et une mauvaise conception des relations.

5.2.5.3. L'algèbre relationnelle

Une fois la base de données constituée, il faut pouvoir l'interroger et la modifier grâce à un langage, lui-même basé sur un ensemble d'opérations élémentaires. Dans le cas du modèle relationnel, ces opérations formelles s'appliquent aux relations et à leur attributs. Les opérations de base sont :

- l'union de deux relations A et B de même schéma : la relation résultante est de même schéma que A et B et a pour *tuples* l'union des *tuples* de A et B.

- la différence de deux relations A et B de même schéma : la relation résultante est de même schéma que A et B et a pour *tuples* la différence ensembliste des *tuples* de A et B.

- le produit cartésien de deux relations de schémas quelconques : la relation résultante a pour schéma la concaténation des schémas de A et de B et pour tuples le produit cartésien des deux ensembles de *tuples* de A et B. Cette opération comme on peut s'en douter, est des plus coûteuses.

- la projection d'une relation sur un certain nombre de ses attributs : la relation résultante a pour schéma les attributs sur lesquels la projection est faite, les *tuples* étant obtenus par élimination des valeurs d'attributs n'appartenant pas au schéma résultant, ainsi que par élimination des *tuples* en double.

- la restriction d'une relation A par une qualification Q : la relation résultante est de même schéma et a pour *tuples* les *tuples* de A qui satisfont la qualification Q.

- la jointure de deux relations A et B selon une qualification Q : la relation résultante est la restriction du produit cartésien de A et B sur la qualification Q. Cette opération est fondamentale, car elle permet de relier deux relations sur un ou plusieurs de leurs attributs et de créer une troisième relation résultant du croisement.

Il est important de noter que la jointure ne conserve pas la notion de clé : si A1 est clé de la relation R1 et A2 clé de la relation R2, A1 ne sera vraisemblablement plus clé de la relation R3 résultat d'une jointure entre R1 et R2. L'ensemble (A1,A2) sera une clé de la relation R3 s'il fait partie de cette nouvelle relation.

Il est possible de répondre à la plupart des questions que l'on peut poser à une base de données relationnelle par un enchaînement d'opérations de base, que l'on appelle alors une requête. Les interrogations d'une base de données relationnelle peuvent être exprimées directement en terme d'opérations relationnelles, ou grâce à des langages d'interrogations basés sur la vérification de formules dont les variables sont soit des *tuples*, soit des valeurs d'attributs.

5.2.5.4. Puissance et limite du modèle relationnel

La puissance du modèle vient surtout de la rigueur de ses concepts et de leur simplicité, deux choses qui permettent une véritable mise en œuvre des objectifs assignés aux SGBD, alors que c'est difficilement réalisable sur les modèles hiérarchique et réseau, et même orienté objet.

Les limites du modèle relationnel sont inhérentes à sa définition même : il ne concerne que des types d'attributs simples. Si les objets ne peuvent être décrits à l'aide d'attributs de type simple, le modèle fonctionne mal ou ne fonctionne pas du tout. Par exemple, il n'est pas prévu de décrire des images ou d'utiliser des données bidimensionnelles; il n'est prévu aucun opérateur d'interrogation lié à une distance entre deux objets.

Le modèle gère mal la hiérarchie, ou alors au prix de coûteuses jointures. Il ne propose aucune méthode sur les données, laissant aux applications le soin d'utiliser le contenu sémantique des relations.

5.2.6. Le modèle objet

Le modèle objet prend justement la démarche inverse. La description est la plus complexe possible, quitte à abandonner les objectifs principaux des SGBD, sauf celui d'indépendance entre structure physique et structure logique [BER 93] [DEL 91].

5.2.6.1. L'approche objet : données et méthodes, encapsulation et héritage

Prenant une démarche inverse à la décomposition en relation disjointe, l'approche objet tente de représenter de façon naturelle des objets dans toute leur complexité, aussi bien descriptive que procédurale. Un objet permet de regrouper sous un même élément une structure de description (des attributs) et un comportement (sa façon d'agir ou de réagir, décrit par des méthodes). Les attributs

peuvent être élémentaires, c'est-à-dire d'un type simple (entier, réel, date, etc.), mais peuvent également être d'autres objets. L'un des intérêts majeurs de l'approche objet consiste dans sa capacité à décrire et gérer les entités du monde réel en un formalisme hiérarchique (des attributs sont eux-mêmes des objets), et en associant données et procédures.

La notion de classe correspond au schéma d'un objet (description des attributs et des méthodes). Un objet est alors considéré comme l'instance d'une classe.

Les systèmes à objets implémentent la notion d'*identité d'objet*, qui permet de d'identifier un objet indépendamment des valeurs de ses attributs. Cet identificateur interne est unique pour chaque objet, il permet de séparer l'existence de l'objet de la valeur de ses attributs. Cette notion est importante dans la gestion des attributs complexes, puisque la valeur de l'attribut d'une classe peut être un objet d'une autre classe, objet qui doit donc pouvoir être pointé indépendamment de ses valeurs qui, elles, peuvent changer. Nous retrouverons rapidement cette notion dans les SIG, qui ont souvent besoin d'identifier un objet indépendamment de la valeur de ses attributs.

L'*encapsulation* est le concept permettant de regrouper dans une même classe des attributs et des méthodes, et de ne permettre la manipulation d'un objet que par les méthodes définies dans la classe à laquelle il appartient. L'encapsulation induit donc deux niveaux de perception et de représentation d'un objet : le niveau externe, accessible par les utilisateurs, et le niveau interne, accessible uniquement par les méthodes. Elle garantit en principe l'intégrité des objets, qui ne peuvent être manipulés par des éléments extérieurs que via des méthodes que la classe propose.

L'*héritage* est également un concept important implémenté par le modèle objet : il permet de modéliser la hiérarchie de classes. Une classe dérivée d'une autre hérite de l'ensemble de ses attributs et de ses méthodes. L'héritage peut être simple (une classe dérive d'une seule classe) ou multiple (une classe dérive de plusieurs classes, et regroupe données et méthodes de plusieurs classes).

L'utilisation du modèle orienté objet s'étend à des domaines très variés, s'appliquant aux langages de programmation, à la représentation des connaissances, aux modèles et bases de données.

5.2.6.2. les SGBD objets

Les systèmes de gestion de bases de données orienté objet tentent de combiner les possibilités des SGBD classiques et les avantages de la modélisation objet, très riche dans l'aspect procédural et plus proche du réel par l'introduction de types complexes pour les attributs. Un SGBDOO est construit pour gérer des collections d'objets, suivant un schéma de données correspondant à la définition de classes

d'objets. Un SGBDOO regroupe donc des caractéristiques des SGBD (persistance des objets, sécurité des accès, etc.) et les avantages de l'approche objet (identité d'objet, encapsulation, objet complexe, héritage, etc.). Mais l'introduction de certaines de ces caractéristiques sont peu compatibles avec les objectifs des SGBD tels que nous les avions exprimés au début de ce chapitre :

- le comportement des objets n'est plus du seul ressort des programmes d'applications. Le SGBD redevient ainsi peu à peu un véritable programme d'application, même si la persistance des objets et gestion des collections est assurée indépendamment du schéma des objets. Il n'y a plus vraiment indépendance entre les données et les programmes d'applications, puisque les objets ne peuvent être manipulés qu'au travers de leurs méthodes;

- l'intégrité et la cohérence des données devient difficile à assurer dans un modèle ou des attributs font références à d'autres objets complexes.

Malgré ces inconvénients, les SGBDOO représentent une avancée conceptuelle importante, même si les systèmes relationnels restent de loin les systèmes les plus employés aujourd'hui. Une approche hybride d'extension du relationnel par certains concepts objet peut être très bénéfique. C'est l'approche que nous développerons dans la suite de ce chapitre, avec dans un premier temps l'introduction de types génériques d'objets et d'attributs complexes.

Les SIG introduisent naturellement la notion d'objet, car l'utilisation du modèle cartographique pour la modélisation des données géographiques introduit des types différents pour les objets. Chaque classe de base possède des méthodes qui lui sont propres (par exemple, la surface ou le périmètre pour le type zone, le plus court chemin dans un réseau pour le type ligne, etc.). Le modèle objet ne permet pas d'assurer les contraintes d'intégrité comme le fait le modèle relationnel, mais il permet de répondre à une modélisation plus complexe en introduisant des types d'objet et des attributs complexes : il correspond mieux à la nécessité de modélisation de l'information géographique, et semblerait mieux adapté aux différents problèmes de modélisation que nous avons rencontrés au chapitre 3.

5.3. SGBD et objets géographiques : l'extension du modèle relationnel

Intégrer l'attribut de localisation dans le schéma relationnel permettrait de bénéficier sur cet attribut de toute la puissance du modèle, de simplifier les opérations de manipulation et les interrogations utilisant la localisation dans l'espace en mettant ces opérations au même niveau que les opérations sur les relations non localisées.

5.3.1. Objectifs

Nous allons donc étendre le modèle relationnel aux attributs prenant leurs valeurs dans un espace de dimension deux ou trois, et aux objets représentant soit un élément de cet espace, soit un ensemble d'éléments (sous-ensembles de dimension 0, 1 ou 2). Nous allons également compléter l'algèbre relationnelle par de nouvelles opérations algébriques liées au caractère multidimensionnel de l'attribut de localisation :

- la restriction spatiale, qui correspond à la sélection d'objets par rapport à leur localisation
- la jointure spatiale, qui correspond à la mise en relation de deux objets par rapport à leur localisations respectives
- la projection spatiale, qui correspond à l'opération d'impression de l'attribut de localisation et qui devient ici une opération de cartographie

Ces nouvelles opérations, et surtout les deux premières, vont permettre de manipuler la localisation grâce à des opérations algébriques et logiques : c'est dans ce sens que la localisation, considérée habituellement comme un résultat graphique, va devenir également le résultat intermédiaire d'une séquence d'opérations de gestion et de manipulation de données qui peut très bien ne pas avoir de résultats finals graphiques.

Le caractère relativement universel de la localisation permet d'envisager de nombreuses mises en relation d'objets par opération une jointure spatiale en relations localisées, et ceci indépendamment du codage ou de la méthode de représentation de la localisation. La seule restriction, et elle est de taille, provient de la précision sémantique liée à cette localisation, et de la modélisation de la réalité, comme nous l'avons vu au chapitre 3.

L'intégration de la localisation dans le schéma relationnel est donc très riche, aussi bien par la puissance de manipulation qu'elle apporte que par les contraintes et les problèmes de représentation et de validité de l'information localisée qu'elle met en évidence.

5.3.2. L'extension du modèle relationnel

5.3.2.1. Les principes de l'extension du modèle

Le modèle relationnel ne concerne que des types d'attributs simples. Ces attributs prennent leurs valeurs dans un ensemble fini ou ordonné de dimension un. Les opérations classiques de l'algèbre relationnelle sont limitées aux attributs de type simple, et sont liées à la structure d'ordre canonique des ensembles de définition.

La localisation spatiale ne rentre pas dans le cadre des attributs simples, puisque cet attribut prend ses valeurs dans un espace de dimension 2 ou 3. Il faut donc étendre le modèle relationnel classique pour qu'il puisse gérer des objets localisés en en conservant la structure multidimensionnelle [SOU 86].

Il n'existe pas de structure d'ordre *canonique* dans un espace de dimension supérieure à 1 qui puisse être utilisée dans la résolution de problèmes spatiaux liés à la proximité ou la distance : la structure d'ordre canonique de la dimension 1 doit être remplacée par une structure topologique ou une structure métrique. Les opérations de l'algèbre relationnelle liées à l'attribut de localisation ne sont plus définies dans un espace à une dimension, mais à deux ou trois. Les relations entre objets ne s'expriment donc plus à partir d'une relation d'ordre, mais à partir d'une topologie ou d'une distance. Ainsi, un domaine de restriction ne sera plus une valeur ou un intervalle, mais un domaine de l'espace métrique. Un critère de jointure géographique ne sera plus une relation entre les valeurs des objets eux-mêmes, mais une relation ensembliste ou une relation entre les valeurs des distances entre les objets [SOU 86].

D'une manière classique, la localisation des objets géographiques s'exprime soit par des éléments de l'espace (pour les objets dont la localisation peut être ramenée à un point), soit par des sous-ensembles de l'espace (lignes ou zones). L'extension du modèle relationnel ne concerne donc pas seulement le domaine de définition des attributs (l'espace de dimension 2 ou 3) : il introduit également des types dans la définition des relations, se rapprochant ainsi du modèle objet. Les contraintes d'intégrité ne seront pas les mêmes suivant le type des objets. Nous leur consacrons un chapitre spécial.

La schématisation géométrique de la localisation des objets géographiques en zone, ligne, point, pixel établit une classification naturelle sur laquelle repose à la fois l'extension du modèle relationnel et la définition de classes d'objets géographiques pour le modèle objet. Cette classification est très discriminante, elle sera donc présente très rapidement comme membre des objets géographiques. Pour l'extension du relationnel, elle intervient au niveau de la définition des relations, puisque d'elle dépend la mise en œuvre des opérations de l'algèbre relationnelle. Pour le modèle objet, peu de classes d'objets non localisés peuvent prétendre encapsuler des classes d'objets géographiques localisés.

Toutes les considérations ultérieures seront donc basées sur la classification suivante, que nous avons déjà vue au chapitre précédent et qui permet de schématiser la géométrie des objets géographiques et de définir des collections d'objet du même type :

– les objets point, dont la localisation est un point de l'espace,

– les objets ligne, dont la localisation est un sous-ensemble de dimension 1 de l'espace, en général schématisée par un pour plusieurs arcs (ligne brisée entre des points),

– les objets zone, dont la localisation est un sous-ensemble de dimension 2 de l'espace, en général défini par un ensemble d'arc (schématisant le contour de la zone),

– les objets pixel, dont la localisation est un sous-ensemble de dimension 2 de l'espace définie par un point (tous les pixels ont une géométrie identique - maille carrée, hexagonale, etc.).

Un arc n'est pas un objet géographique, mais seulement un objet géométrique. Les contraintes d'intégrité géométriques seront exprimées sur les points et les arcs, c'est-à-dire sur les objets géométriques qui permettent de définir la localisation des objets géographiques.

5.3.2.2. Relations localisées et notion de clé géographique

Rappelons la définition d'une clé dans le cas d'un attribut de dimension 1 : une clé est un ensemble minimum d'attributs qui détermine tous les autres. On peut donc introduire la notion de clé géographique, lorsque, dans une relation contenant un attribut de dimension deux, cet attribut de localisation détermine la valeur de tous les autres attributs descriptifs.

En fait, on érige cette dépendance en contrainte : par définition, une relation est dite localisée si et seulement si l'attribut de localisation est une clé pour les attributs descriptifs. Dans une relation localisée, on a donc l'assurance que deux objets distincts ne partagent pas le même espace : il y a unicité d'un phénomène en un lieu. Les objets d'une relation localisée sont appelés objets localisés. Seuls les objets localisés contiennent une description explicite de leur localisation (telle que nous l'avons vue au chapitre 3). Lorsque la localisation n'est pas une clé géographique (donc dans une relation non localisée), elle peut être ramenée à un attribut descriptif classique (nominal, de dimension un) faisant référence à la localisation réelle.

Par exemple, une relation décrivant des départements, avec la localisation comme clé géographique, va contenir un attribut de localisation qui sera décrit de façon interne par des points et des arcs. Cette relation est localisée, et les objets sont des objets localisés. Chaque département a un nom : cet attribut descriptif nominal est également clé de la relation. Une relation décrivant des personnes, avec un attribut donnant le département de naissance, n'est pas une relation localisée, même si elle contient un attribut donnant une localisation (le département), car cette localisation n'est pas une clé de la relation. D'ailleurs, dans ce cas, la localisation ne va pas utiliser des points et des arcs, mais bien un attribut descriptif classique donnant le nom du département. On pourra retrouver la description de la localisation

(c'est-à-dire l'attribut spatial de localisation, et son expression en points et arcs) en utilisant une semi-jointure classique sur cet attribut descriptif.

La notion de clé géographique est très importante : elle est à la base de l'extension du modèle relationnel que nous proposons. En effet, il ne serait pas possible de bien gérer un modèle dans lequel la description de la localisation des objets serait donnée quelle que soit l'attribut (clé ou non).

5.3.3. Les extensions de l'algèbre relationnelle

Toutes les opérations que nous allons ajouter à l'algèbre relationnelle portent sur l'expression de la localisation dans l'espace, et donc intrinsèquement sur le même attribut. Si l'on dispose d'une référence unique pour l'attribut de localisation, et pour chaque relation de la description de la localisation des objets par rapport à cette référence, tout objet pourra être mis en relation avec n'importe quel autre objet ainsi localisé, indépendamment d'un système de référence attaché à telle ou telle entité.

Le système de référence unique est bien sûr l'espace mathématique euclidien : l'attribut de localisation est alors de dimension deux ou trois, sans relation d'ordre canonique (liée à la métrique définissant la structure de l'espace). La qualification des opérations de restriction ou de jointure porte sur un attribut de dimension deux ou trois. Elle n'est plus liée à une structure d'ordre mais à la structure métrique de l'espace euclidien. Quel que soit le type de représentation utilisée pour la localisation, il va falloir formellement passer de nouveau par l'espace euclidien pour mettre les objets en relation sur leur localisation : il s'agit maintenant de repasser de l'espace de représentation à l'espace euclidien, de l'entité dérivée à l'entité initiale, à l'inverse de la démarche exposée dans le chapitre précédent.

L'attribut formel de localisation sera noté *loc* dans la suite, et sa valeur représente l'ensemble des points de l'espace euclidien attaché à un objet. Il est, de par son caractère formel, indépendant du type de représentation utilisé pour la localisation des objets dans la base et des méthodes effectives de réalisation des opérations algébriques. Cet attribut formel, associé s'il y a lieu à un attribut temporel, détermine l'ensemble des attributs descriptifs de la relation : la localisation spatio-temporelle constitue toujours une clé d'une relation localisée.

Nous allons compléter l'algèbre relationnelle par de nouvelles opérations algébriques concernant l'attribut multidimensionnel de localisation : la restriction spatiale, la jointure spatiale, la projection spatiale.

5.3.3.1. La restriction spatiale

C'est l'opération correspondant à la sélection des objets par rapport à un domaine défini de l'espace, noté D. Ce domaine peut être défini soit directement (fenêtre d'étude, domaine défini par l'utilisateur en coordonnées ou sur un écran, etc.), soit par rapport aux objets d'une autre relation : zone autour d'un point, espace autour d'une ligne ou d'une zone. Nous verrons plus tard comment le concept de *masque* permet de manipuler facilement ces domaines de restriction.

Plusieurs types d'opération de restriction spatiale peuvent être envisagés quand les objets sont des ensembles de points, en prenant une qualification ensembliste plutôt que métrique. La qualification peut utiliser l'ensemble des points de l'objet (inclusion, intersection), ou un point particulier de l'objet (appartenance). Le point particulier de l'objet est appelé centroïde : il peut être choisi manuellement lors de la définition géométrique de l'objet, ou automatiquement à partir de la géométrie de l'objet. Si A désigne un objet et D le domaine de restriction, on peut ainsi définir les différentes opérations de restriction spatiale :

- $A \subset D$ l'objet A est inclus dans le domaine D
- $A \cap D \neq \varnothing$ l'objet A a une intersection non vide avec le domaine D
- $A \cap D = \varnothing$ restriction par exclusion, si A et D sont disjoints
- $a \in D$ le centroïde de A appartient à D

Contrairement à la restriction classique, la restriction spatiale peut modifier les objets lorsqu'il s'agit d'une qualification ensembliste concernant l'intersection. En effet, le résultat de la restriction peut être l'objet lui-même (pas de modification de la localisation) ou l'intersection de l'objet et du domaine de restriction. Dans ce cas, il y a modification de l'attribut de localisation. Par contre, il n'y a pas modification du type d'objet (une zone reste une zone, une ligne reste une ligne). Les attributs descriptifs ne sont jamais modifiés par une opération de restriction.

5.3.3.2. La jointure spatiale

De toutes les opérations algébriques, la jointure est certes la plus importante en pratique, car, comme nous l'avons souligné, c'est elle qui permet la mise en relation de deux *tuples* par le biais d'un attribut commun, en créant ainsi un nouvel objet ayant les caractéristiques des deux objets répondant à la qualification de jointure, et correspondant dans la nouvelle relation à un *tuple* formé de valeurs d'attributs des deux *tuples* mis en relation.

La jointure spatiale est donc l'opération de jointure portant sur l'attribut de localisation. La qualification de jointure s'exprime soit par rapport à la distance entre deux objets :

- $a1 \leq d(O1,O2) \leq a2$

soit par rapport à une relation ensembliste entre les objets :

- $O1 \cap O2 = \varnothing$
- $O1 \cap O2 \neq \varnothing$
- $O1 \subset O2$
- $O1 \not\subset O2$

Par exemple, a2 = 0 (d(O1,O2) = 0) signifie que deux points de l'espace sont mis en relation s'ils ont la même localisation . On parle alors d'équijointure géométrique.

Pour bien définir cette opération, il est nécessaire de revenir à l'espace géométrique de référence et de repasser du type de représentation des objets en zones, lignes, points aux objets élémentaires que sont les points de l'espace euclidien avec leur localisation intrinsèque. L'opération de jointure spatiale s'effectue, en théorie, en considérant que les seuls objets logiques sont les points de l'espace. Le résultat de l'opération de jointure pourra à nouveau faire l'objet d'un changement de représentation, du type passage entité principale-entité dérivée. Le cheminement théorique d'une jointure spatiale est donc :

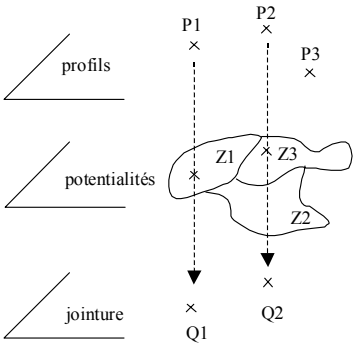
Espace de représentation
↓
Espace euclidien
↓
Jointure sur les points de l'espace euclidien
↓
Espace de représentation (si nécessaire).

La jointure spatiale crée de nouveaux objets. La description de la localisation de ces objets pose les mêmes problèmes que la description de la localisation des objets initiaux : les nouveaux objets sont des points ou des ensembles de points de l'espace euclidien, qu'il est nécessaire de regrouper et de décrire d'une manière ensembliste avec les choix possibles que nous avons vus au chapitre 3. Bien sûr, les méthodes de réalisation de la jointure et de la description de la localisation des résultats seront fonction du type de description des objets initiaux.

La jointure spatiale est une opération très intéressante dans la mesure où elle permet de comparer des objets sur leur localisation, sans avoir à spécifier et décrire cette localisation. Le résultat peut d'ailleurs être considéré d'avantage au niveau de cette mise en relation, plutôt qu'au niveau du résultat graphique formé par les *tuples* de la nouvelle relation. Ainsi par exemple, on pourrait mettre en relation la relation

Parcelle (numéro, nom de propriétaire, nombre de construction,…) et la relation Bornes d'incendie (numéro, débit) sur la qualification d(O1,O2) < 100 mètres pour avoir la liste des numéros de parcelles et le nombre de constructions associées à une borne plutôt qu'une carte des parcelles ou portions de parcelles et bornes résultant de la jointure.

Les jointure du type d(01,O2) = 0 sont appelées des équi-jointures géométriques. Si la qualification de jointure est d(O1,O2) < a, les résultat est plus complexe puisqu'il correspond à un ensemble d'objets pour lesquels la localisation ne constitue plus une clé : un point peut apparaître plusieurs fois avec des descripteurs différents s'il répond à la qualification de jointure pour plusieurs objets de chaque relation. La notion de clé de localisation n'est pas conservée par la jointure spatiale, comme c'est d'ailleurs le cas pour le résultat d'une jointure classique. Dans ce cas, le résultat sera une relation non localisée, et le résultat sera impossible à représenter cartographiquement car nous n'avons plus de dépendance fonctionnelle des attributs descriptifs par rapport à la localisation : on peut avoir des *tuples* formés du même ensemble de points mais ayant des valeurs descriptives différentes.



	couleur	Ph
P1	rouge	7.5
P2	jaune	5.8
P3	ocre	6.5

	pente	pierrosité	salinité
Z1	10-20	faible	faible
Z2	20-30	moyenne	très faible
Z3	10-20	très faible	moyenne

Le résultat de la jointure est composé des points Q1 et Q2 :

	couleur	Ph	pente	pierrosité	salinité
Q1	rouge	7.5	10-20	faible	faible
Q2	jaune	5.8	10-20	très faible	moyenne

fig. 5.2 : jointure entre zones et points sur la qualification d(O1,O2)=0

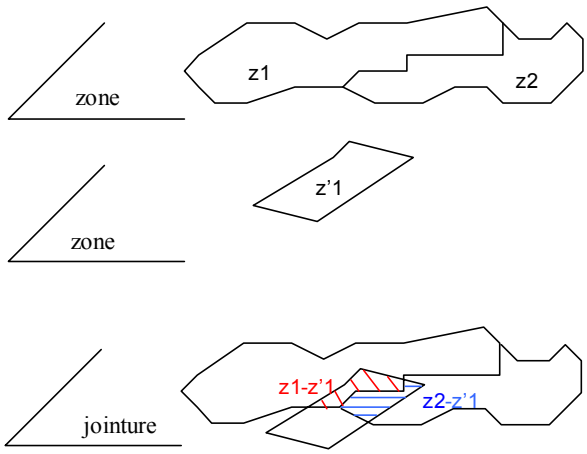


fig. 5.3 : jointure entre zones sur la qualification d(O1,O2)=0

Des exemples de jointures entre relations de type zone sont donnés par les figures 5.2 et 5.3. La qualification $d(O1,O2) = 0$ correspond à l'intersection des zones et conserve la localisation comme clé de la relation résultante. La qualification $a < d(O1,O2) < b$, avec a non nul, donne un résultat plus complexe et difficile à représenter graphiquement dans son intégralité, puisque la localisation n'est plus une clé de la relation résultante.

La jointure spatiale est une opération dangereuse si les objets mis en relation n'ont pas la même échelle de validité, ou si la validité de la localisation par rapport aux données descriptives est trop faibles pour permettre le passage de l'espace de représentation à l'espace euclidien (pour des données à caractère statistique par exemple, ou n'ayant qu'un intérêt de représentation graphique). Ainsi, des logiciels proposent des opérations graphiques pour éliminer les problèmes résultant de différentes précisions sur les collections d'objets mis en relation (élimination de petites zones par exemple). Nous présenterons au chapitre 8 d'autres opérations de mise en relation d'objet, basées sur la jointure spatiale, mais permettant de prendre en compte ces problèmes de précision et de gérer les transferts d'échelle notamment avec l'introduction de procédures d'agrégation spatiale.

5.3.3.3. La projection spatiale

La projection spatiale correspond à l'opération de projection d'attributs de l'algèbre relationnelle classique (elle n'a donc rien à voir avec les projections géographiques de type UTM ou Mercator…). Projeter l'attribut de localisation correspond donc à une opération de cartographie, exactement comme une projection

d'attributs descriptifs a pour objet la représentation en liste des valeurs de ces attributs. Si l'objectif d'une requête est purement graphique, on peut donc présenter l'opération de cartographie comme une projection de l'attribut de localisation, dans le cadre de l'algèbre relationnelle étendue à la localisation. Nous verrons au chapitre 9 les nombreuses possibilités de représentation graphique liées à cette opération.

5.3.3.4 Masques et semi-jointures géométriques

Certaines opérations que nous avons définies comme des opérations de restriction sont en fait des jointures : la restriction d'une relation sur la localisation des objets d'une autre relation est en fait une semi-jointure avec la qualification $d(O1,O2) = 0$: on connaît a priori l'ensemble des valeurs de l'attribut de qualification, représenté par le domaine *D*. Mais l'intérêt de cette opération réside plus dans la restriction apportée à l'une des relations que dans la mise en relation effective des attributs des deux relations. C'est pour cela que nous l'avons classée comme restriction spatiale :

Sélection dans la relation R1
↓
Définition d'un domaine *D* à partir des objets de la relation R2
↓
Jointure de R1 et R2 par rapport à *D*
↓
Projection de la jointure sur les attributs de R1

Ces semi-jointures sont intéressantes dans la mesure où elles ne font pas perdre la clé de localisation : on conserve donc, comme dans toute restriction, la possibilité d'utiliser la localisation dans une opération ultérieure ou de cartographier le résultat.

La définition d'un domaine de l'espace à partir des objets d'une relation prend donc un caractère particulièrement intéressant. C'est pour cela que nous avons nommé *masque* ce type de domaine. Un masque est donc un domaine de l'espace qui n'intervient dans le processus relationnel que par sa localisation.

Des opérations ensemblistes, métriques ou morphologiques peuvent être appliquées à ces domaines de l'espace : union, intersection, différence, inverse, dilatation, érosion, etc. En utilisant ces opérations, on peut créer des domaines de l'espace permettant de résoudre de nombreuses questions faisant intervenir une restriction spatiale, comme par exemple : sélectionner les objets d'une relation se trouvant à telle distance des objets d'une autre relation, sélectionner les objets d'une relation se trouvant dans une zone de caractéristiques données, etc.

5.3.4. Objets spatiaux et opérations relationnelles classiques

Les opérations de l'algèbre relationnelle classique peuvent bien sûr être utilisées sur les attributs descriptifs dans le cadre du modèle relationnel étendu à la localisation. L'incidence de ces opérations est néanmoins importante sur la conservation de la clé de localisation, les problèmes étant posés essentiellement par les objets résultant d'une jointure classique.

La restriction ou la projection sur un attribut descriptif n'a pas d'incidence directe sur l'attribut de localisation. En éliminant des objets, la restriction peut néanmoins modifier des caractéristiques géométriques de la relation résultante (connexité, genre, etc.).

La jointure classique de l'algèbre relationnelle permet de mettre en relation les objets de deux relations différentes de manière à créer une nouvelle collection regroupant les objets répondant au critère de jointure. Lorsque les objets sont localisés, ils sont regroupés dans des relations où tous les objets sont de même type géographique. La jointure classique va donc être difficile à mettre en œuvre entre deux relations localisées sans perdre la localisation du résultat. Si les deux relations ne sont pas du même type, la localisation des objets répondant au critère de jointure ne peut être conservée, car elle ne peut être exprimée par les méthodes que nous avons exposées plus haut et ne rentre pas dans le modèle relationnel étendu. On pourrait décider, pour les objets résultant de la jointure, de ne conserver que la localisation provenant d'une des deux relations, mais on risque alors malgré tout de perdre le principe d'unicité de clé pour cet attribut. Lorsque les deux relations sont du même type, on rencontre le même problème, la localisation risque de perdre son caractère de clé, car les objets résultant de la jointure pourraient partager le même espace géographique, comme c'est le cas pour la jointure spatiale lorsque la qualification de jointure n'est pas $d(O1,O2) = 0$, et la relation résultante ne peut donc être considérée comme une relation localisée et devient donc une relation d'objets non localisés. Pour éviter la perte de la clé de localisation, il est courant d'effectuer une opération d'agrégation dans une des relations d'origine après une jointure classique. L'agrégation doit se faire sur la clé de localisation, et correspond à la suppression des doublons sur la localisation. Nous reviendrons en détail sur ce type d'opérations dans le chapitre 8.

5.4. La structuration et l'indexation de l'attribut de localisation

La structuration des données géographiques en vue d'améliorer les opérations de recherche en mémoire de stockage est un problème général qui concerne l'ensemble des données graphiques de la base, quels que soient les modes de représentation choisis. Il s'agit ici des méthodes de partage ou d'indexation des objets, méthodes

basées sur leur localisation absolue au niveau de l'espace initial de représentation. La nécessité d'une telle structuration est évidente car l'interrogation d'une base de données localisées va faire intervenir en priorité la localisation ; une interrogation classique, dans un SIG, sera en effet souvent du type : retrouver dans telle zone tous les objets de tel type qui répondent à telles conditions. Cette interrogation se décompose d'une manière optimale en :

- Retrouver tous les objets dans une fenêtre contenant la zone d'étude,
- Sélectionner les objets qui répondent aux critères de sélection.

La première opération, la restriction spatiale, est de loin le critère le plus discriminant, car il repose sur un attribut multidimensionnel (n= 2 ou n=3), face aux opérations de sélection sur les attributs descriptifs à une dimension. Devant s'effectuer en priorité pour optimiser le temps de recherche, elle nécessite l'accès possible à l'ensemble des données de la base sur les mémoires de stockage et représente l'opération la plus coûteuse en temps d'exécution. L'organisation des données doit donc être conçue en priorité par rapport à l'attribut de localisation.

5.4.1. Les méthodes classiques d'indexation

Les méthodes classiques de gestion de données ont pour objet de réduire le nombre des opérations d'accès aux mémoires de stockage. Elles sont basées sur des techniques d'indexation d'attributs mono-dimensionnels ordonnés (accès aléatoire, séquentiel indexé) [KNU 73] [LEE 80], et ne sont pas applicables directement à l'attribut de localisation, qui est par nature multidimensionnel sans relation d'ordre compatible avec la structure euclidienne. On peut organiser l'indexation d'un ensemble d'objets par leur nom, grâce à l'ordre lexicographique par exemple, les regrouper en paquets grâce à la relation d'ordre, et stocker les objets de chaque paquet sur un secteur contigu des disques magnétiques, de manière à pouvoir accéder au paquet en une opération d'accès mémoire. En revanche, il n'est pas possible, pour le cas le plus simple de données ponctuelles, de structurer ces points de la même manière, avec un ordre sur les abscisses ou sur les ordonnées, indépendamment l'un de l'autre, sans perdre toute relation de voisinage ou de distance entre deux points (il n'existe pas de relation d'ordre total dans les espaces euclidiens de dimension supérieure à 1). Une telle structuration serait donc inefficace dans le cas qui nous intéresse, à savoir retrouver rapidement tous les objets dans un espace donné ou tous les objets proches d'un objet donné. Cet objectif impose un rapprochement logique (et physique sur les mémoire de stockage) des objets basés sur leur localisation. Il est nécessaire de développer pour cela des méthodes de gestion adaptées aux données localisées, ou plus généralement multidimensionnelles, méthodes bâties sur une structuration basée sur la notion de voisinage et de sous-ensemble borné pour la métrique choisie dans l'espace de représentation.

5.4.2. L'indexation multidimentionnelle

L'idée la plus simple, calquée sur les techniques du séquentiel indexé [GAR 83] est de décomposer l'espace en sous-ensemble bornés, que nous appellerons feuilles, et d'indexer les objets par rapport à ces feuilles et d'effectuer les recherches séquentiellement dans les feuilles sélectionnées pour le domaine d'étude ou de sélection.

On peut classer les différentes méthodes utilisées selon la définition et la structure des feuilles [SCH 95] :

- les structures en grille : les feuilles correspondent à un quadrillage régulier ;
- les structures en arbres quaternaires ou *region-quadtree* ;
- les structures en arbres k-d (*k-dtree*) ;
- les structures en arbres R (*R-tree*).

On trouvera dans [SCH 95] une présentation détaillée des différentes structures permettant de gérer les index spatiaux. Nous ne rappelons ici que les principes élémentaires et les structures les plus employées.

Les structures de type grille utilisent une partition de l'espace en feuilles élémentaires de forme rectangulaire. La structure est extrêmement simple : à chaque feuille est associée une page contenant l'ensemble des objets inclus dans la feuille.

Pour optimiser les temps d'accès à la mémoire de stockage, la taille d'une feuille sera choisie de manière à être lue en une seule opération d'entrée-sortie, ce qui impose un stockage de tous les objets d'une feuille sur un secteur contigu de la mémoire de masse. Le nombre des objets pouvant être très variable d'un endroit à l'autre de l'espace géographique, cette taille ne peut être fixée a priori sans dégrader les performances de l'indexation (les feuilles se trouvant dans des zones de faible densité seraient alors presque vides). La taille d'une feuille devrait donc être variable en fonction de la densité d'objet : si le nombre des objets d'une feuille dépasse le maximum, cette feuille sera divisée en sous-feuille, et chaque objet réaffecté à une des sous-feuilles. Cette division peut se faire en adaptant la taille de la grille au nombre d'objet (grille adaptative) ou de manière à faciliter le choix de la sous-feuille, pour une localisation donnée, et donc naturellement par dichotomie. La feuille est alors divisée en 4m (pour n=2) ou 8m (pour n=3) sous-feuilles, où m n'est pas trop élevé pour ne pas créer un grand nombre de feuilles presque vides.

L'éclatement d'une feuille en sous-feuille est une opération complexe, qui impose également la réorganisation des index basés sur les numéros de feuilles.

L'organisation des feuilles et les différents niveaux d'éclatement peuvent être représentés par un m-quadtree (m-octree pour n=3). Chaque nœud non terminal contient la description de la division et pointe vers ses fils, et chaque nœud terminal contient l'adresse de la feuille dans la mémoire de stockage. Un tel quadtree est appelé « région quadtree », par opposition aux quadtrees vus au chapitre 3 et servant à représenter directement un espace découpé en cellules.

La recherche sur la localisation (sélection géographique) se fait alors par descente dans l'arbre : à chaque niveau sera choisi le fils qui contient la feuille cherchée, jusqu'au nœud terminal qui donne l'adresse de la feuille. Si m est choisi de manière à ce que la lecture d'un nœud ne nécessite qu'une entrée-sortie (la recherche du fils se faisant en mémoire centrale), l'adresse de la feuille est déterminée en quelques entrée-sorties. La recherche des feuilles voisines fait appel à des cheminements dans le quadtree, et ne demande donc elle aussi que quelques opérations [SAM 84].

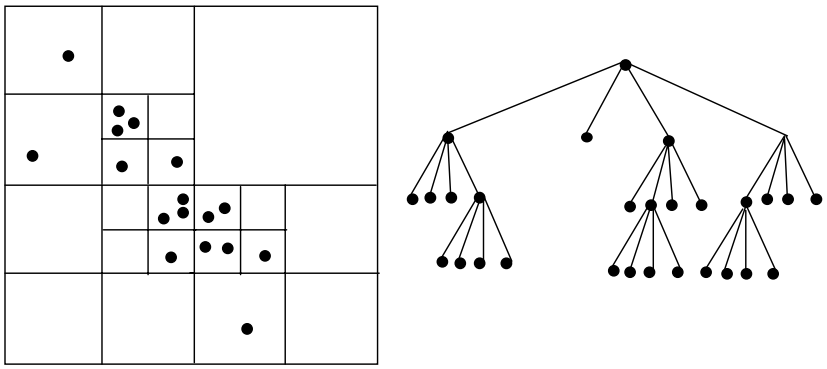


fig. 5.4 : hiérarchie en feuilles et « region quadtree » associé

L'inconvénient majeur de la partition en « region quadtree » réside dans le coté arbitraire du découpage en quatre régions égales, ce qui peut provoquer un déséquilibre important de l'arbre si les données ne sont pas réparties de manière relativement homogène dans l'espace, et donc une baisse notable des performances de l'indexation. Pour remédier à ce problème, on peut utiliser un découpage récursif qui tient compte de la position des objets de manière à optimiser le nombre d'objets présents dans chaque partie du découpage, à l'image des k-dtree [MAT 84]. Bien sûr, le découpage doit alors être mémorisé car il n'est plus implicite comme dans les « region quadtree » ; la recherche des feuilles du k-d tree contenues dans la fenêtre

d'étude se fait également par descente dans un arbre binaire, mais en consultant pour chaque nœud la manière dont l'arbre a été découpé.

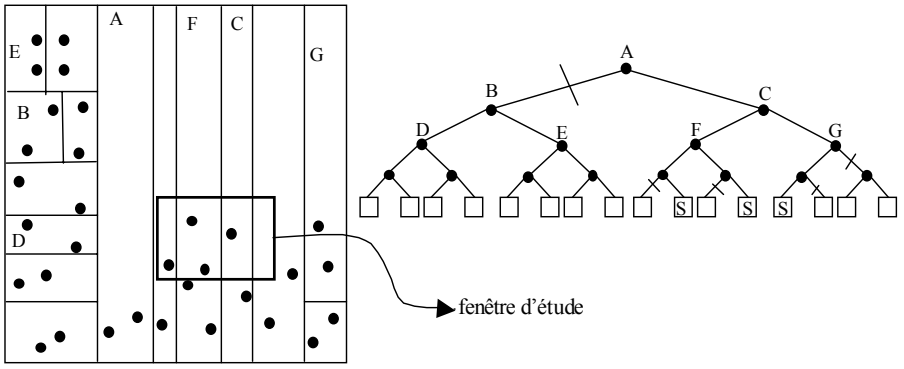


fig. 5.5 : découpage en k-d tree et arbre binaire associé

La recherche d'un objet doit également pouvoir se faire sur la valeur d'une de ses clés, sans connaître la localisation : par exemple, un nom de ville doit permettre de retrouver rapidement l'adresse de la feuille qui la contient. Ce type de recherche nécessite la mémorisation pour chaque objet de l'adresse de la feuille à laquelle il appartient : ceci revient à créer pour la clé choisie un index dense, et d'organiser cet index de manière classique en arbre B+ ou en paquets. Quelques opérations d'entrée-sortie suffiront alors à retrouver l'adresse de la feuille qui contient l'objet. Les objets voisins se trouveront alors dans la feuille sélectionnée ou dans les feuilles adjacentes, faciles à déterminer grâce à l'indexation principale sur la localisation en cheminant dans les arbres associés. Ces index, nécessairement denses, peuvent occuper une place importante.

Comme nous l'avons déjà mentionné, l'éclatement et l'agrégation de feuilles imposent la réaffectation des objets dans les feuilles et donc la réorganisation des index. Toutes ces opérations sont relativement complexes à mettre en œuvre.

L'affectation d'un objet dans une feuille est simple pour les objets d'une relation de type point. Tout se complique lorsque ces objets sont des lignes ou des zones, car ils peuvent alors ne pas être contenus strictement (au sens de l'inclusion) dans la feuille et l'unicité du choix d'affectation n'est plus assurée. Diverses solutions peuvent être apportées à ce problème. La plus simple consiste à affecter un centroïde à chaque objet et à stocker l'objet dans la feuille qui contient le centroïde. Une autre alternative serait de diviser les objets en sous-objets en introduisant le

bord de la feuille, mais cette méthode complique beaucoup les manipulations nécessaires au stockage et à l'extraction des données. Matsuyama propose de conserver avec chaque feuille le numéro des enregistrements de tous les objets totalement ou partiellement contenus dans la feuille : la sélection d'une feuille par descente dans le k-d tree ou dans le « region quadtree » permet de déterminer l'ensemble des feuilles à consulter ainsi que le numéro des objets dans chaque feuille [MAT 84].

Le problème peut également être résolu de manière différente, en conservant une feuille à chaque niveau de la structure arborescente où ne seront stockés que les objets qui ne peuvent être contenus entièrement dans aucune des feuilles des niveaux suivants. Evidemment, la gestion des objets en est fortement compliquée, car ces feuilles ne peuvent être éclatées quand elles sont pleines et doivent alors être chaînées entre elles comme des paquets de débordement. Pour n=2, le nombre des objets possibles à un niveau est néanmoins fortement limité par la surface de la feuille ; les objets fortement convexes (surface/périmètre^2 << 1/4*π) ainsi que ceux de petite surface qui sont coupés par les premières divisions de l'espace sont les plus problématiques. Si l'on arrive à résoudre le problème de l'affectation de ces petits objets mal placés par rapport à la division de l'espace, cette structure multi-niveau introduit une hiérarchie dans l'encombrement des objets qui peut également constituer un critère de sélection intéressant [FRA 81]. Notons pour conclure que peu de systèmes mettent en œuvre ces techniques de structuration. De nombreux systèmes ne sont pas aptes à fournir des performances valables lorsque la taille de la base de données géographique est importante, et n'offrent pas alors les possibilités d'interactivité qui semblent néanmoins nécessaires à un système performant.

5.5. La mise en œuvre du relationnel étendu dans le système SAVANE

Dans ce paragraphe, nous allons détailler la mise en œuvre concrète, dans le système Savane, des principes de structuration énoncés aux chapitres 3 et 5. Nous allons décrire la mise en œuvre concrète de la structuration relationnelle étendue à la localisation dans le système Savane.

5.5.1. Schéma : relations et attributs

L'architecture de SAVANE est celle d'un calculateur base de données : il possède un dictionnaire des données, indiquant les relations et attributs ainsi que leurs caractéristiques (types, définition du schéma interne associé), un dictionnaire des accès permettant de gérer les niveaux externes (par l'allocation de droits et la définition des accès aux données), un langage de commande permettant d'interroger

et de manipuler les données. Chaque opération utilise ces trois structures pour accéder au niveau interne et au système de fichier.

Les données sont structurées en relations correspondant aux entités du monde réel. La structuration reprend le principe du modèle relationnel étendu à la localisation tel que nous l'avons présenté plus haut. Les objets sont donc regroupés en relations, dont le type peut être : zone, ligne, point, mosaïque, non localisé. La localisation est conservée sous forme vectorielle (pour les types zone, ligne, point) et sous forme matricielle (pour le type mosaïque). Les paramètres des relations d'une base de données sont conservés dans le fichier *base* qui correspond au dictionnaire de la base de données.

Pour chaque relation, la description comprend :

- le nom de la relation
- le type de la relation (zone, ligne, point, image, non localisée)
- le nombre d'attributs descriptifs
- le nom des fichiers associés (objets et valeurs descriptives, objets graphiques associés)

La description d'une relation est suivie de la description de ses attributs. Pour chaque attribut, la description comprend :

- le nom de l'attribut
- le type de l'attribut (nominal, ordinal, entier, numérique, date, RVB, etc)
- le nombre de valeurs nominales (dans le cas d'un attribut nominal)
- l'adresse de la première valeur nominale dans le fichier des valeurs d'attributs (dans le cas d'un attribut nominal).

La lecture du dictionnaire permet au système de charger le schéma des données et l'ensemble des paramètres permettant d'accéder aux objets (données graphiques et descriptives). La classe `CDico` permet d'encapsuler l'ensemble des opérations de lecture et d'écriture du dictionnaire de la base de données (A.1). On pourra consulter par exemple, le code des procédures `EcrireRelation()` et `EcrireAttribut()`.

Toutes les opérations de déclaration ou de modification du schéma d'une base de données sont effectuées avec le module SAVATECA (fig. 5.6, fig. 5.7). Il est possible à tout moment de modifier le schéma d'une base de données, en ajoutant, modifiant, ou supprimant relations ou attributs. Toute modification du schéma implique une modification du numéro d'état de la base. Les programmes d'exploitation du système (comme les modules SAVANE, SAVEDIT, SAVAMER)

utilisent ce numéro d'état pour remettre à jour le schéma s'il a été modifié pendant l'exploitation de la base de données.

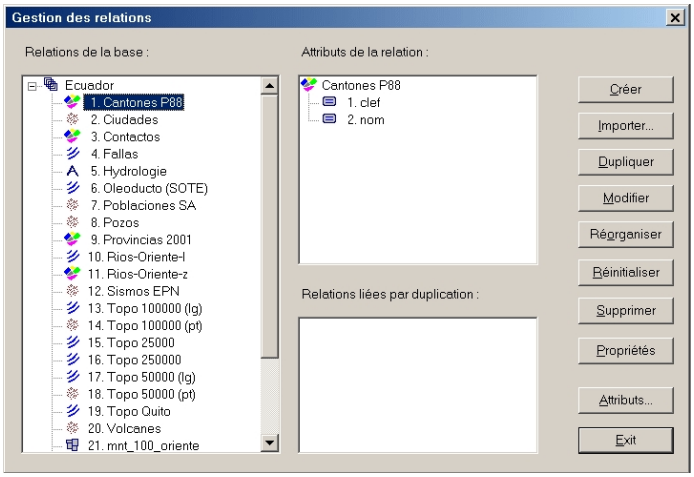


fig. 5.6 : le dialogue pour la gestion des relations

Le dialogue général de gestion des relations de SAVATECA contient de nombreux boutons, permettant de créer, d'importer, de modifier, de supprimer, de réinitialiser ou de réorganiser une relation.

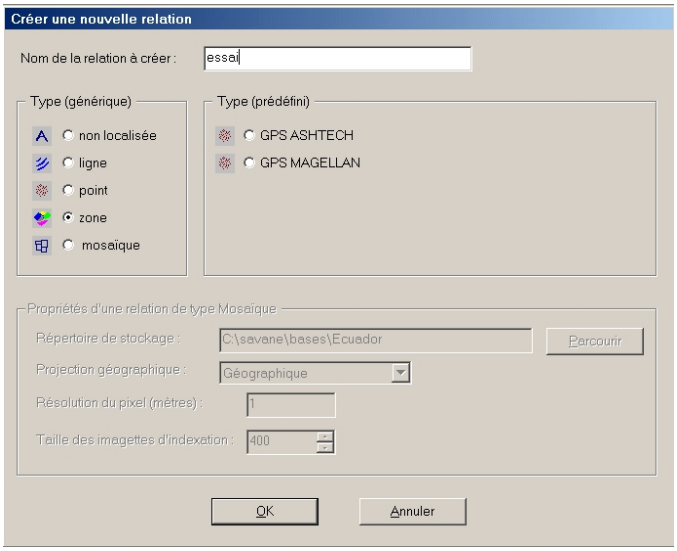


fig. 5.7 : le dialogue de création d'une relation

Pour créer une relation, l'utilisateur doit indiquer le nom et le type de la relation. Le schéma de certaines relations est prédéfini : la relation est alors automatiquement créée avec son type et ses attributs prédéfinis.

Lorsque la relation est du type mosaïque, l'utilisateur doit indiquer le répertoire de stockage de la mosaïque ainsi que la projection géographique utilisée pour conserver les images.

La création de la relation se traduit concrètement par l'écriture de la description de la relation dans le fichier *base*. La suppression d'une relation supprime la description de la relation et de ses attributs dans le dictionnaire et supprime toutes les références à cette relation dans les vues externes.

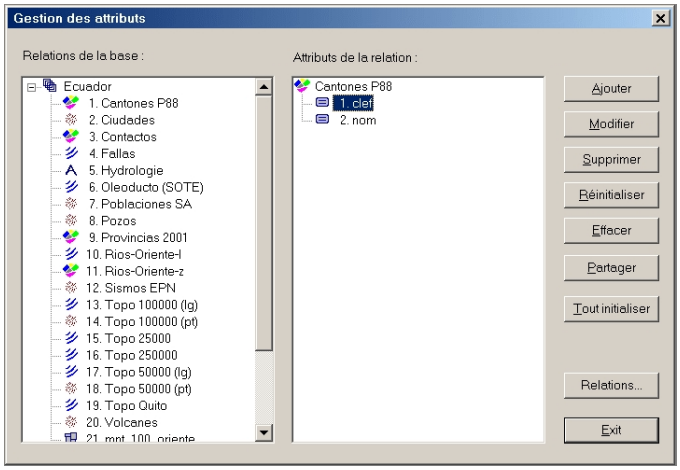


fig. 5.8 : le dialogue de gestion du schéma des attributs d'une relation

Le dialogue de gestion des attributs de SAVATECA permet d'ajouter, modifier, supprimer, ou réinitialiser des attributs (fig. 5.8, fig. 5.9). Comme pour les relations, toute suppression d'attribut est suivie d'une suppression des références de cet attribut dans les vues externes de la base, ainsi que d'une modification du numéro d'état de la base de données.

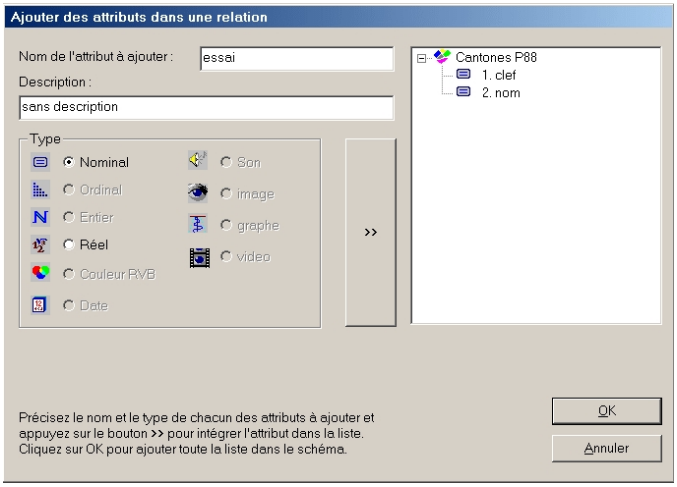


fig. 5.9 : le dialogue de création d'un attribut

La classe `CSchema`, présente dans tous les modules du système, permet de charger le schéma en mémoire à partir du dictionnaire. Elle est associée aux classes `CRelation` et `CAttribut`, qui permettent d'encapsuler données et méthodes correspondant au schéma des relations et de leurs attributs. Par exemple, la méthode `Alloc()` de la classe `CSchema` charge le schéma dans un objet de type `CSchema`, à partir du dictionnaire de la base et de la vue externe utilisée (A.2.1.2.).

Ces classes sont fondamentales dans le système SAVANE. Elles sont au cœur du système et sont utilisées par la majorité des autres classes.

5.5.2. Structuration des collections d'objets

5.5.2.1. Architecture générale et organisation des bases de données

Tous les fichiers correspondant aux relations d'une base de données sont conservés dans un répertoire unique portant le nom de la base de données. Ce répertoire peut être physiquement situé n'importe où sur le réseau local. L'accès aux fichiers d'une relation est géré par le système d'exploitation de l'ordinateur (ouverture, écriture, accès concurrents) à partir de ce répertoire, dont l'emplacement absolu est conservé dans le fichier *fpconf*.

La seule exception concerne les relations de type pixel, pour que ces relations puissent être partagées entre plusieurs bases de données différentes. Les fichiers correspondants à ce type de relation peuvent donc être stockés dans un répertoire

particulier. L'emplacement de ce répertoire est conservé dans le fichier *base* avec les caractéristiques de la relation.

Si les déclarations sont incomplètes, ou si le système n'arrive pas à trouver les emplacements indiqués, l'utilisateur devra indiquer le bon emplacement des fichiers à chaque première utilisation de la relation, lors d'une session de travail.

5.5.2.2. Indexation géographique et structure des fichiers

La structure des fichiers d'une relation dépend du type de la relation. Nous allons décrire cette structure pour chaque type.

D'une manière générale, la description des objets (identifiant d'objet, entête géographique) et leurs valeurs d'attributs sont conservées dans un fichier, la description de la localisation dans un autre (pour les types zone et ligne), et un troisième fichier conserve l'indexation géographique (pour les relations localisées). Toutes les localisations sont conservées en coordonnées géographiques (longitude-latitude) par rapport à un point fixe, propre à la base. Pour toutes les relations d'une même base, les coordonnées géographiques doivent donc être exprimées dans un même *datum*. Il est possible de modifier le datum d'une base de données, toutes les coordonnées étant alors recalculées en utilisant la formule de Molodensky.

Tous les types localisés sont indexés de façon primaire sur la localisation. Cette indexation utilise le principe des grilles adaptatives et une structure séquentielle pour chaque entrée de l'index (séquentiel indexé). Pour définir les cellules, on utilise le découpage naturel en feuille cartographique de numérisation (voir chapitre 7). L'indexation conserve pour chaque feuille l'espace géographique occupé par les objets de la feuille (c'est-à-dire le plus petit rectangle englobant l'ensemble des objets), le nombre d'objet de la feuille, l'adresse du premier objet de la feuille dans le fichier de description. Pour les types zone et ligne, on conserve également le nombre d'arcs utilisés pour décrire la localisation des objets et l'adresse du premier de ces arcs dans le fichier des arcs correspondant à la relation.

5.5.2.2.1. Le type non localisé

Le type non localisé est le plus simple : les objets sont stockés séquentiellement, la description de chaque objet est constituée uniquement des valeurs des attributs. Il n'y a pas d'indexation, ni d'identifiant d'objet.

5.5.2.2.2. Le type point

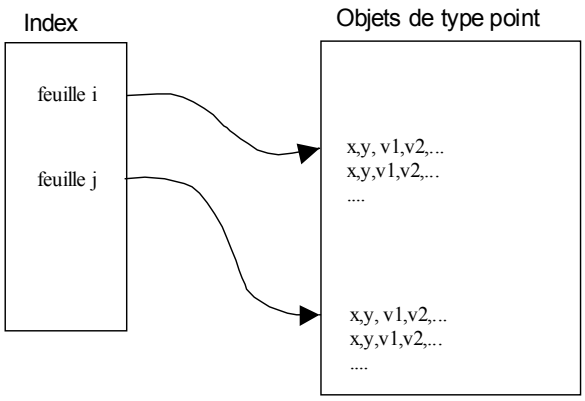
Chaque objet point est décrit par des valeurs d'attributs descriptifs et par une localisation géométrique ponctuelle. Le type point utilise deux fichiers : un fichier index et un fichier descriptif. L'index comprenant pour chaque feuille :

- le nom de la feuille,

- l'espace géographique occupé par les objets de la feuille (point bas gauche, point haut droit, en coordonnées géographiques),
- le nombre d'objets de la feuille,
- l'adresse du premier objet de la feuille dans le fichier descriptif.

Le fichier descriptif comprend pour chaque objet :
- la localisation du point (en coordonnées géographiques),
- les valeurs des attributs, dans l'ordre décrit par le schéma de la base.

On peut donc lire l'ensemble des objets d'une relation en balayant son index, et pour chaque entrée de l'index, en balayant séquentiellement les objets à partir de l'adresse donnée pour le premier objet de la feuille.



5.5.2.2.3. Le type ligne

Chaque ligne est décrite par des attributs descriptif et par un arc géométrique. Nous n'utilisons pas de polyligne, ou de lignes constituées par plusieurs arcs.

Le type ligne utilise trois fichiers : un fichier index, un fichier descriptif, un fichier d'arcs. Le fichier index permet d'accéder aux objets d'une feuille, puis on obtient dans le fichier descriptif la description de chaque objet ainsi que l'adresse de l'arc correspondant dans le fichier descriptif. L'espace géographique occupé par chaque objet est conservé dans le fichier des arcs. Chaque arc est décrit par la suite des points qui le constitue. Les coordonnées sont conservées en longitude, latitude par rapport au point bas de la base.

L'index comprend pour chaque feuille :

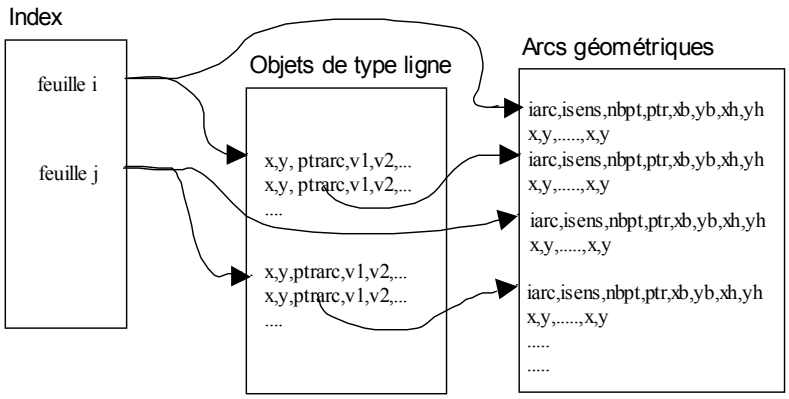
- le nom de la feuille,
- l'espace géographique occupé par les objets de la feuille (point bas gauche, point haut droit, en coordonnées géographiques),
- le nombre d'objets de la feuille,
- l'adresse du premier objet de la feuille dans le fichier descriptif,
- le nombre d'arcs utilisés pour décrire la localisation des objets,
- l'adresse du premier arc dans le fichier des arcs.

Le fichier descriptif comprend, pour chaque objet :
- le numéro de l'arc correspondant à l'objet,
- l'adresse de l'arc dans le fichier des arcs,
- la localisation du centroïde de la ligne,
- les valeurs des attributs dans l'ordre décrit par le schéma de la base.

Le fichier des arcs comprend, pour chaque arc, un entête et un ensemble de points. L'entête comprend pour chaque arc :
- le numéro de l'arc,
- le sens de l'arc,
- le nombre de points de l'arc,
- le pointeur vers l'arc suivant,
- la fenêtre de l'arc, en coordonnées géographiques.

Les coordonnées des points de l'arc suivent l'entête. Ils sont exprimés en longitude-latitude par rapport au point bas de la base.

On peut lire l'ensemble des objets d'une relation en balayant son index, et pour chaque entrée de l'index, en balayant séquentiellement les objets à partir de l'adresse donnée pour le premier objet de la feuille. On retrouve l'arc géométrique correspondant à un objet par son pointeur dans le fichier des arcs. On peut également accéder à l'ensemble des arcs d'une feuille grâce au pointeur du premier arc de la feuille, pointeur qui se trouve également dans l'index.



5.5.2.2.4. Le type zone

Chaque zone est décrite par un identifiant d'objet, par des attributs descriptifs et par des arcs géométriques décrivant la frontière de la zone. Dans une feuille, les arcs ne sont pas répétés : on conserve pour chaque arc le numéro des deux zones adjacentes. Pour reconstituer le contour d'une zone, il est donc nécessaire de balayer l'ensemble des arcs d'une feuille pour retrouver les arcs qui appartiennent à cette zone. Les seuls arcs répétés sont donc les arcs qui sont frontières de zones appartenant à deux feuilles différentes.

La topologie d'une zone n'est pas constituée lors du processus d'intégration de la zone dans la base. La topologie est créée lors du processus de saisie ou d'importation, grâce au module SAVEDIT, que nous étudierons en détail au chapitre 7. Il est très important que les objets d'une feuille constituent un ensemble topologiquement sans erreur : de nombreux contrôles d'intégrité sont développés dans SAVEDIT à cet effet.

Cette méthode de description de la géométrie d'un ensemble de zones est très simple et facile à gérer, puisqu'elle ne nécessite pas la gestion du sens des arcs ou de leur chaînage. Le problème des îles ou des zones incluses et imbriquées ne se pose pas . On privilégie ainsi la simplicité de la structure et de la représentation, au détriment de la simplicité des traitements nécessaires à certaines fonctions.

Le type zone utilise trois fichiers : un fichier index, un fichier descriptif, un fichier d'arcs. Le fichier index comprend pour chaque feuille :

- le nom de la feuille,

- l'espace géographique occupé par les objets de la feuille (point bas gauche, point haut droit, en coordonnées géographiques),

- le nombre d'objets zone de la feuille,
- l'adresse du premier objet de la feuille dans le fichier descriptif,
- le nombre d'arcs utilisés pour décrire la localisation des zones,
- l'adresse du premier arc dans le fichier des arcs.

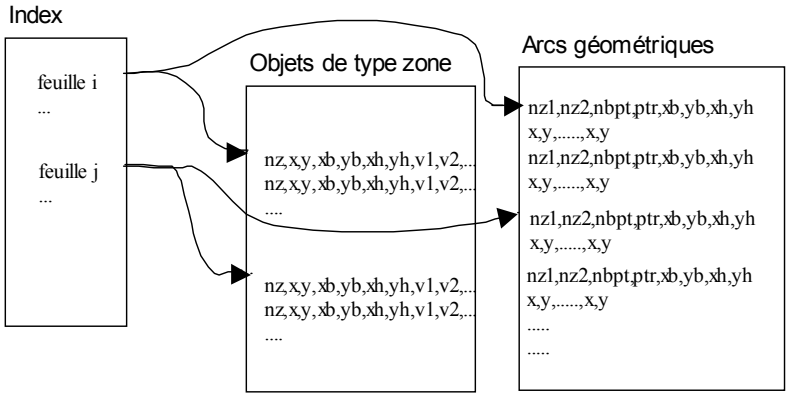
Le fichier descriptif comprend, pour chaque objet :

- le numéro de la zone dans la feuille,
- la localisation du centroïde de la ligne,
- la fenêtre minimale contenant la zone,
- les valeurs des attributs dans l'ordre décrit par le schéma de la base.

Le fichier des arcs comprend, pour chaque arc, un entête et un ensemble de points. L'entête comprend pour chaque arc :

- le numéro de la première zone adjacente,
- le numéro de la seconde zone adjacente. Si l'arc ne borde qu'une zone, ce numéro est 2. Si la zone adjacente appartient à une autre feuille, ce numéro est 19 (si l'arc doit être visible) ou 18 (si l'arc ne doit pas être visible, comme dans le cas d'un découpage arbitraire en feuille par une ligne droite),
- le nombre de points de l'arc,
- le pointeur vers l'arc suivant,
- la fenêtre de l'arc, en coordonnées géographiques.

Les coordonnées des points de l'arc suivent l'entête. Elles sont exprimées en longitude, latitude par rapport au point bas de la base.



5.5.2.2.5. Le type mosaïque

Le type mosaïque correspond aux images. A chaque relation correspond un fichier décrivant les paramètres de la relation (taille du pixel, projection géographique, etc.), et à chaque attribut de la relation correspondent deux fichiers : un fichier index, correspondant au découpage en feuille (on parlera plutôt d'imagettes dans le cas d'une relation de type mosaïque), et un fichier contenant la valeur de chaque pixel présent dans la relation, pour cet attribut. La structure de ce type de relation sera étudiée en détail au chapitre suivant.

5.5.3. Intégration d'objets et de valeurs d'attributs

5.5.3.1.La constitution d'une relation

La constitution d'une relation de la base de données se fait par intégrations successives de groupes d'objets dans la relation. Pour les relations localisées, l'intégration se divise en deux phases distinctes : la première phase consiste à créer les objets en intégrant l'attribut de localisation (saisi ou importé dans le module SAVEDIT) et un identifiant descriptif pour chaque objet. La seconde phase permet d'intégrer les valeurs descriptives des objets en utilisant l'identifiant comme attribut de jointure. Ces valeurs descriptives peuvent provenir d'un fichier classique ou d'une base de données relationnelle classique. Les valeurs des attributs des objets sont alors lues dans ces fichiers et recopiées dans la base savane.

Pour les relations localisées, les objets à intégrer sont regroupés dans des documents à intégrer, correspondant en général à une coupure de carte, ou à un espace géographique. L'idée générale, c'est d'avoir une relative homogénéité du nombre d'objets par document, puisque chaque opération de création d'objets à partir d'un document crée une feuille, c'est-à-dire une entrée dans l'index géographique. L'adresse des objets et la position géographique en deux dimensions sont conservées pour chaque feuille et sont utilisées lors de l'exploitation de la base comme indexation primaire, dans une structure séquentielle indexée en deux dimensions, comme nous l'avons vu dans le paragraphe précédent.

Le découpage de l'espace en grille adaptative doit donc se faire lors de la préparation de la saisie des données, bien avant l'intégration des objets dans la base de données.

5.5.3.2. Codage des valeurs des attributs descriptifs

Le codage des valeurs des attributs descriptifs dépend du type de l'attribut. Rappelons les types d'attribut descriptif gérés par le système SAVANE :

• *Nominal* ou *qualitatif* : ceux qui prennent des valeurs nominales (une chaîne de caractères).

• *Ordinal* : ceux qui prennent des valeurs nominales que l'on veut pouvoir ordonner,

• *Entier* : ceux qui prennent des valeurs numériques entières,

• *Réel* : ceux qui prennent des valeurs numériques quelconques (comme un prix).

• *Couleur* 8 bits, 16 bits, RVB : la valeur de l'attribut correspond au codage d'une couleur. Avec 8 bits, on peut coder 256 couleurs, avec 16 bits 64000. Une couleur RVB comprend un niveau de rouge, un niveau de vert, un niveau de bleu, chaque niveau étant codé sur 8 bits (256 valeurs possibles pour chaque niveau, soit 16 millions de couleurs possibles).

• *Date* : ceux qui prennent la valeur d'une date,

• *Son* : la valeur de l'attribut correspond à la description d'un son,

• *image* : la valeur de l'attribut correspond à une image ou à une collection d'images (album de photographies, documents scannés,…),

• *vidéo* : la valeur de l'attribut correspond à une séquence vidéo,

• *graphe* : la valeur de l'attribut correspond à la description d'un graphe en deux dimensions.

Les valeurs des attributs nominaux sont codées en numérique : les valeurs nominales sont conservées sur 16 caractères dans un fichier global, propre à la base de données (le fichier *fpvaleurs*). La valeur conservée dans le fichier de description des objets est l'indice relatif de la valeur dans ce fichier, par rapport à la première valeur. L'adresse de la première valeur dans le fichier *fpvaleurs* et le nombre de valeurs différentes sont conservés dans le descriptif de l'attribut, dans le fichier *base*. Il est ainsi facile de connaître l'ensemble des modalités d'un attribut nominal et de rechercher une modalité sans avoir à balayer l'ensemble des objets. Un ensemble de modalités peut également être partagé entre plusieurs attributs. Par contre, la gestion de ce codage est plus complexe, surtout en cas de modification ou d'ajout.

Tous les attributs numériques sont conservés sous forme numérique en double précision, directement dans le fichier descriptif de la relation.

Les couleurs (8, 16, 24 bits) correspondent uniquement à un attribut d'une relation de type mosaïque. Nous détaillerons leur codage dans le chapitre suivant.

Les valeurs des attributs de type son, album d'images, vidéo, graphe conservent des noms absolus de fichier, chaque valeur correspondant à un fichier contenant l'objet en question. Ainsi, la valeur d'un attribut de type video pour un objet (localisé ou non) correspond au nom et à l'emplacement du fichier contenant la video. Ces noms de 256 caractères sont en fait conservés dans un fichier global (le

fichier *fpfichier*), et la valeur de l'attribut correspond à l'adresse du nom dans le fichier global.

Il arrive souvent que la valeur d'un attribut d'un objet soit inconnue ou indéterminée. Nous avons donc défini pour chaque type d'attribut une valeur spéciale correspondant à la « valeur inconnue ». Pour les attributs nominaux ou de type son, album d'images, vidéo, graphe, on utilise la valeur 0. Pour les attributs numériques, on utilise la valeur $-1.e^{+30}$. La seule dérogation à cette règle concerne les attributs des mosaïques codés sur 8 ou 16 bits : la valeur inconnue est 0, ce qui nous obligera à modifier les pixels de valeur initiale 0 pour qu'ils ne soient pas considérés comme des objets de valeur inconnue.

5.5.3.3. L'intégration graphique

L'intégration graphique (fig. 5.10) crée physiquement des enregistrements dans le fichier des objets et dans le fichier des arcs (pour les types zone et ligne). Elle calcule la fenêtre géométrique de chaque objet, écrit l'enregistrement avec les valeurs géographiques (centroïde, fenêtre géométrique) et tous les autres attributs avec une valeur réservée correspondant à la valeur « inconnue » (codée $-1e^{30}$). Une fois les objets créés, l'intégration graphique revient sur les objets pour intégrer la valeur d'une clé descriptive (attribut nominal clé des attributs descriptifs de la relation). Pour les objets localisés, cette valeur a été saisie pour chaque objet lors de la saisie graphique.

L'intégration graphique, lors de la création des objets, ne fait que recopier dans les fichiers de la relation les objets digitalisés qui sont regroupés dans des documents distincts. Ces documents sont constitués et édités grâce au module SAVEDIT du système. L'intégration graphique n'effectue aucune création ni contrôle au niveau de la topologie des objets. Cette topologie est constituée et vérifiée lors de la saisie graphique. Nous étudierons en détail dans le chapitre 7 le module SAVEDIT et les diverses contraintes et contrôles d'intégrité liés à l'attribut de localisation. L'indexation en feuille conserve le regroupement des objets provenant des différents documents digitalisés : la topologie conserve donc également cette structure.

La seule difficulté réside donc dans la gestion des relations graphiques entre objets appartenant à des feuilles différentes : connexité d'arcs, frontières entre zones de feuilles différentes. Ces relations doivent être gérées lors de la saisie graphique des objets. Le module SAVEDIT contient donc des procédures permettant de joindre ou d'unir des arcs appartenant à des documents différents.

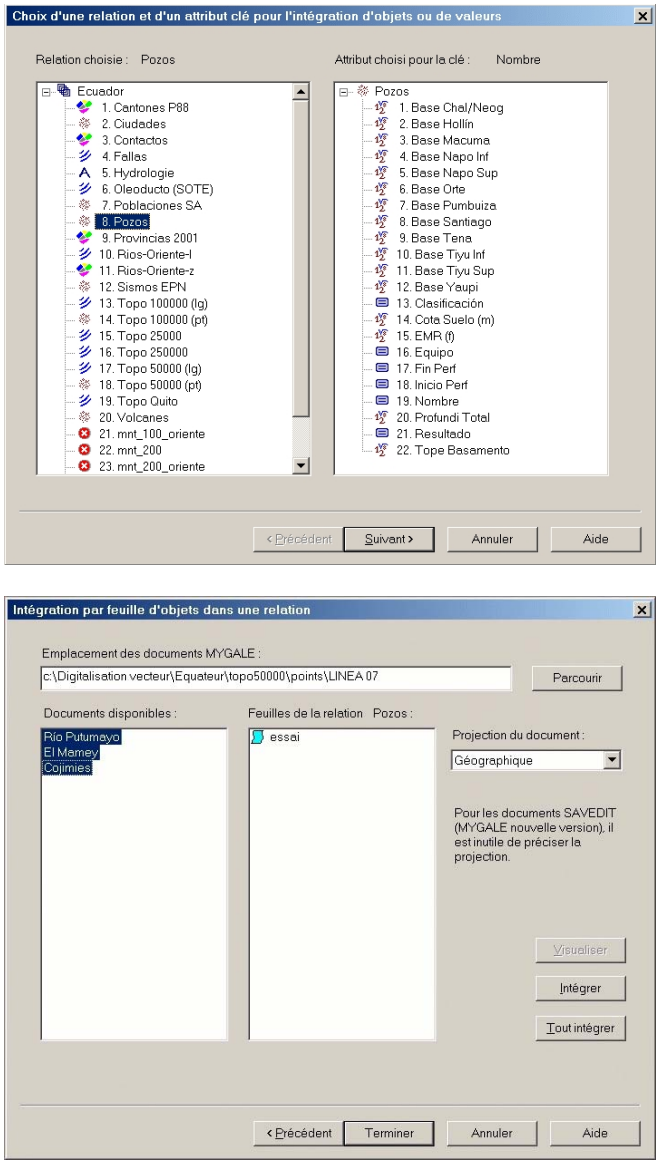


fig. 5.10 : les dialogues de l'intégration graphique

Une autre difficulté apparaît lorsque l'on veut regrouper des objets appartenant à des feuilles différentes, car la structure en feuille et l'organisation de SAVANE ne permet pas à un objet d'avoir des arcs dans des feuilles distinctes.

Un identifiant descriptif, nominal, a été saisi pour chaque objet lors de la saisie graphique. Cet identifiant est intégré comme valeur d'un attribut nominal qui servira ensuite de clé descriptive pour l'intégration des valeurs des autres attributs. Les principes de cette intégration sont les suivants :

- codage de la valeur nominale en numérique,

- écriture de la valeur de chaque objet dans l'attribut correspondant, dans le fichier descriptif de la relation,

- écriture des nouvelles modalités dans le fichier des valeurs nominales (*fpvaleurs*) et actualisation des pointeurs et autres paramètres de l'attribut dans le dictionnaire de la base (fichier *base*).

Le codage de la valeur nominale se fait en plusieurs étapes :

- indexation des valeurs nominales déjà existantes pour cet attribut,

- pour chaque objet, recherche de sa valeur dans l'ensemble des n valeurs déjà existantes. Si la valeur est trouvée, le code correspond à l'indice de cette valeur dans l'ensemble des valeurs de l'attribut. Si la valeur n'est pas trouvée, on la rajoute aux valeurs existantes et le code correspond à $n+1$.

Ces procédures sont également utilisées lors de l'intégration descriptive des autres attributs nominaux, décrite dans le paragraphe suivant. Nous verrons en détail un peu plus loin les classes regroupant ces procédures.

5.5.3.4. L'intégration descriptive

L'intégration descriptive intègre, pour chaque objet, la valeurs des attributs descriptifs. Pour cela, on utilise la clé descriptive intégrée lors de la création des objets, et qui sert de lien entre l'objet et les valeurs à intégrer, qui sont en général regroupées dans un fichier unique. L'utilisateur doit alors indiquer les correspondances entre champs de fichier et attributs, ainsi que la méthode de séparation de champs dans le fichier.

L'intégration d'un attribut descriptif se fait en plusieurs étapes :

- si l'attribut à intégrer est nominal, codage de l'attribut descriptif à intégrer en fonction des valeurs déjà existantes pour cet attribut.

- codage du champ du fichier servant de clé de jointure, en fonction des valeurs de l'attribut nominal choisi comme clé de jointure et déjà intégré dans la base. Le système signale à ce niveau toute valeur nominale présente dans le fichier mais non présente dans la base pour l'attribut choisi comme clé de jointure.

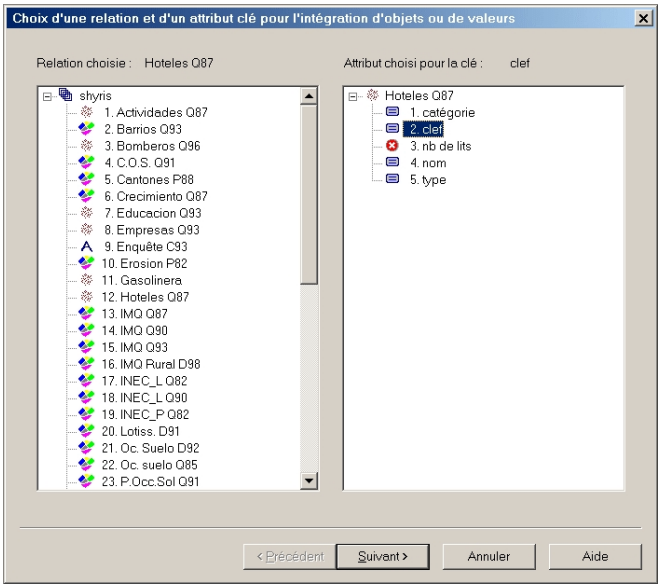
- pour chaque objet de la relation, recherche dans le fichier d'un enregistrement ayant même valeur pour la clé de jointure. Si aucun enregistrement n'est trouvé, la valeur de l'attribut à intégrer reste inchangée. Il est ainsi possible de modifier une

partie seulement des objets intégrés : tout dépend des clés de jointure présentes dans le fichier.

- si l'attribut à intégrer est nominal, intégration des nouvelles modalités dans le fichier des valeurs nominales (*fpvaleurs*) et actualisation des paramètres de l'attribut (nombre de modalités).

Pour les relations localisées, on a en général plus d'objets que de valeurs de clé descriptive, puisque par définition la localisation est une clé de chaque objet. Le fichier descriptif peut ainsi avoir beaucoup moins d'enregistrements que le nombre d'objet de la relation. Une carte des sols est un exemple classique : on peut avoir graphiquement beaucoup de zones (connexes), mais peu de clés descriptives distinctes.

Voici les différents dialogues (fig. 5.11) permettant d'intégrer des valeurs d'attribut à partir d'un fichier ASCII :



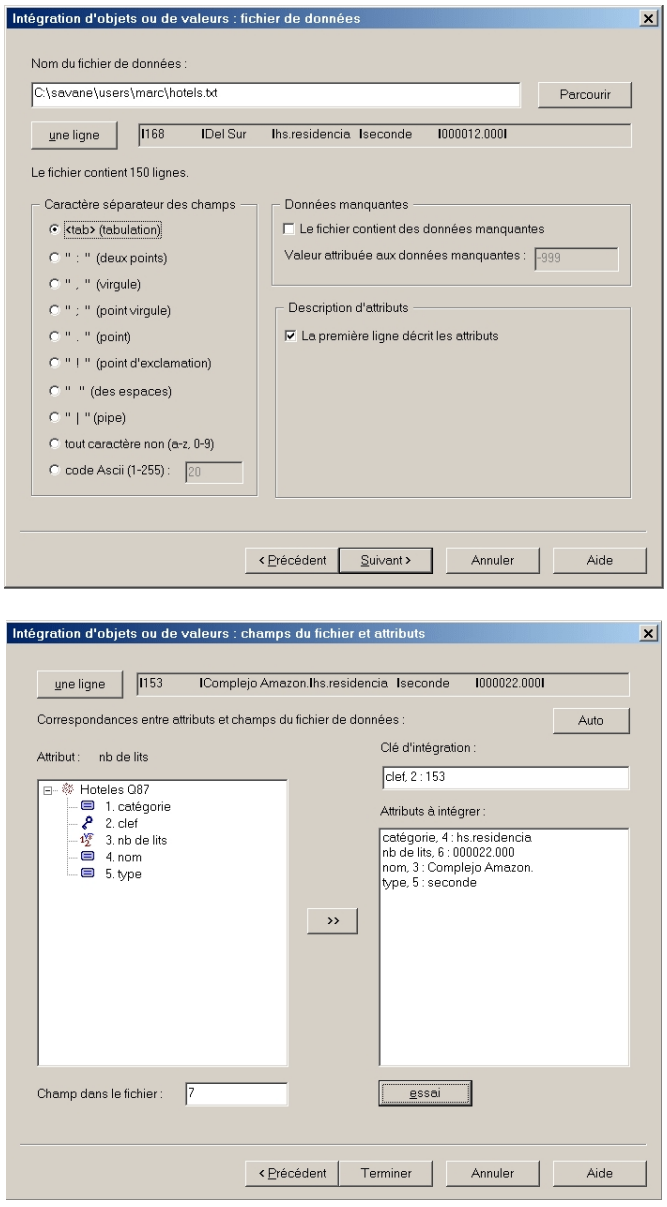


fig. 5.11 : les différents dialogues de l'intégration descriptive

5.5.3.5. La classe CIndexAttribut

La classe `CIndexAttribut` encapsule l'ensemble des structures et opérations nécessaires à l'indexation et la recherche d'une valeur dans l'ensemble des valeurs d'un attribut nominal. On indexe sur les trois premiers caractères des valeurs en utilisant un découpage en paquets liés à l'ordre lexicographique. La recherche est séquentielle à partir de chaque entrée de l'index. Cette classe est utilisée dans les différents modules du système (A.1.5.).

5.5.4. Savane : requêtes et états temporaires

5.5.4.1. Principe d'une requête dans SAVANE

Nous avons vu dans le chapitre 2 (présentation du système SAVANE) les principes de l'exploitation d'une base de données dans le système SAVANE. Rappelons rapidement ces principes.

Le document de base du logiciel SAVANE est une carte, c'est-à-dire une feuille de papier destinée à contenir une représentation graphique de données géographiques. Cette feuille contient par défaut un *cadre géographique*. Un cadre géographique correspond au résultat d'une requête sur la base de données : SAVANE utilise le principe de la requête pour l'interrogation d'une base de données. Il faut enchaîner les opérations, le résultat de l'une pouvant servir d'entrée à la suivante. Une interrogation, une requête, est donc composée d'un ensemble d'opérations que l'on déclenche grâce au menu principal, et que l'on peut conserver sous forme de macro-commande.

Un cadre peut donc être vu comme une fenêtre sur la base de données contenant à un moment donné, un état temporaire de la base correspondant à une interrogation et aux opérations effectuées (calculs, création de nouvelles variables, croisement et jointure d'objets, etc.). Lorsqu'une relation est modifiée, ses objets sont recopiés dans des fichiers temporaires qui sont physiquement stockés dans le répertoire de l'utilisateur. La relation devient alors temporaire, et elle ne contient plus que les objets se trouvant dans la fenêtre d'étude liée au cadre. La structure des fichiers temporaires liés aux relations temporaires est un peu différente de la structure des fichiers de base (voir plus loin la classe `CLecture`). A chaque relation temporaire peut correspondre quatre fichiers temporaires : un pour l'index (feuilles), un pour les objets (descriptif), un pour les arcs, et un pour conserver une image matricielle de la relation (voir paragraphe suivant). Des variables maintiennent le nom de ces fichiers, qui sont libérés automatiquement lorsque la relation est de nouveau modifiée, ou remise dans son état initial. Des procédures globales (`FChoixFeuille()`, `FChoixDescriptif()`, `FChoixArc()`) permettent de connaître le premier fichier disponible pour chaque type (index, descriptif, arc).

La classe `CSchema` de SAVANE permet de décrire l'état de la base de données correspondant à la requête effectuée dans un cadre (A.2.1.2.). Par exemple, la méthode `Actualiser()` permet de recharger le schéma après une modification externe du schéma de la base (avec SAVATECA), tout en conservant l'état des relations temporaires (modifiées par la requête en cours).

Les opérations de manipulation de données sont implémentées dans des fonctions globales qui prennent comme argument un objet de la classe `CCadre` (le cadre sur lequel doit s'effectuer l'opération) ou `CSchema` (le schéma du cadre sur lequel doit s'effectuer l'opération) et l'ensemble des paramètres de l'opération. Ces paramètres sont en général écrits dans un fichier (*fpmenu*) après dialogue avec l'utilisateur ou lecture d'une macro-commande, et relus par la fonction d'exécution de commande. Ces fonctions modifient le schéma qui reflète ainsi l'état de la base après l'exécution de la commande. Ainsi sont implémentées en particulier toutes les fonctions relationnelles classiques (restriction, projection, jointure).

Par exemple, la fonction d'exécution d'une jointure entre deux relations débute par le code suivant :

```
void XQuestJointure(CCadre* pCadre)
  {
  HCURSOR hcurSave = ::SetCursor(LoadCursor(NULL, IDC_WAIT));

  //--- jointure entre deux relations
  CSchema* pSchema=pCadre->GetSchema();
  CWind* pWind=pCadre->GetWind();

  int i,j,k,iNothEmetteur1,iNothEmetteur2;
  int iNoattCleEmetteur1,iNoattCleEmetteur2;
  int iNbAttributsEmetteur1,itabNoattEmetteur1[NB_MAX_ATTRIBUTS];
  int iNbAttributsEmetteur2,itabNoattEmetteur2[NB_MAX_ATTRIBUTS];
  int iJointure;
  char chNomRelation[_MAX_PATH];

  //--- lecture des paramètres
  FILE* FileMenu=fopen(base.m_chNomFtmp,"rb");
  fscanf(FileMenu,"%d%*c",&iNothEmetteur1);
  fscanf(FileMenu,"%d%*c",&iNoattCleEmetteur1);
  fscanf(FileMenu,"%d%*c",&iNbAttributsEmetteur1);
  for(i=0;i < iNbAttributsEmetteur1;i++)
    {
    fscanf(FileMenu,"%d%*c",&itabNoattEmetteur1[i]);
    }
  fscanf(FileMenu,"%d%*c",&iNothEmetteur2);
  fscanf(FileMenu,"%d%*c",&iNoattCleEmetteur2);
  fscanf(FileMenu,"%d%*c",&iNbAttributsEmetteur2);
  for(i=0;i < iNbAttributsEmetteur2;i++)
    {
    fscanf(FileMenu,"%d%*c",&itabNoattEmetteur2[i]);
    }
  fscanf(FileMenu,"%d%*c",&iJointure);
  fgets(chNomRelation,_MAX_PATH,FileMenu);
  fclose(FileMenu);

  ….
```

5.5.4.2. Définition et gestion des macro-commandes

Une macro-commande permet de conserver tous les paramètres nécessaires à l'exécution d'une ou de plusieurs commandes, indépendamment du schéma sur

lequel cette commande doit s'effectuer. On peut ainsi conserver des parties d'un traitement complexe pour les exécuter de nouveau plus tard, ou les conserver dans la structure de la base de données comme des méthodes.

Lors d'une requête, toutes les opérations sont conservées dans une macro-commande spéciale, attachée au cadre et conservée dans la structure du cadre de la carte. Cette macro-commande n'est réinitialisée que si l'utilisateur revient à l'état initial de la base. Elle permet d'exécuter de nouveau la séquence d'opérations effectuée dans le cadre depuis la dernière initialisation du cadre. Cette opération est particulièrement importante dans le cas où la base est modifiée de façon externe, car elle permet de réactualiser le résultat de la requête.

Les macro-commandes sont conservées dans un répertoire de l'utilisateur. La classe `CMacro` de SAVANE permet d'encapsuler toutes les opérations nécessaires à la gestion des macro-commandes (A.2.1.4.).

5.5.5. La classe Clecture

La classe `CLecture` de SAVANE encapsule l'ensemble des procédures de lecture et d'écriture des relations de la base de données, qu'elles soient permanentes ou temporaires (A.2.1.3.). La méthode `Lire()` de cette classe est utilisée en permanence pour lire les valeurs des objets d'une relation, en utilisant le code suivant :

```
CLecture lecture;
if(lecture.Open(pCadre,iNoth))
   {
   int i=lecture.Lire(v);
   while(i != -1)
      {
      if(i == 1)
         {
         //--- traitement de l'objet
         }
      i=lecture.Lire(v);
      }
   lecture.Close();
   }
```

Pour accéder à un attribut d'une relation non temporaire, il faut rechercher le numéro interne de l'attribut, à partir de son numéro externe `iNoAttrExterne` qui provient du dialogue avec l'utilisateur. Le numéro interne est conservé dans la variable membre `m_iNumero` de `CAttribut` (A.2.1.1.) :

```
iNoAttrInterne=pSchema->m_pRelations[iNoRel]->m_pAttributs[iNoAttrExterne]->m_iNumero ;
```

Cette indirection n'est plus nécessaire lorsque la relation est temporaire, car les valeurs des attributs de la vue externe sont réécrites dans l'ordre de la vue, comme nous le verrons dans les paragraphes suivants. On a alors toujours `iNoExterne=iNoInterne`.

5.5.6. SAVANE : les structures internes

5.5.6.1. Vecteur et raster

Comme nous l'avons vu précédemment, la localisation est conservée dans le système SAVANE sous forme vectorielle pour les objets de type zone, ligne ou point, avec la précision inhérente à la définition sémantique des objets. Par contre, le système crée une matrice *raster* pour la résolution rapide de certaines opérations impliquant des relations zonales ou des masques, lorsque la précision de cette matrice est suffisante pour l'opération en question, ainsi que pour améliorer les temps d'affichage sur écran ou pour rechercher les valeurs d'un objet en le désignant sur l'écran. Cette opération de *rasterisation* est transparente pour l'utilisateur.

De nombreuses opérations peuvent ainsi être effectuées sans avoir à reconstituer le chaînage des arcs en polygone, ou sans avoir à se préoccuper du problème des zones incluses. Cette opération de *rasterisation* crée une image, chaque pixel étant codé avec l'indice de l'objet dans le fichier descriptif. On peut ainsi retrouver très rapidement la valeur des attributs d'un point appartenant à une zone, sans avoir à recréer le contour de la zone et tester l'appartenance du point à la zone à partir de ses arcs frontière.

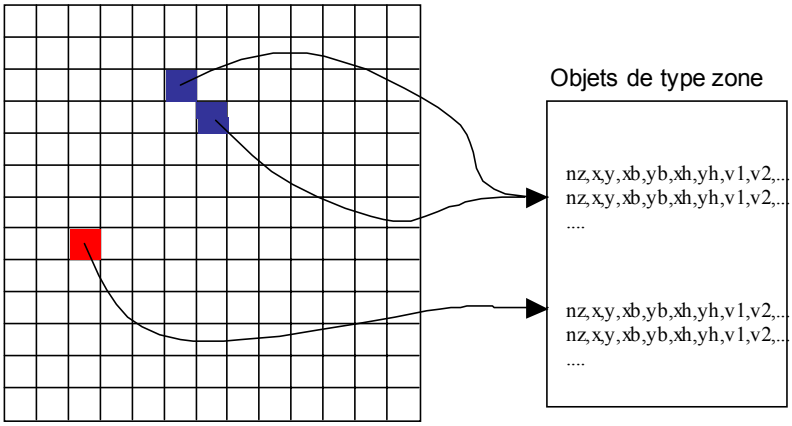


fig. 5.13 : structure de l'image raster associée à une relation zonale

La *rasterisation* est effectuée dans la fenêtre d'étude, et réactualisée à chaque changement de fenêtre d'étude et à chaque modification du fichier descriptif de la relation. La définition de l'image matricielle créée est un paramètre du système (habituellement fixé entre 1600 et 3000). Dans la plupart des cas, cette résolution

offre une précision suffisante. La précision absolue dépend bien sûr de la taille de la fenêtre d'étude : plus la fenêtre est grande, plus la taille du pixel résultant de la *rasterisation* sera petite. Le système conserve cette image dans un fichier temporaire, lié à la relation, et qui est détruit lors de la réinitialisation de la requête, la réinitialisation de la relation, le changement de la fenêtre d'étude, ou le changement de projection géographique.

L'avantage d'une représentation matricielle réside dans la simplicité de la structure, et dans le fait que l'ensemble des relations peuvent ainsi être comparées pixel par pixel, puisque, dans une fenêtre donnée, différentes *rasterisations* vont créer des pixels géométriquement identiques. Si l'utilisateur le souhaite, il peut également conserver la représentation matricielle d'un attribut dans une nouvelle relation, dont le type ne sera plus zone, ligne, point, mais *raster*. Ainsi, cette nouvelle relation nommée *raster*, toujours temporaire, pourra contenir, sous forme matricielle, des attributs provenant de toutes les relations de la base, et sur lesquelles on pourra appliquer très facilement de nombreuses méthodes d'analyse (comparaison, calculs, etc.). Cette relation temporaire *raster* s'apparente ainsi aux systèmes qui conservent la localisation sous forme matricielle, mais avec l'avantage d'avoir une taille de pixel variable, choisie par l'utilisateur, et une localisation provenant de données fiables conservées avec toute leur précision sous forme vectorielle. Cette particularité fait de SAVANE un système à double structure interne *raster-vecteur,* vecteur pour le stockage des objets, *raster* pour la résolution opérationnelle de certaines opérations d'analyse spatiale.

Cette propriété ne doit pas être confondue avec la gestion et le traitement des relations de type mosaïque, dont les objets sont également des pixels mais dont la taille ne varie pas et est fixée par la définition sémantique de la relation. Par contre, une relation de type mosaïque peut également apporter un attribut à la relation *raster* (dans ce cas, l'opération de conversion n'est plus une *rasterisation* mais un ré-échantillonage, voir chapitre 6). Elle ne doit pas non plus être confondue avec la possibilité de créer une relation vectorielle zonale dont les objets sont des mailles (carrée, hexagonale) que nous verrons au chapitre 8.

La représentation matricielle d'une relation zonale permet également d'accélérer la visualisation des zones par plage sur un écran : il suffit d'effectuer un ré-échantillonage de cette image en fonction de la taille du pixel de l'écran, plutôt qu'un chaînage des arcs en contours de taches et un remplissage de chaque tache, avec le problème toujours délicat des trous et des zones incluses. Pour une question de performance, cette représentation matricielle temporaire est donc presque indispensable lorsque la représentation de la localisation des zones utilise le mode topologique (frontières), comme c'est le cas dans le système SAVANE.

5.5.6.2. Un algorithme de rasterisation

L'opération de *rasterisation* étant effectuée de façon automatique en tâche de fond et à chaque modification de la relation, l'algorithme utilisé pour passer d'une description vectorielle à une représentation matricielle doit être très efficace. Nous en présentons ici les principes et la réalisation.

L'algorithme est basé sur le principe des algorithmes YX de *rasterisation* d'une zone (un point de l'intérieur d'une zone est caractérisé par un nombre impair d'intersections précédents le point, le long d'une droite horizontale coupant le contour de la zone). Cet algorithme est étendu à la *rasterisation* simultanée de plusieurs zones. Il comprend trois phases principales :

- calculs des points d'intersection des arcs avec des lignes horizontales séparées d'un pixel. La taille du pixel correspond à la résolution de la matrice à obtenir.

- tri des points d'intersection en Y, puis en X sur chaque ligne,

- pour chaque ligne du balayage, détermination de l'objet zone compris entre deux points d'intersection consécutifs.

Le calcul des points d'intersection des arcs doit gérer avec précision les segments horizontaux, à ne pas prendre en compte, ainsi que les extrémités des segments qui ne doivent pas être dupliqués. La méthode de tri des points doit être efficace.

Chaque arc des zones ayant une intersection avec la fenêtre d'étude est découpé en segments de droite, en reprenant directement les points représentant l'arc. Chaque segment est orienté (du bas vers le haut, et de gauche à droite), les segments horizontaux sont écartés, puis les intersections avec les lignes du balayage sont calculées et stockées par insertion (sur Y, puis sur X) dans une structure qui conserve également pour chaque point le numéro des zones adjacentes de l'arc, ou la valeur d'un attribut nominal de chaque zone adjacente. Le segment est écarté si les deux valeurs adjacentes sont identiques. On obtient donc pour chaque ligne une liste des points d'intersection se trouvant sur la ligne, triée par ordre croissant sur les abscisses. On détermine ensuite pour chaque point le code entrant et le code sortant, en fonction du nombre de points d'intersection confondus en chaque point d'intersection.

Cet algorithme, comme tous ceux de sa catégorie, exige une topologie sans faille : une zone ouverte crée une erreur dans la rasterisation et fait « déborder » la zone sur la zone adjacente.

Nous avons construit plusieurs classes pour mettre en œuvre cet algorithme : une classe `CCellule`, qui permet de stocker chaque point d'intersection et le chaîner avec le point suivant, et une classe `CYX`, qui permet de connaître l'adresse de la première cellule pour chaque ligne (membre `m_Y`) et qui regroupe les procédures de

calcul des points d'intersection (`Points()`), d'insertion des cellules (`InsererCellule()`), et de calcul du balayage (`YX()`) (A.2.2.2.).

L'image qui résulte de la *rasterisation* est stockée dans un fichier sous forme compressée, en utilisant une compression simple par segments horizontaux permettant de regrouper les pixels contigus de même valeur.

L'opération duale, permettant de passer d'une image à une représentation vectorielle des zones, correspond à une opération de vectorisation. Cette opération sera utilisée par exemple pour retrouver une représentation vectorielle après traitement sur une représentation *raster* de la localisation. Nous présenterons au chapitre 8 l'algorithme de vectorisation développé dans le système SAVANE.

5.5.7. Les masques

5.5.7.1. La structure d'un masque

Comme nous l'avons vu plus haut, un masque correspond à un domaine de l'espace. Chaque masque est représenté par une double structure, une image *raster* et un ensemble d'arcs qui correspondent aux contours du domaine. L'image *raster* est créée à la définition interne du système (la définition de *rasterisation*) suivant la projection géographique utilisée, et est stockée sous forme d'une image compressée formée de 0 (hors du domaine) et de 1 (dans le domaine). Les arcs sont conservés sous forme vectorielle, en coordonnées géographiques dans le *datum* de la base de données. La structure du fichier est identique à celle des fichiers des arcs d'une relation zonale. Chaque masque possède également un fichier qui le décrit (fenêtre minimale, nombre d'arcs, taille du pixel de l'image *raster* associée, etc.).

Toute modification de la fenêtre d'étude ou de la projection géographique implique la création d'une nouvelle image *raster* à partir des arcs, par *rasterisation*. On utilise l'algorithme de *rasterisation* que nous avons vu plus haut, simplifié, puisqu'il n'y a ici qu'une seule zone à traiter. Dans la majorité des cas, les traitements faisant intervenir un masque utiliseront la représentation matricielle du masque.

5.5.7.2. Création d'un masque : zone tampon ou définition sur écran

Un masque peut être créé principalement de deux façons : par saisie directe sur écran ou à partir des objets d'une relation.

La saisie directe sur écran ne pose pas de problèmes particulier : les contours du masque sont saisis sur l'écran, polygone par polygone. Les coordonnées sont conservées en longitude-latitude, après avoir été transformées à partir des

coordonnées écran (coordonnées écran → carte → cadre → géographique). L'image matricielle est calculée directement à partir des contours.

La définition d'un masque à partir des objets d'une relation est plus complexe, surtout lorsque l'on fait intervenir une distance : le masque correspond alors au domaine de l'espace se trouvant au maximum à telle distance des objets de la relation (appelé alors zone tampon, ou encore *buffer*) (fig. 5.14). Pour la création de ce type de masque dans SAVANE, on va d'abord créer l'image matricielle correspondante, et l'on obtiendra les arcs par vectorisation de l'image. L'algorithme de création dépend du type de la relation utilisée : pour une création à partir de points, on remplit l'image autour de chaque point en utilisant un disque de la taille du tampon. Pour une création à partir de lignes, on applique ce disque autour de chaque point provenant du dessin des ligne dans l'image *raster*. Pour une création à partir de zones, en utilisant l'image de la relation ligne par ligne : on applique ce même disque sur les extrémités de chaque segment horizontal de l'image, en joignant les deux disques par un rectangle.

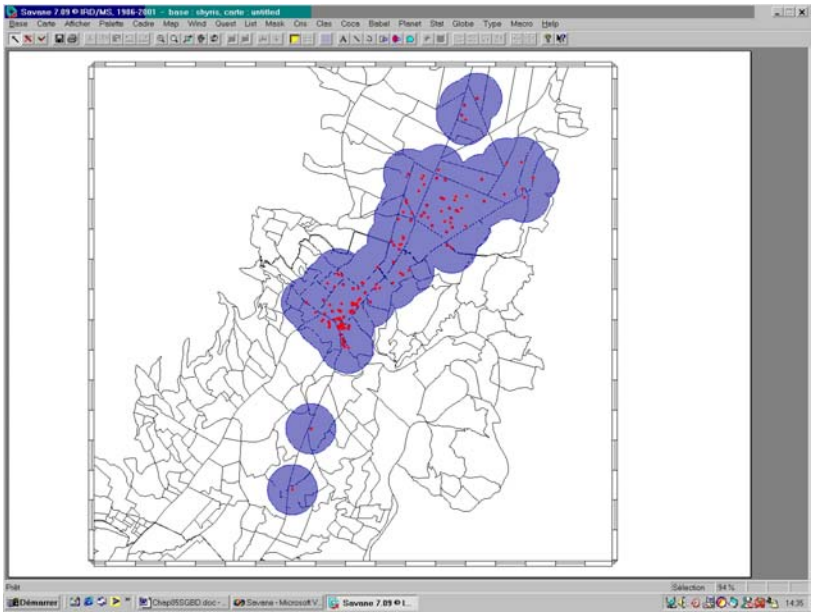


fig. 5.14 : la création d'un masque à partir des points d'une relation et d'une distance à ces points

5.5.7.3. L'algèbre des masques

De nombreuses opérations peuvent être appliquées aux masques dans SAVANE. Il s'agit d'opération unaires (inverser, éroder, dilater) et d'opérations binaires (intersection, union, union exclusive, différence). Toutes ces opérations utilisent la représentation matricielle du masque, et sont simples à mettre en œuvre. Seule la dilatation demande quelque effort algorithmique, nous utilisons ici la méthode présentée pour créer un masque avec une distance. L'érosion correspond à une dilatation de l'inverse du masque.

Grâce à ces opérations, nous pouvons obtenir des domaines de l'espace correspondant à de nombreuses situations géographiques. Par exemple, dans la figure 5.15, le domaine en bleu dans le cadre correspond à la zone se trouvant à plus de trois cent mètres mais à moins de cinq cent mètres d'un hôtel. Il a été obtenu par différence entre deux masques :

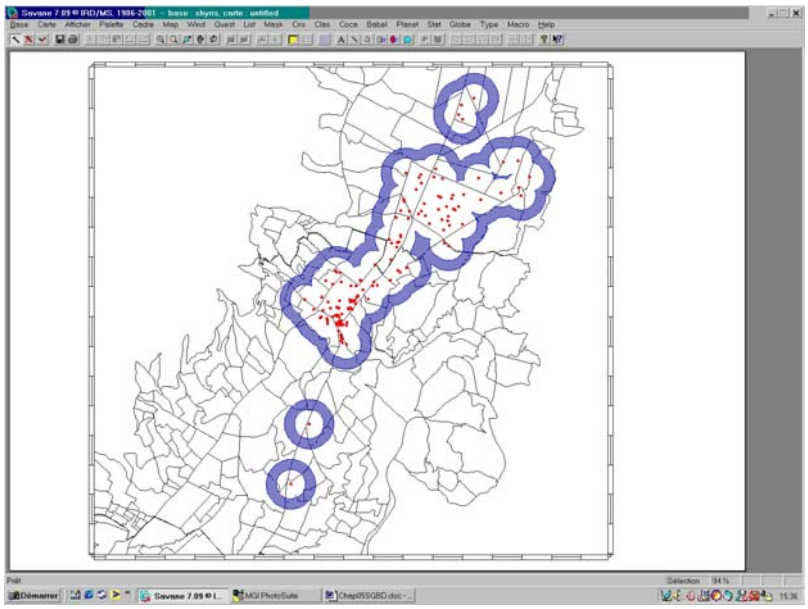


Fig. 5.15 : un masque résultat de la différence de deux masques

Les masques sont conservés dans le répertoire de l'utilisateur. Ils peuvent donc être utilisés dans toutes les cartes construites par cet utilisateur.

5.5.8. La réalisation des opérations relationnelles classiques

5.5.8.1. La restriction

La restriction d'une relation correspond à la sélection d'objets de la relation sur un critère descriptif. La condition de sélection peut porter sur plusieurs attributs : elle utilise une formule logique qui est ensuite vérifiée pour chaque objet (fig. 5.16). Si un objet répond à la condition, il est conservé et recopié dans l'état temporaire de la relation. A la fin de l'opération, la visualisation de la relation doit être actualisée pour refléter ce nouvel état.

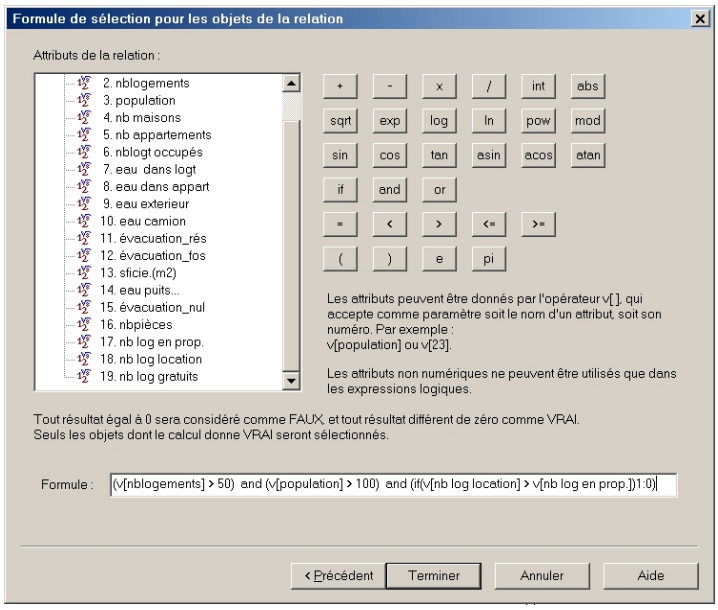


fig. 5.16 : dialogue de restriction par formule

L'opération de sélection utilise un calculateur permettant de vérifier la syntaxe de l'expression et de calculer sa valeur pour un objet. Nous avons développé une classe `CCalculateur` pour encapsuler l'ensemble de ces opérations (A.2.1.5.). La procédure de sélection s'écrit alors de façon très simple :

```
//--- initialisation de la lecture
int iNoFichier=pSchema->FchoixDescriptif();
if(iNoFichier == -1)return;

CLecture lecture;
if(lecture.Open(pCadre,iNoth,iNoFichier,0))
  {
  //--- lecture des objets
  float v[NB_MAX_ATTRIBUTS],fResultat;
  float vprim[NB_MAX_ATTRIBUTS];

  i=lecture.Lire(v);
  while(i != -1)
```

```
         {
         if(i == 1)
           {
           fResultat=(float) calculateur.Calcul(chFormule,v);
           if(fResultat != 0 && fResultat > FINCONNU)
             {
             iNbObjets++;
             if(pSchema->m_pRelations[iNoth]->m_iType < 10)
               {
               for(i=1;i <= iNba;i++)vprim[i]=v[tabp[i]];
               lecture.Ecrire(vprim);
               }
             else lecture.Ecrire(v);
             }
           }
         i=lecture.Lire(v);
         }
       lecture.Close();

       //--- mise a jour de la relation
       pSchema->m_pRelations[iNoth]->m_iNbObjets=-1;
       if(pSchema->m_pRelations[iNoth]->m_iType < 10)
         {
         pSchema->m_pRelations[iNoth]->m_iType+=10;
         pSchema->m_pRelations[iNoth]->m_iNoFichDescriptif=iNoFichier;
         pSchema->m_iFdisDescriptif[iNoFichier]=1;
         for(i=1;i <= pSchema->m_pRelations[iNoth]->m_iNba;i++)pSchema->m_pRelations[iNoth]-
>m_pAttributs[i]->m_iNumero=i;
         pCadre->GetDlgStatExplorateur()->MiseAJour(iNoth);
         }
       else
         {
         int iNo=pSchema->m_pRelations[iNoth]->m_iNoFichDescriptif;
         unlink(pSchema->m_chFprovDescriptif[iNo]);
         pSchema->m_pRelations[iNoth]->m_iNoFichDescriptif=iNoFichier;
         pSchema->m_iFdisDescriptif[iNoFichier]=1;
         pSchema->m_iFdisDescriptif[iNo]=0;

         //--- suppression de l'image si type 24
         if(pSchema->m_pRelations[iNoth]->m_iType == 24)
           {
           pSchema->m_pRelations[iNoth]->m_iType=14;
           int iFim=pSchema->m_pRelations[iNoth]->m_iNoFichImage;
           pSchema->m_iFdisImage[iFim]=0;
           }
         //--- actualisation de l'explorateur cartographique
         if(pCadre->GetExplorateur()->IsRelationInside(iNoth))
           {
           pCadre->Effacer();
           carte.TracerUnCadre(pCadre);
           }
         }
```

La restriction sur un attribut nominal peut également être effectuée en indiquant uniquement les modalités de l'attribut à conserver. Le dialogue est alors différent, et la restriction ne porte que sur un attribut (fig. 5.17).

Après avoir retrouvé et conservé le code interne des modalités sélectionnées, la relation est restreinte en ne conservant que les objets correspondant à ces valeurs. Le principe de l'implémentation est identique à la sélection par formule.

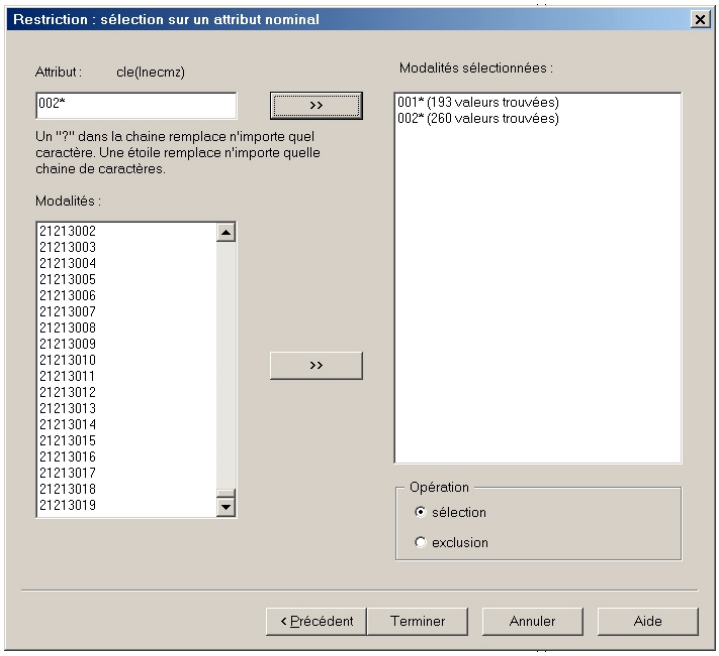


fig. 5.17 : dialogue de restriction sur un attribut nominal

5.5.8.2. La jointure

La jointure de deux relations correspond à la restriction du produit cartésien des deux relations sur un critère descriptif. Le résultat est une relation non localisée, quel que soit le type des deux relations initiales. Une nouvelle relation non localisée est créée, elle contient les objets répondant au critère de jointure. Les attributs proviennent des deux relations initiales.

Si l'on veut conserver la localisation de l'une des relations initiales, il faut être sûr de ne pas dupliquer les objets lors de la jointure. C'est pour cela que nous avons développé une autre opération, appelée *union*, qui s'apparente à une jointure avec les différences suivantes :

- un objet de la première relation est conservé dans la relation résultante si il répond au critère de jointure avec au moins un objet de la seconde relation ;

- la valeur des attributs provenant de la seconde relation correspond à la valeur des attributs du premier objet trouvé dans la seconde relation satisfaisant au critère de jointure.

Cette opération est notamment très utile lorsque le critère de jointure est une égalité (équi-jointure) et qu'il porte sur une clé de la seconde relation, car l'on sait

alors que le résultat de la jointure ne crée pas de doublons : au maximum un seul objet de la seconde relation pourra être mis en relation avec un objet de la première relation. Le résultat d'une union est alors identique au résultat d'une jointure, mais permet de conserver la localisation de la première relation.

L'implémentation de cette opération dans le système SAVANE est donnée dans l'annexe A.2.5.1.

5.5.9. La réalisation des opérations relationnelles étendues

5.5.9.1. La restriction spatiale

La restriction spatiale correspond à la sélection d'objet sur un critère de localisation. Le système SAVANE permet d'effectuer des restrictions spatiales par rapport à une fenêtre géographique rectangulaire (fenêtre de travail) et des domaines de l'espace définis sous forme de masques. L'opération de sélection recopie les objets sélectionnés dans un nouveau fichier descriptif temporaire. Si la relation est visualisée dans le cadre, le dessin est automatiquement réactualisé. Pour les types zone et ligne, le fichier des arcs n'est pas modifié, sauf dans le cas d'une restriction par domaine avec redéfinition de la localisation, comme nous allons le voir un peu plus loin.

5.5.9.1.1. La restriction spatiale par fenêtre d'étude

Chaque cadre, correspondant à une requête, possède une fenêtre de travail. Quel que soit l'opération effectuée dans le système, une restriction spatiale sera automatiquement effectuée dans cette fenêtre. Pratiquement, le système utilise l'indexation spatiale en feuille et n'accède qu'aux feuilles qui ont une intersection non vide avec la fenêtre de travail. Ensuite, il n'accède qu'aux objets dont la fenêtre a une intersection non vide avec la fenêtre de travail. L'indexation spatiale et la fenêtre d'étude permettent d'améliorer considérablement les performances du système, lors de l'interrogation des données.

Lors de la création du cadre, cette fenêtre correspond à l'espace géographique donné lors de la définition de la base. On peut changer cette fenêtre par de nombreuses méthodes : coordonnées géographiques, coordonnées de projection, espace occupé par les objets d'une relation, choix sur écran, déplacement, etc. Chaque changement de fenêtre implique normalement une réinitialisation de la requête. Cette réinitialisation doit être faite par l'utilisateur lui-même.

5.5.9.1.2. La restriction spatiale par domaine

La restriction spatiale par domaine utilise un masque. Pour une relation, elle permet de sélectionner les objets se trouvant dans ou hors du domaine représenté par

le masque. Cette restriction ne pose pas d'ambiguïté pour les objets de type point. Par contre, pour les types ligne ou zone, il faut choisir entre une restriction ensembliste ou une restriction spatiale. On a donc le choix entre une sélection par centroïde (sélection de l'objet si son centroïde se trouve dans le masque), une sélection ensembliste (sélection de l'objet si l'intersection avec le masque est non vide), et une sélection avec redéfinition de la localisation de l'objet (sélection de l'intersection).

La sélection par centroïde ne pose aucune difficulté de réalisation. On utilise la représentation matricielle du masque pour tester l'appartenance du centroïde au masque, en testant dans le masque la valeur du pixel correspondant à la localisation du centroïde de l'objet.

La sélection ensembliste est un peu plus complexe. Pour les zones, on utilise la représentation matricielle de la relation et du masque, ce qui permet de comparer les images pixel par pixel et tester l'intersection d'une zone avec le masque. On conserve la zone si l'intersection est non vide. Pour les lignes, on teste directement l'intersection de la ligne avec le masque en calculant dans le repère matriciel du masque les pixels correspondant à la ligne.

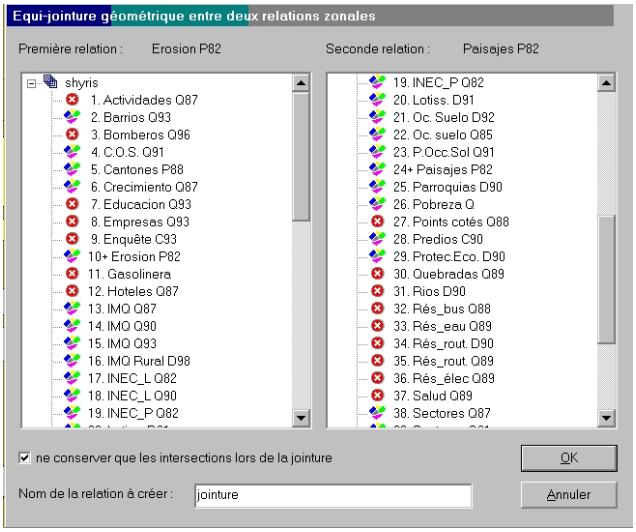
La sélection de l'intersection est un peu plus difficile à mettre en œuvre, puisqu'elle impose une redéfinition de la géométrie des objets résultants de l'opération. Pour les relations de type zone, deux méthodes sont disponibles pour cette opération : une méthode rapide en *raster*, une méthode plus longue mais plus précise en vecteur. La méthode *raster* utilise la représentation matricielle de la relation et du masque : la comparaison pixel par pixel est rapide et permet d'obtenir rapidement une image de la relation qui ne conserve que les pixels correspondant à la valeur 1 du masque. Cette image est ensuite balayée pour obtenir la liste des objets existants et créer le nouveau fichier descriptif à partir de l'ancien. Enfin, l'image est vectorisée pour créer un nouveau fichier d'arcs correspondant aux contours de ces nouvelles zones. La méthode vectorielle utilise directement les arcs en calculant les intersections des arcs des zones avec les arcs du masque. Les zones sont ensuite reconstituées à partir des bouts ainsi formés. L'algorithme classique d'intersection d'arcs segment par segment est utilisé (voir chapitre 3). Il est fortement accéléré par la prise en compte de la fenêtre des arcs, qui permet d'éliminer de nombreuses comparaisons inutiles.

5.5.9.2. Les jointures spatiales

L'implémentation des jointures spatiales dépend du type des deux relations et de la qualification de jointure. Les seules jointures qui ne font pas perdre la localisation sont les jointures avec une qualification d(O1,O2) = 0, le type du résultat correspondant alors à celui de l'intersection des objets. Ainsi, le résultat d'une jointure zone-zone avec une qualification d(O1,O2) = 0 est une relation de type

zone : les objets correspondent aux intersections des objets des deux relations. Le résultat d'une jointure zone-point est une relation de type point, et le résultat d'une jointure zone-ligne est de type ligne.

Comme pour les restrictions de domaine, deux méthodes sont disponibles pour la réalisation des jointures spatiales lorsque l'une des relations est de type zone. La méthode *raster* compare les images des relations, l'image de la relation résultante correspondant à l'intersection des deux images (fig. 5.18). Chaque zone du résultat récupère les attributs des deux zones initiales. L'image du résultat est vectorisée pour créer les contours des arcs correspondant aux zones de la jointure.



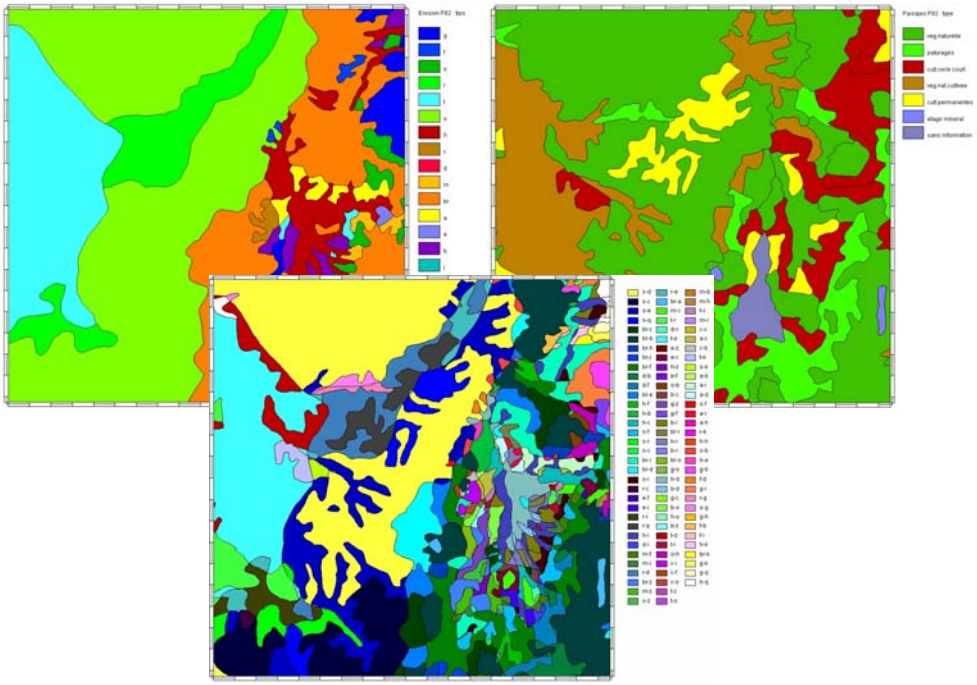


fig. 5.18 : une équi-jointure spatiale zone-zone

La réalisation de la jointure se déroule ainsi en plusieurs phases :

- création d'une nouvelle image par calcul pour chaque pixel d'un code résultant des valeurs du même pixel dans les images des deux relations initiales,

- création des *tuples* descriptifs correspondant aux combinaisons rencontrées dans la première phase (ce qui revient à éliminer les *tuples* descriptifs en double),

- établissement de la relation image-*tuples*,

- vectorisation de l'image.

On trouvera en annexe le code de la procédure d'équi-jointure géométrique entre deux relations de type zone, méthode *raster* (A.2.5.2.).

La méthode vectorielle crée les intersections de zones à partir des arcs de l'ensemble des zones initiales, puis crée les *tuples* descriptifs correspondant aux objets graphiques ainsi créés.

Certaines jointures zone-zone créent de nombreuses petites zones qui n'ont pas de sens sémantique car elles proviennent uniquement des différences de précision de la représentation géométrique et peuvent être considérées comme des artefacts. On

peut supprimer ces zones parasites en éliminant dans la nouvelle relation les zones de trop petite taille, ou celles dont le rapport périmètre/surface est trop faible.

5.5.9.3. Les semi-jointures spatiales

Pour éviter la perte de la localisation de l'une des deux relations, nous avons développé dans le système SAVANE plusieurs procédures qui sont des semi-jointures géométriques suivies d'une agrégation des valeurs descriptives sur la clé de localisation initiale. De même, nous avons développé des semi-jointures classiques qui évitent les doublons d'objets géographiques. Ces opérations, liées à l'analyse spatiale, seront présentées au chapitre 8.

5.5.10. La connexion à d'autres systèmes de gestion de base de données

Le système SAVANE dispose de son propre moteur de base de données relationnel étendu, qui lui permet de gérer de manière optimale les données localisées. La plupart des SIG ne fonctionnent pas de cette manière : ils utilisent un système relationnel classique pour gérer les attributs descriptifs et ne gèrent que l'information de localisation des objets.

Ces deux approches ont leurs avantages et leurs inconvénients. Les avantages sont évidents : on dispose de la puissance et de la fiabilité des SGBD relationnels commerciaux. Les inconvénients sont également nombreux : toute opération nécessite l'union des objets géré par le SIG et ceux, non localisés, gérés par le SGBD classique. Une requête un peu complexe demande donc plusieurs aller et retour entre les deux systèmes, lorsque cela est possible.

Nous avons développé dans SAVANE une solution qui permet de bénéficier des avantages des deux systèmes : des attributs descriptifs peuvent être importés des principaux SGBD relationnels (via ODBC, DAO, ou ADO) temporairement dans le SGBD directement géré par SAVANE. Si l'importation est temporaire, les attributs ne sont utilisés que lors de la requête, mais une fois importés, ils sont considérés comme les autres attributs et gérés directement par le système. Il n'y a donc qu'une seule connexion au SGBD externe. Cette particularité fait de SAVANE virtuellement un SGBD à double moteur de gestion : un moteur interne basé sur une indexation primaire bidimensionnelle, et le moteur externe du système externe associé.

5.6. Conclusion

L'extension du modèle relationnel au données localisées constitue le premier objectif de notre travail : bénéficier de la simplicité et de la puissance du modèle

relationnel pour la gestion des données géographiques. Nous avons ainsi défini de nouvelles opérations permettant d'étendre l'algèbre relationnelle sur l'attribut de localisation. Ces opérations utilisent la structure métrique et la distance euclidienne pour étendre les opérations classiques de l'algèbre relationnelle (restriction, projection, jointure) . D'autres opérations pourraient ainsi être définies, en utilisant la structure ensembliste (inclusion, intersection, appartenance) ou la structure topologique (relations d'adjacence, par exemple).

La jointure spatiale étend formellement l'opération classique de jointure de l'algèbre relationnelle, mais la validité de son emploi s'avère beaucoup délicate que dans le cas classique, où l'opération ne présente aucune ambiguïté. En effet, comme nous l'avons vu au chapitres 3 et 4, l'attribut de localisation n'a pas toujours la même précision et la même validité. L'opération de jointure telle que nous l'avons définie ignore ces différences. Elle est donc peu adaptée à de nombreuses situations, surtout lorsque la jointure implique des relations dont l'implantation spatiale n'a pas le même degré de validité. Pour construire un système performant, il nous faudra donc introduire de nouvelles opérations permettant de résoudre ces problèmes de *transfert d'échelle* et de mise en relation d'objets de précision géographique différentes. Ces opérations sont pratiquement toutes apparentées à une jointure spatiale suivi d'une opération de regroupement, mais nous les présenterons dans le chapitre 8 consacré à l'analyse spatiale : plus que des opérations de gestion, elles utilisent une mise en relation sur la localisation plus pour résoudre des problèmes d'analyse que pour répondre à un besoin de bonne gestion de données.

Bibliographie

[ABI 95] ABITEBOUL S., HULL R., VIANU V., *Foundations of Databases*, 1995, Addison-Wesley

[BER 93] BERTINO E., MARTINO L., *Object-Oriented Database Systems*, 1993, Addison-Wesley

[DAT 95] DATE C.J., *Introduction to Database Systems*, 1995, Addison-Wesley

[DAV 91] DAVID B., *Modélisation, représentation et gestion d'information géographique, une approche en relationnel étendu*, Thèse de doctorat, 1991, Université Paris VI

[DEL 91] DELOBEL C., LECLUSE C., RICHARD P., *Bases de données : des systèmes relationnels aux systèmes à objets*. 1991, InterEditions, Paris

[EGE 91] ENGENHOFER M., *Spatial Query langages*. Thèse de doctorat, 1991, University of Maine, USA

[FLO 93] DE FLORIANI L., MARZANO P., PUPPO E., *Spatial Queries and Data models*, in Frank A. and Campari I.U., Spatial Information Theory, Lecture Notes in Computer Sciences 716, 1993, Springer-Verlag, p. 113-138

[GAR 83] GARDARIN G., *Bases de données : les systèmes et leurs langages*, 1983, Eyrolles, paris

[GRE 89] GREENE D., *Implementation and Performance Analysis of Spatial Data Access Methods*, IEEE, Intl. Conference on Data Engineering, 1989, p.606-615

[GUN 89] GÜNTHER O., *Efficient Computation of Spatial Joins*. IEEE, Intl. Conference on Data Engineering, 1989, p.50-59

[GUT 88] GÜTING R.H., *Geo-Relational Algebra : A Model and Query Language for Geometric Database System*, Proccedings of Intl. Conference on Extending Database Technology, 1988, p. 506-527

[HAS 82] HASKIN R.L. AND LORIE R.A., *On extending the functions of a relational database system*. Association for Computing Machinery, Proccedings of SIGMOD, NY : ACM Press, 1982, p. 207-212

[LAR 93] LARUE T., PASTRE D., VIÉMONT Y., *Strong Integration of Spatial Domains and Operators in a Relational Database System*, Intl. Symposium on Large Spatial Databases, LNCS n° 692, 1993, p. 53-72, Springer-Verlag

[LAU 93] LAURINI R., MILLERET-RAFFORT F., *Les bases de données en géomatique*, Hermès, 1993, Paris

[LOR 84] LORIE R.A., MEIER A., *Using a Relational DBMS for Geographic Databases*. GeoProcessing, 1984, p.243-257

[NEW 79] W. M. NEWMAN, R.F. SPROULL, *Principles of Interactive Computer Graphics*, 1979, Mc Graw Hill

[OOI 89] OOI B.C., SACK-DAVIS R., MC DONELL K.J., 1989, *Extending a DBMS for Geographic Applications*, Intl. Conference on Data Engineering, p. 590-597

[PAV 82] PAVLIDIS T., *Algorithms for Graphics and Image Processing*, Computer Science Press, 1982

[SAM 80] SAMET H., *Tree Structure for region representation*, Aangeenbgug, Autocarto 4, vol. 1, 1980, pp 108-118

[SOU 86] SOURIS M., *Systèmes d'information géographique et bases de données*, Colloques et Séminaires, Traitement des données localisées, 1986,Editions de l'Orstom

[SOU 87] SOURIS M., *A Geographic Information System with relational architecture : principles and example of use*. in Primera conferencia sobre informatica y geografia, San Jose, Costa Rica, 1987, pp 703-728

[ULL 88] ULMANN J.D., *Principles of Database and Knowledge-Base Systems*, 1988, Computer Science Press

[WOR 90] WORBOYS M.F., HEARNSHAW H.M., MAGUIRE D.J., *Object-Oriented data Modelling for Spatial Databases*, International Journal of Geographic Information Systems, 1990, vol. 4, p. 369-383

Chapitre 6

Géo-référencement et intégration d'images

L'objet de ce chapitre est de montrer comment les images numériques (télédétection spatiale, photographies aériennes, modèles numériques, cartes scannées) peuvent être intégrées en tant qu'instances de classes d'objets géographiques dans une base de données relationnelle et bénéficier ainsi de la gestion du système d'information géographique au même titre que les autres classes d'objets localisés (zones, lignes, points).

Nous introduirons le type d'objet défini pour gérer les images, puis les différentes méthodes utilisées pour créer les objets pixels dans un SIG, par redressement et mosaïquage des images brutes ou par création directe à partir d'objets d'un autre type. Nous passerons rapidement en revue les différentes opérations pouvant être utilisées dans le cadre de la gestion relationnelle des objets de type pixel, ainsi que les opérations permettant de passer du type pixel aux autres types d'objets localisés. Enfin, nous montrerons comment la définition de classes d'objet pour les principaux types de capteurs de données géographiques par balayage permet d'encapsuler définition des objets et méthodes d'utilisation de ces objets, facilitant ainsi l'exploitation des ces données dans le cadre du SIG. Le système peut ainsi proposer des schémas de données incluant de nombreuses méthodes associées aux attributs, et se charge de l'ensemble de la gestion de données souvent très volumineuses. Enfin, nous exposerons la mise en œuvre de ces principes et méthodes dans le système d'information géographique SAVANE.

6.1. Introduction

Les données de type image sont souvent considérées comme des données un peu à part dans les SIG, ce qui donne lieu à de nombreuses ambiguïtés sur la validité des traitements effectués par rapport aux autres objets de la base de données. Cette approche ne peut nous satisfaire : il est nécessaire de recourir à un formalisme plus précis, et il convient d'appliquer aux données provenant d'un capteur de type *raster* les mêmes critères qu'aux autres données géographiques [EST 92].

On devra ainsi définir exactement quel est l'objet traité par le SIG. Le pixel d'une image brute est en effet rarement un objet localisé de façon absolue avec précision : la position d'un pixel n'intervient souvent dans une image que de manière relative, par rapport à ses voisins. La plupart des opérations en télédétection et traitement d'image ne considèrent que l'agencement des pixels entre eux, et ne nécessitent pas de géoréférenciation absolue de chaque pixel. Mais lorsque l'on veut joindre ces pixels avec les objets d'une base de données localisées, le premier problème est de localiser les éléments d'une image avec une précision connue. Pour cela, on se rend vite compte que le pixel de l'image brute ne peut être l'objet final. Il est nécessaire de définir un nouvel objet, connu et parfaitement localisé (avec une précision géographique donnée), et de calculer pour cet objet la valeur d'un attribut en utilisant les données de l'image, par une opération de ré-échantillonnage. Mais il est également important de conserver l'information de voisinage présente dans l'image de départ, car l'on connaît la précision de voisinage grâce au capteur (par exemple, le satellite SPOT permet une précision relative de 10 mètres entre les pixels; une photographie aérienne peut être scannée avec une précision relative de un mètre, etc.). La précision n'est bien sûr qu'un ordre de grandeur, puisque la projection géographique des images est en général inconnue (les images sont par définition projetées, puisque elles sont données dans un plan, mais les caractéristiques de cette projection sont inconnues - elle ne pourrait être trouvée qu'au moyen de la résolution d'équations à un grand nombre d'inconnues). On ne peut donc parler de l'unité des coordonnées de cette projection inconnue.

On a deux critères pour définir les objets : un critère de taille pour conserver l'information de voisinage, et un critère de calcul pour conserver l'information descriptive (la valeur physique correspondant au signal). On peut choisir des objets qui existent déjà (des zones par exemples), et calculer une valeur descriptive pour ces objets (mais on perd en général l'information de voisinage). On peut choisir de créer des objets simples (de nouveaux des pixels, mais cette fois-çi considérés comme des mailles), dont on définira exactement les caractéristiques et dont on connaîtra alors la position exacte dans un plan de projection donné. Toute la difficulté vient alors de définir la méthode permettant d'associer à ces objets une valeur descriptive venant de l'image d'origine, par redressement et ré-échantillonnage. Pour cela, il faut essayer de savoir quels sont les pixels de l'image

d'origine qui sont candidats à intervenir dans la valeur d'un objet d'arrivée, et il faut définir la méthode à employer pour associer une valeur descriptive à l'objet à partir de ces pixels d'image.

Nous allons donc définir dans le schéma relationnel un type d'objet pour gérer les images, puis introduire les différentes méthodes utilisées pour créer les objets dans un SIG, par redressement et mosaïquage des images brutes, ou par création directe à partir d'objets d'un autre type. Nous présenterons les principes de la gestion des relations de type mosaïque dans SAVANE, ainsi que les différentes opérations pouvant être utilisées dans le cadre de la gestion relationnelle. Enfin, nous présenterons en détail le module SAVAMER qui permet d'effectuer le redressement des images et la constitution de mosaïques dans le système SAVANE.

6.2. Définition d'une classe d'objet pour les données venant d'images : le pixel

6.2.1. Définition de l'objet

Nous allons définir une nouvelle classe d'objet localisé pour gérer les images géoréférencées. D'abord, nous pouvons remarquer que chaque objet est un ensemble de points de l'espace, correspondant à un élément de l'image : ce sont des zones. La géométrie des objets est la plus simple possible, et est identique pour tous les objets de la classe; la plupart du temps, elle ne sera même pas prise en compte dans les traitements, et la zone sera assimilée à son centroïde. C'est justement cette ambiguïté qui ne doit pas faire oublier la définition des objets, en terme de précision de leur localisation.

Par définition, l'objet est une zone carrée, qui est appelée *pixel* par similitude avec les éléments d'image, et sa localisation est donnée par la position de son centre et la taille exacte du coté du carré dans un plan de projection. Comme tous les objets ont la même taille, il suffit de connaître la position absolue d'un seul objet pour déterminer la position de tous les objets, si leur position peut être donnée relativement à cet objet. L'agencement des objets en grille permet donc de connaître la localisation de chaque objet, pour peu que l'on connaisse la position d'un seul objet, la taille commune des objets, et le nombre de colonnes de la grille. Cette description est donc très efficace en terme de mémoire et de sélection des objets sur un critère de localisation, puisqu'il suffit de stocker les valeurs des pixels et que l'agencement de ces valeurs dans la structure de stockage est suffisant pour connaître la localisation des objets. On évite ainsi de conserver les coordonnées du centre de chaque objet.

Les attributs correspondent aux valeurs affectées à chaque pixel. Les données provenant des images brutes peuvent être de différents types, correspondant à des

attributs nominaux, des attributs entiers (codés sur 1, 8, 16 ou 32 bits), réels, représentant une couleur RGB (24 bits) ou RGBK (32 bits).

Par exemple, une image SPOT fournit trois ou quatre attributs, de type entier codé sur 8 bits (256 valeurs maximum), une image TM en fournit sept, une image scannée en vraie couleur en fournit un de type RGB, etc.

Une relation de type pixel sera donc définie par les critères suivants :

- la taille des objets (en fait la taille du coté du pixel, en mètre) ;
- le plan de projection utilisé (en fait les paramètres de la projection géographique) ;
- les attributs descriptifs.

6.2.2. Gestion des collections de pixels

La localisation d'un objet de type pixel est donc donnée par la position de la valeur de ses attributs dans une grille. Mais le stockage des pixels en grille pose un problème quant à l'existence des objets, car il suppose l'existence de tous les pixels de la grille. Il faut donc ajouter un attribut booléen indiquant l'existence réelle de tel objet dans la relation. Une grille est toujours rectangulaire : pour éviter une importante perte de volume de stockage lorsque le domaine d'existence des objets est de forme quelconque, il convient de trouver une structure qui minimise la place perdue dans la grille pour les pixels non existants, tout en conservant la simplicité du stockage en grille.

Nous avons donc choisi naturellement de diviser l'espace en petites régions (imagettes) et d'indexer ces imagettes. L'ensemble des imagettes constitue donc une mosaïque, et l'ensemble des pixels des imagettes dont l'attribut d'existence est vrai correspond à l'ensemble des objets de la relation. La position d'un pixel quelconque de la mosaïque est donnée par sa position dans l'imagette auquel il appartient, et par la position absolue d'un pixel de l'imagette. Chaque imagette sera donc caractérisée par ces trois paramètres : position absolue d'un pixel dans le plan de projection (on prendra le pixel bas-gauche), nombre de colonnes et nombre de lignes de l'imagette. L'existence des objets est donnée par la valeur de l'attribut d'existence. Les valeurs des attributs des objets sont données par les valeurs de l'imagette correspondant à la position de l'objet dans l'imagette (on a donc une mosaïque d'imagettes par attribut).

Les opérations relationnelles (restriction, jointure) utilisent en priorité l'attribut d'existence. La restriction correspond par exemple à une modification de la valeur cet attribut.

Les relations de type pixel seront donc composées de :

- une liste des imagettes existantes, décrivant pour chaque imagette la localisation du coin bas gauche du pixel bas gauche, le nombre de colonnes et de lignes, et l'adresse du début de l'imagette dans l'ensemble des grilles de valeurs d'attributs ;

- pour chaque attribut, un ensemble de grilles de valeurs.

6.3. Géoréférencement des images et création des objets

Une fois défini le schéma d'une relation de type pixel, les objets peuvent être créés à partir d'autres objets de la base de données (par changement de type, par exemple par interpolation ou par rasterisation), mais le plus souvent, ces objets seront créés à partir d'images numériques provenant d'un capteur (photographie numérique, scanner, satellite, etc.).

Pour chaque pixel potentiel de l'espace, il est alors nécessaire de répondre à plusieurs questions :

- un objet doit-il être créé ?

- quelles valeurs pour les attributs, à partir des valeurs des pixels de l'image d'origine ?

L'opération de géoréférencement d'une image permet de répondre à ces questions. Cette opération vise à créer une nouvelle image possédant les caractéristiques de taille et de projection géographique de la relation, et permettant de connaître la localisation exacte de chacun de ses pixels. Lors de l'intégration de cette image dans la relation, seuls les pixels présents dans les imagettes ayant une intersection avec l'image géoréférencée seront physiquement créés dans la relation. L'attribut d'existence des pixels créés mais non présents dans l'image géoréférencée sera initialisé à FAUX.

La création d'objets de type pixel dans la base de données à partir d'une image correspond donc à deux opérations :

- la géoréférenciation de l'image d'origine, par redressement géométrique et ré-échantillonnage ;

- l'intégration des pixels de l'image géoréférencée dans la relation, par mosaïquage.

6.3.1. Redressement d'image

Les images obtenues par un capteur ne sont pas directement géo-référencées : elles comportent des déformations globales ou locales et doivent être redressées pour être mises au mieux en conformité avec la réalité géographique, dans un plan de projection. Nombreuses sont les causes de déformation [NOV 92]. Pour les photographies aériennes ou les images satellites, on peut citer par exemple les déformations dues à l'optique de prise de vue, aux conditions atmosphériques, à la position de l'instrument de prise de vue, au relief de la surface, à la projection géographique [HOT 99].

Une fois définis les paramètres de l'image d'arrivée (taille du pixel et plan de projection), le problème du redressement consiste à trouver le ou les pixels de l'image d'origine qui correspondent à un pixel de l'image d'arrivée. Le choix des fonctions à utiliser pour assurer au mieux cette correspondance dépend du type de déformation présumée, des possibilités de prise de points de correspondance entre image d'origine et image d'arrivée (points d'appui ou amers), et de la disponibilité d'un modèle numérique de terrain. Corriger géométriquement une image revient toujours à déterminer un modèle de déformation entre les coordonnées dans l'image brute et les coordonnées dans le système de référence utilisé pour la relation à créer.

Trois cas peuvent se présenter :

- on connaît les modèles de déformation avec une grande précision. Dans ce cas, il suffit de modéliser le redressement en prenant en compte les déformations connues ;

- on ne connaît pas les modèles de déformation. On modélise alors arbitrairement ces déformations avec une fonction donnée, en général polynomiale. On calcule les coefficients du modèle à l'aide de points d'appui, points dont on connaît les coordonnées dans les deux repères (image et système de référence de la relation à créer) ;

- on connaît les modèles de déformation, mais avec une précision insuffisante. On ajuste alors le modèle à l'aide de points d'appui. Le nombre de points d'appui nécessaires dépend alors du modèle.

Les deux derniers cas sont les plus courants en pratique. Les possibilités de détermination de points d'appui sont donc fondamentales. Cette opération revient à localiser (dans le plan de projection) des points de l'image d'origine.

Les modèles de déformation et les méthodes de redressement qui en découlent sont nombreux : ils dépendent essentiellement du type de déformations auxquelles l'image à redresser est soumise, des possibilités de prise de points d'appui, de la disponibilité d'un modèle numérique de terrain [HOT 99]. Ils dépendent également

de la précision souhaitée par rapport aux données d'origine. Dans la pratique, on utilise les fonctions suivantes :

- une translation : dans ce cas, l'image d'origine est déjà conforme à la projection géographique, la taille du pixel est connue, la translation n'est utilisée que pour localiser l'image mais n'effectue aucune transformation sur les pixels. Un seul point d'appui est nécessaire pour effectuer une translation.

- une translation et une rotation : c'est la transformation la plus simple, à utiliser dans le cas où l'image d'origine est géographiquement correcte mais doit subir une translation et une rotation pour être en conformité avec le repère de la projection. Les distances dans l'image ne sont pas modifiées. Deux points d'appui sont nécessaires pour effectuer cette transformation.

- une similitude : c'est une translation et une rotation suivie d'une homothétie (mise à l'échelle). La similitude est à utiliser lorsque, pour être mise en conformité avec une projection géographique, l'image doit subir une translation, une rotation, puis une mise à l'échelle. La mise à l'échelle est identique sur l'ensemble de l'image, la déformation est identique quelque soit la direction. Deux amers sont nécessaires pour cette transformation.

- une déformation polynomiale de degré 1 : cette déformation est identique à la similitude, sauf pour la mise à l'échelle qui n'est pas identique quelque soit la direction. Trois points d'appui sont nécessaires pour effectuer cette transformation.

- une déformation projective : c'est une perspective centrale, déformation naturelle idéale obtenue sur une photographie, lorsque le terrain photographié correspond à un plan. Au minimum six points d'appui sont nécessaires pour fixer les coefficients de cette transformation, ce qui revient à positionner le faisceau projectif dans l'espace. Lorsque la distance aux objets est grande par rapport à la taille des objets, la perspective centrale peut être approchée par une déformation polynomiale de degré 1. Lorsque l'on redresse une image de la Terre, cette transformation doit être combinée à une déformation prenant en compte l'altitude et la courbure de la Terre, sinon l'échelle obtenue n'est pas correcte. En effet, la déformation précédente ne tient pas compte de l'altitude des points dans l'image : les points sont projetés dans le plan comme s'ils étaient tous à l'altitude zéro, alors que la projection dans le plan doit suivre une verticale, et non pas un rayon lumineux, à partir de l'altitude supposée du point. Pour prendre en compte cette déformation, il faut connaître le relief en tout point et donc disposer d'un modèle numérique de terrain.

- une déformation par grille ou par triangulation : cette déformation combine une première transformation globale (rotation, similitude, polynomiale, perspective centrale), avec une déformation locale (en général polynomiale) dans chaque triangle résultant d'une triangulation à partir des points d'amers saisis. Cette transformation est très efficace lorsque l'on ne dispose pas d'un modèle numérique de terrain permettant de connaître l'altitude en chaque point. En effet, elle établit un modèle de déformation dans chaque triangle. La déformation correspond à un

découpage de l'espace en facettes (planes dans le cas d'un polynôme de degré 1), et si le réseau de point d'amers est dense et homogène (et d'autant plus dense que les différences d'altitude sont grandes), le redressement permet d'obtenir directement une image en conformité avec la projection géographique choisie. Ce type de déformation permet également d'assurer une jointure parfaite entre différentes images : une fois une image redressée, il suffit de saisir des points d'amers entre l'image redressée et l'image à recaler pour faire coïncider les deux images. La transformation initiale permet de caler grossièrement l'image à redresser : elle doit correspondre au mieux au type de déformation globale à laquelle est soumise cette image. Le redressement polynomial de degré 1 ou projectif (si l'on connaît l'altitude des points d'amers) est souvent le plus adéquat.

6.3.2. Le choix des points d'appui

Disposer du modèle de la déformation est peu courant en pratique. Pour une photographie aérienne par exemple, cela revient à connaître les paramètres de l'optique du capteur, la position exacte du point focal, et l'altitude de l'ensemble des points de l'espace photographié. La prise de points d'appui est donc fondamentale.

Quelque soit la méthode de redressement employée, il est important de multiplier les points d'appui de manière à mieux ajuster le modèle et réduire l'incertitude. Des points de contrôle permettront d'estimer la validité du modèle employé.

Le choix des points d'appui peut être manuel ou automatique. Lorsque la prise des points est manuelle, le choix devra se focaliser sur des points remarquables et précis, comme des intersections de routes, des arbres (pris au niveau du sol), des angles de champs, etc. Dans la majorité des cas, cette prise de point s'effectue sur une carte (c'est l'une des fonctionnalités du module SAVAMER décrit dans ce chapitre). Lorsque l'on ne dispose pas de cartes, ou que la précision est insuffisante, on aura recours à la prise de points sur le terrain, par des méthodes géodésiques classiques ou par GPS.

La prise de points manuelle est une opération longue et fastidieuse ; elle peut même être dangereuse car une grande attention est requise pour ne pas introduire des erreurs. Des techniques de saisie automatique peuvent alors être employées pour déterminer des points remarquables équivalents entre l'image à redresser et le document géo-référencé, en mesurant des indices de corrélation locale. Ces techniques sont particulièrement efficaces lorsqu'il s'agir de redresser une image en utilisant comme référence une autre image géo-référencée. Elle est plus délicate entre une carte et une image pour des raisons de précision. Le principe est le suivant :

- on effectue un premier redressement global de degré 1 ou 2 à l'aide de quelques points d'appui saisi manuellement ;

- on effectue une recherche systématique de points singuliers dans la première image. Pour chaque point singulier, on détermine la forme d'une fonction à partir des valeurs des pixels de l'image proche du point singulier. On recherche ensuite dans la seconde image le point qui offre la même forme pour cette fonction. La recherche est faite autour du point correspondant au redressement global initial. Le choix de la fonction est important : on peut prendre une fonction sur les valeurs des pixels, ou une fonction sur l'agencement local de ces valeurs (comme une texture, par exemple). Le rayon de la recherche est également un paramètre important : il dépend de l'ampleur des déformations locales estimées, ainsi que du relief supposé. Enfin, le seuil de ressemblance ou de variance locale sont également des paramètres importants : ils permettront d'être plus ou moins exigeant sur la qualité du résultat.

6.3.3. Le ré-échantillonnage

Le redressement s'accompagne également d'un ré-échantillonnage : les pixels dans l'image d'arrivée n'ont pas forcément la même taille que les pixels dans l'image de départ. En fait, les pixels ne sont jamais les mêmes, puisqu'ils sont déformés par les nombreuses opérations de redressement. La démarche est donc inverse : la taille et la position des pixels de l'image d'arrivée étant fixées par définition de la relation, on calcule la valeur en cherchant quel est ou quels sont les pixels dans l'image de départ qui lui correspondent, après toutes les déformations inverses. Le ré-échantillonnage consiste donc à choisir une fonction de calcul ou d'interpolation pour la valeur du pixel à partir des pixels qui lui correspondent dans l'image de départ. Les méthodes les plus répandues sont : plus proche voisin, interpolation à partir d'une fonction bilinéaire ou d'une fonction bicubique sur les voisins [GDT 93]:

- la règle du plus proche voisin consiste à affecter au point la valeur du point le plus proche dans l'image de départ.

- la fonction bilinéaire correspond à une interpolation à partir d'un domaine de 2x2 pixels autour du point trouvé, en traitant linéairement les abscisses et les ordonnées.

- la fonction bicubique utilise un polynôme de degré 3 sur un domaine 4x4, en traitant indépendamment les abscisses et les ordonnées.

6.3.4. Le mosaïquage et l'intégration dans une relation

Une fois redressée, les pixels de l'image peuvent être intégrés dans une relation qui possède les même caractéristiques de taille de pixel et de projection

géographique. Cette opération est appelée mosaïquage. Le mosaïquage permet de choisir de façon interactive les pixels de l'image redressée à intégrer dans la relation. Si un objet n'existe pas dans la relation, il est créé et la valeur de l'image est affecté à l'attribut choisi, qui doit être du même type que l'image (nominal, entier, réel, RGB, etc.). Si l'objet existe déjà, la valeur de l'image remplace la valeur de l'attribut choisi. Dans la suite, nous appellerons *mosaïques* les relations dont les objets sont de type pixel.

Une mosaïque provient de diverses images redressées et intégrées dans un même ensemble. La correction géométrique permet d'assurer le bon assemblage géométrique des images. Mais il est courant d'avoir des images qui présentent des différences de valeurs : on constate alors un mauvais raccord des valeurs des pixels. Pour palier à ce problème, il convient d'effectuer un *redressement descriptif* des images après le redressement géométrique. Ce redressement revient à normaliser les valeurs des différentes images avant de les mosaïquer.

La réalisation d'une mosaïque peut donc être décomposée en trois étapes :

- corrections géométriques des images à assembler ;
- corrections des valeurs pour normaliser l'ensemble des images ;
- assemblage des images

6.3.4.1. Les méthodes de correction géométrique pour les mosaïques

Les méthodes de correction géométrique sont celles que nous avons vues au paragraphe précédent. Mais lorsque l'on redresse une image dans le but de l'intégrer à une mosaïque, il est très utile de se servir des zones de recouvrement entre images pour la prise de points d'appui, et assurer ainsi une superposition exacte des zones communes. Le problème du mosaïquage est bien connu en photogrammétrie analytique. Plusieurs méthodes ont été développées pour éviter la propagation d'erreurs lors de la prise de points d'appui dans des zones de superposition [HOT 99].

6.3.4.2. Les méthodes de correction descriptive pour les mosaïques

Les images d'une mosaïque sont généralement obtenues à des dates différentes, ou avec des conditions de prise de vue différentes (ensoleillement, angle, etc.). Comme les objets de même nature doivent apparaître avec des tonalités identiques, il faut minimiser leurs différences de couleurs ou de niveaux de gris de part et d'autre de la ligne de raccord. On opère généralement l'égalisation des valeurs d'une image par étalement, en prenant des valeurs de référence calculées sur la base d'un ajustement des histogrammes sur une zone de recouvrement. Néanmoins, cette méthode n'est pas toujours satisfaisante, car elle suppose l'ajustement linéaire, à la fois dans la zone de recouvrement (pour la constitution du modèle) et dans

l'ensemble de l'image à redresser. Ce n'est que rarement le cas en pratique, car, dans le cas de photographies aériennes ou spatiales par exemple, les différences proviennent de multiples facteurs non linéaires.

6.3.4.3. L'assemblage des images

Pour créer une mosaïque, il est courant d'avoir des images dont une partie est commune. Trois méthodes de mosaïquage peuvent être employées :

- remplacement de la zone de recouvrement par la zone correspondante d'une seule des deux images ;

- calcul de nouvelles valeurs pour les pixels de la zone de recouvrement, par combinaison des valeurs des deux images (par exemple, la moyenne) ;

- détermination d'une ligne de jointure adaptée, dans la zone de recouvrement, aux différences de valeurs des deux images.

Le module SAVAMER, que nous allons décrire dans ce chapitre, met en œuvre l'ensemble de ces techniques.

6.3.5. Un cas particulier : les satellites d'observation de la Terre

Les satellites d'observation de la Terre sont une source très importante d'information géographique. Ils fournissent des images prises par des instruments qui balayent la surface terrestre. Les données fournies par les satellites d'observation de la Terre sont caractérisées par :

- la partie du spectre électromagnétique utilisée (visible, infra-rouge, ondes radar) ;

- la résolution au sol, ou plutôt la taille du plus petit élément échantillonné sur le terrain ;

- la répétitivité de l'observation ;

- la qualité des images.

La plupart des images provenant des satellites d'observation de la Terre de haute résolution (comme SPOT, LANDSAT, IKONOS, ASTER) subissent un pré-traitement de base pour effectuer des corrections géométriques ou radiométriques [GDT 93].

Ainsi, les niveaux de pré-traitement d'une scène SPOT sont :

- Niveau 1A : égalisation des détecteurs dans chaque bande spectrale ;

- Niveau 1B : correction radiométrique et géométrique des déformations systématiques introduites par le système : rotation de la Terre, effet de filé, angle de visée. La précision absolue de la localisation est d'environ 800 m.

- Niveau 2A : corrections géométriques pour restituer la scène dans une projection géographique donnée. Cette correction ne fait pas intervenir de points d'appui, la précision absolue de la localisation est celle du niveau 1B.

- Niveau 2B : corrections géométriques faisant intervenir quelques points d'appui. L'image est donc restituée dans une projection géographique, et la précision absolue de la localisation des pixels ne dépend en principe plus que du relief et des conditions atmosphériques. Elle est d'autant plus précise que la visée est plus verticale et le relief moins important.

- Niveau 3 : c'est une correction de niveau 2B qui tient compte des déformations dues au relief. Il faut alors disposer d'un modèle numérique de terrain pour connaître l'altitude approchée en chaque point.

Le problème de la géoréférenciation est actuellement bien résolu par les fournisseurs d'images satellitales, à condition de disposer de modèles numériques de terrain suffisamment précis dans la zone concernée.

6.4. SAVANE et mosaïques

Le système SAVANE permet de gérer des relations de type mosaïque dans le cadre du modèle relationnel étendu. Nous allons décrire dans ce paragraphe la structure de ce type de relation ainsi que les particularités de gestion qu'elle implique.

6.4.1. Structure des relations de type mosaïque

Les relations de type mosaïque permettent de conserver des objets pixel dont les valeurs sont données sous forme d'image. Dans ce type de relation, les objets sont très nombreux (il n'est pas rare d'avoir plusieurs dizaines de millions de pixels), et représentent, comme nous l'avons vu, à la fois une surface ou un point. A cause de ces caractéristiques et pour gérer ce type de données de façon efficace, la structure de représentation interne est profondément différente de la structure des relations vectorielles :

- les données sont conservées selon une projection géographique particulière, ce qui permet d'utiliser la mosaïque dans cette projection sans avoir à recalculer la position de chaque pixel ;

- la position d'un objet est donné par son agencement dans une grille, appelée encore imagette, et non par ses coordonnées ;

- l'indexation n'utilise pas une grille adaptative, mais une grille régulière ;

- une mosaïque peut être partagée par plusieurs bases de données, pour ne pas avoir à dupliquer une information volumineuse si deux bases de données souhaitent l'utiliser.

A cause de ces caractéristiques, la création et la gestion d'une relation de type mosaïque utilise des procédures particulières dans SAVATECA. Par exemple, le dialogue de création d'une relation de ce type est différent, car il faut indiquer des paramètres supplémentaires : la taille des pixels, le répertoire de stockage et la projection dans laquelle est conservée la mosaïque. La taille de la grille d'indexation peut également être paramétrée (fig. 6.1).

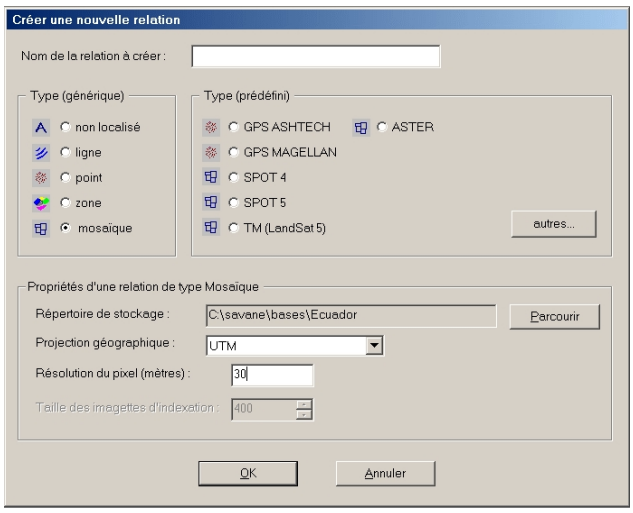


fig. 6.1. : le dialogue de création d'une relation mosaïque

Pour les attributs, nous avons étendu les types de base (nominal, entier, réel) de manière à minimiser l'espace de stockage de chaque valeur. Rappelons qu'un attribut correspond ici à la notion d'image ou de canal (pour les images satellitaires). Ainsi, le type entier peut se décliner en entier 8 bits (valeurs de 1 à 255, valeur inconnue 0), entier 16 bits (valeurs de 1 à 65655, valeur inconnue 0), entier 32 bits (fig. 6.2). Le type RGB correspond à un codage de chaque valeur en 24 bits, pouvant ainsi décrire une couleur par ces trois composantes, chacune sur un octet (rouge, vert, bleu). Le type réel correspond à un codage sur 32 bits (en float) en non sur 64 bits.

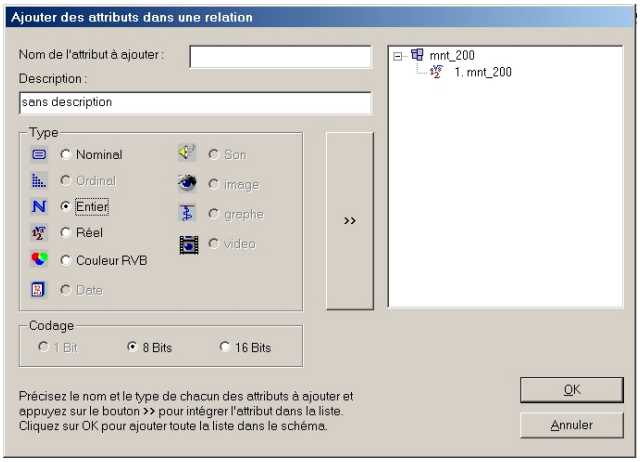


fig. 6.2. : le dialogue de création d'un attribut d'une relation mosaïque

Tous les fichiers d'une relation sont stockés dans un répertoire qui porte le nom de la relation. Ce répertoire peut être n'importe où sur le réseau local, et son adresse (indiquée lors de la création ou de la modification de la relation) est conservée dans le fichier base. La structure d'une relation de type mosaïque comprend un fichier décrivant la relation (taille du pixel, projection, taille des imagettes), et, pour chaque attribut, un fichier index contenant la description des imagettes présentes pour cet attribut et un fichier descriptif contenant les valeurs des pixels (fig. 6.3).

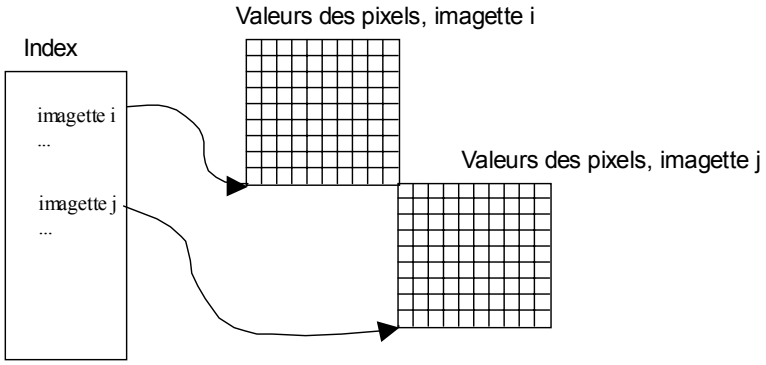


fig. 6.3 : structure interne des relations mosaïques

La description de chaque imagette comprend la fenêtre géographique occupée par l'imagette (dans la projection de la mosaïque), et l'adresse de premier pixel de l'imagette dans le fichier descriptif. Les pixels de chaque imagette sont ensuite agencés comme dans une image, en ligne du bas vers le haut, chaque ligne étant organisée de la gauche vers la droite.

Chaque relation comprend également deux fichiers spéciaux indiquant l'existence des pixels, pour l'ensemble des attributs. Le premier correspond à l'index des feuilles existantes, le second indique, pour chaque feuille, l'existence de chaque pixel dans la feuille.

Des attributs temporaires peuvent être créés dans une relation de type mosaïque, comme dans tout type de relation (nous décrirons ces opérations au chapitre 8). Dans ce cas, le fichier correspondant au nouvel attribut temporaire n'est pas créé dans le répertoire de la mosaïque, mais dans le répertoire de l'utilisateur.

6.4.2. Opérations relationnelles

Les pixels d'une mosaïque sont gérés dans la base de données comme les objets d'une relation, et bénéficient ainsi de l'ensemble des opérations disponibles, sans qu'il soit nécessaire de recourir à un formalisme spécial pour ce type d'objet. Ainsi, sélections, sélections géographiques, calculs de nouveaux attributs, classifications, calculs statistiques, utilisent dans le système SAVANE un formalisme identique quelque soit le type de la relation (zone, ligne, point, pixel, non localisé). Par exemple, le calcul d'un indice de végétation sur un domaine de l'espace à partir d'une mosaïque provenant d'une image satellite crée un nouvel attribut pour la relation : l'utilisateur n'indique que la formule de calcul, et le nom du nouvel attribut à créer. Le système de gestion se charge de l'ensemble des opérations.

Les opérations de restriction sur les mosaïques n'offrent aucune particularité dépendant du type : les objets pixel peuvent être sélectionnés sur un critère descriptif comme sur un critère de localisation (sélection sur un masque). Seul l'attribut d'existence est temporairement modifié. Il est par exemple très simple d'étudier les valeurs d'un attribut d'une mosaïque sur un domaine de l'espace défini à partir des objets d'une autre relation localisée.

Les opérations de jointure classique (sur un critère descriptif) n'ont que peu d'intérêt pour les relations de type mosaïque, car le résultat n'est pas localisé et le nombre d'objets créés peut être très grand. Par contre, les opérations de jointures géométriques, suivies ou non d'une agrégation sur un critère descriptif, sont intéressantes et deviennent également très faciles à mettre en œuvre. Par exemple :

- l'équi-jointure géométrique zone-pixel permet de créer de nouveaux attributs dans la relation mosaïque, par intersection des zones et des pixels.

- l'agrégation de pixels dans des zones permet de créer un attribut pour des objets d'une relation zonale en effectuant, par zone, une opération sur la valeur des pixels se trouvant dans la zone. L'opération peut être une somme, une moyenne, un écart-type, la fréquence d'une valeur nominale, la surface occupée par les pixels, etc. Par exemple, il est ainsi très simple de calculer la moyenne de radiométrie dans une zone de végétation, ou d'ajuster un modèle de calcul entre canaux grâce à la mise en relation avec les valeurs d'un échantillon de zones.

- l'agrégation de pixels dans d'autres pixels effectue une opération de ré-échantillonnage, et permet ainsi très facilement de comparer des images de résolutions différentes.

Basées sur ces opérations relationnelles, les opérations de géostatistique, univariées ou multivariées, deviennent également très faciles à mettre en œuvre.

6.4.3. Création de relations de type mosaïque et opérations de changement de type

La création d'une relation de type pixel peut également être interactive et intervenir à tout moment lors d'une requête :

- par définition directe des paramètres de la relation : taille du pixel, projection géographique ;

- par rasterisation d'une relation zonale, en calculant pour chaque zone les objets pixel correspondants, et en affectant à chaque pixel la valeur d'un attribut de la zone à laquelle il appartient ;

- par interpolation à partir de points ou d'arcs, en calculant pour chaque pixel du domaine d'étude une valeur par interpolation entre les points d'origine.

Les mosaïques créées à partir d'autres objets de la base ont la même structure que les mosaïques permanentes. Elles peuvent ensuite être intégrées de manière permanente à la base de données.

Ces opérations de création d'une relation de type pixel à partir de relations localisées d'autres types (zone, ligne, point, pixel) correspondent à un changement de type d'objet. Inversement, on peut créer des relations de type zone, ligne, point ou pixel à partir d'une relation de type pixel :

- création de zones, par vectorisation et labélisation sur un attribut nominal de la relation mosaïque (par exemple, pour créer les zones correspondant à la classification d'un attribut) ;

- création de lignes, par vectorisation sur un attribut nominal (par exemple, pour créer des courbes d'isovaleur à partir d'une interpolation) ;

- création de points, par calcul direct des coordonnées de chaque pixel de la mosaïque ;

- création de pixels, par ré-échantillonnage permettant modifier la taille du pixel d'origine.

Nous détaillerons l'ensemble de ces opérations dans le chapitre consacré aux changement de type d'objet (chapitre 9).

6.4.4. Utilisation des mosaïques dans le cadre objet

6.4.4.1. La classe de base CMosaique

Les relations de type mosaïque peuvent être décrites en tant qu'instances d'une classe dérivée d'une classe de base que nous appellerons CMosaique. Les attributs de la relation sont des membres de la classe dérivée, et dépendent bien sûr de la définition de la relation correspondante.

Du fait de la forme et de l'agencement des objets, le type mosaïque possède de nombreuses opérations spécifiques, provenant du traitement et de l'analyse d'images, comme par exemple :

- morphologie mathématique (érosion, dilatation, connexité),

- calculs géomorphologiques et hydrologiques : pente, orientation, courbures, volume, drainage, etc.

- analyse d'image, filtrage, texture et segmentation,

- squelettisation et transformations de voisinage,

- anisotropie, reconnaissance de forme.

Ces opérations sont implantées en tant que méthodes de la classe de base `CMosaique`. De même, certaines procédures de visualisation sont propre au type et sont implantées en tant que méthodes de la classe de base : éclairements, perspectives, compositions colorées.

Les méthodes ont en général comme paramètres un ou deux attributs de la relation correspondante à l'instance de la classe dérivée.

Pour la mise en œuvre des méthodes, la structure des mosaïques en imagettes ne pose aucun problème algorithmique majeur. Il faut seulement gérer efficacement les relations de voisinage entre imagettes, pour pouvoir retrouver facilement les

relations de voisinage entre pixels quelque soit l'imagette à laquelle ils appartiennent.

6.4.4.2. Quelques classes dérivées

On peut définir une classe dérivée à partir de la classe de base lorsque l'on connaît les attributs et des méthodes spécifiques associées à ces attributs. Par exemple, les capteurs satellites SPOT ou TM permettent de définir des classes dérivées, car les attributs et leurs caractéristiques sont connus (les canaux radiométriques). Tous les indices particuliers au capteur (comme par exemple les indices de végétation, dont le NDVI, peuvent être implantés en tant que méthodes propre à la classe.

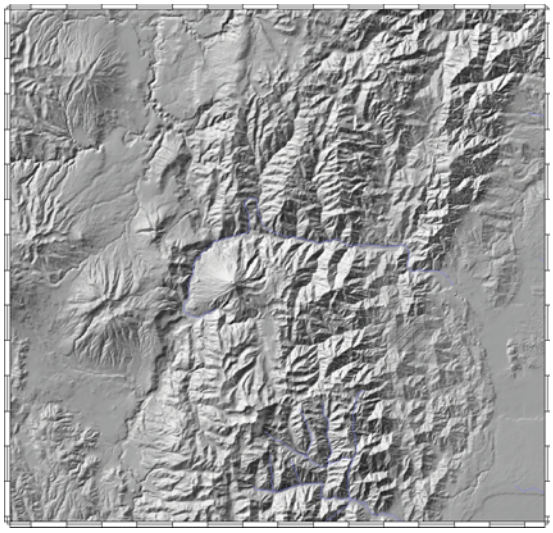


fig. 6.4 : un modèle numérique de terrain

Les modèles numériques de terrain offrent d'autres classes dérivées. Si l'unique attribut correspond à une valeur d'altitude de chaque pixel, la définition de la classe est fonction de la taille de l'objet (la taille du pixel). La classe hérite de nombreuses méthodes de la classe de base, notamment pour toutes les opérations géomorphologiques et hydrologiques.

En proposant ces classes dérivées, le système peut faciliter considérablement l'utilisation des données de type pixel dans le SIG : il propose des schémas de données incluant de nombreuses méthodes associées aux attributs, et se charge de l'ensemble de la gestion de données souvent très volumineuses. L'intégration de ce

type de données dans le système de gestion est complète et l'exploitation de ces données peut être facilement accessible à de non-spécialistes.

6.5. Le module SAVAMER

6.5.1. Présentation générale

Le système SAVANE comporte deux modules de saisie graphique, l'un pour les images, l'autre pour les données vectorielles. Le module SAVAMER permet de géoréférencer des images *raster* et de les intégrer dans une relation de type mosaïque.

A partir d'une image non géoréférencée, appelée *image de départ* dans la suite, le module SAVAMER permet de créer une nouvelle image, appelée *image d'arrivée*, par redressement et ré-échantillonage. Les paramètres de cette nouvelle image (projection géographique, taille de pixel, espace géographique) sont indiqués par l'utilisateur. Une fois cette opération effectuée, le module SAVAMER permet d'intégrer les valeurs de l'image d'arrivée à un attribut d'une relation mosaïque d'une base de données. Les paramètres géographiques et géométriques (projection géographique, taille et forme de pixel) de l'image d'arrivée sont donc connus : ce n'est pas le cas pour l'image de départ.

Le module SAVAMER permet la saisie de points d'appui et le redressement avec ré-échantillonage, en donnant le choix de la fonction de redressement : similitude, projection centrale, fonction polynomiale de degré 1 ou 3, tessellation, etc. Plusieurs fonctions de redressement peuvent être combinées : il est courant d'effectuer une transformation globale (de type projective ou de degré1) suivie de transformations locales (par exemple, redressement de degré 1 dans les triangles issues d'une tessellation à partir de points d'appui). La fonction de ré-échantillonage peut également être choisie : plus proche voisin, fonction bilinéaire sur les voisins, fonction bicubique sur les voisins.

6.5.2. La prise de points d'appui

6.5.2.1. Principes généraux et interface graphique

La première étape du redressement, c'est la prise de points d'appui, c'est-à-dire de couples de points, l'un dans l'image à redresser, l'autre dans l'espace géographique. Le nombre de points d'appui à saisir dépend de la méthode de redressement qui sera employée par la suite : un point suffit pour une translation, deux points pour une rotation, une similitude, ou une fonction polynomiale plus générale de degré 1, six points pour une perspective centrale, neuf points pour une

fonction polynomiale de degré 3, et le plus de points possibles dans le cas d'un redressement local par triangulation.

L'interface de SAVAMER comprend plusieurs fenêtres : une fenêtre générale correspondant à un espace géographique, dessiné dans une projection géographique donnée par l'utilisateur, et des fenêtres permettant d'afficher une partie ou la totalité de l'image à redresser. La fenêtre géographique permet de dessiner tous les objets déjà présents dans les relations de la base de données, ainsi que des images déjà géoréférencées, ou des documents provenant d'une saisie vectorielle effectuée avec le module SAVEDIT.

La prise de points d'appui s'effectue visuellement en associant un point de la fenêtre « image » à un point de la fenêtre « géographique ». Il est également possible d'indiquer directement les coordonnées géographiques d'un point de l'image. C'est la méthode utilisée lorsque l'on redresse une carte scannée, en choisissant des points d'intersection des amorces de projection dans l'image de la carte scannée. On peut également dessiner les amorces de projection dans l'espace géographique pour vérifier le bon placement des points d'appui (fig. 6.5).

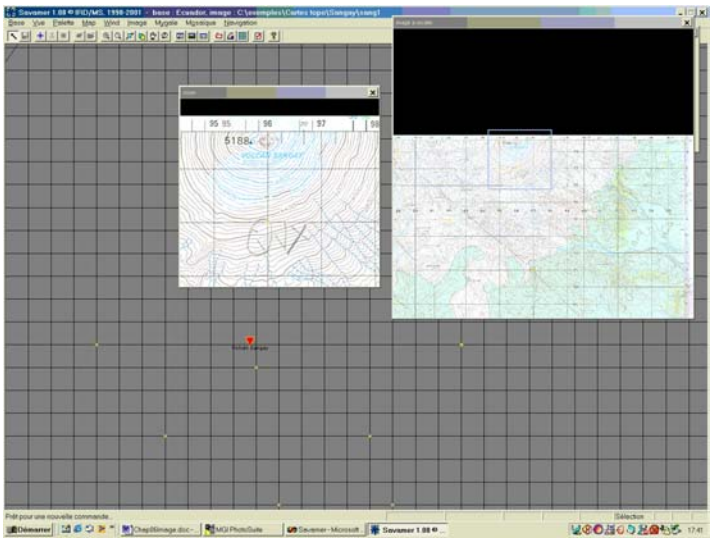


fig. 6.5 : l'interface graphique du module SAVAMER

C'est également la méthode utilisée lorsque les points d'appui proviennent d'une campagne de mesures directes sur le terrain : après avoir pointé un point sur l'image, on indique les coordonnées géographiques qui ont été relevées sur le terrain.

Les points d'appui sont conservés dans l'instance d'une classe `CAmers` (A.3.1.). Cette structure conserve les points d'appui dans divers systèmes de coordonnées. La procédure de sauvegarde crée un fichier qui conserve les points d'appui en coordonnées sphériques (pour le point géographique) et en coordonnées image (pour le point de l'image). Ce fichier est associé à l'image à redresser. S'il existe déjà (des points d'appui ont déjà été saisis), il est automatiquement ouvert lors de l'ouverture de l'image, et les points déjà saisis sont dessinés sur l'écran.

6.5.2.2. La saisie manuelle des points d'appui

La saisie manuelle consiste à associer manuellement deux points, l'un saisi dans l'espace géographique, l'autre dans l'image. L'opérateur peut bien sûr modifier la position de chaque point avant ou après validation du point d'appui (fig. 6.6).

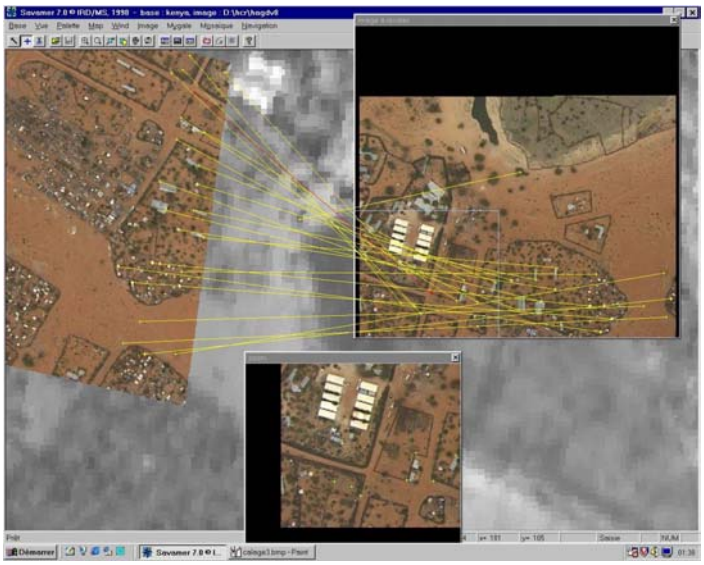


fig. 6.6 : exemple de prise de points d'amers dans une zone commune à deux images

Dès que deux points d'appui ont été saisis, il est possible de calculer une fonction polynomiale de degré 1 (en fait une similitude) pour passer d'un repère à un autre. Une option très utile permet d'utiliser cette fonction pour proposer automatiquement un point lors de la saisie ultérieure de points d'appui : dès que l'opérateur choisit un point dans l'un des espaces (géographique ou image), le système propose automatiquement le point correspondant dans l'autre espace. En général, ce point n'est pas très éloigné du point réel (la différence correspond à la différence entre la similitude et la déformation réelle à modéliser). L'opérateur n'a plus qu'à déplacer ce point à l'aide des touches de déplacement du clavier, pixel par

pixel (la taille du pixel dépend de l'espace visualisé, qui peut bien sûr être choisi à tout moment par l'opérateur).

6.5.2.3. La saisie semi-automatique et automatique

La saisie semi-automatique permet dans certains cas d'améliorer la précision de la prise des points d'appui. On utilise alors un indice de corrélation basé sur l'agencement des pixels dans l'image à redresser et dans une image déjà géoréférencée ou dans une relation de la base de type mosaïque. La recherche d'une meilleure ressemblance est effectuée au voisinage du point proposé par l'opérateur : il permet souvent d'affiner la prise du point de quelques pixels.

La saisie automatique permet de saisir automatiquement des points d'appui dès qu'un redressement polynomial initial peut être effectué. Le principe de la corrélation est identique à celui de la saisie semi-automatique, mais ici les points sont choisis par le système en fonction de leur singularité dans l'image à redresser. Le système balaye l'image à la recherche des pixels qui sont « très » différents de leurs voisins, ou qui sont le centre d'une structure remarquable. Il recherche ensuite pour chaque point celui de l'image déjà redressé qui lui ressemble le plus, au voisinage du point calculé par la fonction polynomiale initiale de degré 1.

Nous avons construit des classes pour encapsuler l'ensemble des opérations de calculs de la transformation polynomiale initiale (`CSimilitude`, `CSystème1`). Ces classes sont également être utilisées pour redresser l'ensemble d'une image. Nous les présenterons dans le paragraphe suivant.

6.5.3. Le redressement et le ré-échantillonnage

Les pixels de l'image d'arrivée sont définis sans ambiguïté, taille et position. Pour savoir quelle valeur affecter à un pixel de l'image d'arrivée à partir des pixels d'une image de départ, il faut calculer, par géoréférencement, quels pixels de l'image de départ participent par leur position à la définition de la valeur du pixel de l'image d'arrivée, puis, par une opération de ré-échantillonnage, il faut calculer une nouvelle valeur à partir de ces pixels. Nous effectuons donc cette opération en deux étapes : calcul des coordonnées dans le repère de l'image de départ d'un pixel de l'image d'arrivée, puis calcul d'une valeur à partir des pixels voisins de cette coordonnée.

6.5.3.1. Le redressement

Le redressement consiste donc à savoir quelles sont les coordonnées dans le repère de l'image de départ d'un pixel de l'image d'arrivée. On utilise pour cela une fonction qui est censée approcher la déformation. Cette fonction peut être

parfaitement connue, mais c'est rarement le cas. Par exemple, si une carte n'a pas subit de déformation du papier, si le scanner utilisé ne provoque aucune déformation locale, la transformation est une similitude (si, bien sûr, on conserve la même projection géographique. Sinon, il faut composer avec un calcul de changement de projection). Si l'optique d'un appareil photographique de prise de vue ne provoque aucune déformation, si les rayons lumineux sont rectilignes, si le terrain est totalement plat, et si la prise de vue est orthogonale, la transformation est une projection centrale suivie d'un changement de projection géographique (tenant alors compte de la courbure de la Terre…). Evidemment, ces conditions ne sont jamais remplies dans la pratique (une optique n'est jamais parfaite, etc.).

On peut essayer de modéliser la déformation de chaque élément (l'optique, le relief, les angles de prise de vue, les conditions atmosphériques) pour composer ces déformations et aboutir à une fonction globale modélisant la déformation. La modélisation de certains éléments nécessitent des données de positionnement (par exemple, les éphémérides des satellites) ou des points d'appui (pour calculer par exemple les angles de prise de vue, la position du point focal d'un appareil de prise de vue aérienne), d'autres nécessitent uniquement des paramètres *intrinsèques* aux appareils employés. Ces paramètres sont en général obtenus par échantillonage initial (pour la déformation optique, par exemple).

Nous avons développé dans SAVAMER des méthodes de modélisation exclusivement basées sur les points d'appui et non sur les paramètres intrinsèques des appareils de prise de vues ou de rasterisation. Ces méthodes sont les suivantes :

- modélisation par une transformation polynomiale de degré 1 (translation, rotation, similitude, ou polynôme général de degré 1) ;
- modélisation par une transformation polynomiale de degré 3 ;
- modélisation par une transformation perspective conique ;
- modélisation par une transformation perspective cylindro-conique ;
- modélisation par une transformation locale polynomiale de degré 1.

Les paramètres des fonctions correspondantes à ces méthodes sont déterminés uniquement par les points d'appui. Les modélisation polynomiales de degré 1 nécessitent de 1 à 3 points d'appui. Les modélisations perspectives demandent de 4 à 6 points. La modélisation locale polynomiale utilise au minimum quatre points d'appui : elle utilise une triangulation calculée à partir de l'ensemble des points d'appui, et une fonction polynomiale de degré 1 dans chacun des triangles (fig. 6.7).

Le système permet de combiner une déformation globale, polynomiale de degré 1 ou perspective, et une déformation locale par triangulation, en effectuant successivement les deux transformations. La déformation par triangulation revient à modéliser la déformation due au relief et aux autres paramètres de prise de vue par

un ensemble de facettes planes, après une première transformation qui est censée modéliser la déformation idéale (optique sans déformation, prise de vue orthogonale, etc.) de manière globale. La surface à modéliser est représentée par un réseau de facettes triangulaires dont les sommets sont les points d'appui redressés par la première transformation globale. On se trouve devant le classique problème mathématique visant à approcher une fonction de deux variables connues uniquement en un nombre fini de points distribués de façon quelconque dans son domaine de définition. La discrétisation du domaine de définition est la première étape de résolution de ce problème. Si les points sont distribués de manière aléatoire, une grille triangulaire dont les sommets sont les points initiaux répond à la question, lorsque cette triangulation permet d'assurer la continuité de la fonction à interpoler. La triangulation de Delaunay, duale de la tessellation de Voronoï, satisfait cette condition. Elle est par ailleurs utilisée, tout comme la tessellation de Voronoï, dans la résolution de nombreux problèmes en géométrie discrète ou algorithmique [SHA 78] [O'RO 93]. Nous présenterons au paragraphe 6.6.3.3. l'algorithme utilisé dans SAVAMER pour calculer la triangulation de Delaunay utilisée lors du redressement par triangulation.

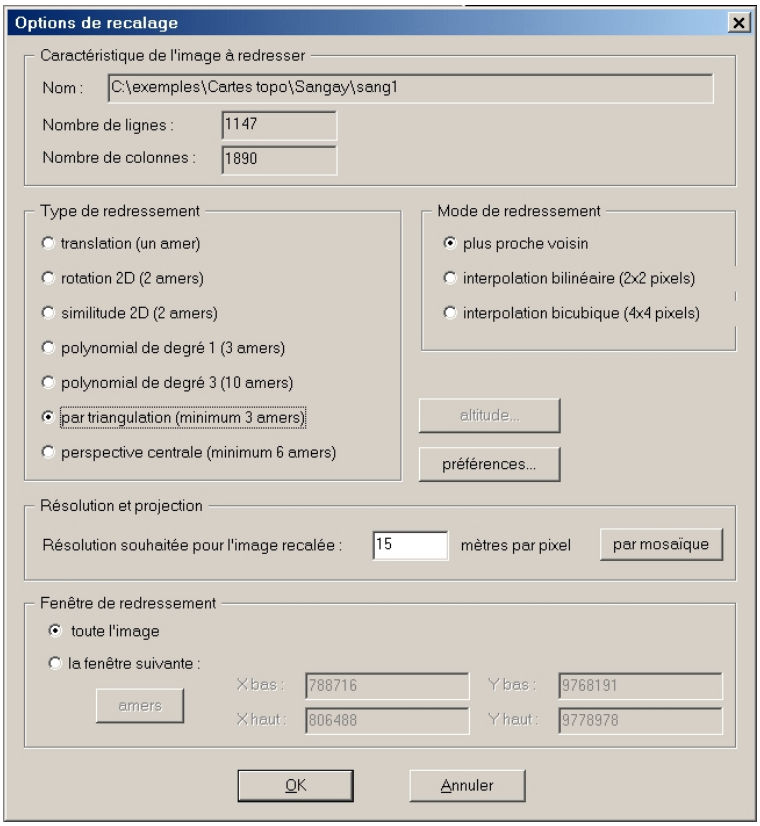


fig. 6.7 : le dialogue de redressement de SAVAMER

La triangulation de Delaunay crée, à partir des points d'appui redressés par le calcul global, un graphe planaire représenté par ensemble de segments (fig. 6.8). Ces segments sont ensuite rasterisés pour obtenir une image codée, chaque code représentant un triangle de la triangulation. Parallèlement, on calcule et on conserve pour chaque triangle les coefficients de la transformation bilinéaire à appliquer aux points du triangle, grâce aux coordonnées redressées des trois points d'appui et de leurs coordonnées correspondantes dans le repère de projection géographique. Le redressement de l'image se fait donc en plusieurs étapes :

- pour un point de l'image d'arrivée, recherche du triangle dans lequel se trouve ce point, et calcul des coordonnées par transformation locale inverse ;

- calcul de ses coordonnées dans l'image de départ par transformation globale inverse.

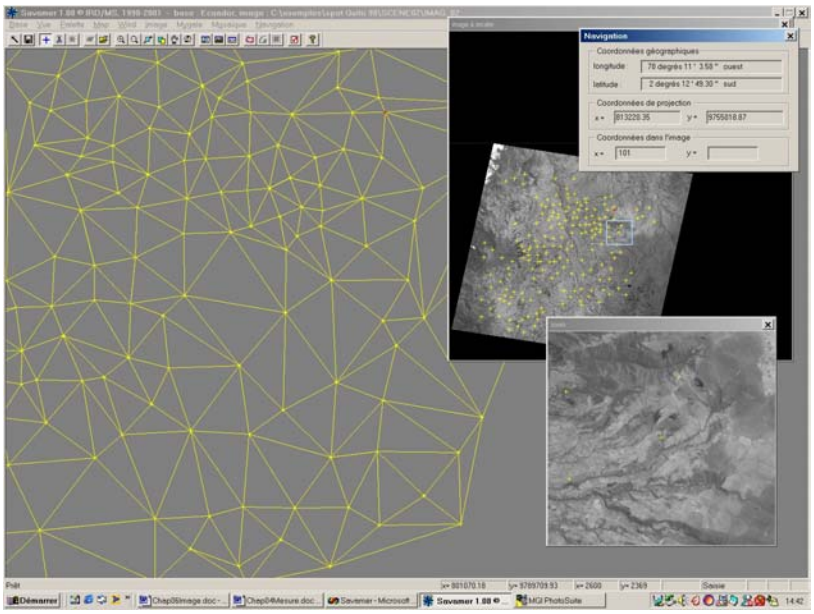


fig. 6.8 : triangulation obtenue à partir des points d'appui

Pour toutes les méthodes de redressement, il faut commencer par calculer la fenêtre de redressement, c'est-à-dire l'espace géographique de l'image d'arrivée concerné par l'image de départ. Cette fenêtre permet de déterminer les paramètres de l'image d'arrivée, dont on ne connaît a priori que la résolution. Le calcul de la fenêtre utilise uniquement le redressement global, qui est effectué sur les quatre coins de l'image de départ.

Plusieurs classes permettent dans SAVAMER d'encapsuler l'ensemble des calculs nécessaires aux opérations de redressement : `CSimilitude`, `CSysteme1`, `CSysteme3` (A.3.2.). La méthode `Equations()` de chacune de ces classes calcule, à partir des trois premiers points d'amers, les coefficients des formules de calculs. Les méthodes `ProjectionToImage()` et `ImageToProjection()` effectuent les calculs de changement de repère. Le calcul de la fenêtre de redressement est effectué par la fonction `FenetreDestination()`. La méthode `Recalage()` effectue à la fois le redressement et le ré-échantillonage.

6.5.3.2. Le ré-échantillonnage

Comme nous l'avons vu, le redressement s'accompagne d'une opération de ré-échantillonnage. Plusieurs méthodes peuvent être employées pour effectuer cette opération [GDT 93]. La plus simple consiste à prendre la valeur du pixel de l'image d'origine le plus proche (plus proche voisin), mais il est également courant de faire

une moyenne sur les pixels voisins, par un calcul bilinéaire ou bicubique. Ces trois opérations de ré-échantillonage sont implantées dans le module de redressement de SAVAMER.

Le calcul bilinéaire calcule la valeur en fonction de la position du point par rapport aux quatre voisins, le calcul bicubique calcule la valeur en fonction de la position du point sur une fenêtre de 4*4 voisins (A.3.2.).

Toutes les opérations de redressement créent une nouvelle image géoréférencée, munie d'un fichier décrivant ses paramètres de localisation (coordonnées du point bas-gauche, taille du pixel, projection géographique, nombre de lignes, nombre de colonnes). Le type de l'image créée (codage sur 8, 16, 24 ou 32 bits) est toujours identique à celui de l'image de départ.

6.5.3.3. Un algorithme de triangulation

Plusieurs algorithmes ont été développés pour créer une tessellation de Voronoï ou un graphe de Delaunay à partir d'un semis de points aléatoirement distribués dans le plan. On peut les classer en deux catégories : les algorithmes par incrémentation, qui construisent la triangulation à partir d'un point intérieur quelconque et en ajoutant les points un par un, en recalculant la triangulation pour chaque point ajouté, et les algorithmes de type « diviser pour régner », qui emploient une méthode récursive en divisant récursivement la région en sous-régions jusqu'à ce que la triangulation finale soit atteinte. Les techniques incrémentales ne sont pas très optimisées car en général de complexité en $O(n^2)$, mais elles sont plus faciles à mettre en œuvre et demandent moins de ressources mémoire.

L'algorithme implémenté dans SAVAMER est du type incrémental. Il est décrit dans [FLO 85]. Il se divise en plusieurs opérations :

- triangulation de Delaunay pour un simple polygone ;
- insertion incrémentale de points dans la triangulation.

La classe `CTessel` regroupe l'ensemble des structures et des opérations nécessaires à la mise en œuvre de l'algorithme de triangulation de [FLO 85] (A.3.3.).

6.5.3.4. Le redressement descriptif des dynamiques

La dernière étape avant l'intégration d'une image dans la base par mosaïquage concerne le redressement des valeurs de l'image géométriquement redressée. En effet, une image peut être géométriquement correcte mais comporter d'importantes différences de valeur avec des images voisines correspondant au même attribut.

Dans SAVAMER, on peut opérer l'égalisation des valeurs d'une image géoréférencée par rapport à une mosaïque par étalement, en prenant des valeurs de références calculées sur la base d'un ajustement linéaire des histogrammes. L'étalement est calculé par régression linéaire à partir des points d'appui ou sur l'ensemble des pixels de la zone de recouvrement. Si l'image est de type RVB, la régression linéaire est calculée séparément pour chaque composante de couleur.

6.5.4. Le mosaïquage et l'intégration dans une relation

L'opération de mosaïquage permet d'intégrer une image géoréférencée à un attribut d'une relation de type pixel. Le système Savane gère ces attribut de manière à permettre des ensembles de pixels plus complexes que des images rectangulaires. L'opération d'intégration consiste donc, à partir d'une image géoréférencée, à créer des objets pixels dans une relation (si ces objets n'existent pas déjà), et à affecter les valeurs des pixels de l'image à un attribut dans cette relation. On constitue ainsi une « mosaïque » dont la gestion est entièrement à la charge du système, et dont la structure externe est identique à celle des autres types de relation.

Le module SAVAMER permet de choisir le domaine à intégrer, ainsi que la méthode d'intégration. Le domaine peut être choisi de plusieurs manières (fig. 6.10) :

- par imagette, en sélectionnant directement sur l'écran les imagettes à créer ;
- par rectangle, en choisissant le rectangle sur l'écran ou en donnant ses coordonnées géographiques ;
- par polygone, en dessinant sur l'écran le polygone à intégrer.

Si la valeur d'un pixel existe déjà, la méthode d'intégration peut être : remplacer la valeur, ne pas remplacer la valeur, faire la moyenne (fig. 6.9).

fig. 6.9 : le dialogue d'intégration d'une image dans une mosaïque

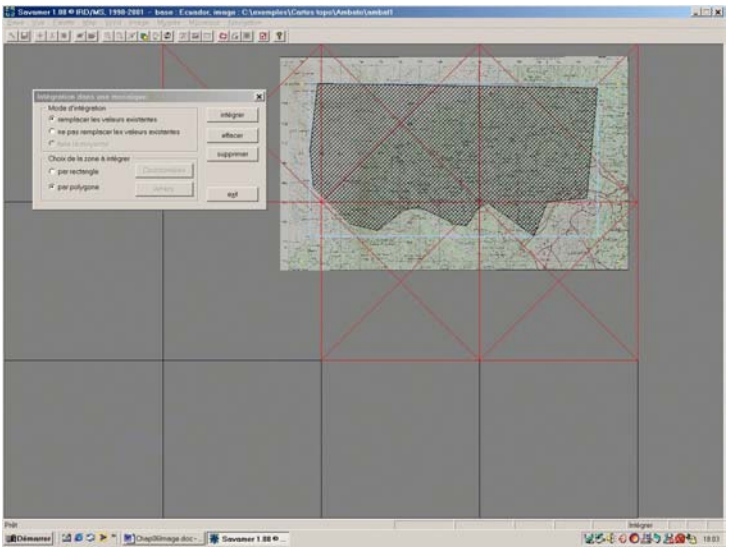


fig. 6.10 : le choix des pixels à intégrer : méthode par polygone

L'intégration d'une image crée des imagettes pour l'attribut sélectionné dans la relation. Une imagette est créée dès qu'un pixel doit être créé dans l'espace correspondant. L'intégration peut ainsi être effectuée en plusieurs étapes, et la mosaïque croît au fur et à mesure de la création de nouvelles imagettes (fig. 6.11). Il est bien sûr également possible de supprimer des pixels. Si tous les pixels d'une imagette doivent être supprimés, l'imagette elle-même est supprimée dans le fichier servant d'index.

fig. 6.11 : exemple de constitution de mosaïque d'images géoréférencées

6.6. Conclusion

L'intégration des données de type image dans le processus relationnel permet de bénéficier de la simplicité du modèle avec des données dont la structure est très différente des objets géographiques dont la localisation est décrite par la géométrie. Elle permet surtout de ne pas avoir à différencier la logique des opérations relationnelles classiques (restriction, jointure) ou des opérations sur les attributs (calculs, classifications, etc.) entre les différents types de localisation. Mais cette intégration exige d'avoir une définition formelle de ce type d'objets, et d'assurer la validité de l'attribut de localisation pour chacun des objets. Nous avons donc introduit un nouveau type d'objet, le pixel, et un nouveau type pour les relations représentant des collections de pixels, les mosaïques, qui sont structurées et gérées de façon très différente des autres types de relations.

La validité de la localisation est assurée par des opérations de redressement d'images. La mise en œuvre de ces opérations souvent complexes a donné lieu au développement d'un module spécialisé dans le système SAVANE. Ce module regroupe le redressement des images, la géoréférenciation des pixels, et leur intégration dans une relation de la base de données. Le redressement peut être effectué par plusieurs méthodes. La plus complète utilise un redressement local de degré 1 dans les triangles issues de la triangulation de Delaunay obtenue à partir de l'ensemble des points d'amers, et a l'avantage de ne pas utiliser les paramètres intrinsèques de la prise de vue.

Bibliographie

[EST 92] ESTES J., *Remote sensing and GIS integration : research needs, status and trends*. ITC Journal, 1992, n° 1, p. 2-9

[FLO 85] DE FLORIANI L., FALCIDIENO B., PIENOVI C., *Delaunay-based Representation of Surfaces Defined over Arbitrary Shaped Domains*, Computer Vision, Graphics, and Image Processing, 1985, n° 32, p. 127-140

[GDT 93] GDTA, *Les Spatiocartes*, Cahiers Pédagogiques du GDTA, 1993

[HOT 99] HOTTIER, *Photogrammétrie analytique*, Cours ENSG, 1999

[NOV 92] NOVAK K., *Rectification of digital Imagery*. Photogrammetry Eng. And Remote Sensing, 1992, vol. 58, p.339-344

[O'RO 93] O'ROURKE J., *Computational Geometry in C*, 1993, Cambridge University Press

[REN 91] RENOUARD L., *Restitution automatique du relief à partir de couples stéréoscopiques d'images du satellite SPOT*, 1991, Thèse de Doctorat présentée à l'Ecole Polytechnique

[SHA 78] SHAMOS M.I., *Computational Geometry*, PHD Thesis, 1978, Yale University

Chapitre 7

Contraintes d'intégrité spatiale et saisie graphique

Les contraintes d'intégrité permettent d'assurer la cohérence des bases de données et de renseigner sur leur niveau de qualité. Dans les systèmes d'information géographique, la qualité des données provient également de la qualité de la localisation des objets et de leur niveau de cohérence spatiale. Ce chapitre a pour objectif de passer en revue l'ensemble des contraintes d'intégrité spatiale applicables à une base de données localisées, contraintes qui permettent, si elles sont remplies, de renseigner sur le niveau de qualité de la base de données, d'assurer la fiabilité des processus utilisant ces données, et d'assurer ainsi la capacité du système à remplir les fonctions attendues par les utilisateurs. De plus, avec les possibilités d'échange ou de partage de données, il est important de pouvoir renseigner l'utilisateur sur le niveau de qualité et de cohérence des données qu'il reçoit ou qu'il souhaite exporter vers un autre système.

Comme nous l'avons vu, les systèmes d'information géographique peuvent utiliser de nombreux modèles géométriques internes pour représenter les objets localisés, mais les contraintes d'intégrité liées à la localisation doivent, lorsque cela est possible, être exprimées indépendamment de ces modèles internes. Par contre, la cohérence s'exprime par rapport à une modélisation de la réalité en entités géographiques, correspondant à des collections d'objets géographiques (classiquement des zones, des lignes, des points, des pixels), et les contraintes d'intégrité spatiale sont exprimées en fonction de cette modélisation, sur les objets et entre les objets. Les contraintes les plus classiques concernent par exemple la fermeture des zones, la non-intersection d'objets, la connexité, les relations spatiales entre les objets de différentes entités géographiques. Nous étudierons les contraintes

d'intégrité spatiale en nous basant sur le modèle relationnel pour la modélisation de la réalité et la gestion des collections d'objets géographiques. Nous étudierons ensuite les possibilités de mise en œuvre de ces contraintes en fonction des différents modes de représentation géométrique de la localisation, lors du processus de saisie graphique.

Nous présenterons enfin en détail le module SAVEDIT de saisie graphique sur écran, qui met en œuvre au sein du système SAVANE un certain nombre de contraintes d'intégrité. Nous présenterons l'architecture de ce module et les algorithmes utilisés pour résoudre les nombreux problèmes graphiques soulevés par la mise en œuvre interactive des contraintes d'intégrité.

7.1. La qualité dans les bases de données géographiques

7.1.1. Le problème général de la qualité dans les bases de données géographiques

La connaissance de la qualité de la localisation dans les bases de données géographiques devient de plus en plus importante pour assurer une bonne utilisation de ces bases de données, avec notamment la multiplication des échanges de données entre les SIG. Il n'est plus rare en effet d'avoir à travailler sur des données très hétérogènes, et il devient de plus en plus important de définir des critères de contrôle et de qualité pour la localisation des données géographiques et leurs relations dans l'espace. Ces critères ne peuvent pas se résumer à de simples problèmes de précision, comme cela a été longtemps le cas pour la cartographie. Les bases de données gèrent des collections d'objets, et l'attribut de localisation prend ses valeurs dans un espace de dimension deux ou trois, ce qui introduit des critères qui ne sont plus seulement liés à une relation d'ordre, mais à la structure de cet espace : topologie, métrique, relations ensemblistes, etc. [LAU 93]. Par exemple, le respect des critères topologiques (fermeture, connexité, relations spatiales entre objets…) pourra contribuer à faire la différence entre des données ne pouvant être utilisées qu'à des fins de dessin automatique et des données pouvant être intégrées dans une base de données et utilisées dans un système d'information géographique.

Il n'existe pas de description simple de l'incertitude liée à la modélisation de la réalité par des objets et des collections d'objets géographiques. Cette modélisation est néanmoins toujours liée à une échelle géographique de description, qui associe choix des attributs descriptifs et description de la localisation. Mais, lors de l'utilisation des objets, la description de la localisation peut ne plus correspondre à la description sémantique de l'objet car les SIG donnent la possibilité de choisir une échelle différente de l'échelle de validité pour l'utilisation et la mise en relation d'objets. Ceci provoque bien sûr des problèmes de cohérence entre les objets résultant des traitements ainsi effectués. De nombreuses réponses ont été proposées

pour tenter de résoudre les problèmes d'intégrité et de cohérence lors de l'utilisation des objets et de leur mise en relation [GOO 89] [MÜL 95]. Mais lorsque cela est possible, il est plus efficace d'assurer la cohérence de la base de données géographiques lors de la constitution des objets, en respectant des contraintes d'intégrité spatiale qui ne dépendent que de la définition formelle des objets.

L'approche que nous proposons ici vise donc à assurer la cohérence intrinsèque des objets dans une base de données géographiques. Cette cohérence ne dépend pas des traitements effectués, elle dépend uniquement de la définition des objets par rapport à l'objectif de modélisation de la réalité. La démarche s'inscrit donc dans l'approche base de données [DAT 90].

D'une manière générale, l'information sur la qualité des données géographiques peut être décomposée en différents domaines : la généalogie (information sur les sources des données), l'actualité (validité temporelle), la cohérence logique (contrôle de l'intégrité des données), la précision géométrique, la précision sémantique, l'exhaustivité [FAI 96]. Nous ne traiterons ici que de cohérence logique liée à la localisation : nous chercherons à décrire l'ensemble des contraintes d'intégrité permettant de renseigner sur le degré de cohérence spatiale d'une base de données géographiques. L'intégrité ne couvre pas le plus vaste domaine de l'erreur dans les SIG, mais le respect des contraintes permet de minimiser la génération d'erreurs lors des traitements, permettant ainsi de gérer ces erreurs de manière plus fiable [GOO 89], [GUP 95].

7.1.2. *La cohérence spatiale*

La cohérence spatiale s'exprime par rapport à une modélisation de la réalité en entités géographiques, correspondant à la définition de collections d'objets géographiques. Comme nous l'avons déjà vu au chapitre 3, une entité, ou collection d'objets géographiques, sera définie par :

– un type de localisation (classiquement : zone, ligne, point, pixel),

– des attributs descriptifs (la description sémantique des objets),

– des relations et des contraintes spatiales entre les objets.

Dès lors, on peut donc exprimer la cohérence spatiale selon ces trois composantes : une cohérence géométrique (adéquation entre la description géométrique des objets et le modèle géométrique défini pour la collection), et une cohérence sémantique (adéquation entre modélisation de la réalité d'une part et relations entre les objets d'autre part). Les règles d'intégrité spatiale visent à assurer ces deux niveaux de cohérence. Par exemple, à assurer qu'un objet d'une collection de type zone soit bien une zone (un domaine fermé de l'espace, c'est une condition

géométrique), que deux parcelles d'un cadastre ne se chevauchent pas (la condition de non-intersection est sémantique, et provient de la définition même d'un cadastre), qu'un bâtiment appartienne à une et une seule parcelle (c'est également une condition sémantique, mais qui porte sur les relations spatiales entre les objets de deux collections distinctes).

7.1.3. Les contraintes d'intégrité spatiale

L'un des objectifs des SIG, en tant que systèmes de gestion de collections d'objets géographiques, est d'assurer la cohérence spatiale, au même titre qu'un système de gestion de bases de données classique (SGBD) doit assurer la cohérence et l'intégrité des données.

Le modèle relationnel conventionnel (les attributs ne peuvent être que quantitatifs ou qualitatifs) est très efficace dans le domaine de l'intégrité des données et de la cohérence, et il a d'ailleurs été en partie conçu dans cette optique. Il propose des règles d'intégrité permettant d'assurer la cohérence des données d'une base. Ces règles concernent l'unicité de la clé (existence et unicité d'un identifiant pour chaque objet), les contraintes par référence entre relations (un ensemble d'attributs d'une relation est défini comme clé dans une autre relation), et les contraintes de domaine (les valeurs d'un attribut doivent vérifier une assertion logique). On peut également étendre les contraintes par référence aux contraintes par jointures, en imposant des contraintes de domaine au résultat d'une jointure entre deux relations [GAR 84].

On a donc tout intérêt à étendre les contraintes d'intégrité du modèle relationnel conventionnel à de nouveaux types d'objets - avec l'introduction de la localisation comme attribut. Le modèle relationnel nous guide alors naturellement dans la modélisation de la réalité et dans la définition des contraintes d'intégrités liées à ces nouveaux types. En effet, si l'attribut de localisation est défini comme une clé primaire des collections d'objets localisés, il doit vérifier le principe d'unicité (qui correspond alors au principe d'unicité d'un phénomène dans l'espace-temps : au même moment, on ne peut avoir deux objets de la même collection au même endroit). Les contraintes de domaine peuvent être étendues à la dimension 2, à la fois sur la géométrie (position) et sur la topologie (fermeture, simplicité, connexité, relations entre objets voisins). Les contraintes par référence peuvent également être étendues à la dimension 2, notamment au niveau de la géométrie (contours communs, hiérarchie entre objets). Enfin, des contraintes par jointure peuvent également être proposées, en utilisant la localisation comme attribut de jointure entre les objets (une bouche d'égout ne peut se trouver que dans une rue : le résultat d'une jointure géométrique entre bouches d'égout et immeubles doit être vide). D'une manière générale, les contraintes d'intégrité spatiale sont exprimées sur les attributs des objets, que ces attributs concernent la description ou la localisation. Le

modèle relationnel étendu nous fournit donc tous les principes permettant d'exprimer la cohérence spatiale d'une base de données géographiques indépendamment du modèle géométrique interne au SIG.

Le modèle objet supprime la notion de domaine pour les attributs, puisqu'un attribut peut lui-même être un identifiant d'objet d'une autre classe. De ce fait, les contraintes d'intégrité sont plus difficiles à définir et à mettre en œuvre pour les bases de données orientées objet que pour le modèle relationnel. Néanmoins, il convient de respecter des contraintes d'intégrité dans le cadre du modèle orienté objet, sous peine de dégrader considérablement les objectifs assignés aux SGBD et de perdre les bénéfices acquis du relationnel [BEN 93].

On peut établir pour le modèle objet des règles de cohérence encapsulées comme les méthodes : la définition d'une classe peut inclure la définition de méthodes de cohérence sur et entre les membres de la classe. Ainsi, lorsqu'une classe dérive d'une autre, et que ces deux classes possèdent un attribut de localisation (hiérarchie de découpage par exemple), on devrait pouvoir exiger que ces attributs suivent la hiérarchie des objets (par exemple, les arcs des objets d'une classe sont forcément des arcs des objets d'une classe qui en dérive).

7.2. Les contraintes d'intégrité pour les bases de données

Les contraintes d'intégrité permettant d'assurer la cohérence spatiale d'une base de données vont donc s'exprimer dans le cadre du modèle relationnel étendu aux données localisées. Nous allons décrire comment ces contraintes s'expriment pour les objets et les collections d'objets dans les bases de données géographiques.

7.2.1. *Le cadre relationnel*

Nous avons vu que la localisation des objets géographiques s'exprime soit par des éléments de l'espace (pour les objets dont la localisation peut être ramenée à un point), soit par des sous-ensembles de l'espace (lignes ou zones).

La schématisation géométrique de la localisation des objets géographiques en zone, ligne, point, pixel établit une classification naturelle sur laquelle repose à la fois l'extension du modèle relationnel et la définition de classes d'objets géographiques pour le modèle objet. Toutes les considérations ultérieures seront basées sur la classification en zone, ligne, point, pixel, classification qui permet de schématiser la géométrie des objets géographiques et de définir des collections d'objet du même type.

Un arc n'est pas un objet géographique, mais seulement un objet géométrique. Les contraintes d'intégrité géométrique sont exprimées sur les points et les arcs, c'est-à-dire sur les objets géométriques qui permettent de définir la localisation des objets géographiques.

7.2.2. *Les contraintes d'intégrité*

L'un des objectifs des SGBD, c'est d'assurer le maintien et la cohérence des données par rapport aux schémas (contrôle des types), ainsi que la cohérence des données entre elles (par exemple, un enfant de moins de cinq ans ne peut avoir un niveau d'étude élevé). On appelle contrainte d'intégrité toute règle que doivent suivre les données, pour spécifier les valeurs permises pour certaines données, éventuellement en fonction d'autres données.

L'une des contraintes d'intégrité souvent utilisée est la contrainte d'unicité de clé: on impose l'existence d'une clé (identifiant unique) pour tout objet d'une entité. Tout objet d'une entité vérifiant cette contrainte devra donc avoir une valeur bien déterminée (et unique) pour cette clé.

On définit également les contraintes dites de référence : on impose à une association de n'associer que des objets existants dans la base (par exemple, un acte de vente ne peut être enregistré que si l'acheteur et le bien existent déjà dans la base).

Les contraintes les plus courantes sont les contraintes de domaine : on précise le domaine de variation de l'attribut (par exemple, l'âge d'une personne est compris entre 0 et 140). La contrainte peut être plus complexe, et faire appel aux valeurs de plusieurs attributs. Elle se présente alors comme une assertion logique qui doit être vérifiée pour l'ensemble des objets de la relation. On peut étendre les contraintes de domaine aux contraintes par jointure : c'est alors le résultat d'une jointure entre deux relations qui devra répondre à une contrainte de domaine (par exemple, un fils ne peut pas être plus âgé que son père).

7.2.3. *Les contraintes d'intégrité pour les données localisées*

Lorsque les attributs sont de type simple, comme c'est le cas dans les SGBD classiques, les contraintes d'intégrité s'expriment par des assertions logiques fondées sur l'égalité ou la relation d'ordre en dimension 1, comme toute opération algébrique sur les attributs.

Lorsque l'on étend le modèle aux données géographiques, l'attribut de localisation doit lui aussi être soumis à des contraintes d'intégrité. Ces contraintes

s'expriment alors dans l'espace, soit par des assertions ensemblistes (appartenance, intersection, etc.), soit par des assertions topologiques (voisinage, fermeture, connexité, etc.), soit par des assertions métriques (contraintes sur les distances entre objets). Par exemple, la contrainte d'unicité de clé est fondamentale pour une bonne modélisation de la réalité en entités géographiques, car cette contrainte pour la localisation permet d'assurer l'unicité d'un phénomène en un lieu, et d'être sûr que la localisation de tout objet géographique est bien définie.

Les contraintes d'intégrité peuvent porter sur l'attribut de localisation lui-même, mais la localisation peut également être utilisée comme attribut de jointure pour exprimer une contrainte de domaine entre attributs descriptifs simples (par exemple, un immeuble de huit étages ne peut se trouver dans une zone limitée à deux étages).

Les contraintes d'intégrité interviennent lors de la constitution de la base de données. La saisie des données graphiques correspondant à la localisation doit ainsi prendre en compte ces contraintes pour assurer la cohérence de la base, sachant que toute modification ultérieure est particulièrement délicate. Dans le paragraphe suivant, nous allons décrire, pour l'attribut de localisation, l'ensemble des contraintes d'intégrité qui peuvent être nécessaires à la bonne cohérence d'une collection d'objets localisés. Certaines contraintes sont nécessaires et doivent toujours être vérifiées. Les autres sont optionnelles, mais les SIG devraient proposer un formalisme (langage et méthodes) pour permettre leur mise en œuvre lors de la constitution ou la modification d'une base de données [LAU 91], [UBE 96].

7.3. Les contraintes d'intégrité spatiale

7.3.1. *Une typologie des contraintes d'intégrité spatiale*

Nous allons étudier les contraintes d'intégrité portant sur l'attribut de localisation selon la classification suivante :

– les contraintes *géométriques*, portant sur les composants géométriques servant à schématiser les objets géographiques (arc, point, segment) : simplicité, extra-simplicité, ultra-simplicité ;

– les contraintes *topologiques de type,* permettant d'assurer l'intégrité géométrique d'un objet, en fonction de son type : forme de l'objet, relations entre les éléments géométriques constituant un objet, (fermeture, connexité, centroïde). Ces conditions dépendent du type de localisation (zone, ligne, point, pixel) ;

– les contraintes *relationnelles*, permettant d'assurer l'intégrité d'une collection d'objets par rapport à sa définition sémantique : unicité de clé (existence et unicité d'un objet en un point), appartenance à un domaine (assertion logique sur l'attribut de localisation), contraintes de voisinage (assertion logique sur les couples d'objets

d'une même collection, répondant à un critère topologique ou métrique), relations géométriques entre les objets d'une collection ;

– les contraintes *topologiques de jointure* : contraintes topologiques entre les objets de deux collections (assertion topologique sur les couples d'objets résultant d'une jointure géométrique entre deux collections : intersection, inclusion, position, héritage, etc.) ;

– les contraintes *descriptives de jointure* : contraintes descriptives (assertion logique sur les attributs descriptifs des couples d'objets résultant d'une jointure géométrique entre deux collections), contraintes d'héritage (par exemple, un canton ne peut avoir plus d'habitants que le département auquel il appartient).

Les contraintes géométriques et topologiques de type correspondent à la cohérence géométrique. Les contraintes relationnelles et topologiques de jointure correspondent à la cohérence sémantique. Certaines contraintes géométriques ou topologiques de type devront toujours être vérifiées (par exemple, une zone doit toujours être fermée), d'autres dépendent de la définition sémantique de la collection (un réseau routier doit toujours être connexe, mais un réseau téléphonique peut ne pas l'être). Du fait des répercussions immédiates de leur non-respect, les contraintes géométriques et topologiques de type sont celles qui ont été mieux étudiées [EGE 91], [HAD 92], [LAU 91 93 94], [UBE 96], [TAN 95].

Cette typologie des contraintes d'intégrité est fonctionnelle : elle est indépendante du mode de stockage et de représentation interne des objets, et ne prend pas en compte les particularités d'un système par rapport à un autre. Par contre, leur mise en œuvre sera fonction du mode de représentation interne utilisé par le système de gestion de ces objets [LAU 93].

7.3.2. *Les contraintes géométriques*

Les contraintes géométriques s'expriment sur les points ou les arcs servant à schématiser les objets géographiques. Rappelons que les arcs sont constitués de segments, les extrémités des segments sont appelés points de l'arc, et les deux points extrémités de l'arc sont appelés des nœuds. Nous pouvons définir les conditions géométriques ou topologiques sur les arcs et les points, conditions qui nous serviront à exprimer les contraintes d'intégrité sur les objets géographiques eux-mêmes :

– la simplicité d'un arc : l'arc ne doit pas avoir d'intersection avec lui-même (aucun des segments qui le constituent ne doit recouper un autre segment qui le constitue). Cette contrainte s'exprime sur l'arc lui-même, et ne fait pas intervenir d'autres objets (fig. 7.1).

fig. 7.1 : Simplicité des arcs

On imposera pratiquement toujours aux arcs d'être simples, même si cette condition n'est pas nécessaire du point de vue de la cohérence des objets. Mais elle simplifie les opérations sur les arcs, notamment dans les cas où le sens de l'arc est pris en compte.

– l'extra-simplicité d'un arc : l'arc ne doit avoir d'intersection avec aucun autre arc constituant un objet de la même collection (seuls les nœuds peuvent être des points communs entre les arcs de la relation) (fig. 7.2).

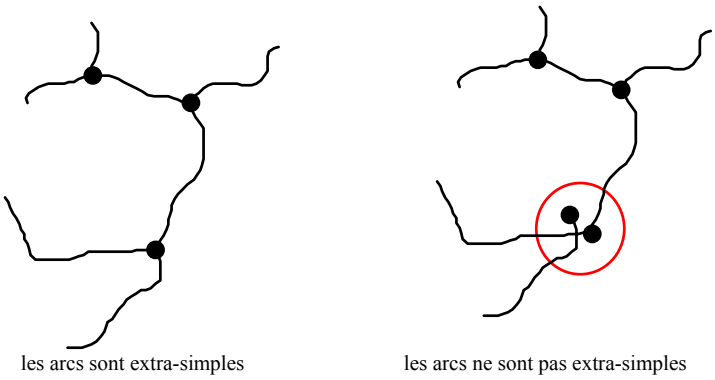


fig. 7.2 : Extra-simplicité des arcs

– l'ultra-simplicité d'un arc par rapport à une autre collection : l'arc ne doit avoir d'intersection avec aucun arc constituant un objet de cette autre collection.

– inclusion d'un arc : un arc est inclus dans un autre si tous les points de l'arc appartiennent à l'autre arc et si deux points consécutifs dans l'arc inclus sont deux points consécutifs dans l'autre l'arc.

– fermeture : un ensemble d'arcs est dit fermé s'il existe un chemin fermé constitué par ces arcs (fig. 7.3).

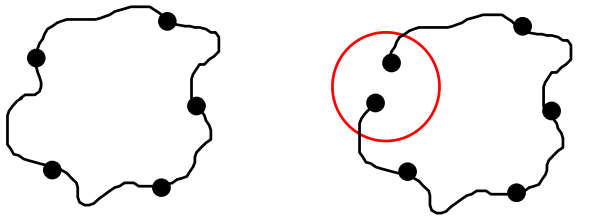


fig. 7.3 : Fermeture d'un ensemble d'arcs

– connexité : un ensemble d'arcs est dit connexe s'il existe un chemin connexe constitué par ces arcs (fig. 7.4).

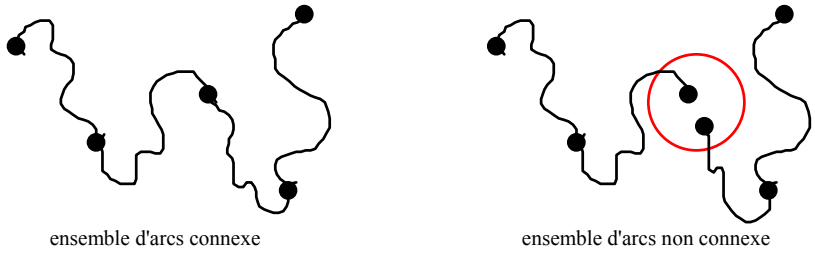


fig. 7.4 : connexité d'un ensemble d'arcs

La plupart des systèmes imposent dès la saisie graphique des conditions de connexité sur les arcs, qu'ils soient composants d'une zone ou d'une ligne, le critère de connexité étant uniquement fonction de la distance séparant leurs nœuds. Si les nœuds de deux arcs sont proches (leur distance est inférieure à un seuil donné), alors ces deux nœuds doivent être identiques (les arcs doivent appartenir à une même composante connexe). Les systèmes assurent alors automatiquement cette contrainte en modifiant la position des nœuds.

Dans une base de données géographiques, on doit imposer aux objets géométriques, arcs et points, d'appartenir à des objets géographiques : les nœuds ou arcs isolés (dit « baladeurs » ou « libres ») sont interdits.

7.3.3. *Les contraintes topologiques de type*

Les contraintes topologiques de type sont les mieux respectées, sous peine de dégradation considérable de la validité des opérations spatiales les plus simples. Elles concernent les objets de type zone et de type ligne. Il n'y a aucune contrainte topologique de type sur les objets point ou sur les objets pixel.

7.3.3.1. *Fermeture et centroïde*

On impose aux zones d'être par définition des fermés de l'espace : les arcs décrivant une zone doivent donc toujours constituer un chemin fermé, définissant la zone de manière unique. Cette condition doit impérativement être vérifiée pour assurer la bonne cohérence des objets zone, et permettre les calculs géométriques ou topologiques (périmètre, surface, remplissage, etc.). Les arcs décrivant une zone doivent également être simples.

On impose également toujours la condition d'appartenance du centroïde à la zone, par définition même du centroïde (un point appartenant à un objet de type zone et représentant le centre de la zone).

On peut également demander aux objets ligne d'être fermés : les arcs décrivant la ligne doivent alors constituer un chemin fermé.

7.3.3.2. *Connexité*

On peut demander aux zones d'être connexes (une zone est alors représentée par un seul polygone). Cette condition n'est pas toujours requise, certains systèmes acceptant que l'objet zone ne soit pas forcément connexe. La connexité est alors une exigence qui dépend de la définition sémantique de la collection d'objet, et non pas du type.

On peut imposer aux réseaux d'être connexes. Les objets sont alors des lignes qui doivent être reliées entre elles pour former un graphe connexe. D'une manière plus générale, un objet constitué de plusieurs lignes forme un graphe. On peut imposer à cet objet d'avoir des propriétés topologiques correspondant à cette structure de graphe : fortement connexe (cas général des réseaux routiers), symétrique, complet, planaire.

Toujours dans le cas des lignes, on peut assortir la condition géométrique de connexité de deux arcs d'une condition sur la valeur des lignes auxquelles

appartiennent les arcs. Par exemple, deux courbes de niveaux doivent être reliées si elles ont un nœud proche et si et seulement si elles ont même valeur.

7.3.3.3 *Surface et périmètre*

On peut imposer une contrainte descriptive sur la surface, sur le périmètre (pour les zones), ou sur la longueur (pour les lignes) d'un objet. C'est en fait une contrainte descriptive classique, puisque la surface, le périmètre, la longueur sont des attributs numériques classiques, leur seule particularité étant de pouvoir être calculés directement à partir de la géométrie des objets.

7.3.4. *Les contraintes relationnelles*

Les contraintes relationnelles concernent la cohérence sémantique d'une collection d'objets. Elles font donc appel aux relations géométriques ou topologiques entre les objets d'une même collection.

7.3.4.1. *Contrainte d'unicité de clé*

Du fait même de la définition d'une entité géographique, la localisation (espace et temps) doit être considérée comme une clé primaire d'une relation localisée. En effet, si tel n'est pas le cas, on pourrait avoir dans une même relation deux objets différents mais situés au même endroit. Dans une base de données géographiques, cette éventualité relève d'une mauvaise schématisation de la réalité et d'une mauvaise définition des objets. On doit donc imposer à la localisation d'être une clé, et à la relation de satisfaire la contrainte d'unicité de clé pour cet attribut. Cela signifie simplement que tout objet doit être localisé et que cette localisation doit être connue pour que l'objet soit présent dans la base, et que deux objets d'une même relation ne peuvent partager le même lieu. Cet identifiant d'objet n'est pas forcément directement visible par l'utilisateur de la base de donnée, mais il doit exister : un objet doit être différencié d'un autre par sa localisation. On peut même caractériser une relation localisée par cette contrainte, en disant qu'une carte doit être une tessellation du plan [PLÜ 97].

La contrainte d'unicité de clé pour la localisation est forte. Elle assure dans les bases de données relationnelles localisées la même fonction que le principe de l'existence d'un identifiant d'objet dans les bases de données orienté objet.

Ce principe doit également être respecté dans les bases spatio-temporelles, et il porte alors sur la localisation dans l'espace-temps (dans une relation, on ne peut avoir deux objets au même lieu et au même instant). A un instant t fixé, on retrouve les contraintes sur la localisation du cas non temporel. On peut également imposer des contraintes d'intégrité sur la localisation dans l'espace-temps (par exemple, en

donnant une contrainte de domaine sur la vitesse de déplacement d'un objet). Dans la suite, nous ne considérerons plus que les bases non temporelles (ou l'état d'une base temporelle à un instant t fixé).

La contrainte d'unicité de clé pour les points ne pose aucun problème : on ne peut pas avoir dans une relation de type point deux objets distincts au même endroit. Cette condition est facilement vérifiable.

La contrainte d'unicité pour les lignes impose à deux lignes d'une même relation de n'avoir aucune intersection. Deux lignes ne peuvent donc se rencontrer qu'aux extrémités de leurs arcs (sur les nœuds), et ne doivent donc pas se croiser : tous les arcs doivent vérifier la condition d'extra-simplicité. Par contre, il n'est pas forcément requis à un arc d'être simple (une ligne peut se croiser sur elle-même). Il est parfois normal que des lignes se croisent (par exemple, c'est souvent le cas dans un réseau de rues avec les passages souterrains ou aériens), mais elles ne se croisent alors que dans leur projection en deux dimensions : une bonne schématisation de la réalité doit, dans ce cas, prendre en compte trois dimensions dans l'espace, et le principe d'unicité est alors vérifié.

La contrainte d'unicité pour les zones impose à deux zones quelconques d'une même relation d'avoir une intersection vide. On considère que les points des arcs servant à définir les zones n'appartiennent pas à la zone : deux zones peuvent donc partager les arcs qui les constituent (ils acquièrent alors le statut topologique de frontière), mais les domaines ouverts (le domaine fermé moins les frontières) constitués par ces arcs doivent avoir une intersection vide. Deux arcs constituant des zones d'une même relation ne peuvent donc pas se croiser : tous les arcs doivent vérifier le principe d'extra-simplicité ou d'inclusion.

On aura donc divers problèmes à résoudre, dépendant du mode de saisie graphique. Les erreurs d'extra-simplicité proviennent souvent d'arcs saisis par erreur deux fois ou d'une mauvaise précision dans le raccord des nœuds. On a également des problèmes d'extra-simplicité, d'inclusion et de fermeture lorsque des arcs sont frontières de zones appartenant à des documents différents.

7.3.4.2. *Contrainte d'appartenance à un domaine*

On peut imposer aux objets d'une relation d'appartenir à un domaine de l'espace. Cette condition correspond à la condition classique de vérification d'une assertion logique pour les données de dimension 1, l'assertion portant ici sur l'attribut de localisation et pouvant être exprimée soit sous forme ensembliste, soit sous forme métrique, suivant la définition du domaine. Cette contrainte permet également de s’assurer de la validité des coordonnées des points : tous les objets situés sur le globe terrestre doivent vérifier une contrainte minimum (longitude de 0° à 360°, colatitude de 0° à 90°).

7.3.4.3. *Contraintes descriptives de voisinage*

On peut imposer une contrainte sur les valeurs d'un attribut en fonction d'une condition de voisinage entre objets d'une même collection. En général, on fixera des conditions sur l'écart entre les valeurs pour deux objets voisins. Ce type de contrainte intervient par exemple pour les courbes de niveaux : la différence des valeurs de deux courbes voisines ne doit pas dépasser une valeur d'équidistance donnée. D'une manière plus générale, on peut imposer à un attribut descriptif de satisfaire à une propriété fonctionnelle par rapport à la position dans l'espace (par exemple, continuité uniforme).

Il faut définir la condition de voisinage pour chaque type d'objet, par exemple :

– deux lignes sont voisines s'il existe un segment de droite reliant un point dans chaque ligne et ne coupant aucune autre ligne (ou encore si tout segment de droite reliant la ligne à une autre ligne coupe l'autre ligne),

– deux points sont voisins au sens de Voronoï (ils sont contigus dans le graphe dual de Delaunay),

– deux zones sont voisines s'il existe un arc frontière appartenant aux deux zones,

– deux pixels sont voisins s'ils ne sont séparés que d'une maille.

Par exemple, imposer à un modèle numérique de terrain de ne pas contenir de cuvette (un point plus bas que tous ses voisins) est une contrainte descriptive de voisinage qui permet d'assurer la validité de nombreux calculs en hydrologie (c'est un des rares exemples de contraintes d'intégrité sur les objets de type pixel).

7.3.4.4. *Contraintes topologiques*

Les contraintes topologiques concernent les relations topologiques entre deux objets d'une même collection. Comme les objets d'une même collection sont de même type (zone, ligne, point, pixel), et que, pour une bonne modélisation de la réalité, ils doivent de plus satisfaire au principe d'unicité de clé (pas d'intersection entre les objets d'une même collection), les contraintes topologiques au sein d'une même collection ne concernent donc que l'adjacence et la connexité.

On peut ainsi contraindre une collection zonale à être une tessellation de l'espace (contrainte d'adjacence) : les arcs schématisant les zones doivent alors constituer un graphe planaire.

Pour les collections de type ligne, les contraintes de connexité ne concernent pas seulement les arcs d'un objet isolé, mais portent sur tous les objets entre eux. L'ensemble des objets doit alors répondre à une condition topologique : former un graphe connexe, fortement connexe, symétrique, complet, planaire.

Il n'y a pas de contraintes topologiques sur les collections de type point ou pixel.

7.3.4.5. *Contraintes métriques*

Les contraintes métriques suivent le même principe que les contraintes de voisinage, mais les conditions topologiques sont remplacées par des conditions sur la distance entre les objets. La distance entre deux zones ou deux lignes est définie comme le minimum des distances entre tous les points des deux objets ($d(O1,O2)=\min(d(x,y), x \in O1, y \in O2)$).

7.3.5. *Les contraintes de jointure*

Les contraintes spatiale de jointure sont des conditions que l'on impose entre les objets de deux collections distinctes, soit directement sur les relations spatiales entre objets, soit sur les valeurs des couples d'objets répondant à un critère spatial. Si les contraintes portent uniquement sur les relations spatiales entre objets (intersection, inclusion, position respective, distance, etc.), on parle de contrainte topologique (par exemple, une route ne doit pas avoir d'intersection avec un immeuble, une bouche d'égout doit se trouver dans une rue, etc.). Si les contraintes portent sur les valeurs des couples d'objets répondant à un critère de localisation, on parle de contrainte descriptive (par exemple, le nombre d'étage d'un immeuble ne doit pas dépasser le nombre d'étage permis par la zone de réglementation auquel l'immeuble appartient). On nomme ces contraintes d'intégrité spatiale *contraintes de jointure,* car elles correspondent à des conditions topologiques, métriques, ou descriptives sur le résultat d'une jointure géométrique entre deux collections d'objets localisées.

Les objets à mettre en relation appartiennent à des collections différentes, et ces collections peuvent être de type géographique différent. Nous allons donc d'abord étudier l'expression des relations spatiales entre entités géographiques quelconques, avant d'étudier la mise en œuvre formelle des contraintes d'intégrité par jointure spatiale.

7.3.5.1. *Les relations spatiales entre objets géographiques*

Pour modéliser la réalité, il n'est pas suffisant de définir des entités géographiques. Il faut également définir des relations spatiales entre ces entités. Ces relations spatiales sont importantes, notamment pour la représentation des connaissances [FRA 93] [EGE 90], le raisonnement spatial [FRA 96] [EGE 94], et, bien sûr, les contraintes d'intégrité spatiale [UBE 96]. Elles peuvent être classées de la manière suivante [PUL 88][EGE 89] :

– les relations basées sur un ordre spatial lié à la définition d'une origine ou d'un axe (par exemple, une direction),

– les relations topologiques qui décrivent voisinages et propriétés ensemblistes (par exemple, *disjoint*, *adjacent*, *intersecte*),

– les relations métriques liées à une distance.

Les relations topologiques ont donné lieu à de nombreux travaux, car ce sont les plus importantes vis à vis du raisonnement et de la cohérence spatiale [EGE 90] [FRA 92b] [CLE 93] [CUI 93] [BOU 94] [MOL 94] [FRA 96]. Ainsi, plusieurs modèles théoriques de représentation des relations topologiques ont été proposés :

– le modèle des 9-Intersections, qui s'appuie sur les notions topologiques d'intérieur et de frontière : une relation topologique entre deux objets est décrite par les 9 intersections de l'intérieur, la frontière, et l'extérieur du premier objet avec l'intérieur, la frontière, et l'extérieur du second objet.

– le modèle RCC (*Region Connection Calculus*) utilise une approche axiomatique fondée sur le calcul de prédicats du premier ordre. La base du formalisme est la relation de connexion « *x* est connecté à *y* », qui signifie que les ensembles *x* et *y* partagent un point. Un ensemble d'autres relations est défini à partir de cette relation de connexion [CUI 93] [COH 93].

– le modèle CBM (*Calculus Based Method*), qui, à partir de cinq relations ensemblistes (*touch, in, overlap, disjoin, cross*) et d'opérateurs de frontières, permet de coder toutes les situations topologiques entre deux objets [CLE 95].

Afin de faciliter l'utilisation des modèles de représentation, on peut grouper les relations topologiques en fonction du type de l'objet et du type de relation [UBE 97]. La définition mathématique comprend les notations suivantes : pour un ensemble Y, Y° représente l'intérieur, ∂Y représente la frontière, Y- représente l'extérieur. Le type polygone correspond au type zone limité aux domaines connexes de genre 1 (zones sans trou). Le nombre de relations indique le nombre de cas possibles de relations topologiques différentes pour chaque regroupement (d'après [UBE 97]).

Le groupe Point / Point		
Nom de la relation	*Définition mathématique*	*Nombre de relations*
EGALITE	$(P_1 \cap P_2 = 0)$	1
DISJOINT	$(P_1 \cap P_2 = \phi)$	1
Le groupe Point / Ligne		
EXTREMITE	$(P \cap \partial L = 0)$	1
SUR	$(P \cap L° = 0)$	1
DISJOINT	$(P \cap \partial L = \phi) \wedge (P \cap L° = \phi)$	1
Le groupe Point / Polygone		

DANS	$(P \cap R^\circ = 0)$	1
SUR	$(P \cap \partial R = 0)$	1
DISJOINT	$(P \cap \partial R = \phi) \wedge (P \cap R^\circ = \phi)$	1
Le groupe Ligne / Ligne		
CROISE	$(L_1^\circ \cap L_2^\circ = 0)$	11
JOINT	$(\partial L_1 \cap \partial L_2 = 0)$	14
RENCONTRE	$(\partial L_1 \cap L_2^\circ = 0)$	23
COUVRE	$(L_1^\circ \cap L_2^\circ = 1)$	13
DISJOINT	$(\partial L_1 \cap \partial L_2 = \phi) \wedge (\partial L_1 \cap L_2^\circ = \phi) \wedge (L_1^\circ \cap \partial L_2 = \phi) \wedge (L_1^\circ \cap L_2^\circ = \phi)$	1
Le groupe Ligne / Polygone		
DANS	$(L^\circ \cap R^\circ = 1) \wedge (L^\circ \cap R^- = \phi)$	9
DEHORS	$(L^\circ \cap R^\circ = \phi)$	10
CROISE	$(L^\circ \cap R^\circ = 1) \wedge (L^\circ \cap R^- = 1)$	12
RENCONTRE	$(\partial L \cap \partial R = 0) \wedge (L^\circ \cap \partial R \neq 1)$	11
BORDE	$(L^\circ \cap \partial R = 1)$	13
DISJOINT	$(\partial L \cap \partial R = \phi) \wedge (\partial L \cap R^\circ = \phi) \wedge (L^\circ \cap R^\circ = \phi)$	1
Le groupe Polygone / Polygone		
CONTIENT-DANS	$(R1^\circ \cap R2^\circ = 2) \wedge (R_1^\circ \cap \partial R_2 = \phi)$	4
DEHORS	$(R_1^\circ \cap R_2^\circ = \phi)$	3
RECOUVRE	$(R_1^\circ \cap R_2^\circ = 2) \wedge (\partial R_1 \cap R_2^\circ = 1) \wedge (R_1^\circ \cap \partial R_2 = 1)$	2
RENCONTRE	$(R_1^\circ \cap R_2^\circ = \phi) \wedge (\partial R_1 \cap \partial R_2 = 0)$	1
BORDE	$(R_1^\circ \cap R_2^\circ = \phi) \wedge (\partial R_1 \cap \partial R_2 = 1)$	1
DISJOINT	$(\partial R_1 \cap \partial R_2 = \phi) \wedge (R_1^\circ \cap R_2^\circ = \phi)$	1

7.3.5.2. *Les contraintes topologiques*

Les contraintes topologiques permettent de spécifier quelles relations topologiques sont interdites ou imposées entre les entités géographiques de la base de données. Elles n'imposent de contraintes que sur la localisation des objets. La difficulté de mise en œuvre vient de la description des situations par des relations topologiques élémentaires ou des regroupements de relations. Des langages de contraintes ont ainsi été proposés par plusieurs auteurs [HAD 92] [UBE 97].

Les contraintes topologiques les plus courantes sont néanmoins assez simples, comme par exemple les contraintes d'inclusion des objets d'une entité dans les objets d'une autre entité (pour un cadastre, les bâtiments doivent être inclus dans les

parcelles), ou les contraintes de non-intersection (les rues ne doivent pas croiser les parcelles).

7.3.5.3. *Les contraintes descriptives*

Ces contraintes s'expriment sur un attribut descriptif obtenu par jointure géométrique entre deux collections (mise en relation d'objets par leur localisation). Elles n'imposent pas de conditions sur la localisation, mais sur les valeurs des attributs des couples d'objets qui satisfont à une condition topologique ou métrique de jointure. Ce sont des contraintes sur le résultat descriptif de véritables requêtes spatiales : zone dans zone, point dans zone, agrégation, zones adjacentes, dispersion…

Par exemple, on peut imposer à un immeuble de n'avoir pas plus d'étages que ne le permet la zone d'occupation du sol dans laquelle il se trouve (pour mettre en œuvre cette contrainte, il faut, pour chaque immeuble, calculer par géo-jointure le nombre d'étages maximum de la zone d'occupation du sol dans laquelle il se trouve, et comparer avec le nombre d'étages de l'immeuble). On peut imposer à un modèle numérique de terrain de ne pas présenter de différences de valeurs avec des valeurs ponctuelles d'altitude (cette contrainte correspond à la condition d'égalité -ou de faible différence- sur le résultat d'une équi-jointure géométrique entre les deux collections d'objets : des points cotés, des pixels d'un modèle numérique).

Pour toutes les contraintes d'intégrité faisant intervenir une distance, par jointure ou par appartenance, on peut introduire une incertitude de manière à gérer l'incertitude de la mesure de la localisation (la précision géométrique). La propagation de cette incertitude rend les contraintes moins fortes et permet ainsi d'éviter les rejets injustifiés dans les contrôles de cohérence.

7.4. Modèles géométriques et mise en œuvre des contraintes d'intégrité spatiale

Jusqu'à présent, nous avons étudié les contraintes spatiales indépendamment de la structure interne et des méthodes de stockage des éléments graphiques dans tel ou tel système. En fait, les méthodes de stockage sont essentiellement construites de manière à faciliter les contrôles de cohérence. Par exemple, la méthode de stockage la plus simple (de type dessin où tous les éléments sont mélangés) ne permet aucun contrôle de cohérence : c'est un problème constant dans les SIG que de récupérer des cartes digitalisées sans aucune relation avec la structure de la base de données.

7.4.1. *Les différentes méthodes de stockage*

Les modèles de stockage peuvent être classés en deux grandes familles : les modèles *spaghetti*, qui offrent uniquement des primitives géométriques pour la représentation des objets géographiques, et les modèles *topologiques*, qui conservent à la fois des primitives géométriques et des relations topologiques entre les primitives ou les objets géographiques. Plus le modèle est simple, plus le coût de la digitalisation est faible, et plus le stockage est direct. A l'inverse, si le modèle est peu structuré, il n'offre que peu de contrôle au niveau des objets, et rend difficiles sinon impossibles les recherches suivant un critère spatial.

7.4.1.1. *Les modèles spaghetti*

Le modèle le plus simple consiste à stocker les éléments graphiques sans aucune structure : les points, les arcs, les libellés, sont digitalisés comme du dessin. Seul l'indication d'un niveau, ou d'un attribut graphique (comme une couleur ou un type de symbole), permet de différencier l'appartenance à une collection. Si la notion de polygone n'est pas présente, il est impossible de reconstituer les objets de type zone, ou plus généralement d'établir une correspondance entre les objets géographiques et les éléments géométriques qui définissent sa position dans l'espace. C'est le modèle spaghetti *anarchique*, qui ne permet pratiquement aucun contrôle de cohérence. Seuls des contrôles de type géométrique sur les arcs sont possibles (de type simplicité ou extra-simplicité). Les documents de ce type sont malheureusement très nombreux, car souvent issus de logiciels de dessin, et donc digitalisés sans structuration au préalable. Pour intégrer ces documents dans un SIG, il est souvent nécessaire de recourir à une édition : labelisation des objets, organisation des arcs en polygones, séparation des niveaux et regroupement suivant les entités géographiques. On arrive alors à la notion de modèle spaghetti *simple*, *polygonal*, *polygonal unifié*, où l'élément géométrique reste l'arc, mais structuré par rapport à son appartenance à un polygone. Ces modèles ne conservent pas de topologie, mais permettent en général de la reconstituer en utilisant les références entre les objets - si des contraintes géométriques sont respectées, telle que fermeture des chemins, simplicité, etc. La séparation des objets par entité géographique reste fondamentale pour la vérification de la cohérence de l'ensemble. On est donc dans un cas où la cohérence spatiale ne peut être assurée que par le respect de contraintes d'intégrité géométrique, souvent difficiles à établir et à respecter du fait de la faible structuration des éléments graphiques digitalisés.

7.4.1.2. *Les modèles topologiques*

On parle de modèle topologique lorsque l'on introduit des relations topologiques d'adjacence, de frontière ou de connexité dans la description géométrique des objets. On peut ainsi conserver les relations de connexion entre arcs, les relations entre arcs et nœuds, l'appartenance d'un arc à un polygone, etc. Le modèle le plus répandu

s'appuie sur les notions de points, de nœuds et d'arcs, avec des relations entre les points initiaux, les points finals, les faces gauche et droite, et les arcs sortants, gauche et droit (modèle surfacique planaire).

Les modèles topologiques offrent alors de nombreuses possibilités de contrôle de cohérence, grâce à la conservation explicite des relations d'adjacences et de connexions entre arcs. Mais, comme pour les modèles spaghetti, la clé d'un bon contrôle de cohérence est la bonne gestion des entités géographiques : on peut alors vérifier la cohérence pour les objets d'une même collection avant de vérifier la cohérence entre collections, en suivant les principes que nous avons énoncés dans les paragraphes précédents. Il est pour cela essentiel de structurer les documents en fonction de la définition des entités géographiques avant de les digitaliser.

7.4.2. *La vérification de la cohérence spatiale*

Pour les contraintes géométriques, les contraintes topologiques de type, et les contraintes relationnelles, la vérification de la cohérence spatiale utilise des opérations géométriques simples. La difficulté de mise en œuvre vient donc plus de la définition des contraintes et de la structuration de l'information que de la réalisation informatique des opérations. Mises à part les contraintes géométriques, toutes les opérations de vérification de la cohérence spatiale exigent de pouvoir reconstituer les relations entre les objets et les éléments graphiques utilisés pour la description de leur localisation. Cette reconstitution est d'autant plus simple que le modèle géométrique utilisé est plus complexe. Une fois cette reconstitution effectuée, les contrôles de cohérence utilisent des opérations géométriques élémentaires, telles que :

– les contrôles de simplicité et d'extra-simplicité basés sur la recherche d'intersections entre segments d'arcs,

– pour les objets de type point, les contrôles d'unicité de clé faisant appel uniquement des calculs de distance,

– les contrôles de fermeture basés sur des parcours d'arcs et donc sur des calculs de distances et de parcours entre les nœuds des arcs constituant un polygone,

– pour l'unicité de clé concernant les objets de type zone, il faut assurer la simplicité et l'extra-simplicité des arcs, et l'ajustement des frontières entre zones adjacentes (union d'arcs),

– les trous dans les zones sont des problèmes relevant de l'unicité de clé (un trou ne doit pas être considéré comme une autre zone, car deux zones d'une même collection partageraient alors un même lieu). Certains systèmes gèrent ce problème par le sens des arcs des polygones, d'autres par la prise en compte directe de l'adjacence. Le contrôle général de l'unicité de clé pour les zones exigent de tester la non-intersection de deux zones d'une même collection. On peut utiliser pour cela, en

plus de la vérification d'extra-simplicité des arcs, une vérification de non-inclusion par un algorithme de type y-x.

– les contrôles de connexion impliquant des calculs de distances entre nœuds.

Les contraintes topologiques de jointure font appel en général à des conditions de non-intersection ou de non-inclusion entre objets dont les types peuvent être différents. Elles utilisent les mêmes algorithmes graphiques que précédemment : vérification de l'extra-simplicité des arcs, non-inclusion, calculs de distance, etc.

7.5. SAVANE, saisie graphique et contraintes d'intégrité spatiale

Le système SAVANE comprend un certain nombre de contrôles d'intégrité spatiale. Ces contrôles se situent essentiellement au niveau de la saisie des données, dans le module SAVEDIT. Ils concernent l'unicité de clé, la fermeture de zones, le placement du centroïde, la connexité, la simplicité des arcs, l'extra-simplicité des arcs, les contrôles de valeur, les contrôles de jointure. Nous allons en décrire l'implémentation après avoir présenté rapidement le module de saisie.

7.5.1. Présentation du module SAVEDIT

7.5.1.1. Les principes de la saisie graphique avec SAVEDIT

Le module SAVEDIT permet la saisie graphique sur écran, à partir de documents scannés puis géoréférencés (notamment grâce au module SAVAMER, comme nous l'avons vu dans le chapitre précédent). La saisie graphique doit suivre la modélisation de la réalité et la structure de la base de données qui en découle : chaque collection d'objet doit être saisie séparément l'une de l'autre, et donnera lieu à la création d'un document séparé. La saisie crée directement des documents dont la structure est proche du modèle de stockage topologique de SAVANE pour la localisation : il ne s'agit pas de saisie de type spaghetti, mais bien d'une saisie avec topologie permettant de nombreux contrôles d'intégrité, dont certains sont effectués de manière interactive ou en arrière-plan.

Á chaque objet saisi sera affectée une valeur, correspondant à un attribut de l'objet servant de clé pour les autres attributs descriptifs. Cet attribut, nommé *clé* dans le logiciel, servira d'attribut de jointure lors de l'intégration dans la base de données des valeurs des autres attributs descriptifs de l'objet. SAVEDIT permet également de saisir directement une valeur numérique pour chaque objet (par exemple, lors de la saisie de courbes de niveaux, on peut saisir directement la valeur de la courbe dans un attribut numérique sans avoir à passer par un identifiant nominal, ce qui serait très lourd. Aucune clé descriptive nominale n'est saisie, et

aucune autre valeur, à part celle directement saisie dans SAVEDIT, ne pourra être affectée aux objets dans la base de données).

L'interface utilisateur de SAVEDIT est identique à celle du module SAVAMER. Lors de l'entrée dans le logiciel, on indique une base de données, un utilisateur et une vue externe. Une fois la base ouverte, l'écran de SAVEDIT correspond à un espace géographique, dans lequel on peut dessiner des objets existants dans la base de données, des documents déjà saisis, des images géoréférencées, et des fonds graphiques géoréférencés, au format DXF. Cet espace géographique peut être modifié grâce aux boutons de zoom de la barre d'outils ou grâce aux commandes du menu WIND. L'utilisateur peut également choisir la projection géographique parmi les nombreuses possibilités qui lui sont offertes, et les objets de la base de données seront dessinés dans cette projection géographique.

La saisie s'effectue ensuite, après avoir créé un document ou ouvert un document existant. C'est lors de la création d'un document que l'on doit indiquer le type du document (zone, ligne, point) et le type de l'identifiant à saisir pour chaque objet (nominal, numérique). On peut également indiquer un certain nombre d'autres paramètres facultatifs (nom de l'opérateur, *datum*, échelle de précision). Le processus de saisie est différent suivant le type du document. La saisie de points ou de lignes est très simple, la saisie de zone est un peu plus complexe. Des options de visualisation sont disponibles pour tous les types de document (affichage de la clé ou de la valeur des objets par exemple).

Le principe de la saisie est donc très différent d'un logiciel de saisie graphique de type CAD. Avec SAVEDIT, les différents niveaux sont séparés dans des documents différents, chaque document correspondant aux objets d'une relation de la base. On n'indique lors de la saisie aucun attribut graphique (de type couleur, épaisseur de trait, …), mais on doit donner pour chaque objet un identifiant descriptif.

7.5.1.2. Saisie et indexation géographique

Comme nous l'avons vu au chapitre 5, l'indexation graphique dans la base de donnée utilise directement le découpage géographique en feuille ou coupure de carte. Il convient donc de définir ou de conserver ce découpage lors de la saisie graphique. La saisie des objets d'une même relation donne donc lieu en général à la création de plusieurs documents de saisie, chaque document couvrant un espace géographique donné, correspondant à ce découpage en feuilles. Pour l'efficacité à la fois du système de saisie (SAVEDIT) et du système d'indexation dans SAVANE, il est préférable de ne pas dépasser un certain nombre d'objets par document (environ 2000 pour les zones, 5000 pour les lignes ou les points), même si le système de saisie supporte un nombre d'objet par document bien supérieur (150000).

Si le contrôle du découpage est assez simple lorsque l'on saisit des objets à partir de cartes existantes, il est plus complexe lorsque les objets sont importés à partir de documents numériques saisis sur un autre système n'utilisant pas ces principes d'indexation. Dans ce cas, il faut utiliser les options de découpage automatique lors de l'importation.

7.5.1.3. La saisie de points

Le processus de saisie de points est très simple : il suffit, une fois dans le mode de saisie de points, de cliquer avec la souris sur l'écran pour saisir un point. Le point est validé, avec sa clé, lorsque l'utilisateur appuie sur le bouton *accepter* du dialogue de saisie (ou sur la touche *Entrée*). Un point peut être ensuite déplacé, supprimé, ou sa clé modifiée, lorsqu'il est sélectionné par l'utilisateur.

7.5.1.4. La saisie de lignes

La saisie de lignes est également très simple, chaque arc saisi correspondant à un objet. Pour saisir un arc, il suffit de saisir les points de l'arc (plusieurs modes de saisie sont à disposition de l'utilisateur : point par point, en continu sur la distance ou sur la surface entre trois points contigus), indiquer la valeur de la clé, puis valider.

Un arc peut être modifié par la suite : ajouter, déplacer ou supprimer des points, filtrer, lisser, modifier la clé descriptive. On peut également le couper en deux, le déplacer, le fermer sur lui-même, etc. Un menu contextuel contenant ces options peut être visualisé lorsqu'un arc est sélectionné dans l'écran.

Certaines procédures permettent de régler directement lors de la saisie de l'arc des contraintes de connexité : une option permet de joindre automatiquement les extrémités des arcs lorsque la distance est inférieure à une valeur qui peut être paramétrée, en pixels écran (la valeur absolue dépend donc de la fenêtre de visualisation : plus le territoire de la fenêtre est étroit, plus le pixel écran est grand, en valeur absolue). De même, déplacer l'extrémité d'un arc proche de l'extrémité d'un autre arc va joindre automatiquement les deux arcs si l'option est activée.

7.5.1.5. La saisie de zones

La saisie de zones se fait en deux temps : dans un premier temps saisie des arcs frontières de la zone, et dans un second temps saisie de l'objet zone lui-même avec indication de la valeur de la clé descriptive. La saisie d'un arc s'effectue indépendamment de la zone, avec les mêmes possibilités que pour la saisie d'arc dans le cas des objets de type ligne. Par contre, lors de la définition de l'objet zone, il faut indiquer la valeur de la clé descriptive ainsi que tous les arcs formant la frontière de la zone. Un arc ne peut bien sûr n'être frontière que de deux zones.

Comme pour les lignes, certaines options permettent d'assurer directement des contraintes de connexité lors de la saisie. Par exemple :

- jointure automatique des extrémités proches,

- création automatique d'un nouveau nœud par division d'un arc déjà saisi, lorsque l'on saisit un nouveau nœud proche de cet arc.

- création automatique d'un nœud avec division des arcs à chaque intersection d'arcs.

De plus, certaines options de saisie d'arcs sont spécifiques aux arcs de zones pour assurer la cohérence topologique :

- inclusion d'un arc dans les arcs frontières d'une zone,

- exclusion d'un arc des arcs frontières d'une zone,

- indication de visibilité (certains arcs, créés uniquement pour assurer la fermeture des zones, ne devront pas être visualisé dans une représentation cartographique).

- option permettant d'assurer la persistance de la connexité des arcs, dans le cas de modification d'un nœud (les extrémités des autres arcs sont alors également déplacées).

A chaque zone est affecté un identifiant interne (un numéro, à partir de 20 ; les 20 premiers numéros sont réservés). Chaque arc conserve les numéros des deux zones adjacentes dont il est frontière. Un arc est dit *libre* s'il n'est affecté à aucune zone (ses deux numéros de zones adjacentes ont pour valeur 0). Un arc est dit *pendant* si l'une de ses extrémités n'est rattachée à aucun autre arc. Un document en fin de saisie ne doit contenir ni arcs libres ni arcs pendants. Nous employons indistinctement le terme *bout* ou *nœud* pour désigner l'extrémité d'un arc.

7.5.1.6. Saisie, datum et projection géographique

Chaque point saisi sur l'écran est transformé immédiatement en coordonnées géographiques (longitude, colatitude) selon la projection géographique utilisée. Tous les points sont donc conservés en coordonnées sphériques. Le *datum* d'un document peut être déclaré pour information, et peut être modifié par la suite : dans ce cas, l'utilisateur a le choix de transformer ou non les coordonnées des points saisis. Ceci peut être nécessaire car tous les objets d'une base de données SAVANE doivent être exprimés dans le même *datum*. Si le document d'origine n'est pas dans le *datum* de la base de données dans laquelle il est destiné à être intégré, il faut d'abord le saisir dans son *datum* d'origine, puis transformer toutes les coordonnées saisies grâce à l'option de changement de *datum*.

Le changement de *datum* utilise les formules de Molodensky telles qu'exposées dans le chapitre 4.

7.5.2. Les contraintes sur les objets géométriques

7.5.2.1. Les classes CArc et CSaveditDocument

La classe `CArc` de SAVEDIT permet d'encapsuler l'ensemble des membres et des procédures nécessaires à la manipulation d'un arc et à la vérification des contraintes géométriques (A.4.1). L'allocation mémoire pour les points (tableau `m_ArrayPoint`) est dynamique. Elle utilise les fonctionnalités de la classe `CArray` de la MFC (Microsoft Fundation Class).

Toutes les variables et procédures liées au document de saisie sont regroupées dans une classe nommée `CSaveditDocument` (A.4.2).

7.5.2.2. Nettoyage des arcs

De nombreux tests peuvent être effectués pour éviter d'avoir des arcs inconsistants. On peut ainsi :

- éliminer les arcs de moins de deux points (cas impossible en théorie, mais pouvant provenir d'un défaut du logiciel),

- contraindre les coordonnées des points saisis à rester dans des limites géographiques données,

- éliminer les arcs trop petits. Certains arcs très petits peuvent être créés accidentellement par les options de division ou de jointure automatique, et sont ensuite difficiles à sélectionner sur l'écran pour les éliminer manuellement.

Ces procédures ne posent aucun problème algorithmique. Elles sont implémentées dans la méthode `NettoyerTousLesArcs()` de la classe `CSaveditDocument` (A.4.2).

7.5.2.3. Simplicité

La simplicité des arcs est une contrainte qui doit toujours être respectée, sous peine d'incohérence future, surtout lorsqu'il s'agit d'arcs formant le contour de zones. En effet, certains logiciels ne supportent pas les arcs qui ne sont pas simples et créent automatiquement un polygone correspondant à la boucle pour éliminer le problème. Mais cette solution n'est pas agréable, car elle peut entraîner des incohérences avec des fichiers de valeurs descriptives. Le contrôle de simplicité teste pour chaque arc l'intersection de tous les segments deux à deux (fig. 7.5). Il utilise la procédure `SegmentIntersection()` que nous avons vue au chapitre 3 (A.4.1).

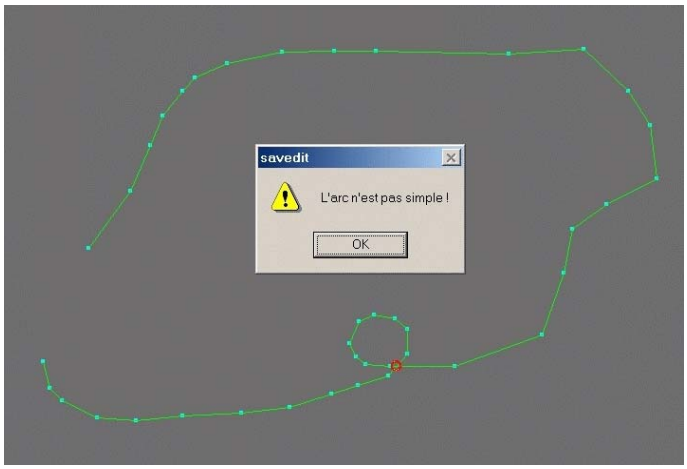


fig. 7.5 : test de simplicité sur un arc dans SAVEDIT

7.5.2.4. Extra-simplicité

L'extra-simplicité teste l'intersection de chaque arc avec tous les autres. On teste pour cela l'intersection de chaque segment d'un arc avec les segments de tous les autres arcs. Bien sûr, les arcs qui ne peuvent avoir d'intersection sont écartés tout de suite (on utilise le rectangle incluant l'arc), sinon le test serait très long.

7.5.2.5. Fusion d'arcs

La fusion d'arc permet de résoudre le problème de l'incohérence entre deux documents, ou d'éviter la saisie directe de portions d'arc en double. Cette procédure peut être activée lors de la saisie (on parle alors de *snapping*), en correction sur un arc, ou pour l'ensemble des arcs du document. En saisie, chaque point digitalisé est comparé à l'ensemble des arcs déjà saisi ; s'il existe un arc proche, c'est-à-dire à une distance inférieure au minimum donné pour la fusion (c'est un paramètre qui peut être choisi par l'utilisateur), il est remplacé par le point de l'arc en question. Ce point est calculé en utilisant la procédure *PointLePlusProche()*. En correction, la procédure est effectuée sur l'ensemble des points de l'arc, arc qui est ensuite filtré pour éliminer les éventuels points en double. Effectué sur l'ensemble du document, la fusion compare l'ensemble des arcs entre eux (A.4.1).

Dans le cas des zones, il faudra après la fusion éliminer les arcs en double, puisque la fusion crée des portions identiques au sein d'arcs différents.

7.5.2.6. Suppression d'arcs en double

Avec le modèle topologique (arcs frontières), l'une des erreurs les plus courantes provient des arcs saisis en double. Un arc en double provoque une erreur d'extra-simplicité, quelquefois difficile à identifier car, si les deux arcs se recouvrent exactement, l'erreur n'est pas visible à l'écran. Il convient d'éliminer les portions d'arcs en double sans détruire la cohérence de l'ensemble. L'algorithme utilisé dans SAVEDIT teste, sur l'ensemble du document, les portions d'arcs identiques, et découpe les arcs pour éliminer l'une des portions en question. La portion qui reste sert alors de frontière aux deux zones concernées. La procédure s'effectue en plusieurs étapes, grâce à `SupprimerLesArcsEnDouble()` de `CSaveditDocument` (A.4.2) :

- pour chaque arc, chercher les autres arcs qui peuvent avoir une intersection,

- si un arc est candidat, rechercher à partir de l'un des nœuds un point commun,

- si l'on a trouvé un point commun, on recherche si les points suivants sont également communs, dans chacun des arcs. Pour le second arc, on doit tester dans le sens de l'arc, puis dans le sens inverse, car les deux arcs n'ont pas forcément été saisis dans le même sens.

- si l'on a trouvé une séquence commune, on découpe le premier arc en trois arcs (à moins que le premier point de la séquence commune soit un bout de l'arc), le second de même, et l'on élimine l'un des deux arcs correspondant à la partie commune.

7.5.2.7. Union d'arcs

L'union d'arcs permet de regrouper plusieurs arcs pour en faire un seul. Elle permet d'alléger le document en diminuant le nombre des arcs et donc des nœuds. Elle intervient en général après une jointure automatique des arcs sur les nœuds. Pour unir plusieurs arcs, il faut chaîner les arcs qui peuvent l'être en testant l'ensemble des nœuds du document, et en changeant l'ordre des points d'un arc si nécessaire. Pour un document de type zone, le chaînage s'arrête dès qu'un nœud apparaît dans plus de deux arcs du document.

7.5.3. Les contraintes sur les objets géographiques

7.5.3.1. Le nettoyage des zones

Comme pour les arcs, quelques procédures permettent d'assurer une bonne cohérence de l'ensemble du document, comme par exemple la suppression des zones sans arc, la suppression des arcs sans zones (arcs libres), ou la détection des arcs pendants. Ces procédures interviennent en général lors de la révision finale du document.

7.5.3.2. La fermeture de zones

On teste la fermeture des zones en vérifiant le nombre d'occurrences d'apparition d'un bout pour tous les arcs constituant la zone. Ce nombre doit toujours être un multiple de 2 (méthode `IsZoneFermee()` de `CSaveditDocument`, A.4.2).

fig. 7.6 : test de fermeture d'une zone

La fermeture des zones est l'un des tests les plus importants, car la cohérence des zones intervient dans de nombreux problèmes graphiques. De plus, les erreurs de fermeture sont très pénalisantes pour la saisie, si elles ne sont pas corrigées au fur et à mesure. Il est en effet difficile de corriger la cohérence d'un document comportant de nombreuses erreurs de fermeture de zone. Dans SAVEDIT, une option permet de visualiser automatiquement toute erreur de fermeture de zone : les nœuds ouverts sont indiqués par un cercle rouge (fig. 7.6). Avec cette option, le test de fermeture est effectué pour l'ensemble des zones à chaque fois que l'on dessine les zones sur l'écran.

On peut également tester la fermeture des zones grâce à l'option de remplissage des zones fermées par une couleur (fig. 7.7). Cette option permet de visualiser le document en cours de saisie en utilisant une image *raster* de l'ensemble des zones fermées. L'image *raster* est créée par rasterisation en utilisant les arcs des zones, après test de fermeture. Elle est actualisée de façon dynamique à chaque modification de zone. On utilise pour cela l'algorithme de rasterisation présenté au chapitre 5, avec une définition de rasterisation égale à la résolution de l'écran.

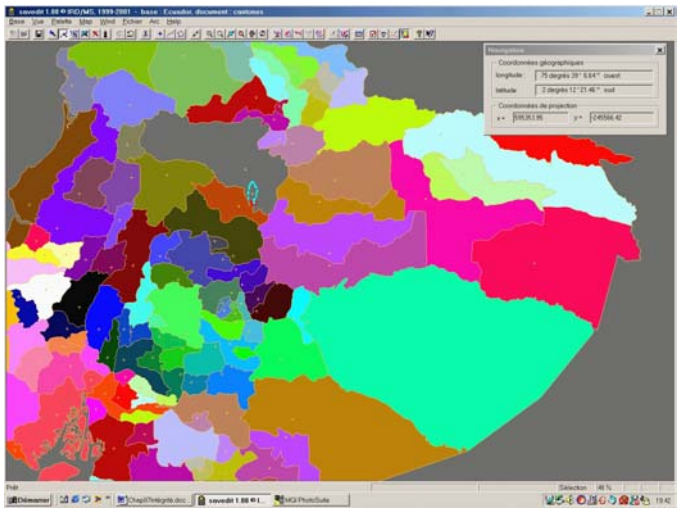


fig. 7.7 : visualisation de la fermeture par remplissage en couleur

7.5.3.3. Connexité et fermeture des arcs pour le type ligne

Pour les objets de type ligne, la connexité ou la fermeture des arcs n'est pas une contrainte géométrique, mais elle s'exprime néanmoins sur les arcs saisis. Il est donc utile d'avoir des outils permettant de joindre deux arcs sur leurs extrémités ou de fermer un arc sur lui-même. L'utilisateur doit effectuer ces opérations manuellement, en sélectionnant l'arc (pour le fermer sur lui-même) ou les deux arcs (pour les joindre), lorsque cela est nécessaire à la cohérence de l'ensemble.

7.5.3.4. Unicité de clé (zones sécantes, zones incluses)

La contrainte d'unicité de clé est fondamentale pour assurer la cohérence du modèle relationnel. Il faut donc éviter que deux zones d'un même document ne se recouvrent.

Le cas de zones sécantes est le plus simple : il peut être contrôlé grâce aux procédures de contrôle sur les arcs (extra-simplicité). Par contre, il faut faire particulièrement attention au problème des zones incluses, car une zone incluse peut facilement être saisie sans que la topologie soit prise en compte, si l'on oublie d'inclure l'arc dans la zone qui la contient. Il faut donc tester l'intersection des zones lorsqu'il y a possibilité d'inclusion.

On teste pour cela les possibilités d'inclusion pour l'ensemble des couples de zone d'un document. Ce test utilise le rectangle minimum des zones. Lorsque deux zones sont candidates, on rasterise ces deux zones dans un espace défini à partir de

la plus petite. Les deux images ainsi obtenues doivent être disjointes. La rasterisation utilise l'algorithme présenté au chapitre 5 pour la rasterisation d'un masque.

7.5.3.5. Centroïde de zones

Comme un centroïde peut être déplacé par l'opérateur de saisie, il est important de tester l'appartenance des centroïdes à leur zone pour assurer la cohérence globale. Rappelons que le centroïde est utilisé lors de l'exploitation des données pour des opérations d'agrégation ou d'appartenance, ainsi que pour placer des éléments cartographiques (symboles, labels, etc.).

On teste l'appartenance d'un centroïde à une zone fermée en utilisant un algorithme YX : une ligne traversant la zone doit couper son contour avec un nombre impair d'intersection avant un point intérieur (théorème de Jordan). Dans la pratique, on calcule le nombre d'intersection d'une ligne horizontale passant par le centroïde avec l'ensemble des arcs formant le contour de la zone, en ne retenant que les points d'intersection qui se trouve avant le centroïde (de la gauche vers la droite). Si la ligne passe par un nœud (extrémité commune à plusieurs arcs) cette intersection ne doit être comptée qu'une seule fois.

7.5.3.6. Valeurs adjacentes (courbes de niveaux)

Cette contrainte descriptive de jointure est très utile pour trouver les erreurs lors de la saisie de courbes de niveaux. Elle permet de trouver les courbes adjacentes dont la différence des valeurs ne se situe pas dans un intervalle donné. Elle est complémentaire d'une recherche d'extra-simplicité des arcs, qui doit normalement être effectuée auparavant. Nous considérerons que deux lignes sont voisines s'il existe un segment de droite reliant un point dans chaque ligne et ne coupant aucune autre ligne.

Pour vérifier cette contrainte, l'algorithme développé dans SAVEDIT discrétise l'espace de manière à trouver rapidement les conditions d'adjacence. Le principe est le suivant :

- on calcule la fenêtre du document ;

- on fait passer n lignes horizontales (y=constante, dans le plan de projection) à travers ce document, en calculant pour chaque ligne horizontale les points d'intersection de cette ligne avec les objets du document. Les intersections sont conservées avec le numéro de l'objet auquel elles appartiennent ;

- pour chaque ligne horizontale, les points sont triés en x. Ensuite, on teste la différence des valeurs des objets de deux points contigus, par rapport à l'intervalle des valeurs admissibles, données par l'utilisateur.

On peut répéter l'opération en inversant *x* et *y*. Bien sûr, la précision du test dépend du pas de calcul. Plus *n* est grand, plus on se rapproche de la définition exacte de la relation d'adjacence, et moins on aura de possibilité de voir une courbe échapper au contrôle. Le temps d'exécution est proportionnel à *n*. Outre les erreurs sur les valeurs, ce test permet de mettre en évidence des problèmes de connexité entre deux lignes : un espace entre deux lignes de même valeur permet à une ligne horizontale ou verticale de traverser cet espace, et donc de provoquer une erreur sur les valeurs.

Voici la procédure `TestValeur()` de la classe `CSaveditDocument` correspondant à ce test (A.4.2). Les points d'intersection sont stockés dans des cellules et triés sur *x* par insertion. La structure est identique à celle utilisée pour la rasterisation.

7.5.3.7. Valeurs adjacentes (points cotés)

Cette contrainte descriptive permet de trouver les erreurs lors de la saisie de points cotés . Elle permet de trouver les points voisins dont la différence des valeurs ne se situe pas dans un intervalle donné. Ici, deux points sont voisins s'ils sont contigus dans la triangulation de Delaunay. La vérification calcule donc la triangulation de Delaunay pour l'ensemble des points saisis (avec l'algorithme présenté au chapitre 6), puis calcule la valeur de la différence de deux points contigus dans la graphe correspondant à la triangulation.

7.5.4. Quelques procédures automatiques : algorithmes

Nous allons présenter ici rapidement quelques procédures et algorithmes un peu plus complexes qui permettent d'améliorer considérablement les performances du logiciel de saisie graphique. Ces procédures sont en cours de développement dans le module SAVEDIT.

7.5.4.1. La détection automatique des contours de zone

La constitution automatique de zones à partir des arcs saisis peut améliorer considérablement les temps de saisie, car l'opérateur n'a plus à indiquer « à la main » les arcs constituant le contour de chaque zone. Mais pour que cette procédure puisse fonctionner, il faut que le document soit déjà en bon état (pas d'arcs libres, pas d'arcs pendants). Cette opération correspond à la création automatique de la topologie pour les zones.

Deux méthodes peuvent être utilisées :

- une méthode semi-automatique, avec l'indication pour chaque zone d'un point intérieur. L'algorithme recherche alors l'ensemble des arcs nécessaires pour fermer le polygone contenant le point.

- une méthode automatique : le programme crée des polygones à partir de l'ensemble des arcs et calcule un centroïde pour chaque polygone. Les objets zone correspondent aux polygones. Cette méthode ne peut résoudre le problème des zones incluses : elles seront toujours considérées comme des objets, et le document créé ne comportera donc aucun « trou ».

7.5.4.2. La saisie semi-automatique d'arc par suivi de contour

La saisie automatique permet de s'affranchir de la saisie manuelle des arcs : elle correspond à une opération de vectorisation à partir d'une image du document scanné. Mais la vectorisation brute est peu efficace, car elle peut créer de nombreux arcs parasites, et ne crée pas les objets ou la topologie de l'ensemble. Nous développons donc une méthode semi-automatique qui permet à l'opérateur de créer des arcs en indiquant la ligne à suivre et en donnant la direction du suivi de contour lorsque l'algorithme rencontre un ambiguïté.

Pour créer les arcs à partir d'un document scanné, nous utilisons une méthode par suivi de contour à partir du document transformé en une image noir et blanc [BAR 88]. Les contours correspondent au squelette obtenu à partir des zones noires. L'opérateur indique le début de l'arc à saisir, puis la direction du suivi en pointant sur un autre point de l'arc. L'algorithme suit la ligne et s'arrête dès qu'il rencontre une branche dans la ligne. L'opérateur indique alors si l'arc est terminé ou la direction dans laquelle le suivi doit continuer.

7.5.5. Le format des documents SAVEDIT

Quelque soit le type du document, SAVEDIT crée un fichier d'extension .car indiquant les caractéristiques du document. Ce fichier texte a l'aspect suivant :

```
.Version 7
.Nom seismes
.Date Wed Feb 23 18:14:59 2000
.Operateur Importation de points
.Datum WGS84
.Echelle 0
.Type 1
.Attribut 2
.TailleCle 40
.Datum non précisé
.Projection
   2.0000000000e+000
   6.3781370000e+006
   6.6943799901e-003
   9.9960000000e-001
   1.6740000000e+004
   0.0000000000e+000
   0.0000000000e+000
   0.0000000000e+000
   0.0000000000e+000
```

```
0.0000000000e+000
0.0000000000e+000
0.0000000000e+000
0.0000000000e+000
0.0000000000e+000
0.0000000000e+000
0.0000000000e+000
0.0000000000e+000
0.0000000000e+000
0.0000000000e+000
0.0000000000e+000
```

Chaque ligne qui commence par un . correspond à un paramètre du document. Par exemple, *.Type* indique le type du document (1 : point, 2 : ligne, 3 : zone), *.Attribut* indique le type de l'identifiant (1 : numérique, 2 : nominal, 3 : les deux). *.Projection* indique la projection géographique privilégiée pour l'affichage du document lors de la saisie (20 paramètres), mais les coordonnées sont conservées en longitude-latitude dans un datum, qui est normalement précisé par une ligne *.Datum*.

Un document de type point comporte un seul fichier, d'extension .pt. Tous les objets sont écrits les uns à la suite des autres (A.4.2).

Un document de type ligne comporte deux fichiers, l'un d'extension .lg, l'autre d'extension .arc. Le fichier d'extension .lg comporte la description des objets, et le fichier d'extension .arc comporte la description des arcs constituant les objets lignes.

Un document de type zone comporte également deux fichiers, d'extension .zon et .arc. Le fichier d'extension .zon comporte la description des objets, et celui d'extension .arc la description des arcs constituant les frontières entre zones. Les fichiers d'extension .arc ont la même structure, quelque soit le type des objets auxquels se réfèrent les arcs (A.4.2).

7.6. Conclusion

La mise en œuvre de contraintes d'intégrité dans les bases de données géographiques est essentielle, car plus que d'autres ces bases sont difficiles à maintenir et mettre à jour, et de nombreux traitements géographiques ne sont valables qu'à la stricte condition d'une bonne cohérence des données. Ce n'est qu'au prix d'un bon respect des contraintes d'intégrité géographique que l'on pourra assurer une bonne utilisation de la base de données.

Les contraintes d'intégrité sont déterminées par la modélisation de la réalité en collections d'objets du même type (qui correspond à la définition de la structure de la base de données). Les contraintes géométriques et topologiques de type peuvent intervenir très tôt dans le processus de constitution de la base. On a même tout intérêt à les traiter de façon interactive lors de la saisie des données elles-mêmes, car

toute modification graphique dans un grand ensemble d'objets géométriques est par la suite délicate, si l'on veut maintenir la cohérence de l'ensemble, du fait de la propagation possible d'erreurs.

Une base de données géographique doit finalement être considérée comme un ensemble complexe, comprenant la définition d'un ensemble de collections, la définition de contraintes d'intégrité pour chaque collection, de contraintes d'intégrité entre les collections, et des méthodes d'utilisation pour les collections d'objets. Un SIG devrait être capable de proposer et de gérer cette structure complète, avec un langage de description et de manipulation des collections et des relations entre collections, que ce soit pour les contraintes d'intégrité comme pour les méthodes d'exploitation des données.

Bibliographie

[BAR 88] BARUCH O., *Line Thinning by Line Following*, Pattern Recognition Letters, 1988, Vol. 8, p. 271-276

[BEN 93] BENKAZEN V., DOUCET A., *Bases de données orientées objet*, 1993, Armand Colin

[BOU 94] BOURSIER P., TAO-YUAN J., *A model for Handling topological relationships in a 2D environment*, Proceedings of the 6th International Symposium on Spatial Data Handling, 1994, Vol 1, pp 73-88, Endinburgh

[CLE 93] CLEMENTINI E., DI FELICE P., VAN OOSTREROM P., *A Small Set of Formal Topological Relationships Suitable for End-User Interaction*, Proceedings of the 3th Symposium on Large Spatial Databases, 1993, Lecture Notes in Computer Sciences 692, pp 277-295, Singapore

[CLE 95] CLEMENTINI E. DI FELICE, CALIFANO G., *Composite Regions iin Topological Queries*, Information Systems, 1995, Vol. 20, n°7, pp 579-594

[COH 93] COHN A.G., RANDELL D.A., CUI Z., BENNETT B., *Qualitaiv Spatial Reasoning and Representation*, Preceedings of QUARDET 93, PP 513-522, Barcelona, Spain, 1993

[CUI 93] CUI Z., COHN A.G., RANDELL D.A. , *Qualitativ and Topological Relationships in Spatial Databases*, Proceedings of the 3th Symposium on Large Spatial Databases, 1993, Lecture Notes in Computer Science 692, pp 296-315, Singapore

[DAT 90] DATE C., *A Contribution to the study of database integrity*, in Relational Database Writing, 1990, Addison-Wesley

[DAV 94] DAVID B., FASQUEL P., *Qualité d'une base de données géographique : concepts et terminologie,* Bulletin d'information de l'IGN, 1994, n° 67, Paris

[EGE 89] EGENHOFER M.J., *A Formal Definition of Binary Topological Relationships*, Proceedings of the 3th International Conference on Foundations of data Organization and Algorithms, 1989, Lecture Notes in Computer Science 367, Paris, France, pp 457-472

[EGE 91] EGENHOFER M.J., FRANZOZA R.D., *Point-Set Topological Relations*, International Journal of Geographic Information Systems, 1991, 5-2, pp 161-174

[EGE 94] EGENHOFER M.J., CLEMENTINI E., SHARMA J., *Modelling topological spatial relations : strategies for query processing*, Computer and graphics, 1994, vol. 18, n°6, pp 515-522

[FAI 96] FAIZ S.O., *Modélisation, exploitation et visualisation de l'information qualité dans les bases de données géographiques*, Thèse de Doctorat, 1996, Univ. Paris XI Orsay

[FRA 92] FRANK A.U., *Qualitative Spatial Reasoning about Distances and Directions in space*, Journal of Visual Langages and Computing, 1992, vol. 3, n° 4, pp 343-371

[FRA 96] FRANK A.U, *Qualitative spatial reasoning : cardinal directions as an example.* International Journal of Geographical Information Systems, 1996, Vol. 10, n°3, pp 269-290

[GOO 89] GOODCHILD M., GOPAL S., *Accurancy of Spatial Databases*, 1989, Taylor & Francis

[GUP 95] GUPTILL S.C., MORRISON J.L., *Elements of Spatial Data Quality*, Pergamon/Elsevier Science, 1995

[HAA 96] HAADZILACOS T., TRYFONA N., *A model for expressing topological integrity constraints in geographical databases*, in Proceedings : Theories and Models of Spacio-Temporal Reasoning in Geographic Space, 1996, International Conference GIS, Pisa, Italy. A. Frank, I. Campari, U. Fomentini (Ed.), pp 252-268

[LAU 94] LAURINI R., *Détection et corrections d'erreurs topologiques dans les bases de données géographiques*, Actes des premières journées de la recherche CASSINI, 1994, Lyon, France, pp 105-114

[LAU 93], LAURINI R., MILLERET-RAFFORT F., *Les bases de données en géomatique,* 1993, *Hermès*, Paris

[LAU 91] LAURINI R., MILLERET-RAFFORT F., *Cohérence dans les bases de données spatiales*, VIIe journées Bases de données Avancées, INRIA, 1991, pp 471-488, Lyon

[MOL 94] MOLENAAR O, KUFONIYI O., BOULOCOS T., *Modelling topological relationships in vector maps*, Proceedings of the 6th International Symposium on Spatial Data Handing, 1994, pp 112-126, Endinburgh

[MÜL 95] MÜLLER J.C., LAGRANGE J.P., WEIBEL R., (ed.), *GIS and Generalization*, GIS DATA I, 1995, Taylor & Francis

[O'RO 93] O'ROURKE J., *Computational Geometry in C*, 1993, Cambridge University Press

[PUL 88] PULLAR D.V., EGENHOFFER M.J., *Toward formal definitions of topological relations among spatial objects*, Procedings of the third International Symposium on Spatial Data Handling, 1988, Sydney, Australia, pp 225-241

[PLÜ 97] PLÜMER, GRÖGER, *Archiving Integrity in Geographic Information Systems – Maps and Nested Maps*, Geoinformatica, 1997, vol.1, n° 4, pp 345-367, Kluwer Academic, Boston

[SOU 86] SOURIS M., *Systèmes d'information géographique et bases de données*, Colloques et Séminaires sur le Traitement des données localisées, 1986, pp 29-87, Editions de l'Orstom, Paris

[TAN 95] TANZI T., UBEDA T., *Contrôle topologique de la cohérence dans les bases de données géographiques : application aux réseaux* , Revue Internationale de Géomatique, 1995, Vol. 5, n°2, pp 133-155

[UBE 96] UBEDA T., SERVIGNE S., *Geometric and Topological Consistency of Spatial Data*, 1996, in Proceedings: First International Conference on Geocomputation, Leeds, UK

[UBE 97] UBEDA T., *Contrôle de la qualité spatiale dans les bases de données géographiques : Cohérence topologique et corrections d'erreurs*, Thèse soutenue à l'INSA de Lyon, 1997

Troisième partie : Analyse spatiale et cartographie

Chapitre 8

Analyser : calculs et analyse spatiale

8.1. Introduction

L'objectif des SIG n'est pas seulement de représenter des objets élémentaires pour les gérer et les restituer graphiquement avec des cartes. A partir d'une modélisation de la réalité qui se veut la plus complète mais aussi la plus simple à gérer, le système doit nous permettre d'analyser les objets ainsi modélisés pour en construire d'autres, plus complexes, plus synthétiques, soit pour construire une modélisation d'un problème spécifique, soit pour découvrir par une démarche empirique des situations nouvelles. Cette démarche n'est pas spécifique aux données localisées, mais la localisation, qui était souvent vue comme un fait et non une causalité, doit également intervenir dans ce processus d'analyse.

L'analyse spatiale recouvre un ensemble de méthodes et d'outils qui permettent d'interpréter et de rendre compte des relations qui existent entre des objets localisés, afin de découvrir ou de mettre en évidence des règles générales d'organisation de l'espace [PUM 97]. Son objectif est essentiellement de décrire une disposition particulière de certains objets, de leur organisation spatiale, de repérer des structures, d'expliquer une localisation par d'autres. Pour cela, il faut déceler en quoi la localisation apporte un élément utile à la connaissance des objets étudiés et peut en expliquer les caractéristiques, en partie ou en totalité. L'analyse spatiale permet d'introduire la localisation comme une cause explicative d'un phénomène, ou l'introduire dans une analyse statistique qui habituellement l'ignore.

Les démarches de l'analyse spatiale sont multiples. Elle peut tendre à un simple résumé de l'information contenue sur une carte thématique, par exemple en localisant le centre de gravité d'un semis de points ou en dessinant des zones homogènes. Elle peut chercher à faire apparaître des structures spatiales connues, comme certains types de réseaux ou une organisation de type centre-périphérie. Elle peut aussi viser à tester la pertinence d'un modèle spatial, ou aider à simuler un processus spatial. Quelle que soit la méthode employée, l'analyse portera toujours sur un ensemble d'objets, et non sur un individu isolé.

L'analyse spatiale a besoin pour cela de beaucoup de données localisées. Et c'est là qu'interviennent les SIG, pour aller au-delà de leurs simples possibilités de gestion de données localisées – bien efficaces en tant que telles mais insuffisantes pour étudier un phénomène et répondre à des problèmes d'organisation du territoire étudié. Au delà de cette fonction de gestion de données, les SIG doivent contenir les outils et instruments permettant de remplir tout ou partie des fonctions de l'analyse spatiale. Et c'est bien ce qui fait alors la puissance de ces systèmes, car il sont les seuls à permettre à la fois la gestion de la localisation et son introduction dans un processus d'analyse. Mais ces outils s'ajoutent aux outils statistiques ou mathématiques existants – qu'ils utilisent d'ailleurs largement – et ne les remplacent pas. Un SIG se doit donc également d'incorporer ou de donner accès aux méthodes classiques de l'analyse de données. L'un des intérêts majeurs de l'utilisation de la statistique dans un SIG réside dans la possibilité d'étudier facilement la localisation des variations d'un attribut descriptif, et de rechercher les relations entre variation d'un attribut et localisation des objets [SAN 89].

Après un bref rappel des principaux concepts de l'analyse spatiale, nous présentons dans ce chapitre les opérations qui permettent de mettre en œuvre des procédures d'analyse spatiale dans le module d'exploitation des données du système SAVANE. Ces opérations ne sont plus des opérations de l'algèbre relationnelle étendue, et s'apparentent plus à des programmes d'application ou des méthodes. Le SIG, initialement système de gestion de type relationnel, s'oriente maintenant vers une approche objet, avec l'introduction de méthodes sur les collections d'objets. Les méthodes liées à la localisation dépendent pour la plupart du type des objets.

8.2. Rappels sur l'analyse spatiale

8.2.1. Structures spatiales et objets géographiques : généralisation et agrégation

Comme nous l'avons vu à l'occasion des exemples donnés au chapitre 3, la description d'un phénomène géographique complexe, statique ou dynamique, fait appel à de nombreuses échelles de description et à la définition de nombreux objets suivant une modélisation qui n'est pas toujours évidente à construire. C'est à partir

de cette schématisation que l'utilisateur géographe va pouvoir construire des objets plus complexes, extraire des modèles, des organisations, des structures, pour aboutir à une vision plus synthétique [HAG 73]. En fait, c'est bien souvent le travail d'analyse spatiale qui va permettre de définir ces échelles de description et les attributs qui lui correspondent, à partir d'une information de base donnée par rapport à d'autres objets géographiques. C'est ce qui fait l'ambiguïté et la difficulté de cette démarche souvent récursive, et qui amène souvent un SIG à contenir des collections d'objets correspondant à différents niveaux d'analyse. C'est à la fin d'une analyse géographique qu'apparaît l'identification de nouveaux objets spatiaux : on utilise l'information connue pour des objets géographiques pour en définir d'autres, plus cohérents par rapport à un problème donné, ou relevant d'une autre échelle d'observation, plus synthétique. Les géographes font référence à ces différents niveaux d'observation en parlant de changement d'échelle géographique de l'analyse. L'analyse spatiale permet donc de passer d'un modèle de la réalité à un autre.

Différents modèles concernent bien souvent différentes échelles de description. L'analyse spatiale consiste alors à utiliser des opérations permettant de passer d'une échelle à une autre sans provoquer d'erreur dans la conception de ces nouveaux objets. Par exemple, le passage d'une échelle à une autre peut s'effectuer en réunissant des unités spatiales de niveau inférieur. On parlera alors d'un processus d'agrégation. Mais les agrégations possibles sont très nombreuses. Une grande partie du travail d'analyse spatiale consiste à explorer quels sont les regroupements d'unités de base qui sont les plus pertinents, selon différents types de critères, pour produire des découpages intéressants à une autre échelle, généralement pour mettre en évidence des structures spatiales impossible à voir à partir des données non agrégées. Cette étape ne modifie pas seulement la nature spatiale des objets géographiques, elle en modifie également la description et les attributs. L'analyse spatiale va donc bien plus loin qu'une simple mise en relation d'objets sur un critère de localisation. Elle permet de définir des opérations permettant d'étendre la jointure spatiale en fonction de la validité sémantique des objets.

Dans un processus de changement d'échelle, la définition de nouveaux objets est le résultat d'une abstraction qui repose sur un processus d'élimination de « détails », appelé généralisation [RIM 90] [PUM 97]. Ainsi, un ensemble d'objets hétérogènes à un niveau d'agrégation géographique peut apparaître homogène à un niveau supérieur, parce que l'on oublie volontairement l'hétérogénéité interne pour ne retenir que les traits qui permettent la comparaison avec d'autres unités spatiales situées au même niveau d'observation. La même remarque s'applique lorsqu'on agrège des unités spatiales de base en fonction non plus de leur ressemblance pour un attribut ou un ensemble d'attributs, mais d'après les relations qu'elles entretiennent les unes avec les autres.

L'opération d'agrégation est toujours délicate et il est nécessaire d'observer des règles précises pour contrôler la perte d'information lors de la généralisation, surtout lorsqu'il n'y a pas hiérarchie spatiale des découpages. La distribution spatiale d'un phénomène est un critère important dans de nombreux cas [PUM 97]. Ces règles nous ont amené à développer plusieurs méthodes d'agrégation spatiales dans le système SAVANE.

La désagrégation – le passage à un découpage plus fin - pose encore plus de problèmes de validité, et s'apparente dans de nombreux cas à un problème d'interpolation. En effet, elle est souvent employée pour remédier à une carence d'information : on va alors chercher de l'information à un autre niveau de validité (plus faible), en faisant implicitement une hypothèse d'uniformité de la répartition spatiale de cette information. Cette hypothèse très forte ne doit pas être oubliée lors de l'utilisation du résultat obtenu.

La plupart des procédures d'agrégation ou de désagrégation sont dangereuses lorsqu'elles s'apparentent à des interpolations : pour un point de l'espace, la donnée n'est plus qu'approchée et sa validité dépend de l'opération qui a été effectuée et des hypothèses sous-jacentes. Nous sommes bien face à la définition de nouveaux objets, avec leur validité spatiale propre, et le résultat d'une opération de ce type doit bien être considéré comme une nouvelle modélisation de la réalité : l'opération employée fait partie du processus de définition du modèle. L'oubli de cette règle peut amener de graves erreurs dans l'utilisation ultérieure des objets ainsi construits.

8.2.2. Distribution spatiale d'un phénomène

Où ? Comment ? Pourquoi là et pas ailleurs ? Ces questions, comme nous l'avons déjà souligné au chapitre 3, sont au cœur de nombreuses questions géographiques [LAU 93]. La répartition d'un phénomène est une question essentielle dans l'analyse dès que l'on décide de prendre en compte la localisation de ce phénomène. On s'intéresse à la répartition de l'ensemble formé par tous les objets : la structure spatiale propre de chaque objet n'importe plus et peut être ramenée à un point, car on cherche à étudier l'agencement des objets les uns par rapport aux autres. On cherche à caractériser l'arrangement géographique des lieux en explicitant les principes de leur distribution, indépendamment d'un attribut descriptif particulier : les méthodes présentées ici ne prennent en compte que la localisation des objets étudiés (qui peuvent bien sûr avoir été sélectionnés auparavant sur un caractère descriptif).

La simple cartographie de la distribution d'objets par leur centroïde donne déjà une vision synthétique d'un phénomène. Pour aller plus loin, on peut définir le centre d'un ensemble de point, point remarquable le plus représentatif possible de

l'ensemble (point médian, centre de gravité). Bien sûr, cette représentation, image de la médiane ou de la moyenne en deux dimensions, ne rend pas compte de l'hétérogénéité de la situation. On peut utiliser pour cela une mesure de la dispersion. Cette dispersion peut être évaluée par rapport à un point remarquable ou calculée comme une mesure généralisée des écarts entre l'ensemble des points. On peut également « classer » le territoire, et compter le nombre d'objets par classe, par analogie à un histogramme sur une variable de dimension 1. Les classes seront habituellement des objets de surface égale et de géométrie simple, telle des mailles carrées. Et l'on calculera plutôt des densités que des effectifs, pour avoir des valeurs qui ne dépendent pas directement de la taille des mailles. Bien sûr, on peut également utiliser une classification de l'espace en utilisant un découpage déjà existant, mais ce découpage peut introduire un biais dans l'étude de la dispersion, car il ne faudrait pas que le découpage utilisé soit corrélé à la distribution que l'on cherche à étudier.

Enfin, la distribution spatiale peut être également caractérisée par la forme du semis de points, correspondant à l'arrangement et l'espacement entre les points. Cette distribution peut être comparée à quelques forme de références : distribution spatiale régulière, concentrée, aléatoire. On peut pour cela utiliser de nouveau des mailles et des calculs de dispersion comme les courbes de concentration ou l'indice de Theil, ou encore comparer la distribution avec une distribution spatiale de référence dont la loi de formation est connue.

8.2.3. Homogénéité spatiale, relations de voisinage, regroupement

Dans le paragraphe précédent, nous avons vu quelques méthodes d'analyse concernant uniquement la position des objets, indépendamment de la valeur d'un attribut de ces objets. Mais bien souvent la valeur d'un attribut est corrélée à la position relative des objets entre eux ou à la forme du semis de point. Il est donc naturel de chercher à définir des indices de ressemblance provenant à la fois de la position relative des objets dans l'espace et de la valeur d'un caractère de ces objets. L'idée de voisinage, de continuité, de connexité est sous-jacente à ces indices reflétant l'homogénéité spatiale d'un ensemble d'objets.

L'analyse spatiale s'intéresse donc directement à la définition ou la détermination d'ensembles homogènes à partir d'une collection d'objets élémentaires. La notion d'homogénéité est toujours relative à une série d'attributs et à une certaine échelle d'observation. Les mesures d'homogénéité, le plus souvent exprimées sous forme d'indice, sont nombreuses. La statistique classique fournit déjà un certain nombre d'outils permettant de mesurer le degré d'homogénéité d'une collection d'objets, mais sans toutefois prendre en compte les relations de voisinage entre les objets.

Mais on sait que bien souvent il y a interaction spatiale entre les objets d'une collection : c'est le cas lorsque la répartition spatiale des objets par rapport à un attribut descriptif n'est pas aléatoire. On se doit alors d'intégrer la localisation dans l'étude statistique du phénomène, sous peine, par exemple, de biaiser la définition d'un échantillon dans lequel on utiliserait ultérieurement la localisation dans le processus d'analyse. La géostatistique fournit des outils permettant de vérifier l'existence d'une corrélation entre voisinage et variation d'un attribut descriptif. Si l'on veut introduire le voisinage dans le calcul de la ressemblance, il faut utiliser ces outils, et notamment l'autocorrélation spatiale qui permet de mesurer l'intensité de la relation entre la proximité des objets et leur degré de ressemblance.

Enfin, le regroupement d'objets suivant ces critères n'obéit plus seulement à une classification descriptive, mais introduit le voisinage et la continuité pour aboutir à la définition de nouvelles partitions de l'espace minimisant certains critères. En particulier, l'analyse de la variance permet d'obtenir une agrégation d'objets géographiques qui crée le plus ou le moins de différence entre les classes ainsi définies.

8.3. SAVANE : exploration des données et statistique

L'utilisateur qui souhaite analyser un ensemble d'objets géographiques commencera sans aucun doute par représenter graphiquement l'ensemble des objets pour visualiser leur répartition spatiale, et il poursuivra son exploration par l'analyse statistique des attributs liés à ces objets, et la poursuivra en essayant de déterminer les relations entre répartition spatiale et variation descriptive.

La démarche exploratoire peut ainsi se décliner :

- analyse de la répartition spatiale d'un ensemble d'objets ;
- analyse de la répartition statistique d'un ou plusieurs attributs descriptifs d'un ensemble d'objet ;
- mise en évidence et analyse des relations entre les variations de la localisation et variations d'un ou plusieurs attributs descriptifs.

Un système d'information géographique tourné vers l'analyse devra donc comporter à la fois des fonctions de représentation graphique et de cartographie et des fonctions statistiques permettant d'explorer les attributs des objets [BEG 94]. C'est le cas du système SAVANE, et nous exposerons en détail au chapitre suivant les principes de la représentation cartographique du module d'exploitation. L'exploration graphique de la répartition des objets est très simple, car on utilise en général des paramètres cartographiques proposés par défaut par le système. L'exploration des attributs peut quant à elle porter sur des objets individuels ou sur

des collections d'objets. L'exploration individuelle utilise des procédures élémentaires d'interrogation sur une valeur d'attribut descriptif ou sur un point de l'espace. L'exploration d'une collection d'objet fait appel aux procédures statistiques classiques et à celles, moins répandues, de la géostatistique.

8.3.1. L'exploration individuelle

L'exploration individuelle consiste à rechercher les valeurs des attributs d'un objet, que l'on peut choisir graphiquement sur l'écran (fig. 8.1) ou en indiquant la valeur de l'un de ses attributs.

Dans le premier cas, un seul objet est sélectionné dans la relation. Il est retrouvé grâce à sa localisation. On utilise pour cela les procédures de distance à un point, à une ligne, ou le test d'inclusion d'un point dans une zone.

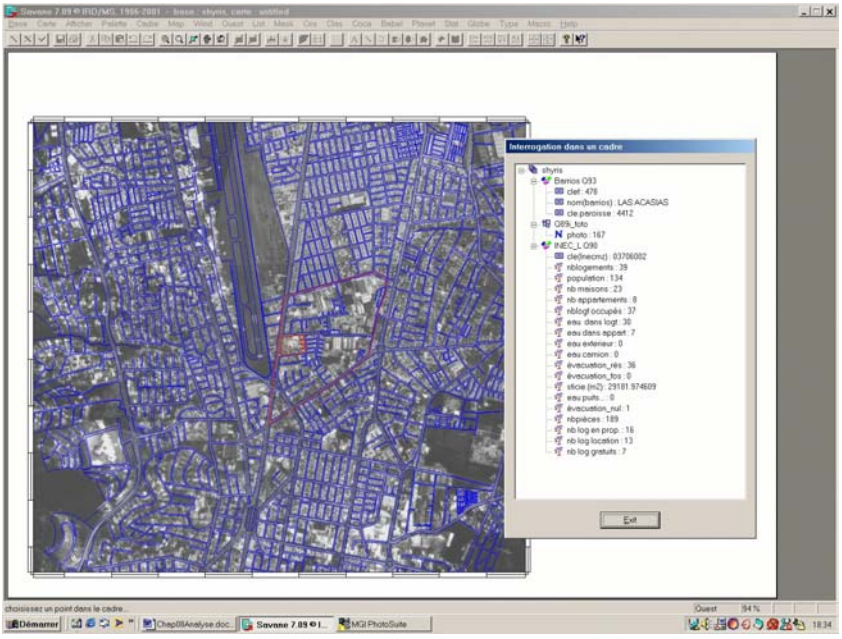


fig. 8.1 : interrogation sur écran des objets présents dans le cadre géographique

Dans le second cas, plusieurs objets peuvent correspondre à une valeur d'un attribut. Le résultat est donné dans une liste, pour les attribut descriptifs, et dessiné sur une carte, pour l'attribut graphique lorsque la relation est localisée.

8.3.2. L'exploration statistique

L'exploration statistique des attributs des objets d'une base de données est une opération fondamentale dans un processus d'analyse et d'interprétation. Cette exploration fait appel à la statistique univariée (moments d'un attribut numérique) ou multivariée (corrélations, régressions, analyse factorielle, etc.).

Si l'on peut facilement avoir accès à des logiciels statistiques extérieurs au SIG, il nous a semblé néanmoins nécessaire de développer quelques fonctions élémentaires qui permettent rapidement de visualiser la répartition statistique d'un phénomène. Nous avons donc développé un module d'exploration statistique permettant de calculer les moments statistiques d'un attribut, de calculer et visualiser un histogramme, de calculer un indice de corrélation, de visualiser une droite de régression linéaire ou un nuage de points suivant deux attributs (fig. 8.2). Toutes ces opérations sont appliquées aux objets d'une même relation.

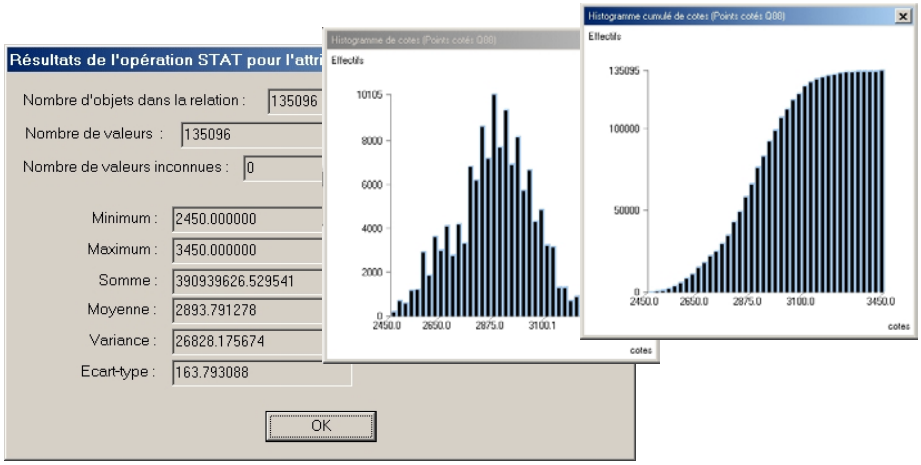


fig. 8.2 : statistiques univariées sur un attribut d'une collection d'objets

Enfin, le calcul du résidu par rapport à une distribution standard (linéaire, exponentielle, normale, de Poisson, etc.) peut être ajouté au schéma comme un nouvel attribut sur les objets, selon le principe de création de nouveaux attributs que nous allons exposer au paragraphe suivant. La cartographie du résultat permet de mettre en avant la répartition géographique des écarts à une distribution donnée (fig. 8.3).

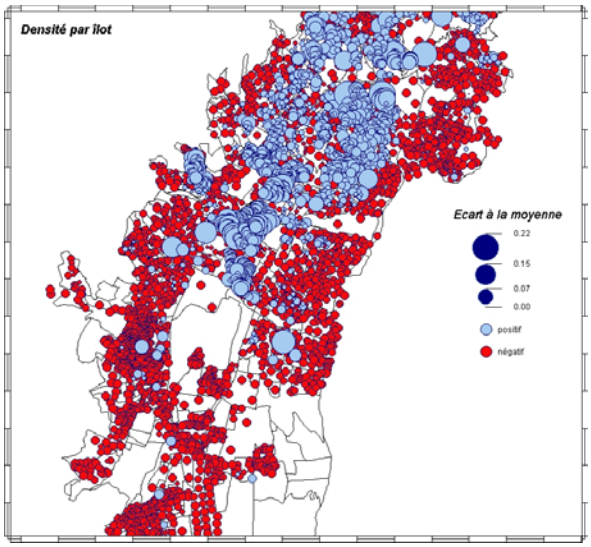


fig. 8.3 : exemple de représentation des écarts par rapport à une distribution donnée

8.4. SAVANE et méthodes sur les attributs descriptifs

Nous allons présenter dans ce paragraphe de nouvelles opérations concernant les attributs descriptifs d'une relation. Ces opérations sont à la base des opérations d'exploration et d'analyse des données. Elles permettent pour la plupart de modifier temporairement le schéma des relations en ajoutant de nouveaux attributs. Combinées aux opérations de l'algèbre relationnelle étendue, elles sont une étape importante dans la mise en œuvre de nombreuses opérations d'analyse spatiale. L'analyse spatiale ne s'effectue pas en utilisant des procédures toutes faites : l'utilisateur devra construire lui-même sa démarche en utilisant les nombreuses opérations élémentaires qui sont à sa disposition et qui s'inspirent pour la plupart des principes que nous avons rappelés lors de l'introduction : généralisation, agrégation, changement d'échelle, classification.

8.4.1. Le principe de la création de nouveaux attributs descriptifs

La création de nouveaux attributs à partir des attributs descriptifs d'une relation est une opération fondamentale dans un processus de requête exploratoire. C'est l'opération de modélisation la plus élémentaire : on peut ainsi modifier temporairement le schéma d'une relation en introduisant de nouveaux attributs calculés à partir des attributs initiaux. Le module SAVANE permet la création de nouveaux attributs par calcul numérique, par calcul logique, par classification, par

agrégation, par combinaison, par comparaison, par tri. Toutes les opérations que nous présenterons dans ce paragraphe concernent les attributs d'une seule collection d'objet : les calculs ne font jamais intervenir de mise en relation d'objets de plusieurs relations. Elles peuvent s'appliquer à tous les types de relation, puisque la localisation n'intervient pas dans ces calculs. Nous verrons dans le paragraphe suivant des opérations qui créent de nouveaux attributs en faisant intervenir la localisation des objets, et qui dépendent donc du type des relations mises en jeu. Mais le principe de la création de nouveaux attributs reste le même.

Tous les attributs créés lors d'une requête sont temporaires : la base de données n'est jamais modifiée dans SAVANE. La relation temporaire est une copie de la relation permanente et se trouve physiquement dans le répertoire du cadre correspondant à la requête, qui lui-même se trouve dans le répertoire de la carte. La gestion des états temporaires permet à plusieurs utilisateurs différents de travailler sur la même base, sans risque d'interférence, et la base garde son intégrité. Si l'on veut intégrer de façon définitive le résultat d'un calcul entre attributs, il faut exporter cet attribut pour l'intégrer avec SAVATECA dans une opération d'administration. Il est bien sûr préférable de conserver le calcul sous forme de macro-commande ou de méthode, car on peut alors être sûr que l'exécution de la macro-commande sera basée sur des valeurs actualisées des attributs.

Pour gérer la vue externe du schéma, il faut connaître la place dans le schéma interne de chaque attribut du schéma externe. Cette place est conservée dans la variable `m_iNumero` membre de la classe `CAttribut`. Lorsqu'un nouvel attribut est créé dans une relation, celle-ci devient temporaire si elle ne l'était pas déjà : la relation résultante ne contient plus que les objets de la fenêtre d'étude et, pour chaque objet, elle ne contient plus que les attributs permanents présents dans la vue externe, plus les attributs temporaires. La classe `CRelation` contient une variable permettant de connaître le nombre d'attribut de la relation dans son état initial (`m_iNb0`) et dans son état temporaire (`m_iNba`). Lors de l'écriture d'une relation temporaire, les attributs permanents sont réécrits dans l'ordre de la vue externe, les attributs temporaires à la suite. La variable `m_iNumero` devient alors toujours égale à la place de la variable dans le schéma externe.

Lors de l'accès à un attribut d'une relation non modifiée, il faut d'abord rechercher le numéro interne de l'attribut, à partir de son numéro externe `iNoatt` qui provient du dialogue avec l'utilisateur :

```
iNovar=pSchema->m_pRelations[iNoRel]->m_pAttributs[iNoAttr]->m_iNumero ;
…
val=v[iNovar] ;
```

La procédure `NouvelAttribut()` de la classe `CSchema` regroupe l'ensemble des opérations nécessaires à la création de l'attribut temporaire dans le schéma de la base de données (A.2.1.2.).

8.4.2. Calculs numériques et logiques

On peut créer un attribut par calcul numérique ou logique entre les attributs d'une même relation. Il suffit d'indiquer la formule de calcul, comme dans le cas d'une restriction par condition de sélection (fig. 8.4).

L'attribut créé contient pour chaque objet de la relation la valeur résultant du calcul. La vérification de la formule et le calcul pour chaque objet utilise la classe `CCalculateur` dont nous avons vu le schéma au chapitre 5 (A.2.1.5.). Le calcul est basé sur le parcours d'un arbre (fils gauche-frère droit) d'expressions (binaires) et de termes (unaires), représentant la formule de calcul. L'expression peut contenir des opérateurs numériques ou des opérateurs logiques.

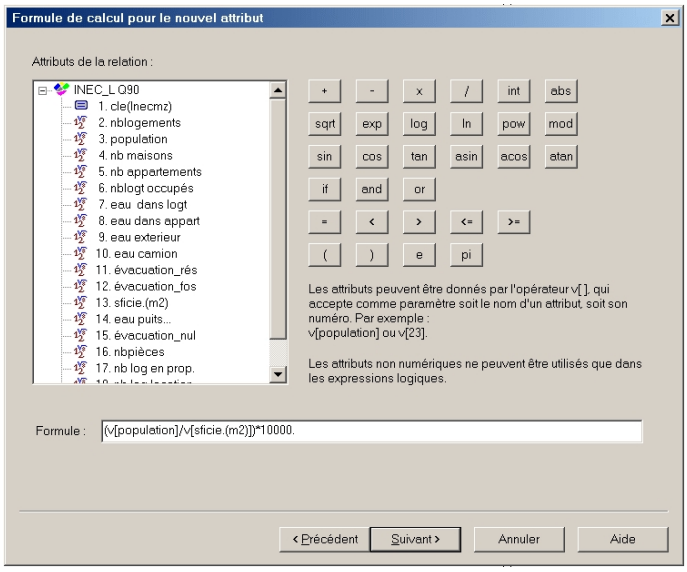


fig. 8.4 : le dialogue de création d'attribut par calcul numérique ou logique

Pour une relation non mosaïque, le code de la procédure permettant de créer un nouvel attribut par calcul est donné dans l'annexe A.2.5.3.

Pour une relation de type mosaïque, le principe est le même mais le code est plus complexe, car les valeurs d'attributs doivent être lues dans des fichiers différents, et le résultat du calcul peut ne pas correspondre au type de l'attribut à créer, qui est choisi par l'utilisateur. Par exemple, une somme de deux attributs codés sur un octet

(correspond à des valeurs entières de 0 à 255) peut dépasser 255. Si l'attribut à créer doit être codé sur 8 ou 16 bits, il faut donc également choisir une méthode de ré-étalement des valeurs (fig. 8.5 et 8.6).

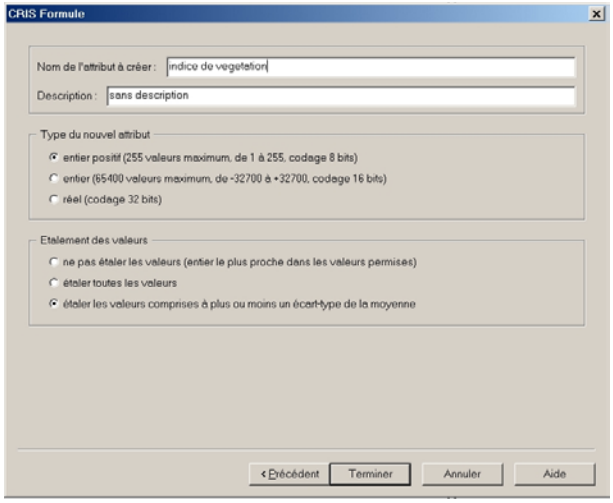


fig. 8.5 : le dialogue de création d'un attribut d'une mosaïque

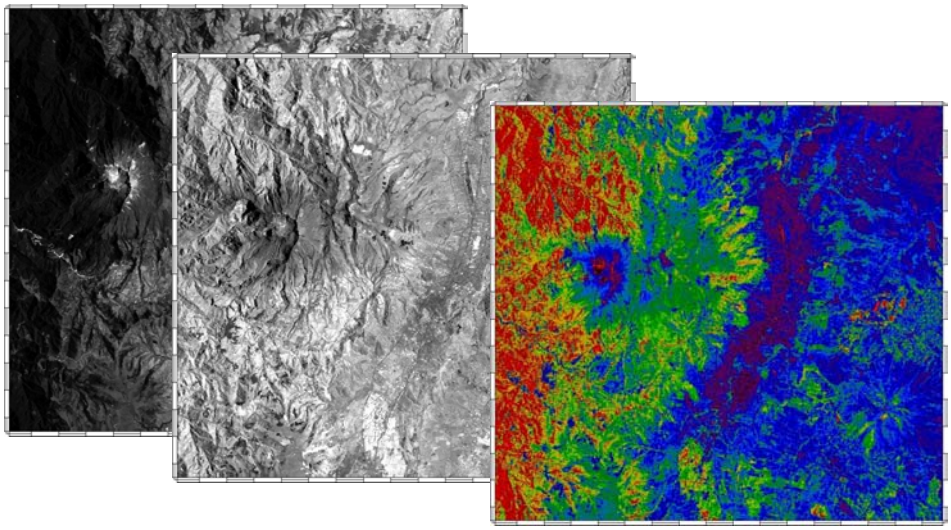


fig. 8.6 : le calcul d'un indice de végétation (NDVI) à partir des canaux 3 et 4 d'une image Thematic Mapper

8.4.3. Classifications et méthodes de discrétisation

L'opération de classification permet de transformer un attribut quantitatif en un attribut qualitatif, c'est-à-dire prenant ses valeurs dans un ensemble fini. Cette opération est très importante car la vision de la distribution de l'attribut est synthétique, et il est facile de représenter un nombre fini de valeurs sur une carte. La classification est un processus très classique en analyse de données.

La mise en œuvre de la classification d'un attribut ne pose pas de problèmes particuliers. L'attribut créé est nominal : ses modalités, qui ont été entrées par l'utilisateur du système, sont conservées dans un fichier temporaire, nommé *ftvaleurs*, qui a la même structure que le fichier *fpvaleurs*, qui lui contient l'ensemble des valeurs des attributs nominaux permanents. Le fichier *ftvaleurs* est propre à une requête, et donc à un cadre d'une carte : il se trouve physiquement dans le répertoire du cadre de la carte. Le type d'un attribut est conservé dans la variable membre `m_iType` de la classe `CAttribut`. Il est codé 1 pour les attributs nominaux permanents, 5 pour les attributs nominaux temporaires. En consultant cette variable, le système sait dans quel fichier (*fpvaleurs* ou *ftvaleurs*) il doit aller chercher les modalités de l'attribut.

Nombreuses sont les méthodes de définition des classes. La classification peut être arbitraire (des intervalles donnés, par exemple), mais bien souvent l'utilisateur choisira la méthode de définition des classes en fonction de la répartition des valeurs de l'attribut. Dans SAVANE, nous avons implémenté les procédures les plus courantes utilisées pour discrétiser une variable numérique (fig. 8.7) :

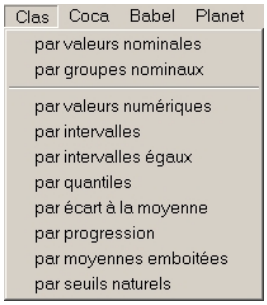


fig. 8.7 : le menu de classification de SAVANE

Par exemple, pour la classification par quantile (classification par intervalles avec des classes d'effectifs égaux), la détermination des bornes des classes impose le tri des valeurs de l'attribut. On doit donc lire la valeur de tous les objets de la relation, pour l'attribut à classer, puis trier ces valeurs pour déterminer les bornes des classes. Pour les relations de type mosaïque, le nombre d'objet peut être très

grand, et le tri serait alors trop long pour une procédure interactive. On regroupe donc les valeurs en effectuant une première discrétisation arbitraire, puis en calculant le nombre d'objets pour chaque classe de cette première discrétisation. Dans tous les cas, le calcul des classes est automatique et s'effectue lors du dialogue avec l'utilisateur du système (fig. 8.8) :

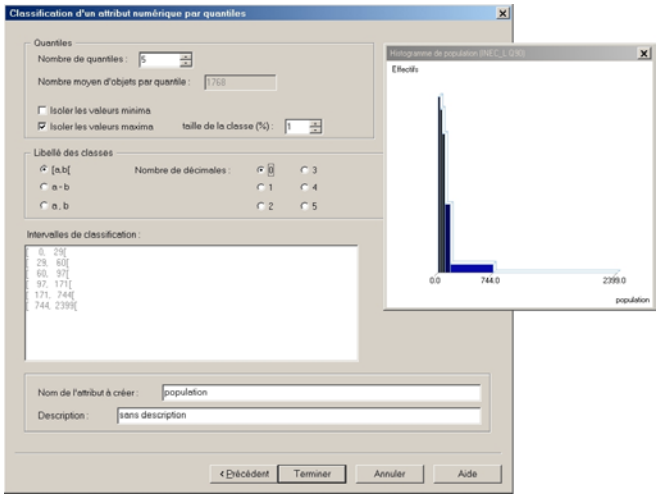


fig. 8.8 : dialogue de définition des classes par quantiles

Le code de la procédure permettant de lire et de trier les valeurs d'un attribut d'une relation non mosaïque peut être consulté dans A.2.5.4.

8.4.4. Agrégations descriptives

L'agrégation descriptive concernent le regroupement des valeurs d'un attribut par rapport à un autre attribut, toujours qualitatif nominal, et servant de clé d'agrégation. Elle permet de modifier le niveau de perception en changeant, en quelque sorte, de clé descriptive. Par exemple, les données d'un recensement, disponibles au niveau de la commune, peuvent être agrégés au niveau du département. Bien sûr, l'agrégation n'est possible que si l'on dispose d'un attribut indiquant pour chaque commune le nom du département auquel elle appartient : il s'agit d'une agrégation descriptive, où les deux attributs utilisés sont descriptifs. Cette opération ne fait pas intervenir la localisation des objets, mais seulement la valeur des attributs descriptifs choisis. Nous verrons plus loin les géo-agrégations, qui, elles, utilisent la localisation comme clé d'agrégation, et qui ne sont valables qu'entre relations localisées.

Notons qu'il y a dépendance fonctionnelle entre l'attribut choisi comme clé d'agrégation et l'attribut créé : la relation résultante n'est plus en deuxième forme normale. Si cette situation ne doit pas exister dans une base de données bien constituée, ici nous sommes dans un état temporaire au milieu d'une requête, et cette dépendance est tout à fait admise. Le nouvel attribut contenant le résultat de l'opération peut être créé dans la même relation, ou dans tout autre relation comportant un attribut nominal ayant le même ensemble de modalités que la clé d'agrégation. Dans l'exemple précédent, si l'on dispose dans la base d'une relation « département », il est possible de créer le nouvel attribut dans cette relation. Si celle-ci est localisée, il sera alors possible d'utiliser sa localisation soit pour dessiner le résultat, soit pour poursuivre l'analyse en faisant, si nécessaire, intervenir la localisation des départements.

Plusieurs opérations sont disponibles pour effectuer l'agrégation et créer le nouvel attribut (fig. 8.9). Si l'attribut à agréger est numérique, on peut effectuer une somme, une moyenne, un écart-type, une variance, un minimum, un maximum. Si l'attribut est nominal, on peut calculer la fréquence, relative ou absolue, d'une modalité particulière. Le choix de l'opération dépend bien sûr de la nature sémantique de l'attribut (par exemple, on ne peut faire de somme que s'il s'agit d'effectifs, et non de taux – mais le choix de l'opération est laissé à l'utilisateur, car il n'y a pas de renseignement sémantique de ce type dans la base de données).

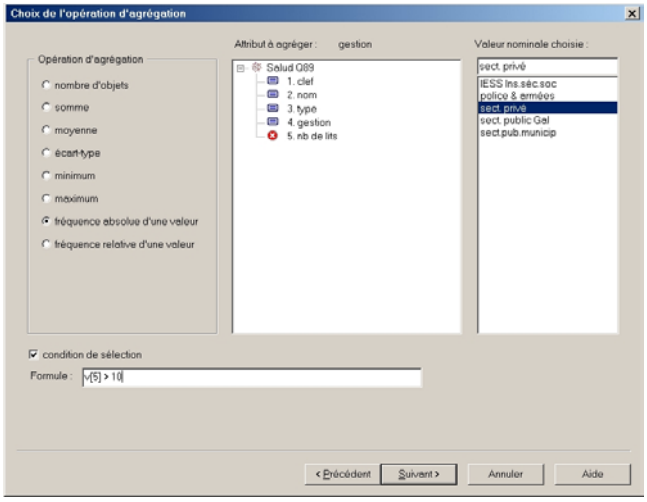


fig. 8.9 : le choix de l'opération d'agrégation descriptive

Notons enfin que si la clé d'agrégation est une clé de la relation, l'opération d'agrégation n'a pas d'intérêt. Et dans ce cas, si le résultat doit être créé dans une

autre relation, l'opération d'agrégation est identique à l'opération d'union que nous avons vue au chapitre 5.

8.4.5. Comparaison d'attributs

Il est souvent utile de pouvoir créer un nouvel attribut par comparaison d'attributs numériques d'une même relation. Cette opération crée deux nouveaux attributs : l'un est nominal, et correspond au nom de l'attribut qui correspond au rang choisi pour la comparaison, l'autre est numérique et contient la valeur de cet attribut. Cette opération permet de répondre aux questions du type : dans chaque commune, quel est le parti politique arrivé en seconde position, et combien d'électeurs sont-ils concernés ? Quelle est l'essence la moins représentée dans ces zones forestières, et combien d'arbres sont-ils concernés ?

8.4.6. Combinaison d'attributs

La combinaison d'attributs concerne les attributs nominaux. Elle permet de combiner deux attributs nominaux en créant un troisième attribut, nominal lui aussi, qui combine les valeurs des deux autres.

8.4.7. Ordre

La création par ordre perme de créer un nouvel attribut numérique à partir d'un attribut numérique de la relation. Le nouvel attribut donne le rang (croissant ou décroissant) de l'objet dans l'ensemble des objets de la relation par rapport aux valeurs de l'attribut choisi. Il permet par exemple de classer les entreprises par rapport à l'impôt sur les bénéfices. On pourrait par la suite facilement calculer le pourcentage que représente les vingt premières entreprises dans l'impôt national.

8.5. L'utilisation de la localisation pour la création de nouveaux attributs descriptifs

Les opérations relationnelles étendues à la localisation permettent déjà de répondre à un certain nombre de requêtes en utilisant la localisation de certains objets. Ainsi, la restriction spatiale d'une relation sur un masque créé à partir des objets d'une autre relation permet de répondre à toutes les interrogations du type « où ? ». La jointure spatiale permet de mettre en relation des objets de deux relations en utilisant leur localisation, sur un critère de distance. Mais ces opérations sont avant tout des opérations de gestion, visant à utiliser le principe de la décomposition relationnelle pour gérer au mieux des collections d'objet. Elles ne

créent jamais de nouveau attributs. La géo-jointure a en particulier le défaut majeur de faire perdre la localisation du résultat lorsque l'on n'utilise pas un critère d(O1,O2)=0. Ces opérations ne permettent pas non plus de mettre en œuvre le transfert d'échelle. Nous avons donc développé un certain nombre d'opérations pour répondre à ces questions. Ces méthodes, pour la plupart, sont spécifiques au type des relations et peuvent être considérées comme des méthodes sur les types généraux que sont les zones, les lignes, les points, les pixels.

Les opérations que nous présentons dans ce paragraphe permettent de créer de nouveaux attributs descriptifs, mais elles ne modifient pas la géométrie des objets ni ne créent de nouveaux objets ou de nouvelles relations. Nous verrons au paragraphe 6 de ce chapitre les opérations permettant la modification d'objets ou la création de nouveaux objets à partir des relations existantes.

8.5.1. Surface, périmètre, coordonnées

La surface et le périmètre sont typiquement des méthodes pouvant être appliquées uniquement au type zone.

Le calcul du périmètre d'une zone utilise les arcs frontière de la zone et la distance euclidienne dans le plan de projection géographique. Il ne pose aucune difficulté. Le résultat est toujours exprimé en mètres.

Le calcul de la surface d'une zone utilise également les arcs frontière de la zone. La surface est calculée dans le plan de projection, sans tenir compte de l'altitude des points (c'est une surface projetée). Elle peut être exprimée en mètres carrés, en hectares, en kilomètres carrés. Nous avons mis en oeuvre deux méthodes de calcul, une méthode par rasterisation et une méthode par calcul direct :

(1) La méthode par rasterisation calcule pour chaque zone une image *raster* binaire de la zone dans la plus petite fenêtre contenant la zone. La rasterisation utilise une résolution fixe de 1600*1600 pixels. On compte ensuite le nombre de pixels appartenant à la zone, et l'on approche ainsi sa surface puisque l'on connaît la taille du pixel de la rasterisation. Cette méthode est rapide mais elle introduit une incertitude sur le calcul due à la définition de la rasterisation.

(2) La seconde méthode calcule la surface directement à partir des arcs. Cette méthode rend nécessaire le chaînage des arcs en polygones fermés et la détection des polygones inclus. Le contour des polygones doit être orienté. La surface d'un polygone dont le contour est représenté par les points de coordonnées $(x_0,y_0,x_1,y_1,\ldots,x_n,y_n)$, dans le sens trigonométrique, est donné par la formule :

$$S=1/2 \sum_{i=1}^{n} (x_i y_{i+1} - y_i x_{i+1})$$ (le point x_{n+1},y_{n+1} correspond au point x_0,y_0)

La surface d'un polygone inclus doit être soustraite ou ajoutée suivant le degré d'inclusion. La méthode utilisée pour chaîner les arcs d'une zone en polygones et calculer le degré d'inclusion d'un polygone est identique à celle utilisée pour exporter des zones au format ShapeFile (chapitre 3, A.2.2.6.).

Dans SAVANE, les coordonnées des objets n'apparaissent pas directement dans le schéma des attributs : la localisation est implicite au type et ne peut être utilisée *a priori* qu'au travers des opérations de gestion et d'analyse spatiale. Mais il peut être utile de faire apparaître explicitement les coordonnées du centroïde dans le schéma des attributs, notamment lorsque l'on désire introduire la localisation dans une opération statistique multivariée ou pour exporter cette localisation vers un autre système. Une opération permet donc de créer deux attributs, l'un pour l'abscisse, l'autre pour l'ordonnée du centroïde. Les coordonnées sont alors exprimées dans le plan de projection géographique utilisé dans le cadre.

8.5.2. Orientation et pente

Le calcul d'orientation peut être appliqué à des objets de type ligne (fig. 8.10). Ce n'est une valeur exacte que dans le cas de lignes représentées par un segment de deux points. Sinon, l'orientation correspond à la moyenne de l'orientation des segments pondérée par la longueur de chaque segment.

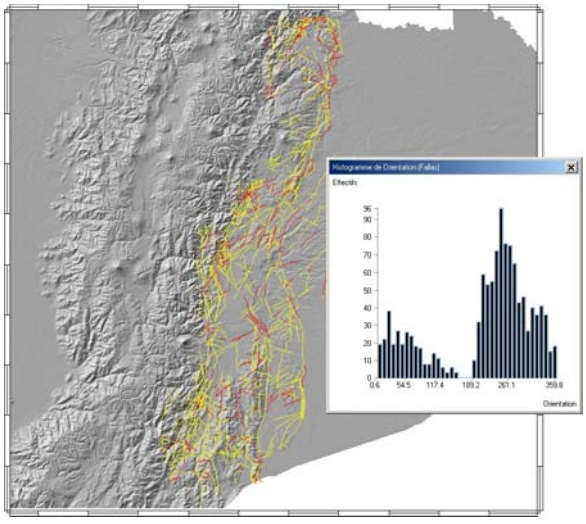


fig. 8.10 : dessin et histogramme de la répartition de l'orientation de failles géologiques, calculée à partir de la géométrie des objets

Le calcul de la pente utilise une relation de type mosaïque, contenant un attribut correspondant à l'altitude (souvent un modèle numérique de terrain créé à partir de courbes de niveaux). Le système cherche par géo-jointure l'altitude des deux extrémités de chaque ligne pour calculer la pente de la ligne. Comme pour l'orientation, la valeur calculée ne représente la pente réelle que dans le cas de lignes représentées par un segment de deux points

8.5.3. Distance à un point, à une relation, à un masque

Cette opération calcule la distance de chaque objet d'une relation à un point, à un masque, ou à l'objet le plus proche d'une autre relation localisée. Par distance, on entend la plus courte distance euclidienne de l'objet au point donné (choisi sur l'écran ou par ses coordonnées), au masque, ou à l'objet le plus proche dans l'autre relation. On utilise pour calculer la distance d'un point à un arc la procédure que nous avons présentée au chapitre 7.

8.5.4. Géo-agrégations

Les géo-agrégations permettent de mettre en œuvre des opérations de transfert d'échelle. Elles correspondent à une géo-jointure entre deux relations suivie d'une opération d'agrégation descriptive sur la clé géographique de la relation réceptrice. Elles utilisent donc la localisation des objets pour les mettre en relation. On utilisera la terminologie *émetteur* et *récepteur* pour différencier la relation qui émet les objets et la valeur d'un de ses attributs, et la relation qui reçoit le nouvel attribut créé par agrégation. La relation réceptrice est donc celle qui généralise le phénomène, chaque objet de la relation réceptrice étant censé recevoir plusieurs objets de la relation émettrice. On calcule la valeur du nouvel attribut de l'objet récepteur à partir des valeurs de ces objets, par agrégation descriptive. Grâce à cette agrégation descriptive, le résultat d'une géo-agrégation conserve la localisation des objets récepteurs. L'opération d'agrégation descriptive est classique : somme, moyenne, écart-type, variance, minimum, maximum, dans le cas d'un attribut numérique, et fréquence relative ou absolue dans le cas qualitatif.

8.5.4.1. La géo-agrégation géométrique par distance

L'opération de géo-agrégation par distance cherche, pour tous les objets de la relation réceptrice, quels sont les objets de la relation émettrice qui doivent lui être agrégés. Les relations émettrice et réceptrice doivent être localisées, mais peuvent être de n'importe quel type (zone, ligne, point, pixel). On peut également utiliser une distance autour des objets récepteurs, pour ne pas limiter la jointure à une opération d'inclusion des objets émetteurs aux objets récepteurs. L'opération reste néanmoins ambiguë dans certains cas.

(1) Lorsque la relation émettrice est de type point, il n'y a pas d'ambiguïté : pour chaque objet récepteur, les objets émetteurs à joindre sont ceux dont la distance du point à l'objet récepteur est inférieure à la distance d'agrégation. L'opération utilise le calcul de la distance d'un point à un point (objet récepteur de type point ou pixel), à un arc (objet récepteur de type ligne), ou à un ensemble d'arcs s'il n'y a pas inclusion (objet récepteur de type zone) (fig. 8.11).

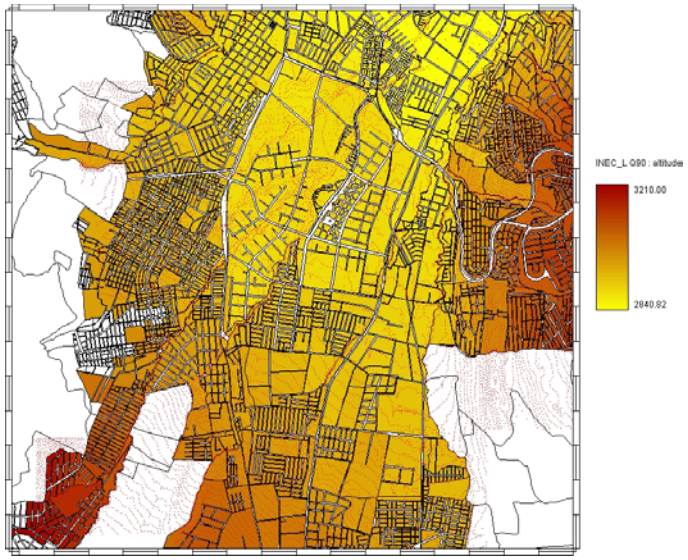


fig. 8.11 : exemple de géo-agrégation de points dans des zones : calcul de l'altitude moyenne par îlot à partir de points cotés (moyenne, d < 100 m)

(2) Lorsque la relation émettrice est de type ligne, il peut y avoir ambiguïté : pour chaque objet récepteur, les objets émetteurs à joindre sont ceux dont la distance de l'arc à l'objet récepteur est inférieure à la distance d'agrégation. L'opération utilise le calcul de la distance d'un arc à un point (objet récepteur de type point ou pixel), à un arc (objet récepteur de type ligne), ou à un ensemble d'arcs s'il n'y a pas intersection (objet récepteur de type zone). Le critère de jointure utilise donc dans tous les cas la distance d'un arc à un objet, c'est-à-dire la distance minimale de tous les points de l'arc à cet objet : il suffit qu'un point de l'arc soit proche de l'objet récepteur pour qu'il soit pris en compte dans l'agrégation.

(3) Lorsque la relation émettrice est de type zone, il peut également y avoir ambiguïté : pour chaque objet récepteur, les objets émetteurs à joindre sont ceux dont la distance de la zone à l'objet récepteur est inférieure à la distance d'agrégation. C'est le cas lorsque l'objet récepteur est inclus ou intersecte la zone,

ou que la distance entre l'objet récepteur et les arcs formant le contour de la zone émettrice est inférieure à la distance d'agrégation.

Dans tous les cas, si un objet émetteur est proche de plusieurs objets récepteurs, il est mis en relation avec tous ces objets. Si l'agrégation est une somme et porte sur un attribut numérique représentant un effectif, agréger plusieurs fois l'objet émetteur dans des objets récepteurs différents revient à dupliquer la valeur. Pour éviter cette erreur, il est possible de choisir une option permettant de n'agréger l'objet émetteur que dans l'objet récepteur le plus proche (fig. 8.12).

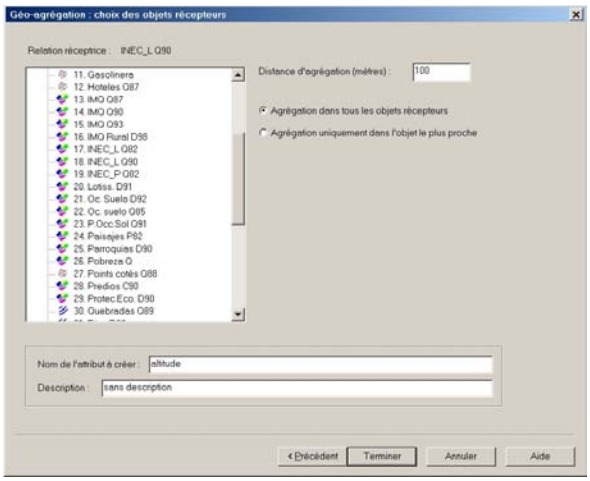


fig. 8.12 : la page finale du dialogue de géo-agrégation par distance

L'opération de géo-agrégation par distance doit être utilisée avec prudence, comme nous l'avions déjà signalé au début de ce chapitre. Sa validité dépend de la précision sémantique et graphique de chaque relation, et dépend du type des objets et des ambiguïtés qui sont créées lorsque les découpages ne sont pas hiérarchiques.

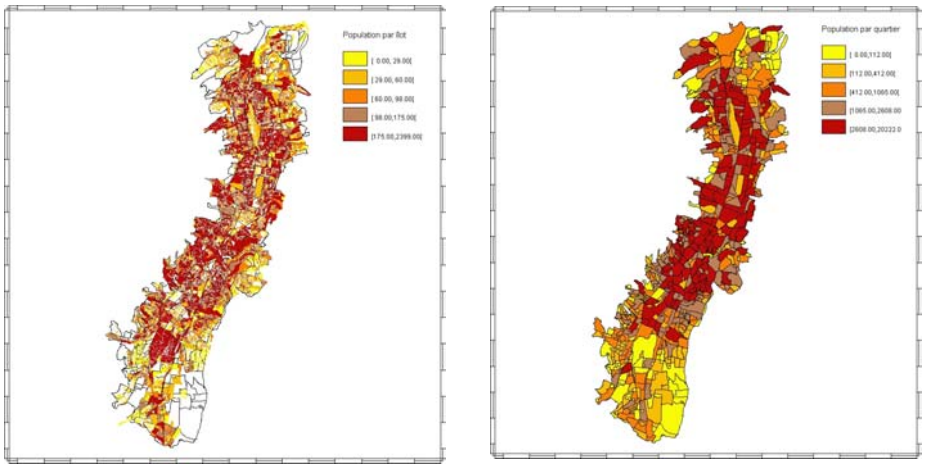


fig. 8.13 : exemple de géo-agrégation de zone à zone sans ambiguité : agrégation de la population d'îlots aux quartiers (somme, d=0)

Elle ne doit être employée en principe que pour résoudre des opérations de transfert d'échelle, et donc lorsque la précision de la relation émettrice et supérieure à la précision de la relation réceptrice, ou lorsqu'il y a hiérarchie géographique des objets (fig. 8.13).

Notons enfin que l'opération d'agrégation par distance utilise la distance entre objets émetteurs et récepteurs pour savoir s'ils doivent être mis en relation, mais n'utilise pas cette distance dans le calcul d'agrégation descriptive qui suit. Par exemple, l'opération de géo-agrégation entre deux relations de type pixel correspond en quelque sorte à un ré-échantillonage, dans lequel seules les options de plus proche voisin ou de moyenne seraient disponibles. L'utilisation de la distance dans le calcul est employée dans d'autres opérations que nous verrons plus loin : pour un véritable ré-échantillonage entre relations de type pixel, ou pour la création de nouveaux objets par interpolation.

8.5.4.2. Géo-agrégation par surface d'un masque

Ce type d'agrégation cherche à résoudre l'ambiguïté de la géo-agrégation zone-zone lorsqu'il n'y pas hiérarchie des découpages, et lorsque l'on cherche à rendre compte de la distribution d'un caractère (nominal) dans un autre découpage. C'est bien souvent le cas lorsque les deux thèmes ne sont pas liés mais d'une précision équivalente. On a alors deux collections d'objets de type zone avec de multiples intersections entre les objets des deux relations.

L'opération permet de calculer pour chaque zone réceptrice le pourcentage de la surface de la zone occupée par un masque. Si ce masque a été créé à partir des objets d'une autre relation, après restriction sur un caractère nominal, le résultat correspond pour chaque zone réceptrice au pourcentage de la surface occupée par le caractère (fig. 8.14).

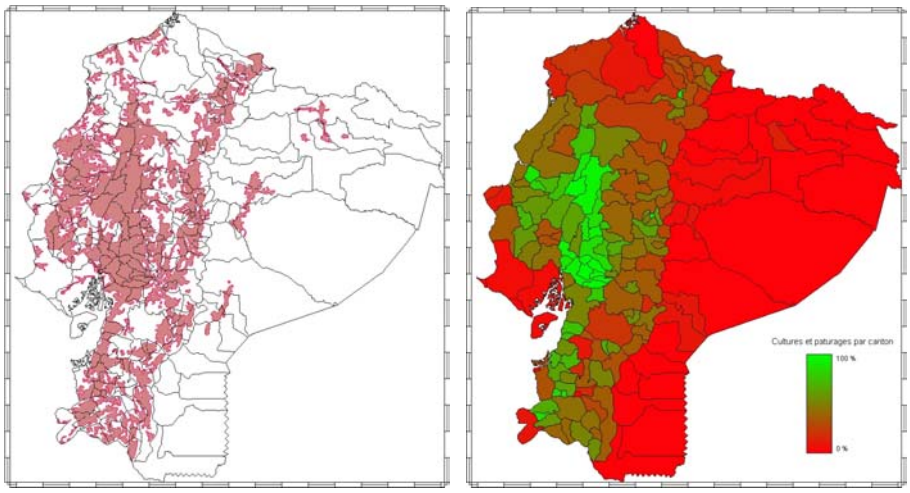


fig. 8.14 : un masque (cultures et paturages) et le pourcentage de la surface dans un découpage (cantons)

8.5.4.3. Géo-agrégation par valeur pondérée par la surface

Toujours dans le cas d'une géo-agrégation zone-zone lorsque les découpages ne sont pas hiérarchiques, mais lorsque l'on veut rendre compte d'un attribut numérique de l'un des découpages dans l'autre, il peut être intéressant de pondérer par la surface occupée par chaque valeur dans la zone réceptrice. On fait ainsi une hypothèse sur la répartition linéaire des valeurs en fonction de la surface occupée.

On n'utilise pas de masque dans cette opération qui effectue directement le calcul du nouvel attribut en faisant par zone la moyenne des valeurs pondérée par la surface occupée (fig. 8.15).

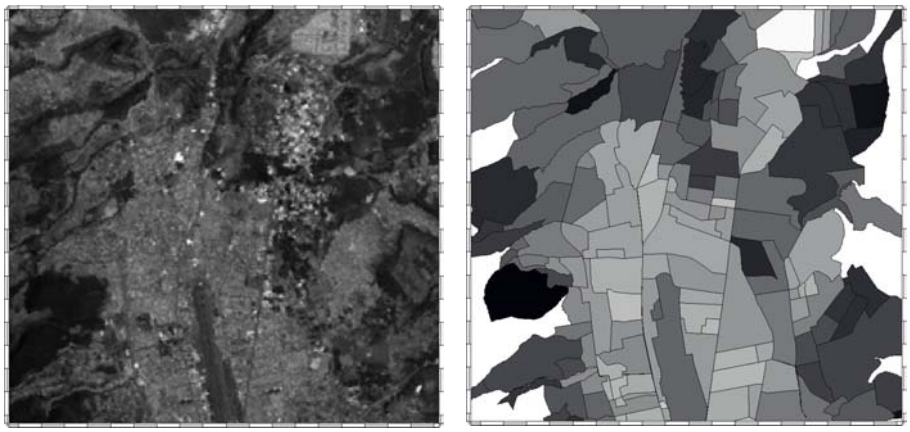


fig. 8.15 : calcul de la moyenne d'un indice radiométrique (TM, canal 1) dans des zones prédéfinies (quartiers) par géo-agrégation pondérée par la surface

8.5.5. Appartenance

L'appartenance est l'opération inverse de l'agrégation. Elle consiste à aller chercher dans une relation émettrice l'objet qui contient le centroïde de l'objet récepteur. La valeur de l'attribut à émettre est alors affectée à l'objet récepteur. La réalisation informatique dans SAVANE est soit vectorielle (test de l'inclusion ou de la distance du centroïde à l'objet émetteur en utilisant les arcs), soit matricielle (on utilise alors la représentation matricielle de la localisation : l'exécution est plus rapide mais la précision dépend de la taille du pixel de rasterisation et donc de la taille de la fenêtre d'étude).

L'opération utilise le centroïde de l'émetteur et peut donc être ambiguë, sauf s'il y a inclusion des objets récepteurs dans les objets émetteurs. Elle ne devrait être utilisée que dans ce cas.

8.5.6. Géoagrégation sur les pixels : le ré-échantillonage

L'agencement des objets d'une relation de type mosaïque permet de définir un certain nombre d'opérations propre à ce type de relation. Par exemple, l'opération de géoagrégation par distance peut être affinée en utilisant les méthodes de ré-échantillonage que nous avons vues au chapitre 6. La relation réceptrice est de type zone ou mosaïque, et la valeur d'une zone est calculée à partir des pixels émetteurs par voisinage plutôt que par distance ou appartenance. On peut ainsi utiliser une fonction bilinéaire ou bicubique pour calculer la valeur d'objet récepteur.

8.5.7. Opérations sur les modèles numériques : orientation, pente, écoulements

Un modèle numérique de terrain est un attribut d'une relation de type mosaïque représentant une altitude. Cet attribut est souvent obtenu par interpolation, comme nous le verrons dans le paragraphe suivant. De nombreuses opérations sont spécifiques à ce type dérivé du type mosaïque : pente, orientation, écoulements, calculs hydrologiques, calculs de volume ou de visibilité. Nous sommes encore dans le domaine de l'analyse spatiale, mais spécialisée ici à un domaine particulier. Nous pouvons ici appliquer des méthodes correspondant au contenu sémantique de l'attribut, alors que nous n'avions vu jusqu'à présent que des opérations valables quel que soit le contenu sémantique de l'attribut.

8.5.7.1. Pente et orientation

Pour chaque pixel, la pente ou l'orientation est calculée sur les voisins : c'est l'angle formé par le vecteur unitaire normal à la surface et le plan z=0 (pente) ou x=0 (orientation). En chaque point, on approche la surface en utilisant le plan de régression le plus proche des quatre voisins du point $p_{i,j}$ ($p_{i-1,j-1}$,$p_{i+1,j-1}$,$p_{i+1,j-1}$,$p_{i+1,j+1}$). L'opération crée un nouvel attribut de type entier 16 bits : les pentes et orientation sont conservées en minutes d'arc.

8.5.7.2. D'autres indices en géomorphologie et hydrologie

De nombreux indices peuvent ainsi être calculés à partir de l'attribut altitude pour créer de nouveaux attributs : convexité verticale, convexité horizontale, convexité transversale, encaissement, modèle de drainage, bassins-versants, longueurs de drainage, surfaces drainées, distances à l'exutoire, etc. Tous ces nouveaux attributs temporaires peuvent bien sûr être utilisés dans la suite de la requête (fig. 8.16).

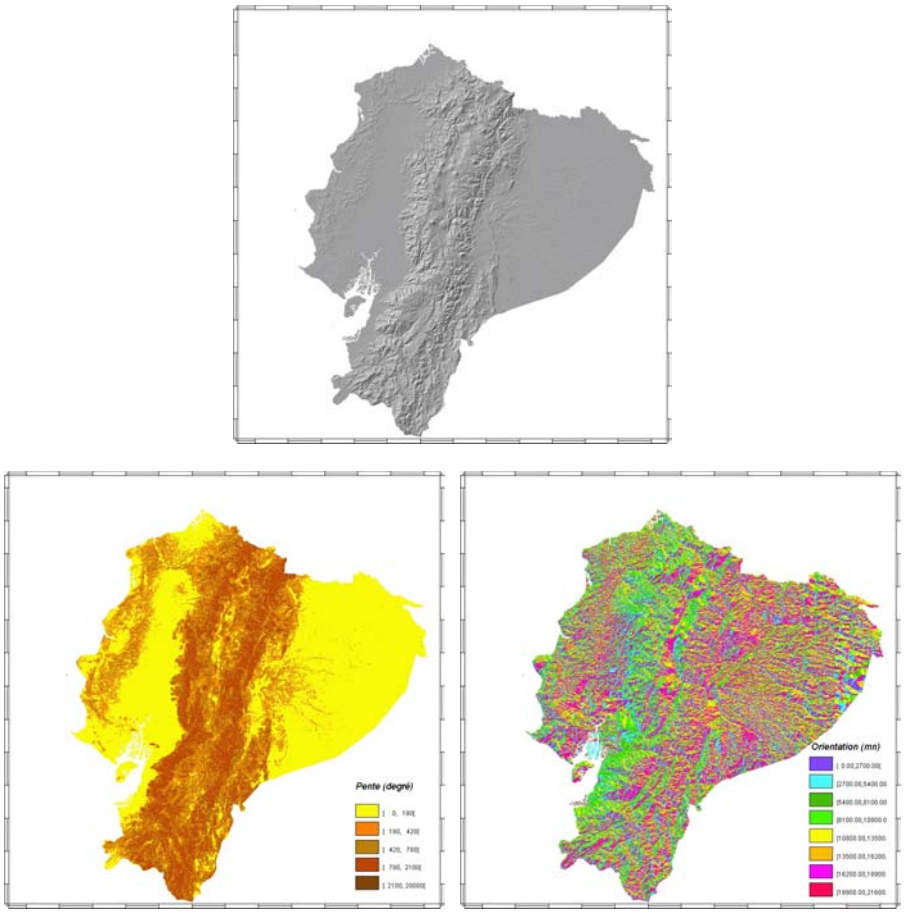


fig. 8.16 : pente et orientation calculées à partir d'un modèle numérique de terrain

8.6. La création de relations temporaires

Jusqu'à présent, nous avons vu des opérations d'exploration ou de création de nouveaux attributs à partir de relations existantes. Aucune de ces opérations ne créent de nouveaux objets. Mais bien souvent, l'implantation spatiale des objets d'une collection ne permet pas d'avancer dans l'analyse, et il faut modifier l'implantation spatiale en créant de nouveaux objets, soit de toute pièce, soit à partir des objets d'une collection existante. Nous allons voir dans ce paragraphe les nombreuses opérations permettant la création de nouvelles collections d'objets. Ces opérations s'inscrivent toujours dans le cadre de l'analyse spatiale, mais elles vont plus loin que la simple réponse à une question posée. Elles sont fondamentales pour

assurer un usage efficace de l'extension du modèle relationnel, car elles permettent de s'affranchir de la barrière imposée par le type de l'implantation spatiale. En effet, les opérations que nous avons vues plus haut sont dans de nombreux cas limitées à un type de localisation. Le passage d'un type à un autre permet alors de les utiliser et met l'accent sur la difficulté d'utilisation d'objets dont la définition et la précision diffèrent, au sein d'une même base de données.

Après avoir rappelé le principe de création de nouvelles relations temporaires dans le schéma de la base de données, nous avons organisé ce paragraphe en regroupant les opérations par le type d'implantation spatiale des objets créés.

8.6.1. Le principe de la création de nouvelles relations

Lors d'une requête, il est possible de créer de nouvelles collections d'objets. Ces relations sont temporaires, le schéma de la base de données est modifié temporairement comme lors de la création d'attributs. Mais la création d'une nouvelle relation localisée implique la création de toute la structure des fichiers correspondants : fichier index, fichier descriptif, fichier graphique. La procédure `NouvelleRelation()` de la classe `CSchema` encapsule les différentes opérations nécessaires à la modification du schéma (A.2.1.2.).

Il existe plusieurs manières de créer une nouvelle collection d'objets :

- par duplication (valable quel que soit le type de la relation) ;

- par définition des objets (mailles) ou par digitalisation des objets sur écran (valable pour les types zone, ligne, point) ;

- à partir d'une relation existante, par modification géométrique des objets (érosion, dilatation, ou par masque) ou par changement de type d'implantation spatiale.

Nous allons voir rapidement les deux premières opérations, avant de consacrer un paragraphe plus détaillé à la création de relation par modification des objets d'une relation existante.

8.6.2. La duplication

La duplication d'une relation permet de recopier tous les objets d'une relation dans une nouvelle relation identique. Pour le type zone, les arcs ne sont pas recopiés : les deux relations partagent le fichier des arcs.

Cette opération permet de conserver une copie d'une relation avant de la modifier par restriction ou projection.

8.6.3. La définition directe sur écran ou par masque

La définition directe sur écran permet à l'utilisateur de créer une nouvelle relation temporaire en numérisant directement les objets dans le cadre géographique.

La définition par masque crée une relation de type zone, possédant un seul objet correspondant au masque utilisé.

8.6.4. La création de mailles

Bien souvent, on ne dispose pas d'un découpage de l'espace représentatif d'une échelle de description donnée, ou suffisamment régulier pour être employé dans une opération de changement d'échelle et de géo-agrégation. En effet, il ne faut pas que le découpage utilisé pour agréger soit corrélé à la répartition spatiale des objets que l'on agrège. La solution la plus simple consiste donc à créer de nouveaux objets dont la géométrie ne puisse pas a priori interférer dans le processus d'agrégation : les mailles carrées ou hexagonales sont les plus employées, car elles correspondent à une répartition uniforme des objets dans l'espace (fig. 8.17).

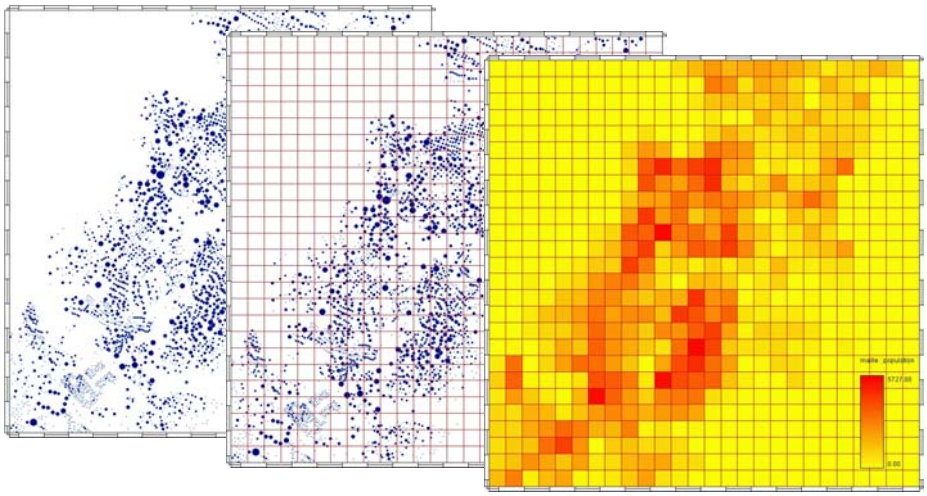


fig. 8.17 : Données ponctuelles, mailles, et géo-agrégation dans les mailles

Si la forme des objets est prédéfinie, on peut par contre choisir la taille des mailles. La variation de la dimension de la maille n'a pas le même effet selon que le semis de point des objets à agréger est plus concentré ou plus dispersé. Dans le premier cas, la diminution de la taille de la maille a tendance à augmenter la

dispersion relative et les disparités de densité entre les mailles. Dans le second cas, les effets sont plus faibles. Le choix de la taille des mailles doit tenir compte du nombre de points à agréger et de la forme du semis de points. Souvent, la meilleure taille est celle qui minimise la variance intra-maille et maximise la variance inter-maille pour l'attribut à agréger.

La création de maille correspond à la création de nouveaux objets : on crée ici une nouvelle relation, temporaire, de type zone.

8.7. Les opérations de changement de type d'objet

Il est très courant d'utiliser les objets d'une ou plusieurs relations existantes pour créer de nouveaux objets. Bien souvent, cette création d'objet s'accompagne d'un changement de type d'implantation spatiale des objets des relations d'origine. De nombreux algorithmes sont employés dans ces opérations : interpolation, extrapolation, vectorisation, triangulation et tessellation, morphologie mathématique, etc. Nous allons détailler ces diverses méthodes en les classant en fonction du type de la nouvelle relation créée.

8.7.1. Vers le pixel : interpolation ou rasterisation

8.7.1.1. Mosaïque et type « Raster »

L'objectif est ici de créer une nouvelle relation dont les objets sont des pixels, c'est-à-dire des objets de géométrie identique et de taille connue, comme nous l'avons défini au chapitre 6. Il y a analogie avec la création de mailles, mais ici le nombre d'objets créés sera en général beaucoup plus grand et la valeur de l'attribut sera calculée directement lors de la création des objets et non par géo-agrégation comme dans le cas des mailles. La relation créée est de type mosaïque et non de type zone : la gestion d'un nombre de mailles dépassant le million n'est pas très efficace, alors que la gestion d'une relation de type mosaïque est conçue pour recevoir sans problème plusieurs dizaines de millions de pixels.

Alors que les mailles sont construites pour recevoir des valeurs par géo-agrégation, la valeur des pixels est construite directement par interpolation en utilisant la localisation du centroïde des objets initiaux dans le calcul de l'interpolation : le changement d'échelle sous-jacent s'apparente donc plus à une opération d'appartenance qu'à une opération d'agrégation. La taille du pixel sera en général inférieure ou égale à la précision des objets initiaux, alors que la taille des mailles est le plus souvent choisie pour effectuer une généralisation. La surface du pixel n'est jamais utilisée dans le calcul d'interpolation.

La relation créée peut être de type mosaïque ou de type « raster ». Cette relation particulière fixe la taille des pixels en fonction de la fenêtre d'étude. Elle est gérée comme une relation de type mosaïque, mais ne contient qu'une seule feuille, est toujours temporaire, et n'existe qu'à un seul exemplaire dans la requête. Elle est automatiquement détruite lors d'un changement de fenêtre d'étude ou de projection géographique.

8.7.1.2. Méthodes d'interpolation

La création d'une relation de type mosaïque crée des pixels sur l'ensemble de la surface de la fenêtre d'étude ; on peut limiter cette création au domaine représenté par un masque. La création de la relation et des objets s'accompagne de la création d'un attribut pour tous les pixels : la valeur est obtenue à partir des objets des relations initiales choisies par l'utilisateur, qui peuvent être de type zone, ligne, ou point. On utilise pour cela des méthodes d'interpolation qui prennent en compte la position des objets initiaux par rapport au pixel. Les méthodes d'interpolation sont très nombreuses : leur choix dépend du contenu sémantique de l'attribut à créer.

Par exemple, la création d'un modèle numérique de terrain relève de cet type d'opération : on cherche à créer des pixels avec un attribut altitude à partir d'objets initiaux qui peuvent être des courbes de niveaux ou des points cotés. L'interpolation utilisée doit correspondre à ce type de problème : de nombreuses méthodes ont été proposées et font l'objet d'une abondante littérature. On peut citer notamment les méthodes barycentrique, statistique, par triangulation, par fonction polynomiale, spline, etc.

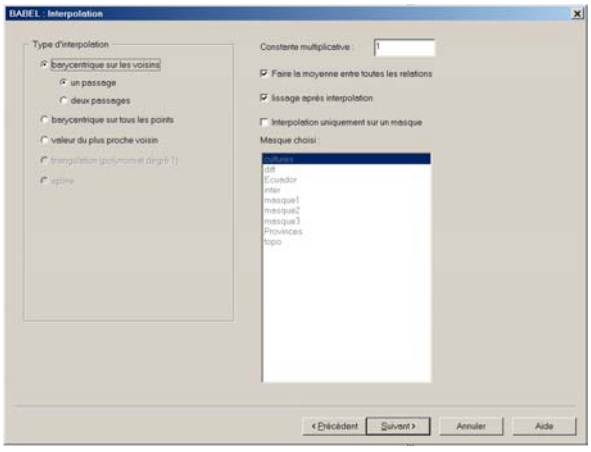


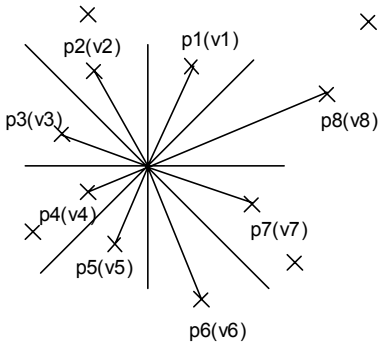
fig. 8.18 : le dialogue du choix de l'opération d'interpolation

Nous avons développé dans SAVANE quelques méthodes d'interpolation : plus proche voisin, barycentrique sur les voisins, barycentrique sur tous les points, bilinéaire dans triangulation, spline après barycentrique. Ces méthodes permettent de répondre à la plupart des besoins d'interpolation (fig. 8.18).

8.7.1.2.1. La méthode barycentrique

La méthode d'interpolation barycentrique est la plus utilisée en pratique. Elle permet notamment d'obtenir de très bons résultats pour la création des modèles numériques de terrain lorsque les données initiales sont suffisamment denses. Nous appellerons « points de référence » les points du plan qui servent de support au modèle et permettent l'interpolation. Ils proviennent de la localisation des objets de relations existantes dans la base de données. La méthode de calcul consiste en une interpolation de type barycentrique sur les huit points de référence les plus proches du point à interpoler, répartis dans les huit portions du plan. L'importance d'un point de référence est d'autant plus grande dans le résultat du calcul que sa distance au point à interpoler est faible (par rapport aux distances des autres points au point à interpoler). Cette méthode ne donne pas une surface différentiable, et n'utilise pas de modèle local ou de modèle de variance : elle nécessite donc une densité de points de référence d'autant plus grande que les variations de l'attribut sont fortes. C'est en fonction de cette variation que les points de référence doivent être choisis si cela est possible.

La méthode d'interpolation barycentrique est stable : elle ne modifie pas la valeur des points de référence dans le résultat de l'interpolation. Cette propriété est essentielle dans de nombreuses applications.



$$v(Pij) = \frac{\sum_{k=1}^{8} v(Pk) \prod_{\substack{m=1 \\ m \neq k}}^{8} d(Pij,Pm)}{\sum_{k=1}^{8} \prod_{\substack{m=1 \\ m \neq k}}^{8} d(Pij,Pm)}$$

Après création des pixels, le calcul de la valeur de l'attribut par interpolation s'effectue en deux temps : dans un premier temps, on affecte à chaque pixel la moyenne des valeurs des points de références dont la distance au pixel est inférieure à la taille du pixel. Ces pixels deviennent alors les points de références utilisés pour calculer la valeur des autres pixels (une option permet d'éviter de faire la moyenne entre les points provenant de différentes relations). Dans un second temps, on cherche pour chaque pixel les pixels les plus proches dans chacun des huit cadrans, et on affecte au pixel la valeur interpolée à partir de ces pixels. Enfin, le résultat peut être lissé (par une fonction polynomiale de Béziers de degré 6) sur chaque ligne et chaque colonne pour supprimer les erreurs éventuelles dans les points de référence. La recherche des points valués voisins est l'étape la plus longue de l'opération : elle peut être très longue si la densité de points de référence est trop faible (fig. 8.19).

L'opération est effectuée imagette par imagette. Le nombre d'imagettes créées dans la relation dépend de la taille de pixel choisie pour la nouvelle relation. La matrice de pixel utilisée pour conserver les points de référence doit être plus grande que l'imagette pour ne pas exclure arbitrairement les points extérieurs qui pourraient participer au calcul de la valeur d'un point de l'imagette, notamment pour les points près du bord, et assurer ainsi la continuité des valeurs d'une imagette à l'autre.

La classe `Cbabel` encapsule l'ensemble des variables et des méthodes utilisées pour le calcul d'interpolation barycentrique (A.2.1.6.).

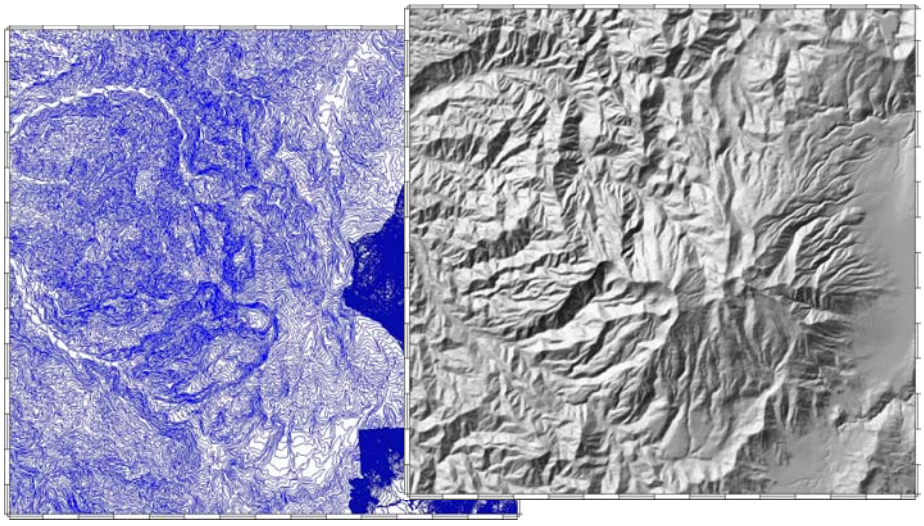


fig. 8.19 : résultat de l'interpolation barycentrique à partir de courbes de niveau

Les relations donnant les points de référence peuvent être de type zone, ligne, ou point. Dans la cas de zone, on utilise le centroïde de la zone comme point de référence. Dans le cas de ligne, on utilise l'ensemble de l'arc représentant la ligne, ce qui donne de nombreux points de référence. La méthode `Draw()` de la classe `CArc` calcule les points de référence à partir des points de l'arc.

L'option *deux passages* proposé pour le calcul barycentrique d'interpolation permet de palier au problème posé par une mauvaise densité des points de référence et d'éviter des temps de calcul trop longs. Le calcul s'effectue alors en deux passages : un premier passage permet de calculer par interpolation des points suivant un pas donné (un pixel sur dix, par exemple). Ces points calculés sont ensuite ajoutés aux points de référence et utilisés dans le calcul général d'interpolation. La recherche de voisins est alors limitée au maximum au pas de la première interpolation.

8.7.1.2.2. Barycentre sur tous les points

L'interpolation barycentrique sur tous les objets s'apparente à un calcul de potentiel ou d'influence. La valeur calculée pour chaque pixel est la moyenne des valeurs des points de référence pondérées par la distance. La méthode ne doit être employée que lorsque le nombre de points de référence est faible (fig. 8.20). On peut néanmoins limiter la calculer aux points dont la distance au point à interpoler est inférieure à une valeur donnée.

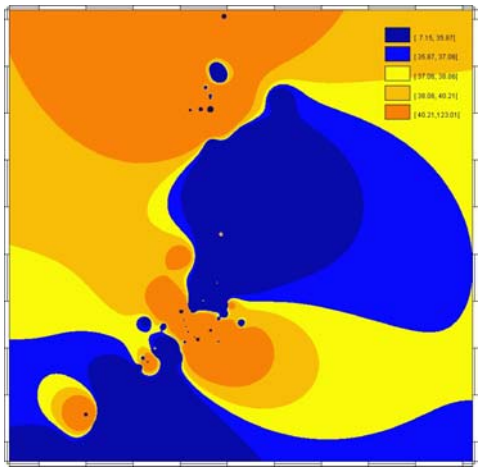


fig. 8.20 : exemple d'interpolation barycentrique sur tous les points de référence (classée par quantiles)

8.7.1.2.3. Interpolation barycentrique sous contrainte

Le caractère à interpoler est parfois directement lié à un autre caractère, connu sur l'ensemble de l'espace d'étude. Par exemple, on ne peut interpoler la température au sol sans tenir compte de l'altitude. On a donc le choix d'interpoler des valeurs normalisées (par exemple des températures ramenées au niveau de la mer), puis de calculer les variations dues à la dépendance des variables (par exemple, la dépendance linéaire de la température en fonction de l'altitude), ou de calculer directement le nouvel attribut par une interpolation sous contrainte. Cette option est disponible dans SAVANE pour les contraintes linéaires.

8.7.1.3. Zone vers pixel : rasterisation

Les opérations que nous avons décrites au paragraphe précédent ne prennent en compte que le centroïde des zones dans le processus d'interpolation. L'opération de rasterisation d'une relation zonale permet d'affecter des valeurs aux pixels de la relation « raster » en fonction de leur appartenance aux zones. Elle correspond à l'opération d'interpolation par plus proche voisin ou d'appartenance d'un pixel à une zone.

On peut ainsi créer des attributs dans la relation « raster » à partir de nombreuses relations différentes. Ces attributs peuvent ensuite être combinés, pixel par pixel, en utilisant les opérations que nous avons vues au paragraphe 4 de ce chapitre. Comme nous l'avons déjà souligné, la valeur du pixel provient d'une opération d'appartenance et non d'agrégation, comme dans le cas de mailles. Les deux approches sont donc complémentaires.

La rasterisation d'une relation zonale utilise l'algorithme de rasterisation que nous avons déjà vu à plusieurs reprises.

8.7.2. Vers la zone : vectorisation ou tessellation

Nous avons déjà vu la création directe d'une nouvelle relation de type zone, par création de mailles ou directement par digitalisation. Mais il est également très intéressant de créer des zones à partir des objets d'une autre relation. Les méthodes de création de zones dépendent du type des objets de la relation initiale :

- à partir d'une relation de type pixel, on utilisera un algorithme de vectorisation et labélisation pour créer les arcs et les objets ;

- à partir d'une relation de type ligne, on utilisera un algorithme de construction de zone pour créer des polygones à partir d'un ensemble d'arcs, si cela est possible ;

- à partir d'une relation de type point, on utilisera un algorithme de tessellation pour diviser l'espace en zones à partir du semis de point ; cette méthode peut

également être utilisée pour tous les types d'objets vectoriels, à partir du centroïde des objets ;

- à partir d'une relation de type zone, on peut redéfinir de nouvelles zones par morphologie mathématique (érosion ou dilatation) ou par regroupement.

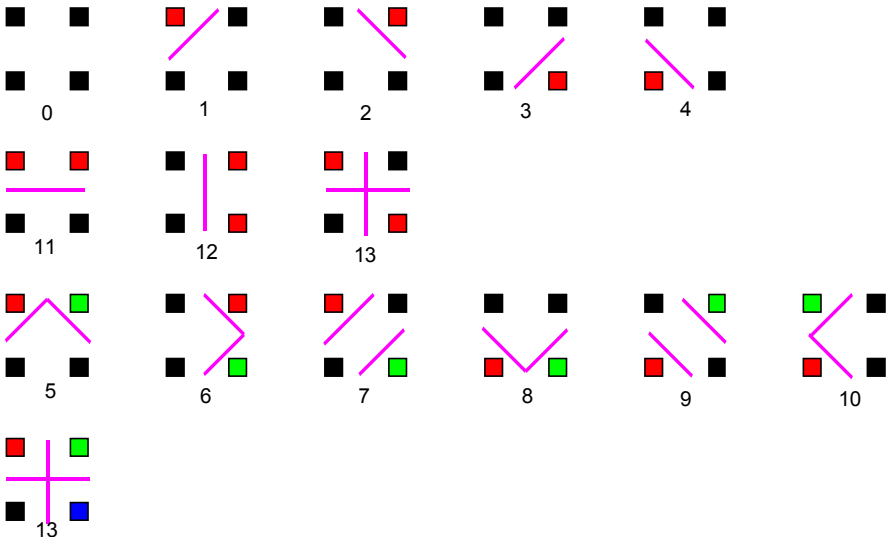
L'algorithme de vectorisation est le plus complexe. Il est utilisé également pour calculer le contour des masques après des opérations sur leur représentation matricielle.

8.7.2.1. Un algorithme de vectorisation

L'opération de vectorisation est l'inverse de la rasterisation : elle vise à créer des arcs à partir d'un ensemble de pixels, en faisant passer un arc entre les pixels qui n'ont pas la même valeur. Nous avons développé un algorithme de vectorisation d'une image codée dont nous allons rapidement décrire le principe.

L'algorithme de vectorisation que nous avons développé cherche à faire passer les arcs entre les pixels, et non sur les pixels. On ne cherche donc pas le cheminement d'un pixel à un autre, mais on cherche les configurations de valeurs quatre pixels adjacents qui permettent le passage d'un arc : un arc doit passer entre deux pixels si leurs valeurs sont différentes. Dans une première phase, l'algorithme teste les configurations de quatre pixels adjacents pour déterminer si une ligne doit passer entre ces pixels. Dans une seconde phase, les segments d'arcs ainsi déterminés sont chaînés entre eux pour créer des arcs. Dans une troisième phase, les objets sont créés en fonction des valeurs de l'image initiale. La classe `CVectoriser` encapsule l'ensemble des opérations de la vectorisation d'une image (A.2.2.3.).

Les configurations possibles de quatre pixels adjacents sont au nombre de 13. Elles sont codées suivant le schéma suivant, où deux pixels de même valeur sont représentés par la même couleur, et où le trait représente le trajet du segment d'arc à créer :



L'image initiale est donc remplacée par l'image des configurations codées (procédure `SegmentType()` de `Cvectoriser()`).

La seconde phase de l'algorithme, la plus délicate et la plus longue, vise à chaîner l'ensemble de ces segments d'arcs pour constituer des arcs. Les arcs conservent le code des pixels adjacents qu'ils séparent. Le principe est assez simple : à partir du premier segment qui n'a pas encore été chaîné, on cherche le segment qui peut le continuer, et ainsi de suite, dans un sens, puis dans l'autre. Le type du segment donne la direction dans laquelle il faut chercher le suivant. La recherche doit s'arrêter lorsque l'on tombe sur une configuration à trois segments, qui correspond à un nœud (type 5, 6, 8 , 10, 13). Des types temporaires doivent être créés pour gérer les configurations à plusieurs segments (5, 6, 7, 8, 9, 10, 13 ; par exemple, le type 6 devient type 2 ou 3 après chaînage d'un premier segment, mais sans oublier que la configuration correspond à un nœud). Les bords de l'image sont chaînés à part.

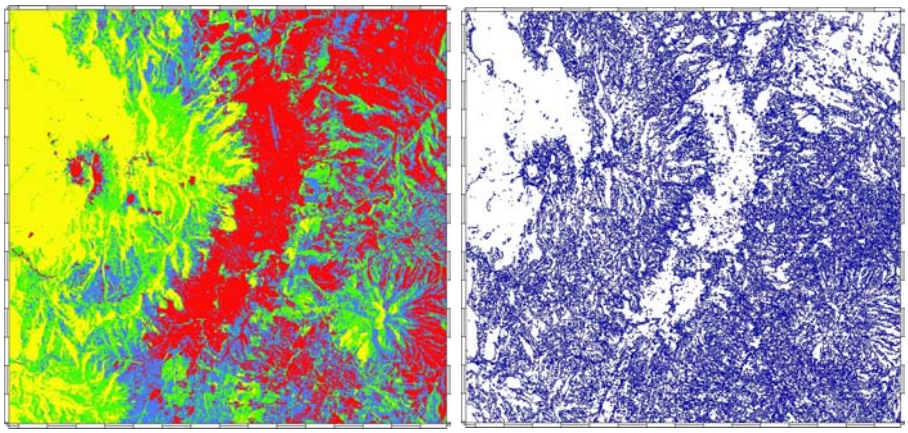


fig. 8.21 : vectorisation après classification d'une mosaïque (image TM)

Enfin, les arcs ainsi créés sont lissés pour éliminer les points superflus (de nombreux points créés sont alignés) (fig. 8.21).

L'algorithme de vectorisation est employé pour créer une relation de type zone à partir d'une relation de type mosaïque. La vectorisation est effectuée pour chaque imagette et crée dans la relation vectorielle une feuille par imagette.

Les objets zone sont créés à partir des codes rencontrés dans la mosaïque initiale, par imagette. Chaque zone représente en général un ensemble de polygones non connexes, car il n'y a pas labélisation de chaque polygone. La création d'une zone s'accompagne de la création de son centroïde.

La localisation d'un centroïde à partir d'un ensemble de polygones quelconques n'est pas triviale : il faut s'assurer que le centroïde se trouve bien dans le polygone et est bien placé par rapport à la forme du polygone. Plusieurs méthodes peuvent être employées, avec la même idée : déterminer un point central, puis bouger ce point s'il ne se trouve pas dans la zone. Comme point central, on peut prendre le centre du plus petit rectangle contenant les polygones, le centre de l'un des polygones, ou le centre du cercle inscrit à l'un des polygones. Le test de l'inclusion du point dans les polygones peut être effectué par l'algorithme YX que nous avons déjà vu au chapitre 7. Si le point est en dehors de la surface de la zone, il peut être déplacé en X pour se situer au milieu du polygone qui le précède ou qui le suit.

L'algorithme de vectorisation est également employé pour vectoriser les masques, ou pour vectoriser des objets après des opérations effectuées sur des représentations matricielles.

8.7.2.2. Un algorithme de tessellation

On utilise pour la création de zones à partir de points un algorithme de tessellation, c'est-à-dire de partition de l'espace en zones disjointes. Nous avons mis en œuvre dans SAVANE la tessellation de Voronoï, duale de la triangulation de Delaunay que nous avons déjà vue au chapitre 6. La tessellation est calculée à partir de la triangulation : celle-ci donne la liste orientée des arêtes des triangles partant de chaque point, il suffit donc de calculer les intersections deux à deux de ces arêtes pour déterminer les arcs. Il y a ici bijection entre les points initiaux et les zones créées, et chaque zone conserve l'identifiant de l'unique point initial qu'elle contient. Les points servent également de centroïdes aux zones (fig. 8.22). Le bord de la fenêtre géographique de travail est utilisée pour fermer les zones extérieures.

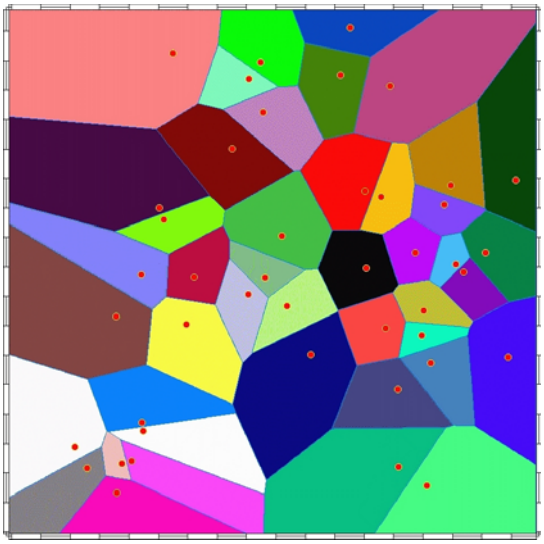


fig. 8.22 : création de zones à partir de points : tessellation de Voronoï

8.7.2.3. De la ligne à la zone : créer la topologie

Il est possible de créer des zones à partir de lignes destinées à former le contour des zones. Celles-ci doivent alors former un graphe planaire fermé. L'opération s'effectue en deux temps :

- vérification de la planéarité et de la fermeture du graphe ;
- création des polygones.

Pour vérifier la planéarité du graphe, on teste les arcs deux à deux à la recherche d'une intersection non située sur une extrémité, après avoir éliminé les arcs ou portions d'arc en double. On teste la fermeture en comptant le nombre d'apparition

de chaque nœud dans l'ensemble des arcs. Chaque nœud doit apparaître en nombre pair. La création des polygones utilise l'algorithme vu au chapitre 7 pour la reconstitution de polygones à partir d'arcs.

8.7.2.4. De la zone à la zone : regrouper, éroder, dilater

On peut également créer une nouvelle relation zonale à partir des zones d'une autre relation, par regroupement de zones ou par modification de la géométrie des zones initiales en utilisation des algorithmes de morphologie mathématique (érosion, dilatation).

Le regroupement de zones se fait sur un attribut nominal de la relation initiale. Cet attribut devient une clé descriptive des objets de la relation résultante. La topologie doit donc être modifiée pour supprimer les arcs frontière de deux zones initiales de même valeur et recréer de nouvelles zones à partir de ces arcs. On prendra comme centroïde des nouvelles zones le centroïde de l'une des zones initiales de même valeur.

Le découpage en feuille pose ici un problème difficile à résoudre, car deux zones de feuilles différentes mais de même valeur pour l'attribut qualitatif choisi doivent être regroupées dans un même objet de la nouvelle relation. Cette opération fait donc perdre l'indexation géométrique, et la relation résultante ne contient plus qu'une seule feuille. Les arcs en double, présents dans deux feuilles distinctes mais séparant deux zones adjacentes de même valeur descriptive doivent être éliminés.

L'érosion et la dilatation sont des opérations d'un autre type. Elles modifient les contours des zones sans modifier les valeurs descriptives des objets. Le nombre d'objets reste le même (fig. 8.23).

Dans SAVANE, la réalisation de ces opérations utilise la représentation matricielle des objets. Après érosion ou dilatation des objets dans l'image, cette image est vectorisée pour créer les contours des nouvelles zones. Voici par exemple la méthode de dilatation d'une image matricielle. On cherche, pour chaque pixel susceptible d'être rempli (les pixels qui ne correspondent à aucune zone initiale) la zone initiale la plus proche, en limitant la recherche à la distance de dilatation. On peut limiter la dilatation à un masque. Pour améliorer les performances de l'algorithme, on construit toujours un masque autour des objets à dilater, avec la distance de dilatation : seuls les pixels de ce masque sont susceptibles d'être remplis. La procédure de dilatation est implémentée dans la procédure globale `DilaterImage()` dont le code est décrit dans l'annexe A.2.2.4. (pour éroder, on inverse le processus : les pixels susceptibles d'être remplis sont ceux qui appartiennent à une zone initiale, et on dilate l'espace vide).

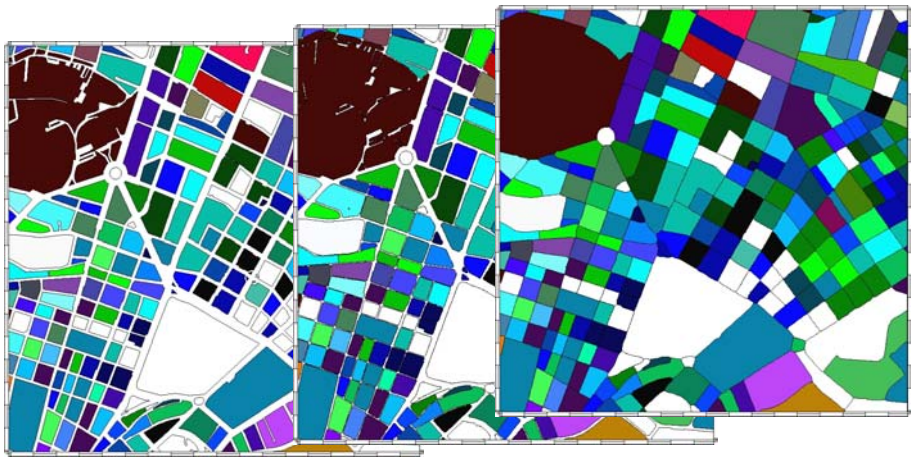


fig. 8.23 : deux dilatations à partir des objets initiaux (5 m, 20 m). L'objectif est ici de diminuer ou de remplir l'espace interstitiel en ville

On peut également utiliser cet algorithme pour créer des zones à partir de lignes ou de points. Chaque objet donne le jour à une zone de la nouvelle relation (fig. 8.24).

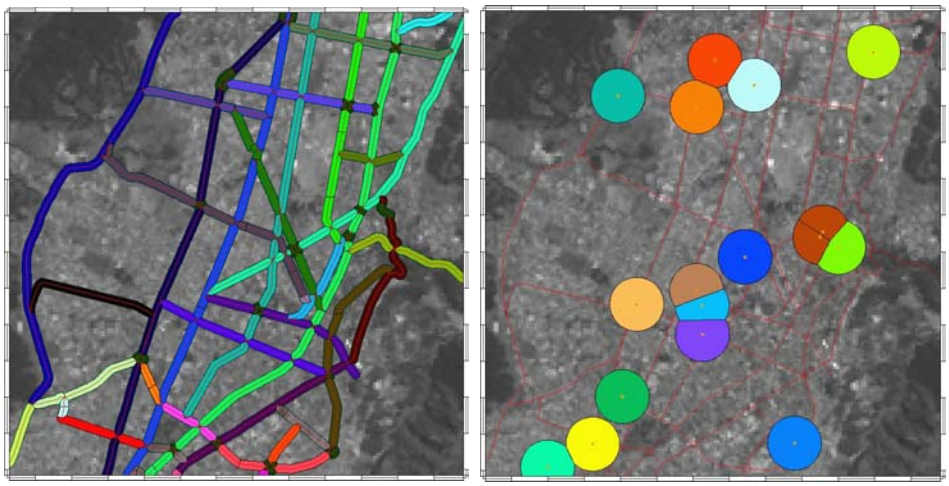


fig. 8.24 : création de zones par dilatation à partir de lignes ou de points

8.7.3. Vers la ligne

On peut créer de nouveaux objets de type ligne à partir de zones, par squelettisation, ou à partir de pixels, par vectorisation. On peut également considérer les frontières des zones comme des objets de type ligne, en affectant à chaque frontière une valeur déduite des valeurs des deux zones.

8.7.3.1. Squelettisation de zones

La squelettisation d'une zone est l'opération qui permet de réduire la surface de la zone à la ligne la plus représentative de la forme de cette surface. Le squelette d'une zone peut être obtenu à partir de son contour : le squelette est l'ensemble des points de la zone qui ont au moins un point du contour à la même distance que le point du contour le plus proche. Il peut également être obtenu à partir d'un attribut qualitatif d'une relation mosaïque : on emploie alors des méthodes itératives de morphologie mathématique (érosions successives jusqu'à stabilité, par exemple).

8.7.3.2. Vectorisation et courbes d'isovaleurs

La vectorisation permet d'obtenir des lignes à partir d'un attribut d'une relation de type zone ou mosaïque, par suivi des différences de l'attribut. Si l'attribut est numérique, il est automatiquement classé en intervalles égaux : les objets correspondent alors à des courbes d'isovaleurs pour cet attribut. L'algorithme de vectorisation est toujours le même, mais les objets créés ne sont plus considérés comme des contours de zones mais comme des lignes. Si la relation initiale est de type zone, on utilise l'image *raster* de l'attribut de la relation.

La création de courbes d'isovaleurs à partir d'un attribut créé par interpolation permet de tester l'idempotence de la composition des deux opérations (fig. 8.25) :

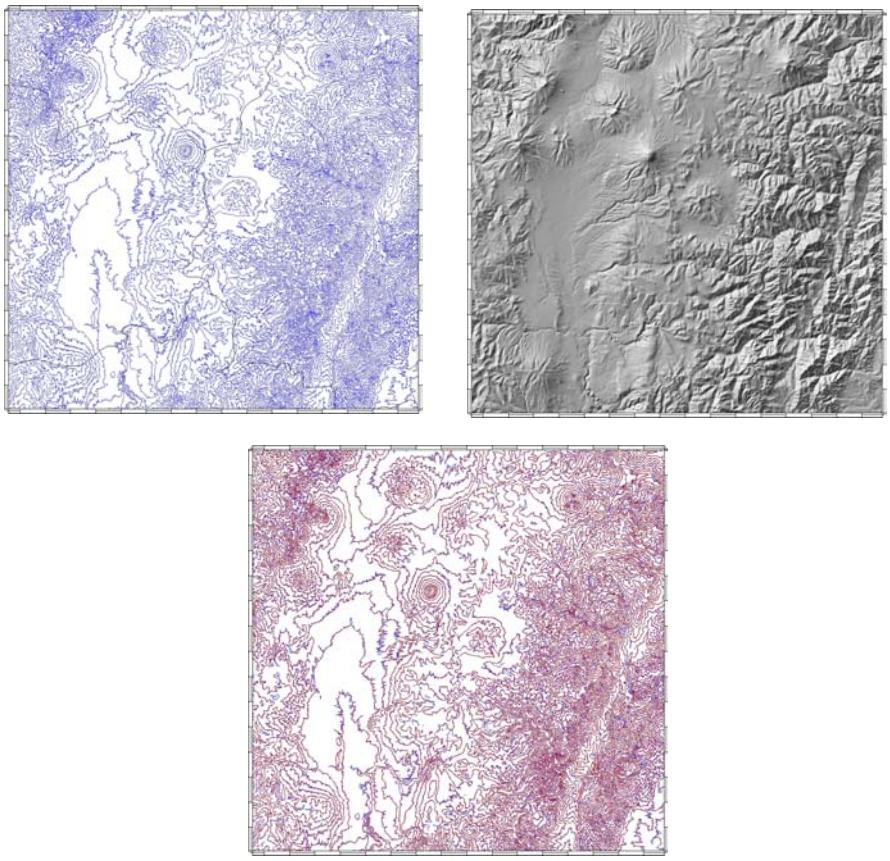


fig. 8.25 : Courbes de niveau initiales, interpolation, puis création de courbes de niveau par vectorisation (en rouge, superposées aux courbes initiales en bleu)

8.7.4. Vers le point

Le type point est le plus simple de tous les types de relations localisées. De plus, les types zone et ligne conservent pour chaque objet un centroïde qui permet de considérer ces objets comme des points dans de nombreuses opérations de dessin ou de géo-agrégation. Il n'y a donc pas lieu de créer une nouvelle relation de type point lorsque l'on souhaite utiliser des objets de type zone ou ligne dans ces opérations. Nous avons déjà vu comment trouver un centroïde à partir d'une zone définie par ses arcs frontières, ou à partir de l'arc d'une ligne, lors de la création d'objets de ce type.

Par contre, il peut être intéressant de définir des objets de type point à partir de la localisation d'objets d'une relation suivant la valeur de l'un des attributs des objets, de manière à avoir une vision synthétique d'un phénomène ou représenter un semis de points par un seul point.

La gestion des réseaux impose souvent de différencier ligne et nœud, et de traiter les deux collections séparément, sans oublier néanmoins les relations topologiques entre les deux entités.

8.7.4.1. Centre d'un semis de points

Cette procédure de création de relation ponctuelle permet de mettre en œuvre les opérations de représentation d'un semis de points par un point remarquable, opérations que nous avons évoqué au début de ce chapitre. On peut ainsi créer un point comme point médian ou comme point moyen d'un ensemble de points ou de centroïdes (fig. 8.26). On peut également pondérer le calcul du point moyen par la valeur d'un attribut numérique.

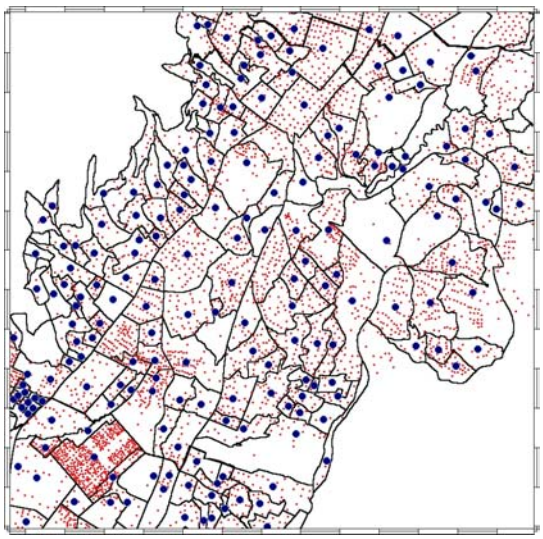


fig. 8.26 : Création de points centraux (en bleu) à partir d'un semis de points (provenant des îlots, en rouge) et de l'appartenance au quartier (attribut créé par géo-appartenance)

8.7.4.2. De la ligne aux points : l'utilisation des nœuds

La création d'une relation ponctuelle à partir des extrémités de lignes permet de séparer les nœuds des arcs. Cette séparation est importante car, s'il est possible de gérer un réseau avec une relation de type ligne, certaines propriétés du réseau doivent être exprimées sur les nœuds et non sur les arcs. Ces propriétés seront donc

calculées à partir de la localisation des nœuds regroupés dans une relation ponctuelle. Par exemple, la création d'une relation ponctuelle à partir des extrémités des objets d'une relation de type ligne s'accompagne de la création d'un attribut indiquant pour chaque nœud le nombre d'occurrence de l'extrémité dans chaque objet ligne de la relation initiale (fig. 8.27).

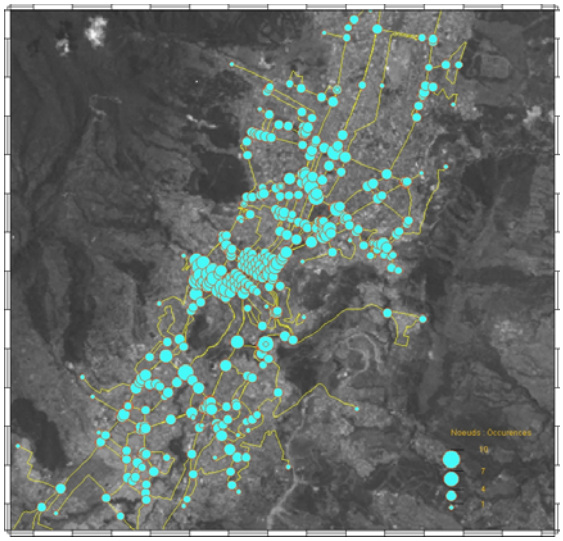


fig. 8.27 : occurrence des nœuds dans un réseau de lignes de bus

8.8. Conclusion

Avec l'analyse spatiale, le SIG quitte le strict domaine du système de gestion de données et devient également programme d'application, puisque, comme nous l'avions vu au chapitre 5, les seules opérations de gestion relationnelle ne sauraient permettre de répondre aux problèmes liés à la schématisation de l'espace. Dès lors, nous ne pouvons que développer et présenter ici quelques méthodes d'application des données localisées, tant les champs d'application sont vastes et variés. L'analyse spatiale s'adresse plus particulièrement aux géographes, mais la démarche pourrait être la même quelque soit le domaine d'application envisagé, comme l'hydrologie, la géologie, la géomorphologie, l'urbanisme, la gestion du territoire, etc. Le SIG s'enrichit non seulement de structures de données propres à certaines applications, il s'enrichit également de méthodes spécifiques pour devenir un véritable système objet. Le développement d'un système comme SAVANE peut donc se poursuivre selon trois voies : l'implémentation de nouvelles opérations d'analyse ou de simulation directement dans le module d'exploitation, l'appel de fonctions externes à partir d'un schéma de méthodes développées par les concepteurs d'objets ou de

collections d'objets et spécialisées à un domaine d'application particulier (l'appel de fonction externe est gérée directement en tant qu'opération élémentaire de la requête), et le développement d'une librairie permettant aux développeurs d'applications d'intégrer l'ensemble des méthodes de gestion de données dans le programme d'application.

Bibliographie

[BEG 94] BEGUIN M., PUMAIN D., *La représentation des données géographiques*, 1994 Armand Colin, « Cursus », Paris

[HAG 73] HAGGETT P., *L'analyse spatiale en géographie humaine*, 1973, Armand Colin, Paris

[LAU 93] LAURINI R., MILLERET-RAFFORT F., *Les bases de données en géomatique*, 1993, Hermes, Paris

[MON 82] MONMONIER M., *Computer-Assisted Cartography, Principles and Prospects*, 1982, Prentice-Hall

[O'RO 93] O'ROURKE J., *Computational Geometry in C*, 1993, Cambridge University Press

[PAR 97] PARKER J.R., *Algorithms for Image Processing and Computer Vision*, 1997, Wiley Computer Publishing

[PUM 97] PUMAIN D., SAINT-JULIEN T., *L'analyse spatiale*, 1997, Armand Colin, « Cursus », Paris

[RIM 90] RIMBERT S., *Carto-graphies*, 1990, Hermès, Paris

[SAN 89] SANDERS L., *L'analyse des données appliquée à la géographie*, 1989, RECLUS, Montpellier

[VOI 95] VOIRON C., *Analyse spatiale et analyse d'images*, 1995, RECLUS, Montpellier

Chapitre 9

Représenter

Ce dernier chapitre traite de la représentation des données géographiques dans le système SAVANE. Après la modélisation, la saisie, le traitement, l'analyse, c'est la dernière étape d'un long processus et l'un des objectifs du système à construire : pour être complet, un système d'information géographique doit intégrer des fonctionnalités classiques d'édition graphique ainsi que des capacités spécialisées dans la représentation cartographique, de manière à présenter les résultats d'une requête. Cette dernière étape est fondamentale et passionnante, car la représentation cartographique est le moyen privilégié pour présenter des données dès lors qu'elles sont localisées, et cette représentation correspond de nouveau une schématisation des objets et des phénomènes – contrairement aux bases de données classiques qui présentent les résultats sous forme de listes ou de formulaires. Nous n'allons ici que rappeler brièvement les principes de la cartographie ou de la sémiologie graphique pour décrire plus en détail le principe et l'implémentation des procédures de cartographie à partir des données gérées par le système SAVANE.

9.1. Les principes de la représentation cartographique

La représentation graphique est la transcription à l'aide de signes graphiques d'une information connue à l'aide d'un autre système de signes. La représentation graphique est une partie de la sémiologie, science qui traite de tous les systèmes de signe [BER 67]. Le système de signes qui nous intéresse ici ne concerne que ce qui est représentable sur un écran d'ordinateur ou sur une feuille plane de papier blanc, par tous les moyens graphiques disponibles. On traitera un peu à part l'intervention du mouvement et l'animation, car les lois graphiques du cinéma sont très différentes des lois graphiques de la cartographie. On traitera également à part l'intervention de

la troisième dimension et la représentation en perspective, car bien que J. Bertin ne la préconise pas pour visualiser des données quantitatives, elle s'avère être une aide conséquente à l'interprétation des photographies aériennes ou des images satellites.

9.1.1. Le langage cartographique

Le langage cartographique regroupe les moyens graphiques permettant l'utilisation de ces signes pour transmettre une information. C'est « une forme d'expression dont les signes graphiques élémentaires (le point, le trait, la tache) seraient l'alphabet, dont le vocabulaire est fait de variables visuelles et dont la syntaxe est définie par les règles de la perception visuelle. Les éléments du dessin constituent le signifiant, le contenu du message, dont le signifié, le contenu informatif, réside dans la correspondance établie entre les signes et l'information à transmettre. » [BEG 94].

Ce langage doit satisfaire à plusieurs objectifs : obéir aux règles générales de la perception visuelle, être universel, c'est-à-dire compréhensible par tous ceux qui ont les mêmes règles de perception visuelle, être cohérent en évitant la redondance, la surcharge ou l'ambiguïté. En combinant les diverses variables dont il dispose, le langage cartographique offre une très large gamme de possibilités pour représenter un phénomène, mais le cartographe, dans son choix d'une représentation, doit s'efforcer de garantir la clarté et cohérence du message. De nombreuses règles ont été établies en vue du respect de ces principes [BER 67].

Dans les limites que nous avons fixées plus haut, on considère que le système graphique dispose de sept variables :

- une variable d'implantation en deux dimensions : une tache visible peut représenter un point, une ligne, ou une zone ;
- six variables de représentation : taille, valeur, grain (ou trame), couleur, orientation, forme.

A partir d'objets géographiques, le cartographe réalise une carte en utilisant des signes élémentaires qu'il distribue selon une certaine implantation graphique. Une carte est constitué de *figurés*, ensemble de dessins visant à représenter ou schématiser le phénomène à cartographier. Le langage cartographique n'utilise que trois signes graphiques élémentaires pour dessiner un figuré : un point, un trait, un aplat de noir ou de couleur. L'implantation graphique est la manière de représenter un figuré à partir de ces signes élémentaires. Par exemple, l'implantation graphique est ponctuelle lorsque le figuré est un symbole. Il n'y a pas forcément de correspondance entre implantation géographique et implantation graphique : par exemple, il est courant de représenter une zone par un symbole (implantation ponctuelle), ou par un contour (implantation linéaire). Ce passage correspond au

pouvoir de modélisation et de schématisation de la cartographie dont nous avons déjà parlé au chapitre 3.

9.1.2. Les variables visuelles

Les variables visuelles de représentation (forme, taille, couleur, valeur, grain ou trame, orientation) permettent de faire varier les signes graphiques en fonction des valeurs des objets géographiques à schématiser. Elles sont utilisées pour représenter graphiquement, à partir d'objets localisés, des informations qualitatives, des variations quantitatives, des répartitions ordonnées, permettant ainsi de schématiser le phénomène que l'on cherche à exprimer. L'utilisation d'un ordinateur permet de simplifier considérablement les opérations visant à dessiner un figuré correspondant aux valeurs d'un objet géographique. C'est l'objet de la cartographie automatique.

9.1.2.1. La forme

La forme d'un élément graphique permet de transcrire une information qualitative. Il existe une infinité de formes disponibles, allant des symboles les plus simples – carrés, cercles, etc. - aux pictogrammes symboliques évocateurs de tel ou tel phénomène. Une carte, pour rester lisible, ne devra pas en contenir plus d'une dizaine. La forme est utilisée dans tous les types d'implantation graphique : elle est à la base des figurés ponctuels, et sert également à différencier des lignes de natures différentes (frontières, courbes topographiques, failles, etc.). La forme d'un aplat est en fait une texture obtenue à partir de la répétition d'une forme élémentaire. Quelle que soit l'implantation graphique, la forme ne doit jamais être employée pour représenter une variation quantitative.

9.1.2.2. La taille

La taille d'un figuré est définie par sa longueur, sa hauteur, sa surface ou son volume. Contrairement à la forme, la taille est utilisée pour représenter des variations quantitatives. De ce fait, elle permet non seulement de représenter des quantités mais également l'ordre des différents objets, car l'œil ordonne naturellement, du plus petit au plus grand, une forme en fonction des différences de longueur ou de surface, et d'autant plus facilement que l'étalement des tailles est également réparti entre la plus petite et la plus grande taille : on s'efforcera donc souvent de rendre la correspondance phénomène-taille proportionnelle, en agissant sur la variable géographique à représenter. La taille peut être utilisée pour les implantations graphiques ponctuelles et linéaires ; on peut faire varier l'épaisseur d'un segment de droite en fonction d'un attribut numérique ; on peut faire varier la surface d'un symbole proportionnellement à une variable.

9.1.2.3. La couleur

L'œil humain perçoit facilement de nombreuses teintes, et la couleur est abondamment utilisée en cartographie. Une couleur peut s'exprimer par la proportion de chacune des couleurs primaires (cyan, magenta, jaune), ou, sur un écran, par la proportion des couleurs fondamentales (rouge, vert, bleu). L'homme perçoit plus facilement les différences de couleur grâce aux variations de ton, d'intensité, et de saturation. La couleur permet de différencier aisément des variations aussi bien quantitatives que qualitatives, mais c'est la variable visuelle la plus utilisée pour représenter des différences qualitatives. On a d'ailleurs établi dans de nombreuses thématiques des palettes de couleurs standards pour représenter des modalités de la thématique (et géologie, en pédologie notamment). Par contre, la couleur seule ne suffit pas pour représenter un ordre, car il est impossible d'agir sur l'une des composantes de la couleur sans altérer visuellement l'ordre les deux autres. Dans un dégradé de couleur, l'ordre des couleurs ne correspond pas de façon intuitive à un ordre numérique. Néanmoins, on utilise parfois des dégradés de couleurs pour représenter des variations numériques, et l'on s'efforce alors de n'utiliser que quelques tons différents.

Si la carte est destinée à l'impression, il faut être très attentif aux couleurs employées. Le rendu d'une couleur est toujours différent sur un écran et en impression. On choisira les couleurs en fonction de la technique employée pour l'impression, et le plus souvent à partir d'une charte imprimée fournie par l'imprimeur et donnant le rendu des couleurs en fonction de leurs niveaux dans les trois couleurs primaires (benday).

9.1.2.4. La valeur

La valeur correspond au rapport entre les quantités de noir et de blanc perçues dans une surface donnée. La variation de valeur est la progression continue que l'œil perçoit dans la suite des gris qui s'échelonnent du noir au blanc. La valeur d'une surface colorée est désignée par le gris qui s'égalise en valeur avec cette surface. Cette variable visuelle est toujours ordonnée. Si l'œil peut distinguer un grand nombre de niveaux de gris, en pratique il faut limiter le nombre de niveaux à 6 ou 7. On utilise essentiellement cette variable pour représenter des valeurs qualitatives ordonnées sur des implantations zonales, et la surface ne doit pas être trop petite pour que l'œil puisse apprécier le rapport entre le blanc et le noir.

9.1.2.5. Grain, trame, texture

Le grain est la quantité de taches séparables dans une surface donnée. La tache est un motif, généralement simple : carré, disque, ligne. La variation de grain est la sensation qui résulte de la succession des réductions photographiques du semis de taches. Les réductions augmentent le nombre de taches mais ne font pas varier la

valeur de la surface. Toute valeur peut donner lieu à une variation de grain, mais c'est dans les valeurs moyennes que les paliers de grain sont les plus sensibles.

C'est en implantation zonale que le grain est le plus utilisé. Mais cette variable visuelle est d'un usage délicat, car il est difficile d'en maîtriser la perception : à valeur égale une trame à gros grain parait plus noire qu'une trame à petit grain qui sera perçue comme plus couvrante et plus grise.

9.1.2.6. Orientation

L'orientation est définie par l'angle que fait un figuré avec la verticale. Pour que la variation devienne significative, il faut pouvoir déceler des catégories d'orientation, ce qui en limite considérablement le nombre disponible. En général, on se limite à quatre directions : verticale, horizontale, et deux obliques à 45°.

9.1.3. La représentation cartographique

A partir d'objets géographiques, le cartographe dispose d'une grande liberté pour représenter ou schématiser ces objets sur une carte en utilisant figurés et variables visuelles. Il peut différencier implantation géographique et implantation graphique ; il peut choisir des variables visuelles en fonction des implantations et de la nature des informations à représenter.

L'utilisation de la cartographie automatique supprime tout travail de dessin manuel, mais augmente la liberté de choix graphiques. Ainsi, l'essor des SIG doit s'accompagner d'un effort de formation à la cartographique, puisque ces systèmes mettent à disposition de l'utilisateur à la fois des outils de traitement et de représentation de l'information géographique, avec de nombreuses possibilités de schématisation et de modélisation. Les SIG rapprochent géographie et cartographie, comme nous l'avons déjà souligné au chapitre 3, mais ce rapprochement ne peut être fructueux qu'à la condition du respect mutuel des règles imposées par les deux disciplines.

Généralement, le géographe-cartographe cherchera à conserver pour la représentation cartographique l'implantation géographique des objets, et ce d'autant plus qu'il cherchera à rendre compte directement de la modélisation initiale de la réalité. Par contre, cette correspondance sera souvent modifiée lorsque la carte rend compte d'un travail de synthèse ou de modélisation. Par exemple, l'échelle de la carte est un élément important de cette schématisation ; elle va de pair avec une simplification de l'information et une modélisation qui induit souvent une représentation des objets de type zone par des symboles ponctuels. L'implantation géographique zonale est la plus difficile à représenter telle quelle, sans en changer

l'implantation ; on ne dispose pas pour l'implantation graphique zonale de variables visuelles permettant facilement de transcrire la valeur absolue ; les différences de surface entre les objets induisent fréquemment des erreurs d'interprétation de la représentation. Elle sera donc souvent remplacée par une implantation graphique linéaire ou ponctuelle. Le choix des variables visuelles est fonction du type d'implantation et de la nature des variables à représenter, suivant les principes énumérés au paragraphe précédent. Pour résumer :

- les attributs qualitatifs sont représentés au moyen de variations de forme et de couleur (implantation ponctuelle et linéaire), de couleur et de texture (implantation zonale) ;

- l'ordre d'un attribut quantitatif est représenté par une variation de valeur, un dégradé de couleur, un variation de taille en implantation ponctuelle, une variation de taille ou un dégradé de couleur en implantation linéaire, une variation de valeur, un dégradé de couleur, une variation de grain en implantation zonale ;

- la valeur absolue d'un attribut quantitatif est représentée par une variation de taille en implantation ponctuelle ou linéaire.

9.1.4. Légendes et habillage

La transmission de l'information à l'aide du langage cartographique s'accompagne également de règles de mise en forme, d'une part pour représenter la correspondance entre variables visuelles et information descriptive (la légende), d'autre part pour décrire la position spatiale de l'espace représenté (amorces géographiques, rose des vents, indication de la projection, etc.). Enfin, il est courant d'ajouter des éléments graphiques pour faciliter la lecture de la carte ou de modifier la position de signes pour répondre à des règles générales de présentation.

9.2. La cartographie avec SAVANE : carte et cadres

L'ergonomie du module SAVANE du SIG est construite pour faciliter l'expression graphique du résultat d'une requête. Le document de base du logiciel est donc une carte, c'est-à-dire une feuille de papier avec un cadre géoréférencé destiné à contenir une représentation graphique d'une requête sur des données géographiques.

9.2.1. La carte

A l'entrée dans le module d'exploitation de SAVANE apparaît une feuille de papier sur l'écran, représentation graphique de l'unique instance de la classe `CCarte`. Cette feuille de papier est destinée à contenir la représentation graphique

des différents cadres géographiques, instances de la classe `CCadre`, chaque cadre correspondant à une requête sur la base de données comme nous l'avons vu depuis le chapitre 5. Une carte doit contenir au minimum un cadre, et peut en contenir jusqu'à dix. Elle est également destinée à contenir les objets graphiques dessinés par l'utilisateur (textes, logotypes, images, légende, dessins, etc.) (fig. 9.1).

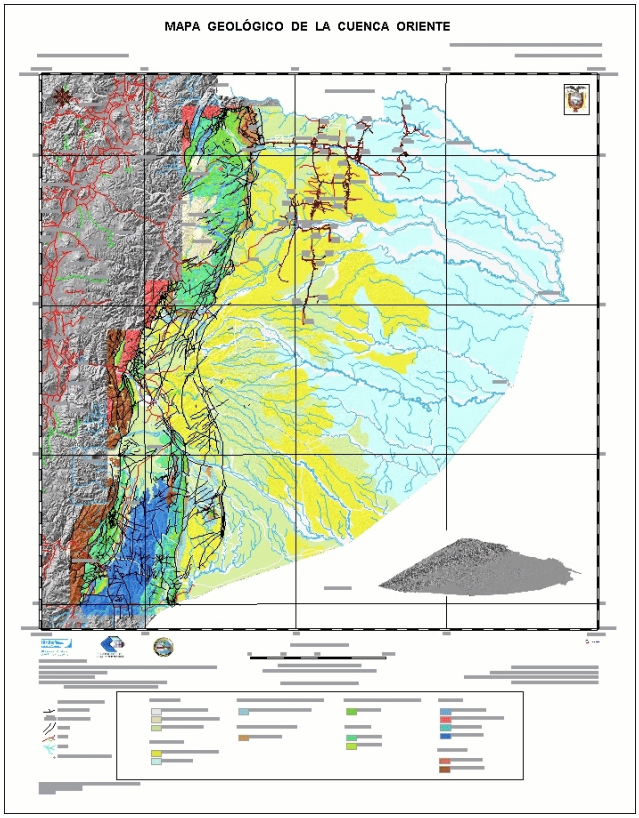


fig. 9.1 : une carte réalisée avec SAVANE

L'objet carte, instance de la classe `CCarte`, contient de nombreux paramètres graphiques permettant de la dessiner sur l'écran ou de l'imprimer : taille, orientation, couleur du fond, paramètres de visualisation sur l'écran (position et agrandissement) (A.2.4.1). Elle contient également un tableau donnant les pointeurs des objets graphiques qu'elle contient (de types dérivés du type `CODD`, A.2.4.2). L'utilisateur agit directement sur ces paramètres grâce à l'éditeur graphique, conçu comme un logiciel de dessin graphique : la fenêtre principale du module SAVANE correspond à un éditeur graphique traditionnel permettant de dessiner sur une feuille de papier avec des outils de dessin.. Il a donc à sa disposition des fonctions de

dessin tout à fait classiques, qui lui permettent de se déplacer dans la page et d'y dessiner des figurés. Chaque objet de dessin correspond à une classe particulière dérivée de `CODD`, et contenant les paramètres et les méthodes permettant de le dessiner ou de l'imprimer (A.2.4.2.). Les objets peuvent être groupés, et un groupe peut être sélectionné par l'utilisateur pour modifier les paramètres des objets graphiques. Nous avons construit deux classes permettant de manipuler groupes et sélection sur écran (`CGroupe`, `CSelection`, A.2.4.3.). Les cadres sont des objets graphiques plus complexes que nous allons décrire dans la paragraphe suivant, mais ils sont manipulés grâce à l'éditeur graphique comme les autres objets graphiques (sélection, déplacement, duplication, modification).

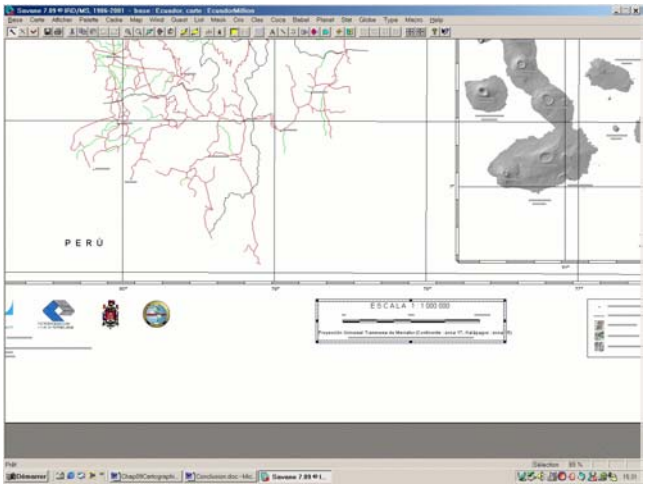


fig. 9.2 : la fenêtre principale de SAVANE est un éditeur graphique

Les légendes et les échelles graphiques proviennent des cadres d'interrogation. Mais une fois posés dans la carte, ces éléments sont des objets graphiques de type `CODD` comme les autres, et peuvent être manipulés grâce à l'éditeur graphique, contrairement aux figurés qui proviennent des objets géographiques, comme nous allons le décrire dans le paragraphe suivant.

9.2.2. Les cadres

Un cadre peut être vu comme une fenêtre sur la base de données contenant à un moment donné, un état temporaire de la base correspondant à une interrogation et aux opérations effectuées (calculs, création de nouvelles variables, croisement et jointure d'objets, etc.). L'état temporaire de la base lié au cadre est conservé avec la carte qui contient le cadre, ainsi que la macro-commande correspondant à toutes les

opérations effectuées dans ce cadre. En plus de la requête et de l'état temporaire de la base, au cadre sont attachés des paramètres qui lui sont propres et qui permettent de le dessiner dans la carte (espace géographique visualisé, projection géographique, échelle de tracé, visualisation des amorces géographiques ou de projection, type de dessin pour le bord, position dans la carte) (A.2.4.4.). Un dialogue permet de choisir l'ensemble de ces paramètres (fig. 9.3). L'échelle de tracé correspond à la définition exacte de l'échelle en cartographie, à savoir le facteur de réduction entre coordonnées de projection et coordonnées de dessin. La taille du cadre dans la carte est donc fonction de l'espace géographique visualisé et de l'échelle. L'espace géographique visualisé peut être différent de la fenêtre géographique de travail correspondant à la requête (de type `CWind`). L'utilisateur peut modifier directement l'espace géographique sur l'écran en modifiant la position des bords du cadre.

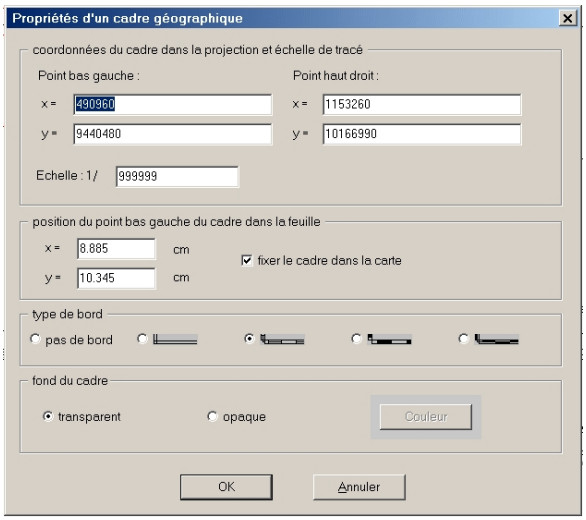


fig. 9.3 : dialogue de propriétés d'un cadre géographique

L'échelle graphique dépend du cadre auquel elle est attachée, puisqu'elle dépend de l'échelle de représentation du cadre. Le dialogue de création d'une échelle graphique donne le choix de plusieurs types de dessin. Comme les signes graphiques constituant une échelle graphique n'ont pas vocation à être modifiés, nous avons créé une classe `CODD_ECHELLE` dérivée de `CODD` pour représenter ce type d'objet graphique (A.2.4.2.).

La représentation cartographique des objets des relations contenues dans un cadre utilise les principes de la sémiologie graphique que nous avons énoncés au début de ce chapitre. Pour chaque cadre, les paramètres permettant d'associer implantation graphique et variables visuelles, à partir des relations et des attributs à

représenter, sont conservés dans un objet de type `CCartExplorateur`, que nous appelons *explorateur cartographique*, et auquel nous consacrerons le paragraphe 9.3., après avoir indiqué comment est traité la couleur dans le système SAVANE.

9.2.3. Couleurs et palettes

Une couleur peut être décrite grâce aux proportions de couleurs fondamentales qu'elle contient. En informatique, on utilise habituellement des niveaux de rouge, vert, bleu, chaque niveau étant codé de 0 à 255 (de manière à occuper un octet en mémoire). On a ainsi la possibilité de coder environ 16 millions de couleurs, même si certaines couleurs peuvent difficilement être codée de cette manière (notamment les teintes d'or ou d'argent, ou les couleurs fluorescentes). Dans le codage RVB, le niveau de chaque couleur primaire correspond directement à la quantité d'électrons émise par le tube cathodique dans chacune des composantes de couleur. Pour du papier, on résonne à l'inverse, puisque intervient une quantité d'encre dans chacune des composantes. On exprime alors une couleur par les proportions de cyan, magenta, jaune qu'elle contient. Parfois, on sépare également le noir, de manière à séparer teinte et saturation, et améliorer l'impression en quadrichromie (une couleur s'exprime alors en proportions de cyan, magenta, jaune, noir qu'elle contient).

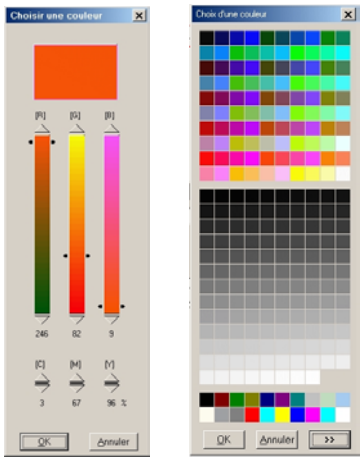


fig. 9.4 : dialogues de définition d'une couleur et de choix dans une palette

Dans le système SAVANE, une couleur est définie par ses niveaux de rouge, vert, bleu, ou ses proportions de cyan, magenta, jaune. La définition d'une couleur s'effectue directement sur l'écran grâce à un dialogue (fig. 9.4). Pour éviter d'avoir à redéfinir une couleur chaque fois que l'on a besoin de définir la couleur d'un élément, le système SAVANE utilise des palettes, c'est-à-dire des ensembles de

couleurs prédéfinies et conservées par le système (classe `CPaletteSavane`, A.2.4.8.). L'utilisateur choisit donc une couleur dans une palette qu'il a constitué auparavant ou que le système lui propose (palettes prédéfinies). On peut ainsi créer une fois pour toute des ensembles de couleurs (des palettes) correspondant à une thématique particulière (topographie, géologie, pédologie, etc.). L'utilisation d'une palette est d'ailleurs nécessaire dans les opérations de représentation par dégradé, puisque c'est le système qui va automatiquement affecter la couleur du dégradé en fonction de la valeur de la variable descriptive à représenter. Il suffit alors de modifier la palette de couleurs pour modifier la correspondance valeur numérique-couleur. Même chose pour une image codée : il serait très fastidieux d'avoir à définir la couleur correspondant à chaque valeur d'une image (par exemple, une image satellite codée de 0 à 255). On utilise alors une palette et une fonction de correspondance linéaire entre code de l'image et indice de la couleur dans la palette.

Le système SAVANE utilise deux palettes distinctes pour dessiner des dégradés dans une carte : une palette dite thématique, et une palette dite imagerie. Les palettes thématiques contiennent 100 couleurs, les palettes imageries 128 couleurs. Il est donc possible de dessiner simultanément deux dégradés. Par exemple, pour dessiner une photographie aérienne codée sur 256 niveaux, on peut utiliser une palette de gris et une fonction de correspondance entre chaque niveau de l'image et la palette imagerie (par défaut, f(x)=x/2). Pour augmenter le contraste de l'image, on peut soit modifier la palette de gris (en restreignant la variation de gris), soit modifier la fonction de correspondance (f(x)=(b-a)x/255 + a, pour une variation entre la couleur d'indice a et la couleur d'indice b de la palette initiale). La carte conserve ses palettes et les charge automatiquement lors de son ouverture.

Certains ordinateurs sont limités dans l'utilisation de la couleur sur écran : il est encore courant, dans certaines résolutions d'écran, de ne pouvoir afficher que 256 couleurs simultanément. Dans ce cas, SAVANE utilise les couleurs des deux palettes de la carte, plus les couleurs réservées par le système d'exploitation (20 couleurs). Si la couleur d'un élément graphique ne fait pas partie des couleurs des palettes courantes ou des couleurs système, l'élément est dessinée sur l'écran avec la couleur la plus proche (on utilise pour cela une fonction de distance euclidienne sur les couleurs en utilisant les trois composantes comme des coordonnées).

9.3. L'explorateur cartographique

9.3.1. Principes de l'explorateur cartographique

L'explorateur cartographique regroupe l'ensemble des informations permettant de dessiner dans un cadre à partir des objets géographiques des relations qu'il contient. A chaque cadre est donc associé un explorateur cartographique. Il conserve

le choix des relations à dessiner, leur ordre de dessin, et pour chaque relation choisie, le choix de l'implantation graphique, des attributs et des variables visuelles associées. De nombreux choix sont possibles, et les combinaisons possibles sont quasiment illimitées. Une même relation peut être choisie plusieurs fois, et être dessinée dans le cadre avec des paramètres différents. Elle peut également ne pas être tracée mais néanmoins conserver l'ensemble de ses paramètres de dessin. A chaque implantation graphique correspond une classe regroupant les paramètres de dessin propre à cette implantation graphique (`CDessinZone`, `CDessinLigne`, `CDessinSymbole`, `CDessinPixel`, A.4.2.6.). Le type de la relation détermine les choix possibles pour l'implantation graphique. Ensuite, pour chaque type d'implantation graphique choisi, l'utilisateur indique l'attribut à représenter et la ou les variables visuelles associées à cet attribut. Nous allons décrire les possibilités et les paramètres correspondant à chaque type d'implantation graphique. Pour chaque relation choisie, l'utilisateur accède aux paramètres de dessin grâce à un bouton « propriétés » du dialogue de l'explorateur (fig. 9.5).

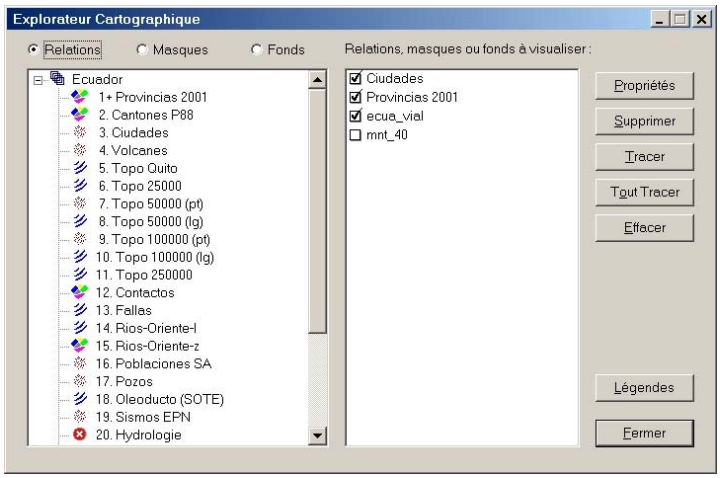


fig. 9.5 : le dialogue principal de l'explorateur cartographique

L'explorateur cartographique donne également la possibilité de tracer des masques, des fonds graphiques, ou des documents de saisie au format SAVEDIT. Ces éléments ne contiennent aucune information descriptive : les paramètres de dessin choisis seront appliqués à tous les éléments (`CDessinFond`, A.4.2.6.).

Les figurés dessinés à partir des objets ne sont pas des objets de dessin au même titre que les objets dessinés directement sur l'écran grâce à l'éditeur graphique. L'utilisateur ne peut les sélectionner ou les modifier indépendamment les uns des autres. Ils sont attachés au cadre et se déplacent automatiquement avec lui. Faire une carte avec SAVANE utilise donc deux outils : d'une part les cadres et leur

explorateur cartographique, d'autre part l'éditeur graphique qui est actif en permanence dans la fenêtre principale du programme.

9.3.2. L'implantation graphique ponctuelle

L'implantation graphique ponctuelle peut être utilisée par tout type de relation, sauf pour le type mosaïque. En implantation ponctuelle, plusieurs types de figurés peuvent être dessinés : un symbole, un diagramme sectoriel, un texte. Pour chaque objet de la relation, l'élément dessiné utilise la localisation du centroïde de l'objet (rappelons que pour une ligne, le centroïde est situé automatiquement au milieu de la ligne ; pour une zone, il est choisi lors de la saisie graphique ; pour un point, il correspond à la position de l'objet lui-même).

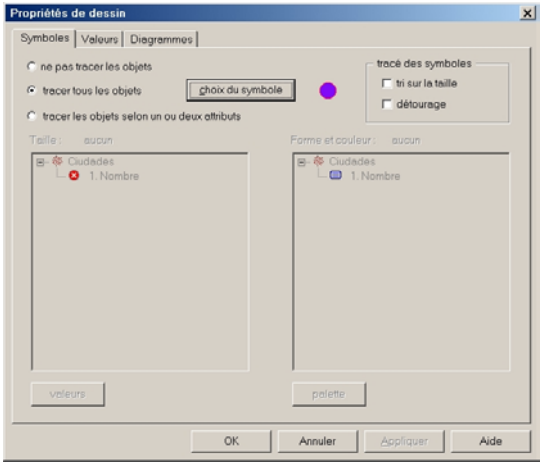


fig. 9.6 : les trois possibilités de figurés en implantation ponctuelle : symboles, valeurs, diagrammes sectoriels

La représentation par symbole permet d'associer un symbole à chaque objet. La forme du symbole peut être la même pour tous les objets, ou varier en fonction d'une variable qualitative. L'utilisateur dispose pour cela d'une palette de formes de symboles prédéfinies. Lorsqu'il souhaite faire varier la forme en fonction d'un attribut nominal, il établit une correspondance entre chaque valeur à représenter et une forme de symbole. La couleur de l'intérieur du symbole peut également varier selon une variable qualitative. La taille du symbole peut être la même pour tous les objets ou varier selon une variable numérique. La couleur et l'épaisseur du trait sont choisies par le cartographe et sont les mêmes pour tous les objets de la relation. La représentation par symbole permet donc d'utiliser simultanément trois variables visuelles : forme, taille, couleur, suivant au maximum deux variables descriptives

distinctes. Le centre du symbole est toujours situé sur le centroïde de l'objet. La surface totale des symboles représentés ne doit pas dépasser 20 à 30 % de la surface totale du cadre [BER 77].

fig. 9.7 : la représentation par symbole selon deux caractères

La représentation par diagramme permet d'associer un diagramme sectoriel (camembert ou histogramme) à chaque objet, en fonction des valeurs de plusieurs attributs numériques. On peut ainsi représenter pour chaque objet les proportions relatives de chacun des attributs numériques (fig. 9.8).

La représentation de texte permet de dessiner la valeur d'un attribut pour chaque objet. L'attribut peut être numérique ou nominal. Il est possible de choisir la

position de la chaîne de caractère par rapport au centroïde, mais de façon identique pour tous les objets. Comme nous l'avons déjà mentionné, les figurés dépendent de l'explorateur cartographique et ne peuvent être modifiés « à la main » directement par l'utilisateur. Mais il est courant d'avoir à déplacer un texte lorsque le placement automatique calculé par l'explorateur cartographique n'est pas satisfaisant. Pour cela, il est possible de « détacher » les textes de l'explorateur et de les transformer en éléments de dessin de type `CODD`. L'utilisateur pourra alors sélectionner un texte pour le déplacer si nécessaire, mais ce texte ne sera plus automatiquement attaché au cadre.

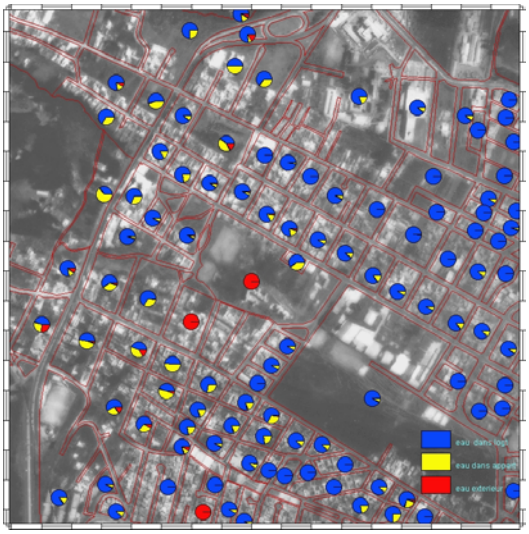


fig. 9.8 : la représentation par diagramme sectoriel suivant trois attributs descriptifs

La classe `CDessinSymbole` permet de conserver l'ensemble des paramètres de dessin en implantation ponctuelle (A.2.4.6). Elle contient notamment une procédure de tracé par type de représentation (`TracerSymbole()`, `TracerDiagramme()`, `TraceValeurs()`).

9.3.3. L'implantation graphique zonale

L'implantation graphique zonale ne peut être choisie que pour les relations de type zone. Elle apparaît dans le dialogue des propriétés de tracé d'une relation de type zone sous l'onglet « Zone ». Il est alors possible de choisir un attribut pour tracer l'intérieur des zones (fig. 9.9). Le tracé des contours relève de l'implantation graphique linéaire que nous verrons au paragraphe suivant.

Si l'attribut choisi est nominal, il faut établir la correspondance entre chaque valeur à représenter et la couleur, la valeur, le grain associé. On peut donc agir simultanément sur ces trois variables visuelles, grâce au dialogue de constitution de ce que nous appelons une palette de valeur (association entre des valeurs nominales, et des couleurs et des trames). L'utilisateur dispose pour cela de la palette de couleur courante et d'une palette de trames correspondant à des variations de forme, de valeur ou de grain d'un aplat.

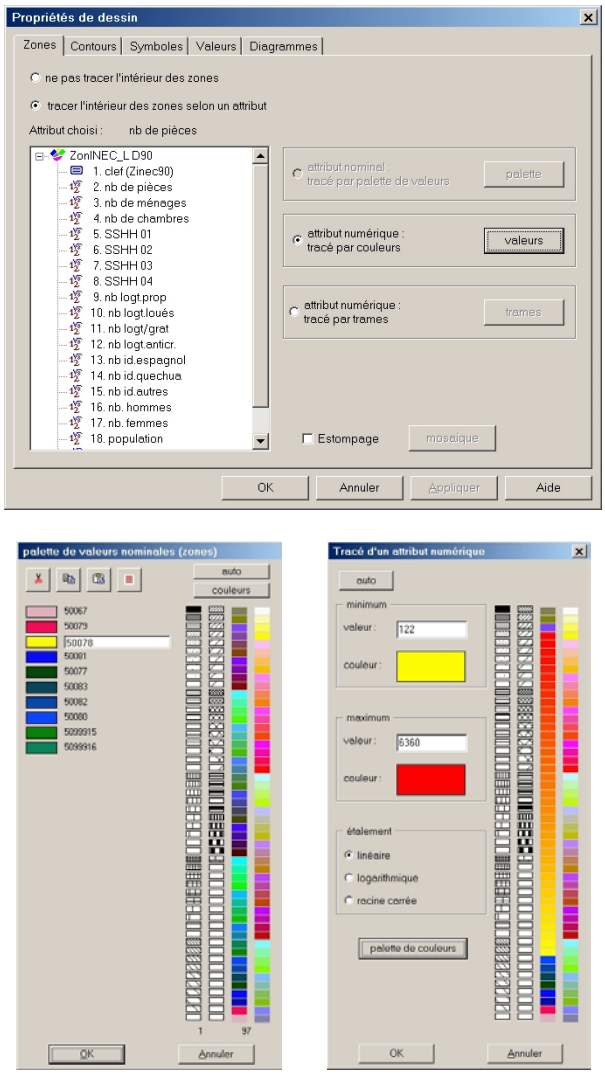


fig. 9.9 : les dialogues pour le tracé en implantation zonale

Si l'attribut choisi est numérique, on peut faire varier la couleur ou la valeur en fonction de l'attribut. Pour faire varier la couleur, l'utilisateur établit une correspondance entre les valeurs minimum et maximum à représenter et deux couleurs de la palette, ainsi qu'une fonction d'étalement (linéaire, logarithmique, exponentielle). Le système affecte alors à chaque objet, en fonction de la valeur de l'attribut, la couleur de la palette correspondant au rang calculé en utilisant la fonction d'étalement. La valeur, correspondant à la trame choisie, est alors la même pour tous les objets. Pour faire varier la valeur, il établit une correspondance entre les valeurs minimum et maximum à représenter et deux valeurs de trames. Le système calcule de la même manière pour chaque objet la trame correspondante.

Une option particulière permet d'estomper la couleur en fonction de la valeur d'une relation de type mosaïque. On fait alors varier l'intensité de la couleur (sans changer le ton), en fonction de la valeur de chaque pixel de la mosaïque. Ce type de tracé correspond en fait au tracé d'une mosaïque, puisque l'on trace finalement les pixels et non les zones. La mosaïque est le plus souvent un modèle numérique de terrain, et l'estompage est calculé en fonction de la quantité de lumière réfléchie par chaque pixel, à partir d'une source lumineuse.

La classe `CDessinZone` permet de stocker l'ensemble des paramètres de dessin en implantation zonale (A.2.4.6).

9.3.4. L'implantation graphique linéaire

L'implantation graphique linéaire ne peut être choisie que pour les relations de type zone ou ligne. Pour les zones, elle correspond au tracé des contours des objets. Pour les lignes, elle correspond au tracé des objets eux-mêmes.

Lorsque la relation est de type zone, on peut tracer tous les arcs avec les mêmes paramètres (couleur, épaisseur, forme), ou différencier les paramètres en fonction de l'égalité d'un attribut pour les deux zones adjacentes. L'attribut choisi peut être nominal ou numérique.

Lorsque la relation est de type ligne, on peut agir directement sur la couleur, l'épaisseur, et la forme de chaque arc. On peut donc faire varier simultanément trois variables visuelles : couleur, valeur, forme. Le principe est le même que pour le tracé de symboles : la couleur et la forme peuvent varier en fonction d'un attribut nominal (on établit alors une palette de valeurs nominales), et l'épaisseur du trait peut varier en fonction d'un attribut numérique. L'utilisateur dispose d'une palette de formes pour les types de ligne.

Lorsque la relation est de type ligne, une option supplémentaire permet de tracer les lignes comme des courbes de niveaux, en différenciant les paramètres de tracé (couleur, épaisseur) en fonction d'un attribut numérique.

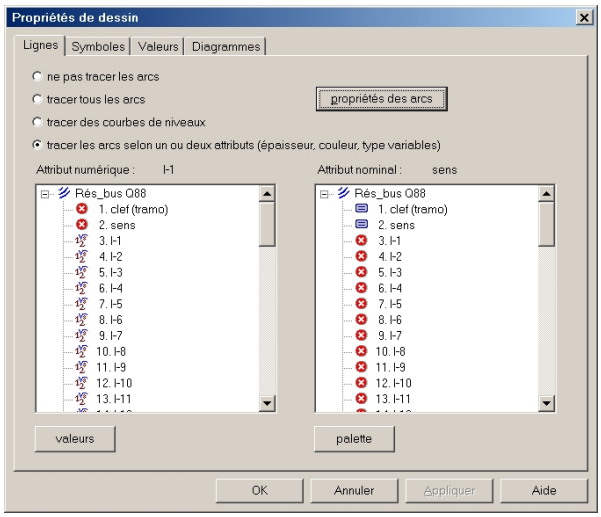


fig. 9.10 : propriétés de dessin pour les relations de type ligne

La classe `CDessinLigne` permet de stocker l'ensemble des paramètres de dessin en implantation linéaire (A.2.4.6).

9.3.5. La représentation des images

La représentation des images correspond à une implantation zonale, chaque pixel pouvant être considéré comme une zone. Mais il est très rare de vouloir tracer les contours des pixels, ou de vouloir représenter chaque pixel d'une mosaïque par un symbole. De plus, certaines méthodes de visualisation utilisent l'agencement des pixels, et sont donc propre au type mosaïque. Enfin, seul le type mosaïque utilise le type d'attribut RVB, qui conserve une couleur absolue pour chaque objet. Nous avons donc séparé les procédures de tracé des relations de type mosaïque des procédures de tracé des relations de type zone.

Le tracé d'une relation de type mosaïque ne comporte donc que le choix de l'implantation zonale. Le dialogue ne comporte pas de page correspondant au tracé par symbole ou par contour.

Le principe du tracé d'un attribut nominal est le même que pour les zones : il faut établir une palette de valeurs donnant la correspondance entre les valeurs nominales à représenter et la couleur et la trame.

Si l'attribut choisi est de type numérique, il peut être représenté par valeur, comme pour les zones. On établit alors une correspondance entre les valeurs à représenter et les couleurs à dessiner, comme pour les zones.

Un attribut numérique peut également être représenté par illumination : l'indice de la couleur dans la palette de couleur est alors calculé en fonction de la quantité de lumière émise par le pixel. Cette quantité de lumière dépend de la position de la source lumineuse et de la normale au pixel, calculée en fonction des pixels voisins (sur une matrice 3x3). C'est ainsi que l'on représente en général les modèles numériques de terrain, c'est-à-dire l'attribut « altitude » d'une mosaïque calculée à partir de courbes de niveaux.

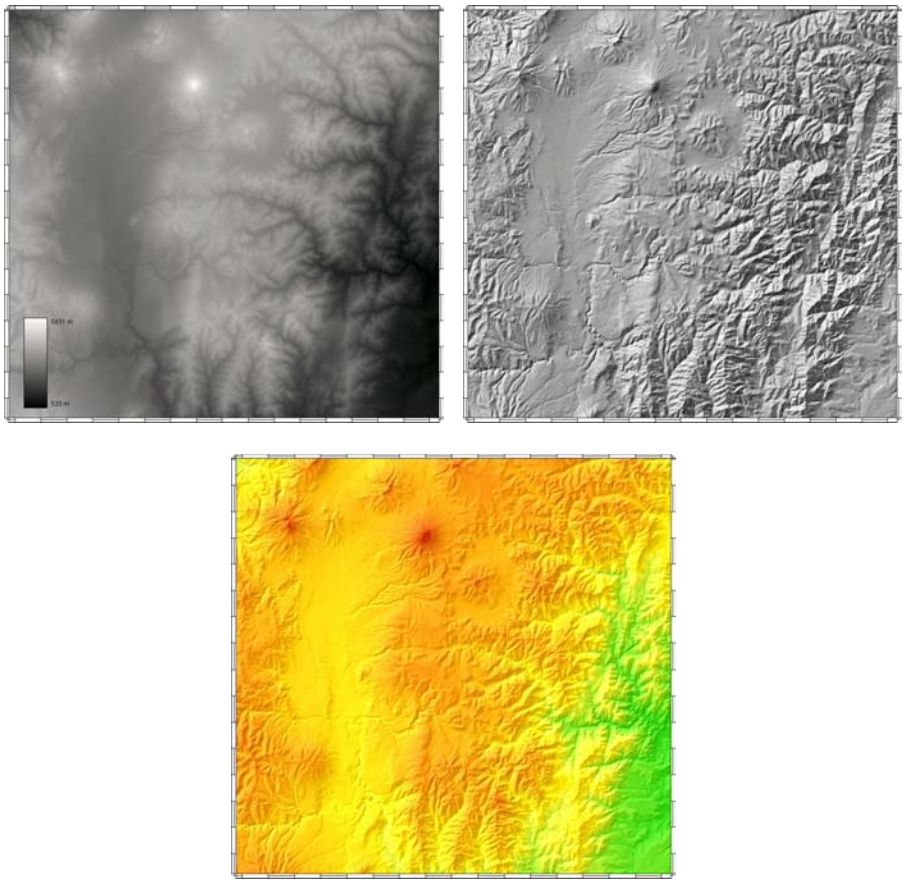


fig. 9.11 : représentation d'un mnt par valeur, par illumination, par estompage (combinaison de valeur et illumination)

La classe `CDessinPixel` permet de stocker l'ensemble des paramètres de dessin en implantation de type pixel (A.2.4.6).

9.3.6. Masques, fonds graphiques, documents digitalisés

L'explorateur cartographique donne également la possibilité de tracer dans un cadre des masques, des fonds graphiques, ou des documents digitalisés.

Un masque n'est relié à aucun attribut : c'est seulement la définition d'un domaine de l'espace. Pour le représenter, on peut donc jouer sur le figuré du contour, de l'intérieur ou de l'extérieur. On peut donc choisir la couleur, la forme et l'épaisseur du contour ; la couleur et la trame de l'intérieur ou de l'extérieur.

L'explorateur cartographique donne également la possibilité de tracer un document graphique géo-référencé, au format DXF. Les paramètres graphiques ne peuvent être modifiés dans SAVANE : le document est tracé tel qu'il a été constitué à l'extérieur du système. Il s'adapte à la projection géographique du cadre s'il contient un fichier décrivant sa propre projection géographique.

9.3.7. Les procédures de tracé

Les procédures de tracé des éléments vectoriels (points, lignes, contours) utilisent tous les paramètres de position du cadre pour dessiner les figurés. Toutes les coordonnées vectorielles des objets et des arcs sont conservées en coordonnées géographiques dans SAVANE. Le dessin d'un point dans la carte nécessite plusieurs changements de coordonnées :

- une transformation correspondant à la projection géographique associée au cadre géographique ;
- une transformation correspondant à l'espace géographique délimité par le cadre et à l'échelle cartographique du cadre (translation-homothétie) ;
- une transformation correspondant à la position du cadre dans la carte (translation).

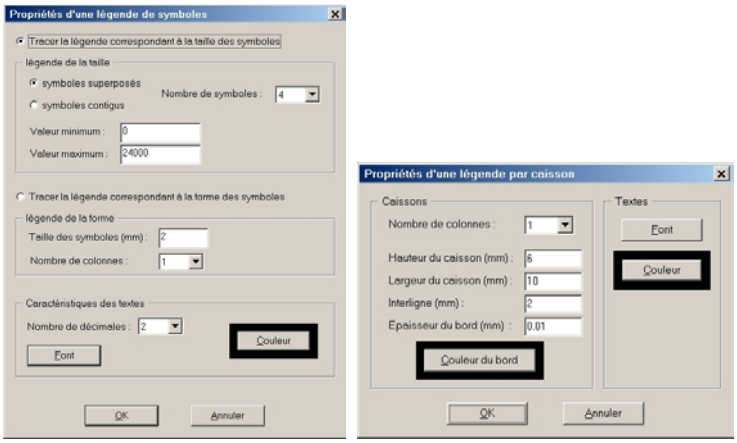
Lorsque le tracé correspond à l'écran, il faut ajouter une transformation correspondant à la position de la carte dans l'écran et au facteur d'agrandissement (translation-homothétie).

Les procédures de tracé d'une mosaïques sont différentes : le dessin est effectué en traçant les lignes des imagettes de la mosaïque, par balayage. Chaque imagette est géoréférencée en coordonnées de projection, et la taille du pixel permet de connaître l'emplacement exact de la ligne dans le cadre, et donc dans la carte. L'impression sur papier utilise quant à elle des procédures de dessin de *bitmap*, contenue dans la MFC (Microsoft Fundation Class).

Le dessin d'aplat pour les zones rend nécessaire la constitution du contour de chaque zone, si l'on veut utiliser une procédure vectorielle de remplissage. Le système SAVANE, comme nous l'avons vu, ne conserve pas cette structure pour chaque zone. Reconstituer les contours par chaînage des arcs à chaque rafraîchissement du tracé sur écran serait trop long : les aplats sont tracés sur l'écran à partir de l'image *raster* associée à chaque relation de type zone. Cette image *raster* est donc calculée lors du premier tracé, si elle n'existe pas encore. Par contre, l'impression d'aplat utilise une procédure vectorielle, avec reconstitution des contours, en utilisant la classe `CZone` (A.2.2.6.).

9.3.8. Les légendes

La légende correspond à l'explication de la correspondance entre valeurs des objets et implantation graphique et variables visuelles utilisées pour la représentation cartographique. L'explorateur cartographique d'un cadre contient l'ensemble des paramètres de cette correspondance, et il est donc également utilisé pour établir une légende.



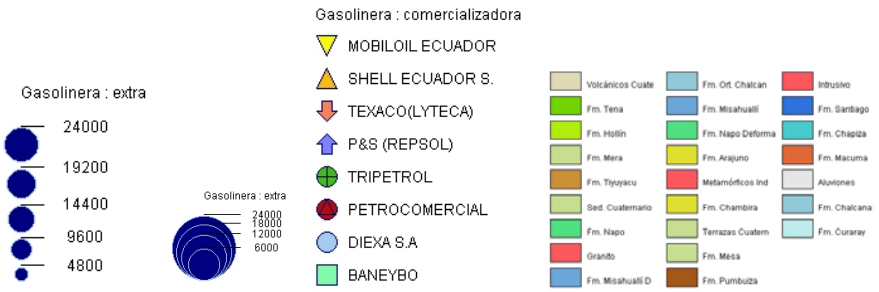


fig. 9.12 : différents types de légendes générés à partir de l'explorateur cartographique et des dialogues de propriétés de légende

Une légende dépend bien sûr du type d'implantation graphique choisi pour représenter les objets. Par exemple, pour une implantation zonale et un attribut nominal, la légende sera constituée de caissons donnant la correspondance entre valeur de l'attribut et couleur et forme utilisée. Pour une implantation ponctuelle et un attribut numérique, la légende indique la variation de taille du symbole utilisé. Pour constituer une légende, on choisit donc une entrée dans l'explorateur cartographique : le dialogue de constitution de la légende dépend ensuite du type d'implantation graphique, du type de l'attribut représenté, et de la variable visuelle utilisée. Une fois la légende posée dans la carte, elle est considérée comme du dessin graphique, et peut être modifiée grâce à l'éditeur graphique. Par contre, le lien entre la légende et l'explorateur cartographique est perdu : la légende n'est pas mise à jour automatiquement en cas de modification des paramètres de représentation cartographique.

Comme une légende est un ensemble de figurés qui dépendent de nombreux paramètres (par exemple, taille des caissons, largeur de l'interligne, couleur et épaisseur du bord des caissons, etc.), nous avons développé des classes propres à chaque type de légende (`CLegendeCaisson`, `CLegendeSymboleParTaille`, `CLegendeSymboleParValeur`, `CLegendeDegrade`, `CLegendeDiagrammeCaisson`, `CLegendeDiagrammeParTaille`, `ClegendeLigneParTaille`, `CLegendeLigneParValeur`).

La légende ne replace pas la notice de la carte, explication et analyse du phénomène représenté.

9.4. La représentation en perspective

La sémiologie graphique se limite à la représentation sur une feuille plane d'objets en utilisant des implantations graphiques limitées à deux dimensions. La

représentation en perspective ou l'animation sortent de ces limites. La cartographie classique n'utilise donc pas, ou très peu, la représentation en perspective (nommée 2.5D) ou en volume.

Dans un SIG, la localisation de l'ensemble des objets géographiques n'est que très rarement décrite en trois dimensions (longitude, latitude, altitude). Mais il est assez courant de disposer d'une relation décrivant l'altitude dans la zone étudiée, avec une certaine précision : soit on dispose d'un modèle numérique, soit de courbes de niveaux permettant de le calculer. Par jointure entre le modèle numérique de terrain et les autres collections d'objets localisés, le SIG permet donc d'avoir une idée de l'altitude de l'ensemble des objets géographiques de la zone étudiée. Il est donc dommage de ne pas utiliser cette altitude pour la représentation des objets et de toujours se limiter à une représentation projetée dans les deux dimensions du plan de projection géographique.

La représentation en perspective est la solution la plus élégante pour rendre compte du relief sur une feuille de papier plane. Elle est facile à mettre en œuvre avec un ordinateur (classe `CPerspective`, A.2.4.9.). Elle consiste à représenter le modèle numérique de terrain illuminé en projetant chaque pixel du modèle sur un plan, selon une perspective centrale ou cavalière, en éliminant les parties cachées. Une carte est représentée en perspective en superposant les figurés aux pixels du modèle numérique.

Représenter des figurés en perspective n'est techniquement pas difficile si l'on peut connaître l'altitude des points qui en définissent le tracé. Par contre, l'interprétation de la carte peut alors être plus complexe, car la perspective change l'impression visuelle de nombreux signes graphiques et altère la signification ou la lisibilité de la carte. Ainsi, la surface d'un symbole projeté en perspective peut varier considérablement en fonction de la pente ou de l'orientation du terrain ; une partie de texte peut être caché par une montagne ; les trames sont sujettes aux effets de moirés. D'une manière générale, les variables visuelles « taille » et « valeur » sont difficiles à représenter, quelle que soit l'implantation graphique des figurés, et l'on doit éviter de les utiliser dans une représentation en perspective. Par contre « couleur » et « forme » sont les variables visuelles qui sont le moins altérées par la représentation en perspective.

Le système SAVANE permet la représentation en perspective à partir d'un modèle numérique de terrain. Le système crée une image (de type bitmap), indépendante de la carte, mais que l'utilisateur peut ajouter à la feuille de papier représentant la carte sur l'écran. L'utilisateur choisit les paramètres de la perspective (cavalière ou centrale ; point de vue ; position de la source lumineuse), puis, parmi les entrées de l'explorateur cartographique, celles qu'il souhaite superposer au

modèle numérique de terrain. Les paramètres cartographiques sont inchangés : seule la représentation graphique est modifiée par la perspective.

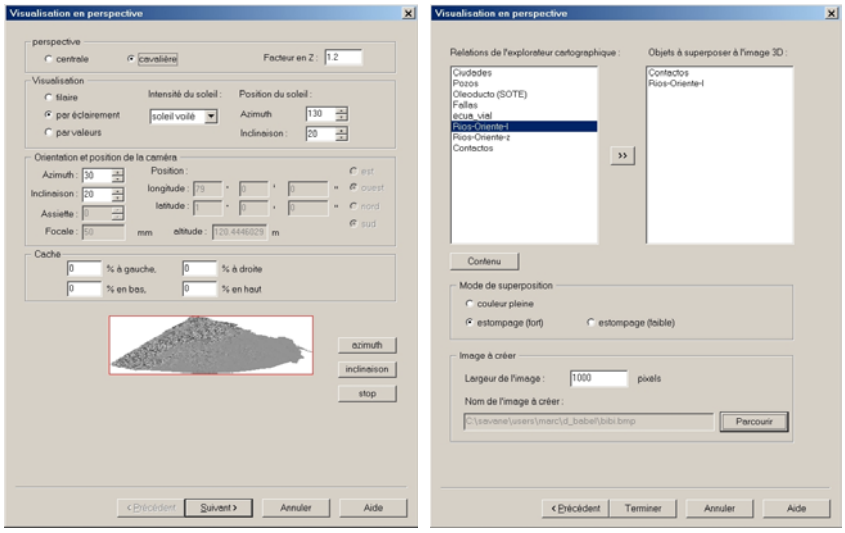


fig. 9.13 : dialogues pour créer une vue en perspective

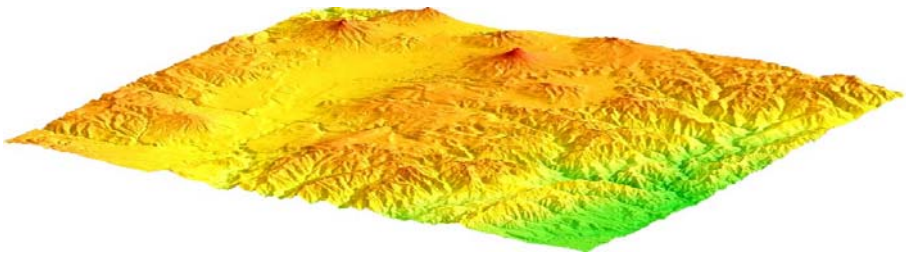


fig. 9.14 : représentation en perspective cavalière des valeurs d'altitude d'un mnt

9.5. L'ouverture ou la sauvegarde d'une carte dans SAVANE

La carte, document de base dans le module d'interrogation SAVANE, regroupe l'ensemble des paramètres d'interrogation et de représentation cartographique d'une requête. Tous les fichiers correspondant à une carte sont conservés dans un répertoire qui se trouve dans le répertoire *d_carte* de l'utilisateur. Sauver une carte permet de conserver l'ensemble des paramètres ainsi que l'état de la base de données correspondant à chaque cadre de la carte. La procédure d'une carte comprend plusieurs phases :

- sauvegarde des objets graphiques dessinés par l'utilisateur (conservés dans un fichier d'extension .obj) ;

- sauvegarde des paramètres de la carte et des cadres (conservé dans un fichier d'extension .sav).

La sauvegarde d'un cadre comprend la sauvegarde des paramètres graphiques du cadre, des palettes de couleurs, la sauvegarde de l'état de la base de données correspondant au cadre conservée dans un objet de la classe `CSchema`.

La sauvegarde d'un schéma comprend la sauvegarde du schéma temporaire de la base de données. Les fichiers temporaires des relations modifiées sont créés dans le répertoire du cadre, qui lui-même se trouve dans le répertoire de la carte.

L'ouverture d'une carte relit l'ensemble de ces paramètres. L'utilisateur retrouve donc l'état complet de la carte, à la fois pour le dessin et pour les requêtes liées à la base de données. Mais la base de données a pu subir des modifications de schéma ou de valeurs d'objets entre la sauvegarde de la carte et son ouverture. De même, la vue externe peut avoir été modifiée. La procédure de lecture teste ces différences de manière à mettre à jour le schéma des cadres. Par contre, si la valeur de certains objets a été modifiée, les relations temporaires ne sont pas mises à jour automatiquement, puisqu'elles conservent les anciennes valeurs des objets. Pour actualiser la carte, il faut exécuter de nouveau l'ensemble des opérations effectuées dans chaque cadre. Pour cela, il suffit de déclencher la macro-commande associée à chaque cadre : cette macro-commande conserve automatiquement l'ensemble des opérations qui ont été effectuées dans le cadre.

Tous les objets intervenant dans une carte (cadres, objets graphiques, schéma) possèdent des méthodes permettant de sauver ou de lire ces objets à partir des deux fichiers de sauvegarde.

Bibliographie

[BEG 94] BEGUIN M., PUMAIN D., *La représentation des données géographiques*, 2000, CURSUS, Armand Colin

[BER 67] BERTIN J., *Sémiologie graphique,* 1967, Gauthiers-Villars, Paris

[BER 77] BERTIN J., *Le graphique et le traitement graphique de l'information,* 1977, Flammarion, Paris

Conclusion

Tout au long de cet ouvrage, nous avons décrit la construction d'un système d'information visant à répondre aux impératifs que nous lui avions fixés au départ. Nous avons abordé au cours de cette construction de nombreux domaines et appliqué de nombreuses méthodes pour aboutir à la description de ce système complet. Dans cette conclusion, nous souhaitons à la fois revenir sur les points forts de cette construction et décrire les enseignements de son utilisation dans un cadre opérationnel, par rapport aux objectifs fixés, et présenter en quoi ce système pourrait être amélioré pour mieux répondre aux difficultés rencontrées lors de son utilisation. Ces difficultés se situent essentiellement dans le domaine de l'intelligibilité des données, de la documentation des bases de données et de l'explication des traitements effectués dans une carte.

1.1. La construction d'un SIG : synthèse

Tout d'abord, il faut souligner que le système que nous avons construit cherche à privilégier la rigueur et la logique sur la facilité d'utilisation. Il n'esquive donc pas la complexité lorsque celle-ci est nécessaire pour maintenir la cohérence et la rigueur des principes théoriques, principes que nous avons largement détaillés pour cela tout au long de cet ouvrage. D'autre part, il faut également préciser que ce système a vocation à être utilisé aussi bien dans un cadre industriel que dans un cadre scientifique. Il s'agit donc de manipuler des données très diverses, aussi bien du point de vue sémantique que du point de vue qualitatif, dans leur genèse comme dans leur précision. Cette remarque qui peut paraître anodine recouvre en fait une réalité complexe, comme nous avons essayé de le développer au chapitre 3. Enfin, si

la présentation de ce système met en avant la logique de sa construction, cette logique a été longue à construire : l'ordre des chapitres ne reflète pas toujours l'ordre de sa genèse ; l'architecture et les choix techniques s'inscrivent dans cette logique mais ils ont donné lieu à une réflexion approfondie, réflexion qui a elle-même permis de construire cette logique globale.

Nous avions fixé plusieurs objectifs dans notre introduction : le premier était la gestion centralisée et la pérennisation de l'information avec des objectifs de partage et d'efficacité. Pour assurer l'intégrité des données, le modèle relationnel s'impose, à condition de l'étendre aux données localisées. Pour une gestion efficace, il est nécessaire d'utiliser une indexation primaire sur la localisation, puisque la localisation est un premier critère de sélection pour la plupart des requêtes dans un SIG. Enfin, la gestion multi-utilisateur et le partage des bases de données imposent la séparation entre administration et exploitation avec gestion des états temporaires. Toutes ces conditions imposent la construction d'un moteur de gestion de données propriétaire au système. Ce moteur, comme nous l'avons décrit, est basé sur le principe de la modélisation et de la gestion relationnelle avec des fonctionnalités orienté-objet sur lesquelles nous reviendrons. La gestion relationnelle est étendue aux attributs représentant une localisation bi- ou tridimensionnelle dans un espace euclidien. L'indexation primaire est basée sur cet attribut lorsqu'il est présent. La gestion multi-utilisateur est délicate : elle impose de séparer l'administration de l'exploitation et interdit à un utilisateur de modifier une base de données si cela peut entraîner un disfonctionnement de l'utilisation par d'autres. Le système que nous avons construit est ainsi différent des systèmes commerciaux les plus répandus : il ne se base pas sur un SGBD relationnel commercial classique et conserve le résultat des requêtes non pas dans la base initiale mais dans des états temporaires, propre à chaque utilisateur. Cette gestion rend la conception du système plus complexe, mais elle permet le partage des bases de données sans provoquer d'interférence entre utilisateurs. Elle peut paraître trop complexe à un utilisateur isolé qui serait le seul utilisateur de sa base de données, car elle lui impose des contraintes dont il ne voit pas forcément l'utilité. Par contre, elle sera perçue comme transparente et naturelle dans un environnement où les données doivent être pérennisées et gérées pour être partagées. Elle permet la séparation claire du serveur de base de données et du client, tout en autorisant une approche exploratoire lors de l'interrogation des données – qui nous paraît essentielle en analyse spatiale - et la gestion d'états temporaires de la base de données lors d'une requête. L'interrogation via Internet se base sur ce même principe, sans avoir à changer la structure du système : le client est simplement hors du réseau local. Seules les techniques d'accès aux fichiers de la base de données et la gestion des reprises sur erreurs diffèrent. Par contre, les temps de transfert de l'information entre le client et le serveur sont encore un frein important à ce développement, que nous n'avons pas souhaité aborder dans cadre de cet ouvrage.

La gestion intégrée de la localisation impose de pouvoir manipuler l'ensemble des types d'objets géographiques disponibles ; elle impose également la maîtrise de la mesure et de la précision de la localisation. Nous sommes donc revenu en détail sur la modélisation du monde réel et sur la schématisation de la réalité en objets géographiques. La gestion interne de l'attribut de localisation dépend du type des objets. Mais le système se charge de cette gestion : l'utilisateur n'intervient jamais à ce niveau. C'est le moteur interne qui se charge de l'ensemble des opérations de gestion de cet attribut qui est toujours vu comme formel par l'utilisateur. Par exemple, il n'a jamais à intervenir sur le codage interne de la localisation, qu'il peut d'ailleurs ignorer. Ainsi, il manipule des pixels d'images de la même manière que des lignes d'un réseau ou que des zones d'une couverture végétale. Cette gestion intégrée n'est possible qu'à condition que la localisation soit exprimée dans un référentiel unique. Nous sommes revenu longuement sur ce sujet important au chapitre 4.

La saisie de la localisation est fondamentalement différente de la saisie d'attributs classiques, nominaux ou numériques. Elle est directement liée au type des objets localisés et doit prendre en compte l'ensemble des contraintes d'intégrité liées à ces types. Elle doit s'appuyer sur la modélisation de la réalité et gérer la précision liée au modèle. Les méthodes de saisie emploient diverses techniques, du redressement d'image à la saisie de contours de zones. Nous lui avons consacré deux chapitres de cet ouvrage. L'opération de saisie est une opération indépendante de la gestion de base de données. Pour faciliter l'utilisation du système, ces fonctions donnent donc lieu à des modules distincts, indépendants du module d'administration ou d'exploitation des bases de données. Ils assurent l'intégrité de l'attribut de localisation. Ces modules n'ont pas été conçus comme des éditeurs graphiques sur lesquels nous aurions ajoutés des fonctions des contrôles : la vérification des contraintes d'intégrité en est le noyau, et ils ont été techniquement construits autour de ces fonctionnalités.

Enfin, le système doit aller au-delà de simples fonctionnalités de gestion et doit intégrer des fonctionnalités très diverses qui en font autant un programme d'application qu'un système de gestion de base de données. La première de ces fonctionnalités, c'est de permettre la représentation cartographique des données. Le module d'exploitation permet donc l'édition graphique et la représentation cartographique à partir des objets de la base de données utilisée. Cette fonctionnalité incontournable détermine largement l'ergonomie de ce module. La plupart des autres fonctionnalités sont liées à l'exploration statistique ou aux types génériques des objets localisés (zone, ligne, point, pixel). Elles sont intégrés dans le module d'exploitation et sont indépendantes du contenu sémantique des relations de la base.

Des fonctionnalités plus spécialisées peuvent être intégrées au module d'exploitation. C'est la cas par exemple de l'exploitation des modèles numériques

de terrain pour l'hydrologie. Mais la démarche habituelle sera plutôt celle du couplage avec des programmes d'application, le système se chargeant alors de l'interface entre les deux systèmes. L'autre approche consiste à intégrer dans un programme d'application spécialisé des ordres d'accès aux objets géographiques d'une base de données, à partir d'un langage de programmation et d'une bibliothèque de primitives (nous avons pour cela construit une librairie de primitives SAVANE en C++). Mais on perd alors les capacités d'exploration, d'analyse spatiale ou d'édition cartographique du SIG, en se limitant à la gestion des objets. L'ambivalence entre gestion d'une base de données et programme d'application est en fait au cœur des difficultés d'utilisation d'un SIG, à qui on demanderait volontiers de répondre à l'ensemble des besoins.

Plus généralement, de nombreux objets géographiques ont des caractéristiques d'utilisation propres à leur définition sémantique, au-delà des types d'objets géographiques qui proviennent de la modélisation cartographique de la réalité. En général, ces caractéristiques ne sont pas intégrées dans la base de données, qui se limite aux attributs et n'intègrent pas les méthodes d'utilisation de ces attributs. On perd ainsi une grande partie de l'information, celle qui serait peut-être la plus utile dans le cadre de l'exploitation de la base de données. L'utilisation d'un SIG est en effet très différente d'une base de données classique : la requête fait souvent appel à une démarche exploratoire ; l'exploration des données influence largement l'utilisateur dans les traitements qu'il envisage d'appliquer aux données. De plus, il n'y a pas capitalisation du savoir acquis par l'exploitation de la base de données, le SIG étant exploité à peu près comme pourrait l'être une base de données classique. Cette déficience est majeure, car dans un SIG, les requêtes sont souvent difficiles à exprimer, beaucoup plus en tout cas que dans le cadre des base de données relationnelles classiques où la seule vrai difficulté, pour répondre à une question, est celle de l'utilisation d'un formalisme dépendant de la méthode de gestion.

1.2. Vers plus d'intelligibilité pour les bases de données géographiques

En résumé, le principal enseignement de l'utilisation d'un système comme celui que nous avons construit, c'est d'une part sa difficulté d'utilisation face à un corpus de données habituellement hétérogène et d'autre part la faible capitalisation de la connaissance induite par son utilisation. La difficulté d'utilisation ne vient pas de l'ergonomie ou de la structuration et de la gestion de l'information, car nous l'avons justement construit avec l'objectif de libérer l'utilisateur de toute tâche de gestion de données. Le système remplit avec satisfaction ces tâches et assure avec succès la pérennité des bases de données.

La difficulté d'utilisation vient essentiellement de la diversité des domaines que nous avons abordé : la connaissance de ces domaines est nécessaire à une bonne

utilisation d'un SIG quel qu'il soit. En particulier, le système doit remplir, au delà des fonctions de gestion, des fonctions d'analyse et de modélisation pour aboutir à des résultats synthétiques. La difficulté est inhérente à l'aspect programme d'application, car le système ne propose pas assez de méthodes sur l'information qu'il gère. Or, les utilisateurs attendent plus qu'une simple aide ou qu'une liste d'opérations susceptibles d'être effectuées sur un jeu de données. Ils souhaiteraient souvent que le SIG se charge tout seul de l'analyse ou leur indique quel chemin suivre parmi les nombreuses possibilités qui leur sont offertes. Paradoxalement, la multiplication des possibilités fonctionnelles nuit à l'utilisation et à la simplicité recherchée par la plupart des utilisateurs. Enfin, la difficulté d'utilisation provient également de la structure même de l'information géographique : elle induit une dépendance fonctionnelle entre localisation et description, et elle résulte le plus souvent d'une modélisation initiale qui n'est pas toujours connue ou bien documentée.

La question qui pourrait nous être posé est donc maintenant : « comment construire un logiciel SIG dont l'utilisation ne nécessite pas l'ensemble de cette connaissance *géographique* ? ». Cette question n'a sûrement pas de réponse globale. Il est peu probable que l'on pourra arriver à concevoir un système qui conserverait la rigueur scientifique nécessaire à la gestion et au traitement de l'information géographique mais qui n'exigerait de l'utilisateur qu'une faible connaissance des contraintes liées à cette rigueur. On peut malgré tout essayer d'améliorer le système pour aller dans ce sens. Ces améliorations concernent l'intelligibilité des bases de données, et vont dans trois directions :

- améliorer la connaissance de l'information par l'intégration systématique de méta-données pour renseigner sur la genèse des données et le rapport entre localisation et description (la précision géographique) ;

- améliorer la connaissance en intégrant dans la base de données des méthodes d'utilisation ;

- permettre la capitalisation de la connaissance en permettant la documentation ou l'intégration de méthodes d'utilisation des données, autant au niveau de la base de données elle-même que sur les résultats cartographiques des requêtes. Une carte devrait en toute logique comporter toujours une notice permettant de comprendre quelle a été la démarche qui a abouti à son élaboration.

L'architecture du système SAVANE permet d'avancer dans ces trois directions. Les méta-données peuvent être gérées avec un système de gestion classique à partir du dictionnaire de la base de données gérée par SAVANE. Elles doivent comporter des champs fixes (définis pour l'ensemble des relations), des champs dépendant du type géométrique de relation, et des champs variables laissés au choix de

l'administrateur ou du fournisseur de l'information. Il est très difficile d'établir de façon rigide la structure des champs fixes de cette base de données, car l'on arrive rapidement à concevoir un schéma qui ne sera en fait pas respecté en pratique : il est courant d'avoir moins de 20 % de réponses sur un schéma de méta-données… Il semble donc plus efficace d'avoir une approche plus souple, c'est-à-dire de limiter les variables requises et de mettre l'accent sur la gestion de données hétéroclites, non structurées en attributs, décrivant l'information de chaque relation.

La vision du schéma peut également être améliorer en introduisant un regroupement thématique de relations ou d'attributs au niveau des vues externes. Une vue externe est alors une hiérarchie de thèmes dont seules les feuilles correspondent à des relations ou des attributs réels de la base de données.

Le second point est sans doute le plus riche pour l'utilisateur : il consiste à introduire dans la base de données des méthodes d'utilisation construites à partir des opérations relationnelles ou d'analyse disponible dans le module de base, ou à partir de modules d'applications externes appelés par le système. Ces méthodes sont conservés techniquement sous la forme de macro-commandes ; l'utilisateur accède au schéma des méthodes comme il accède au schéma des données. La base de données passe donc du schéma relationnel au schéma objet, tout en conservant la gestion interne relationnelle qui lui permet d'assurer la cohérence de l'information. Cette approche permet d'introduire de la connaissance dans des bases de données souvent difficiles à exploiter ; elle permet de proposer à l'utilisateur des méthodes qui peuvent aller du simple calcul d'une densité ou d'un indice de végétation à une application complète. Une base de données contenant des méthodes devient en soi un objet de connaissance, et disposer ce cette connaissance change l'approche dans l'élaboration même d'une base de données : dans de nombreux cas, le schéma de la base de données sera déterminé par les méthodes que l'on souhaite pouvoir utiliser par la suite. On peut ainsi proposer des schémas de base de données géographiques permettant de répondre à un certain nombre de questions, grâce à des méthodes fournies avec le schéma. L'approche classique dans l'élaboration est donc inversée : les méthodes d'application ne sont plus déterminées par la structure de l'information, c'est la structure de la base qui est déterminée par les méthodes que l'on souhaite pouvoir utiliser. Cette approche est bien sûr fortement contextuelle : elle doit déboucher sur la définition d'objets géographiques, valables dans un contexte donné, et contenant méthodes et données permettant de les utiliser dans ce contexte.

Cette approche permet de capitaliser de la connaissance au niveau de la base de données, mais elle n'est pas suffisante pour conserver l'ensemble de la connaissance élaborée dans l'utilisation du SIG. En effet, elle conserve des méthodes d'utilisation sous forme de macro-commandes, mais ne renseigne pas sur le cheminement général d'une application ou sur l'élaboration cartographique du résultat. Il est

également nécessaire de conserver cette information. Dans SAVANE, nous développons donc également des méthodes de documentation au niveau d'un cadre, dans une carte, comme au niveau de la carte elle-même. Cette documentation est difficile à structurer par un schéma analytique : nous avons choisi de la conserver sous forme de texte, à l'image d'une notice ; le texte est élaboré par l'auteur de la carte. A chaque cadre dans une carte et à chaque carte est donc attaché un document de type texte qui est à la disposition de l'utilisateur et auteur de la carte, contrairement aux méthodes qui sont gérées au niveau de l'administrateur de la base de données.

Ces trois approches sont en cours de développement au sein du système SAVANE. Elles s'appuient sur la gestion relationnelle étendue aux données localisées : les méta-données à partir du dictionnaire de la base, les méthodes à partir d'opérations de gestion ou d'applications externes, les notices à partir du résultat d'une requête. Elles permettent d'étendre la gestion relationnelle en intégrant une approche orienté objet et hypermédia, tout en conservant la rigueur de ce modèle de gestion pour les données localisées.

Bibliographie générale

[ABE 84] ABEL D.J., *A B+-Tree Structure for Quadtrees*, Computer Vision, Graphics, and Image Processing, 1984, vol. 27, n° 1, p. 19-31

[ABI 95] ABITEBOUL S., HULL R., VIANU V., *Foundations of Databases*, 1995, Addison-Wesley

[AOK 79] AOKI M., *Rectangular region coding for image data compression*, Pattern Recognition, 1979, vol. 11, p. 297-312

[AMI 71] AMIDON E.L. AND ATKIN G.S., *Algorithmic selection of the best method for compressing map data string* : Communications, ACM, 1971, vol. 14, n° 12, p. 769-774

[BAL 98] BALMINO G., *Champ de pesanteur terrestre et géoïde. Principes, progrès et connaissance actuelle*, 1998, Bureau Gravimétrique International, Toulouse, France

[BAR 88] BARUCH O., *Line Thinning by Line Following*, Pattern Recognition Letters, 1988, Vol. 8, p. 271-276

[BEG 94] BEGUIN M., PUMAIN D., *La représentation des données géographiques*, 2000, CURSUS-Armand Colin, Paris

[BEK 92] BEKKER J.H., *Semantic Data Modeling*, Prentice-Hall, 1992, New-York

[BEN 93] BENKAZEN V., DOUCET A., *Bases de données orientées objet*, 1993, Armand Colin

[BER 67] BERTIN J., *Sémiologie graphique,* 1967, Gauthiers-Villars, Paris

[BER 77] BERTIN J., *Le graphique et le traitement graphique de l'information,* 1977, Flammarion, Paris

[BER 93] BERTINO E., MARTINO L., *Object-Oriented Database Systems*, 1993, Addison-Wesley

[BOT 98] BOTTON S ., DUQUENNE F., EGELS Y., EVEN M., WILLIS P., *GPS, localisation et navigation*, 1998, Hermès, Paris

[BOU 81], BOURSIER P., *Représentation compacte des cartes dans les systèmes d'information géographique*, Thèse de 3-ième cycle, Université Paris VI, 1981,

[BOU 82] BOURSIER P. ET SCHOLL M., *Performance Analysis of compaction techniques for map representation in geographic data cases*, Computer and Graphics, 1982, vol. 6, p. 73-81

[BOU 94] BOURSIER P., TAO-YUAN J., *A model for Handling topological relationships in a 2D environment*, Proceedings of the 6th International Symposium on Spatial Data Handling, 1994, Vol 1, pp 73-88, Endinburgh

[BRE 65] BRESENHAM J.E., *Algorithm for computer control of a digital plotter*, IBM System Journal, 1965, vol. 14, n° 1, p.44-50

[CAP 84] CAPSON D.W., *An improved algorithm for the sequential extraction of boundaries from a raster scan*, Computer Vision, Graphics, and Image Processing, 1984, vol. 28, n°1, p. 109-125

[CAZ 94] CAZENAVE A., FEIGL K., *Formes et mouvements de la Terre*, 1994, Ed.Belin-CNRS

[COO 67] COOK B.G., *A Computer Representation of plane Region Boundaries*, Autralian Computer Journal, 1967, vol. 1, n°1, p.44-50

[CLE 93] CLEMENTINI E., DI FELICE P., VAN OOSTREROM P., *A Small Set of Formal Topological Relationships Suitable for End-User Interaction*, Proceedings of the 3th Symposium on Large Spatial Databases, 1993, Lecture Notes in Computer Sciences 692, pp 277-295, Singapore

[CLE 95] CLEMENTINI E. DI FELICE, CALIFANO G., *Composite Regions iin Topological Queries*, Information Systems, 1995, Vol. 20, n°7, pp 579-594

[COH 93] COHN A.G., RANDELL D.A., CUI Z., BENNETT B., *Qualitaiv Spatial Reasoning and Representation*, Preceedings of QUARDET 93, PP 513-522, Barcelona, Spain, 1993

[CUI 93] CUI Z., COHN A.G., RANDELL D.A. , *Qualitativ and Topological Relationships in Spatial Databases*, Proceedings of the 3th Symposium on Large Spatial Databases, 1993, Lecture Notes in Computer Science 692, pp 296-315, Singapore

[DAN 81] DANGERMOND J., *Some trends in the evolution of GIS technology*, Marble Ed., 1981, Kensington Workshop, p. 25-57

[DAN 83] DANGERMOND J., *Classification des éléments logiciels utilisés habituellement dans les système d'information géographique*, fascicule n° 96 du comité français de cartographie, 1983, p. 7-20

[DAT 90] DATE C., *A Contribution to the study of database integrity*, in Relational Database Writing, 1990, Addison-Wesley

[DAT 95] DATE C.J., *Introduction to Database Systems*, 1995, Addison-Wesley

[DAV 91] DAVID B., *Modélisation, représentation et gestion d'information géographique, un approche en relationnel étendu*. Thèse de doctorat, Université Paris VI, 1991

[DAV 94] DAVID B., FASQUEL P., *Qualité d'une base de données géographique : concepts et terminologie,* Bulletin d'information de l'IGN, 1994, n° 67, Paris

[DEL 91] DELOBEL C., LECLUSE C., RICHARD P., *Bases de données : des systèmes relationnels aux systèmes à objets*. 1991, InterEditions, Paris

[DIM 70] DIME, *Technical description of the DIME System*, U.S. Bureau of Census : *Census and study, the DIME Geocoding System*, Report n° 4, Washington D.C., 1970, p. 25-30

[DUF 98] DUFOUR J.P., *Cours d'introduction à la géodésie*, Ecole nationale des sciences géographiques,1998, IGN-ENSG

[EGE 89] EGENHOFER M.J., *A formal definition of binary topological relationships*, Proceedings of the third International Conference FODO, Paris, Lecture Notes in Computer Science 367, 1989, Sprinter Verlag, Berlin, p.457-472

[EGE 90] EGENHOFER M.J. AND HERRING J.R., *A mathematical framework for the definition of topological relationships*, Proceedings of the 4th International symposium on Spatial Data Handling, 1990, Zurich p. 803-813

[EGE 91A] ENGENHOFER M., *Spatial Query langages*. Thèse de doctorat, 1991, University of Maine, USA

[EGE 91B] EGENHOFER M.J., FRANZOZA R.D., *Point-Set Topological Relations*, International Journal of Geographic Information Systems, 1991, 5-2, pp 161-174

[EGE 94] EGENHOFER M.J., CLEMENTINI E., SHARMA J., *Modelling topological spatial relations : strategies for query processing*, Computer and graphics, 1994, vol. 18, n°6, pp 515-522

[EST 92] ESTES J., *Remote sensing and GIS integration : research needs, status and trends*. ITC Journal, 1992, n° 1, p. 2-9

[FAI 96] FAIZ S.O., *Modélisation, exploitation et visualisation de l'information qualité dans les bases de données géographiques*, Thèse de Doctorat, 1996, Univ. Paris XI Orsay

[FLO 85] DE FLORIANI L., FALCIDIENO B., PIENOVI C., *Delaunay-based Representation of Surfaces Defined over Arbitrary Shaped Domains*, Computer Vision, Graphics, and Image Processing, 1985, n° 32, p. 127-140

[FLO 93] DE FLORIANI L., MARZANO P., PUPPO E., *Spatial Queries and Data models*, in Frank A. and Campari I.U., Spatial Information Theory, Lecture Notes in Computer Sciences 716, 1993, Springer-Verlag, p. 113-138

[FRA 81] FRANCK A., *Application of DBMS to Land information systems*, CH 1701-2, IEEE 81, 1981, p. 448-453

[FRA 92] FRANK A.U., *Qualitative Spatial Reasoning about Distances and Directions in space*, Journal of Visual Langages and Computing, 1992, vol. 3, n° 4, pp 343-371

[FRA 96] FRANK A.U, *Qualitative spatial reasoning : cardinal directions as an example.* International Journal of Geographical Information Systems, 1996, Vol. 10, n°3, pp 269-290

[FRE 61] FREEMAN H., *On the Encoding of arbitrary geometric Configuration*, Inst. Radio Engineers Trans. Elec. Computers, 1961, vol. EG 10, p. 260-268

[FRE 75] FREEMAN J., *The modelling of spatial relations*, Computer Graphics and Image Processing, 1975, vol. 4, p. 156-171

[GAR 83] GARDARIN G., *Bases de données : les systèmes et leurs langages*, 1983, Eyrolles, paris

[GDT 93] GDTA, *Les Spatiocartes*, Cahiers Pédagogiques du GDTA, 1993

[GEO] *Géophysique*, Encyclopédie de la Pléiade, Gallimard, Paris

[GOO 89] GOODCHILD M., GOPAL S., *Accurancy of Spatial Databases*, 1989, Taylor & Francis

[GRE 89] GREENE D., *Implementation and Performance Analysis of Spatial Data Access Methods*, IEEE, Intl. Conference on Data Engineering, 1989, p.606-615

[GUN 89] GÜNTHER O., *Efficient Computation of Spatial Joins*. IEEE, Intl. Conference on Data Engineering, 1989, p.50-59

[GUP 95] GUPTILL S.C., MORRISON J.L., *Elements of Spatial Data Quality*, Pergamon/Elsevier Science, 1995

[GUT 88] GÜTING R.H., *Geo-Relational Algebra : A Model and Query Language for Geometric Database System*, Proccedings of Intl. Conference on Extending Database Technology, 1988, p. 506-527

[HAA 96] HAADZILACOS T., TRYFONA N., *A model for expressing topological integrity constraints in geographical databases*, in Proceedings : Theories and Models of Spacio-Temporal Reasoning in Geographic Space, 1996, International Conference GIS, Pisa, Italy. A. Frank, I. Campari, U. Fomentini (Ed.), pp 252-268

[HAG 73] HAGGETT P., *L'analyse spatiale en géographie humaine*, 1973, Armand Colin, Paris

[HAS 82] HASKIN R.L. AND LORIE R.A., *On extending the functions of a relational database system*. Association for Computing Machinery, Proccedings of SIGMOD, NY : ACM Press, 1982, p. 207-212

[HOT 99] HOTTIER, *Photogrammétrie analytique*, Cours ENSG, 1999

[LAR 93] LARUE T., PASTRE D., VIÉMONT Y., *Strong Integration of Spatial Domains and Operators in a Relational Database System*, Intl. Symposium on Large Spatial Databases, LNCS n° 692, 1993, p. 53-72, Springer-Verlag

[LAU 91] LAURINI R., MILLERET-RAFFORT F., *Cohérence dans les bases de données spatiales*, VIIe journées Bases de données Avancées, INRIA, 1991, pp 471-488, Lyon

[LAU 93] LAURINI R., MILLERET-RAFFORT F., *Les bases de données en géomatique,* Paris, Ed. Hermès, 1993

[LAU 94] LAURINI R., *Détection et corrections d'erreurs topologiques dans les bases de données géographiques*, Actes des premières journées de la recherche CASSINI, 1994, Lyon, France, pp 105-114

[LEV 88] LEVALLOIS J.J., BOUCHER C., BOURGOIN J., COMOLET-TIRMAN A., ROBERTOU A., *Mesurer la Terre : 300 ans de géodésie française, De la toise du Châtelet au satellite*, 1988, Association française de topographie, Presse de l'Ecole Nationale des Ponts et Chaussées, AFT, Paris

[LOR 84] LORIE R.A., MEIER A., *Using a Relational DBMS for Geographic Databases*. GeoProcessing, 1984, p.243-257

[MAG 97] MAGELLAN SYSTEMS CORPORATION, *User guide for the Magellan GPS ProMARK X-CM*, 1997

[MAT 84] MATSUYAMA T., LE VIET H., NAGAO M., *A File Organisation for Geographic Information System based on Spatial Proximity*, Computer Vision, Graphics, and Image Processing, 1984, vol. 26, n°3, p.303-318

[MER 73] MERRILL R.D., *Representation of contours and region for efficient computer search*, Communication ACM, 1973, vol. 16, n°2, p. 69-82

[MOL 94] MOLENAAR O, KUFONIYI O., BOULOCOS T., *Modelling topological relationships in vector maps*, Proceedings of the 6th International Symposium on Spatial Data Handing, 1994, pp 112-126, Endinburgh

[MON 82] MONMONIER M., *Computer-Assisted Cartography, Principles and Prospects*, 1982, Prentice-Hall

[MÜL 95] MÜLLER J.C., LAGRANGE J.P., WEIBEL R., (ed.), *GIS and Generalization*, GIS DATA I, 1995, Taylor & Francis

[NEW 79] W. M. NEWMAN, R.F. SPROULL, *Principles of Interactive Computer Graphics*, 1979, Mc Graw Hill

[NOV 92] NOVAK K., *Rectification of digital Imagery*. Photogrammetry Eng. And Remote Sensing, 1992, vol. 58, p.339-344

[OOI 89] OOI B.C., SACK-DAVIS R., MC DONELL K.J., 1989, *Extending a DBMS for Geographic Applications*, Intl. Conference on Data Engineering, p. 590-597

[O'RO 93] O'ROURKE J., *Computational Geometry in C*, 1993, Cambridge University Press

[PAL 75] PALMER J.A.S., *Computer Science aspects of the mapping problem*, from Davis and Mc Cullagh, Display and analysis of spatial Data, New York, John Wiley and Sons, 1975

[PAR 97] PARKER J.R., *Algorithms for Image Processing and Computer Vision*, 1997, Wiley Computer Publishing

[PAV 82] PAVLIDIS T., *Algorithms for Graphics and Image Processing*, Computer Science Press, 1982

[PLÜ 97] PLÜMER, GRÖGER, *Archiving Integrity in Geographic Information Systems – Maps and Nested Maps*, Geoinformatica, 1997, vol.1, n° 4, pp 345-367, Kluwer Academic, Boston

[PUL 88] PULLAR D.V., EGENHOFFER M.J., *Toward formal definitions of topological relations among spatial objects*, Procedings of the third International Symposium on Spatial Data Handling, 1988, Sydney, Australia, pp 225-241

[PUM 97] PUMAIN D., SAINT-JULIEN T., *L'analyse spatiale*, 1997, Armand Colin, « Cursus », Paris

[REN 91] RENOUARD L., *Restitution automatique du relief à partir de couples stéréoscopiques d'images du satellite SPOT*, 1991, Thèse de Doctorat présentée à l'Ecole Polytechnique

[RIM 90] RIMBERT S., *Carto-graphies*, 1990, Hermès, Paris

[ROS 80] ROSENFELD A. AND SAMET H., *Tree structure for region representation*, Aangeenburg, Autocarto 4, 1980, vol. 1, p. 108-118

[ROU 91] ROUET P., *Les données dans les systèmes d'information Géographique.* Traité des nouvelles technologies, série Géomatique, 1991, Hermès, Paris

[SAM 80A] SAMET H., *Region Representation : quadtrees from boundaries codes*, Communication ACM, 1980, vol. 23, n°3,p. 163-170

[SAM 80B] SAMET H., *Tree Structure for region representation*, Aangeenbgug, Autocarto 4, vol. 1, 1980, pp 108-118

[SAM 90] SAMET H., *Applications of Spatial Data Structures*, Addison-Wesley, 1990

[SAN 89] SANDERS L., *L'analyse des données appliquée à la géographie*, 1989, RECLUS, Montpellier

[SCH 89] SCHOLL M. ET VOISARD A., *Thematic map Modeling*, Int. Symposium on Large Spatial Databases, 1989, LNCS n°409, p. 167-192

[SCH 96] SCHOLL M., VOISARD A., PELOUX J.P., RAYNAL L., RIGAUX P., *SGBD Géographiques, Spécificités*, Thomson Publishing, 1996, Paris

[SHA 78] SHAMOS M. I., *Computational Geometry*, PHD, 1978, Yale University

[SHL 88] SHLAER S, MELLOR S-J., *Objet Oriented System Analysis : Modelling the World in Data*, Englewood Cliffs, 1988, Prentice-Hall

[SOU 86] SOURIS M., *Systèmes d'information géographique et bases de données*, Colloques et Séminaires sur le Traitement des données localisées, Paris, Editions de l'ORSTOM, 1986, p. 29-87

[SOU 87] SOURIS M., *A Geographic Information System with relational architecture : principles and example of use*, in Primera conferencia sobre informatica y geografia, 1987, San Jose, Costa Rica, pp 703-728

[SOU 90] SOURIS M., *Le système SAVANE, Manuels de référence*, 1990-2002, IRD

[TAN 95] TANZI T., UBEDA T., *Contrôle topologique de la cohérence dans les bases de données géographiques : application aux réseaux* , Revue Internationale de Géomatique, 1995, Vol. 5, n°2, pp 133-155

[TOM 76] TOMLINSON, CALKINS AND MARBLE, *Essential part of a geographic information system*, Computer Handling of geographic Data, UNESCO Press, Paris, 1976

[UBE 96] UBEDA T., SERVIGNE S., *Geometric and Topological Consistency of Spatial Data*, 1996, in Proceedings: First International Conference on Geocomputation, Leeds, UK

[UBE 97] UBEDA T., *Contrôle de la qualité spatiale dans les bases de données géographiques : Cohérence topologique et corrections d'erreurs*, Thèse soutenue à l'INSA de Lyon, 1997

[ULL 88] ULMANN J.D., *Principles of Database and Knowledge-Base Systems*, 1988, Computer Science Press

[VOI 92] VOISARD A., *Bases de données géographiques : du modèle de données à l'interface utilisateur*. Thèse de doctorat, Université d'Orsay, 1992

[VOI 95] VOIRON C., *Analyse spatiale et analyse d'images*, 1995, RECLUS, Montpellier

[WOR 90] WORBOYS M.F., HEARNSHAW H.M., MAGUIRE D.J., *Object-Oriented data Modelling for Spatial Databases*, International Journal of Geographic Information Systems, 1990, vol. 4, p. 369-383

[WOR 95] WORBOYS M. F., *GIS, a Computer Perspective*, Taylor and Francis, 1995

Annexes

Annexe A

Structures et méthodes : implémentation

Cette annexe présente le code C++ de classes correspondant à l'implémentation des structures, des méthodes, et des algorithmes décrits dans les chapitres qui précèdent. Nous ne présentons ici qu'une partie des classes du système, en excluant en particulier toutes les classes concernant l'interface utilisateur, propre au système de développement utilisé (Visual C++ de Microsoft).

Chaque module fera l'objet d'une section distincte. L'ordre de présentation des classes suivra en général leur apparition dans le texte principal de ce mémoire, car il est difficile d'établir une hiérarchie selon leur importance dans la construction du système. Certaines classes sont utilisées dans l'ensemble des modules : elles ne seront bien sûr présentées qu'une seule fois.

A.1. SAVATECA

A.1.1. Les classes CAttribut et CRelation

Ces classes représentent les objets fondamentaux que sont les relations et les attributs :

```
class CAttribut
  {
  public :
    CAttribut();
    int m_iType;
    int m_iPtr;
    int m_iNbcar;
    int m_iNbvalb;
    int m_iPtrval;
    int m_iNumero;
    char m_chNom[20];
    int m_iRangDansLaVue;
  };
```

La variable `iType` donne le type de l'attribut (1 ou 5 : nominal; 2 : ordinal; 3 : entier;4 : réel; 6 : pixel entier 8 bit; 7 : pixel entier 16 bitsclass ; 8 : pixel RVB ; 9 : pixel 32 réel bits). La variable `m_iNbcar` indique le nombre de caractères utilisé pour les modalités dans le cas d'un attribut nominal. La variable `m_iNbvalb` indique le nombre de modalités intégrées dans la base de données. La variable `m_iPtrval` donne l'adresse de la première modalité dans le fichier *fpvaleurs*. Ces trois variables permettent de retrouver dans le fichier fpvaleurs une modalité à partir de son codage interne `iValeur` :

```
iPtr=(m_iPtrval+iValeur-2)*m_iNbcarAttr;
fseek(FileValeurs,iPtr,SEEK_SET);
fread(chNomValeur,m_iNbcarAttr,1,FileValeurs);
```

La classe CRelation est également utilisée dans tous les modules du système :

```
CRelation
  {
  public :
    int m_iPtr;
    int m_iNrec;
    int m_iType;
    int m_iNba;
    int m_iPtrLibreArc;
    int m_iPtrLibreTuple;
    char m_chNomFichF[16];
    char m_chNomFichZ[16];
    char m_chNomFichA[16];
    char m_chNom[100];
    char m_chFichierFeuille[_MAX_PATH];
    char m_chFichierZone[_MAX_PATH];
    char m_chFichierArc[_MAX_PATH];

    CAttribut* m_pAttributs[NB_MAX_ATTRIBUTS];
    int m_iRangDansLaVue;

  public:
    CRelation();
    ~CRelation();
  };
```

Le tableau `CAttribut* m_pAttributs[NB_MAX_ATTRIBUTS]` conserve la liste des pointeurs vers les objets de type `CAttribut` de la relation. La variable `m_iType` indique le type de la relation (1 : non localisée ; 2 : ligne ; 3 : point ; 4 : zone ; 5 : pixel).

A.1.2. La classe CBase

La classe `CBase` décrit dans SAVATECA la structure d'une base de données :

```
class CBase
  {
  private:

  public:
    char m_chNomBase[100];
    char m_chDirAdmin[_MAX_PATH];
    char m_chDirBase[_MAX_PATH];
    char m_chDirDigit[_MAX_PATH];
    char m_chDirDoc[_MAX_PATH];
    char m_chNomFichierAcces[_MAX_PATH];
    char m_chNomFichierBase[_MAX_PATH];
    char m_chNomFichierValeurs[_MAX_PATH];
    char m_chNomFtmp[_MAX_PATH];
```

```
    double m_dMaxx;
    double m_dMaxy;
    double m_dMinx;
    double m_dMiny;
    int m_iOuvert;
    int m_iNbrel;
    CRelation* m_pRelations[NB_MAX_RELATIONS];

  public:
    CBase();
    ~CBase();

    int IsOuverte() {return m_iOuvert;};
    void Ouverte() {m_iOuvert=1;};
    void Fermer();
    BOOL Init(char* chNomBase,char* chDirBase,char* chDirDoc);
    void InitRang();

    BOOL Alloc();
    void Desalloc();
    BOOL Realloc();
    int RelRecherche(const char *nom);
    int RelDupRecherche(int iNoth, int* itabNoth);
    int AttDupRecherche(int iNoth, int iNoatt, int* itabNothAtt);
    int FiczRecherche(char *nom);
    int FicfRecherche(char *nom);
    int FicaRecherche(char *nom);
    int AttrRecherche(int iNoth,char *nom);
    BOOL NouveauFichier(int iType,char *chNom);

    BOOL InsertionNewValeurs(char *ctabNewValeurs,int iNbNewValeurs,int iNoth,int iNoatt);
    int NbObjets(int iNoth);
    int NbValeurs(int iNoth,int iNoatt);

    void GeoToSav(double dLongitude,double dColatitude,float& fXSav,float& fYSav);
  };
```

On remarquera en particulier la variable membre `m_iNbrel`, qui donne le nombre de relations de la base, lorsque celle-ci est ouverte, et `CRelation* m_pRelations[NB_MAX_RELATIONS]` qui donne les adresses des objets de type `CRelation` de la base. Les méthodes `RelRecherche()` et `AttrRecherche()` renvoie le numéro interne d'une la relation ou d'un attribut à partir de son nom. Les méthodes `Alloc()` ou `ReAlloc()` permettent de charger la structure de la base à partir du dictionnaire, conservé dans le fichier *base*. Le fichier *base* est maintenu par la classe `CDico`.

```
BOOL CBase::Alloc()
  {
  //--- initialisation des accès aux données suivant le dictionnaire de la base ouverte
  int iPtr,j,k;
  int iNbRelations;
  int iTypeRelation,iNbAttributs,iNrec,plibz,pliba;
  int iTypeAttribut,iNbvalb,iPtrval,iNbcar;
  long adr;
  char chNomRelation[64];
  char chNomAttribut[64];
  char chDescription[128];
  char chNom2[_MAX_PATH],chNom3[100],chNom4[100];

  CDico dico;
  if(dico.Ouvrir(m_chNomFichierBase,MODE_READ) == FALSE)return FALSE;
  if(dico.LireNbRelations(iNbRelations) == FALSE)return FALSE;

  m_iNbrel=iNbRelations;

  iPtr=3;
  j=1;
  while(j <= m_iNbrel)
    {
    adr=(iPtr-1)*dico.GetTailleRecord();
```

```
  if(dico.LireRelation(adr,chNomRelation,iTypeRelation,iNbAttributs,iNrec,chNom2,chNom3,chNom4
,plibz,pliba) == FALSE)return FALSE;

     m_pRelations[j]= new CRelation;
     m_pRelations[j]->m_iPtr=adr;
     strcpy(m_pRelations[j]->m_chNom,chNomRelation);
     m_pRelations[j]->m_iType=iTypeRelation;
     m_pRelations[j]->m_iNba=iNbAttributs;
     m_pRelations[j]->m_iNrec=iNrec;
     m_pRelations[j]->m_iRangDansLaVue=0;
     if(iTypeRelation == 5)
       {
       strcpy(m_pRelations[j]->m_chFichierZone,chNom2);
       strcpy(m_pRelations[j]->m_chFichierFeuille,"");
       strcpy(m_pRelations[j]->m_chFichierArc,"");

       strcpy(m_pRelations[j]->m_chNomFichF,"");
       strcpy(m_pRelations[j]->m_chNomFichZ,"");
       strcpy(m_pRelations[j]->m_chNomFichA,"");
       m_pRelations[j]->m_iPtrLibreArc=0;
       m_pRelations[j]->m_iPtrLibreTuple=0;
       }
     else
       {
       strcpy(m_pRelations[j]->m_chNomFichF,chNom2);
       strcpy(m_pRelations[j]->m_chNomFichZ,chNom3);
       strcpy(m_pRelations[j]->m_chNomFichA,chNom4);
       m_pRelations[j]->m_iPtrLibreArc=pliba;
       m_pRelations[j]->m_iPtrLibreTuple=plibz;

       strcpy(m_pRelations[j]->m_chFichierFeuille,m_chDirBase);
       strcat(m_pRelations[j]->m_chFichierFeuille,"\\");
       strcat(m_pRelations[j]->m_chFichierFeuille,chNom2);
       strcpy(m_pRelations[j]->m_chFichierZone,m_chDirBase);
       strcat(m_pRelations[j]->m_chFichierZone,"\\");
       strcat(m_pRelations[j]->m_chFichierZone,chNom3);
       strcpy(m_pRelations[j]->m_chFichierArc,m_chDirBase);
       strcat(m_pRelations[j]->m_chFichierArc,"\\");
       strcat(m_pRelations[j]->m_chFichierArc,chNom4);
       }

     iPtr++;
     k=1;
     while(k <= iNbAttributs)
       {
       adr=(iPtr-1)*dico.GetTailleRecord();

  if(dico.LireAttribut(adr,chNomAttribut,iTypeAttribut,iNbvalb,iPtrval,iNbcar,chDescription)
== FALSE)return FALSE;

       m_pRelations[j]->m_pAttributs[k]= new CAttribut;
       m_pRelations[j]->m_pAttributs[k]->m_iNumero=k;
       m_pRelations[j]->m_pAttributs[k]->m_iPtr=adr;
       strcpy(m_pRelations[j]->m_pAttributs[k]->m_chNom,chNomAttribut);
       m_pRelations[j]->m_pAttributs[k]->m_iType=iTypeAttribut;
       if(iTypeAttribut == 1)m_pRelations[j]->m_pAttributs[k]->m_iNbcar=iNbcar;
       else m_pRelations[j]->m_pAttributs[k]->m_iNbcar=0;
       m_pRelations[j]->m_pAttributs[k]->m_iNbvalb=iNbvalb;
       m_pRelations[j]->m_pAttributs[k]->m_iPtrval=iPtrval;
       m_pRelations[j]->m_pAttributs[k]->m_iRangDansLaVue=0;

       iPtr++;
       k++;
       }
     j++;
     }
  dico.Fermer();

  return TRUE;
  }
```

A.1.3. La classe CDico

La classe `CDico` encapsule l'ensemble des variables et des fonctions nécessaires à la lecture et l'écriture du dictionnaire d'une base de données SAVANE. Le dictionnaire est conservé dans le fichier *base* qui doit toujours se trouver dans le répertoire de stockage de la base de données. Cette classe est utilisée dans tous les modules du système, pour charger en mémoire la structure d'une base de données lors de son ouverture dans l'un des modules. Les fonctions d'écriture ne sont utilisées que dans le module SAVATECA, lors de la création ou de la modification d'une base de données. Cette classe permet également de gérer l'évolution de la structure du fichier *base*, en y ajoutant des méthodes de mise à jour lorsque cela est nécessaire.

```
class CDico
  {
  private:
    FILE* m_FileBase;
    char m_chBuffer[256];
    char m_chLecture[256];
    int m_iTailleRecord;
  public:
    CDico();
    ~CDico();
    int GetTailleRecord() {return m_iTailleRecord;};
    BOOL Ouvrir(const char* chNomFichierBase,int iMode);
    void Fermer();
    BOOL LireEntete1(int& iDefecr,int& iPbl,int& iVersion,char* chUtilisateur);
    BOOL LireEntete2(char* chNomBase,int& iNbRelations,int& iXb,int& iYb,int& iXh,int&
iYh,int& iPvl);
    BOOL LireNumero(int& iNumero);
    BOOL LirePbl(int& iPbl);
    BOOL LirePvl(int& iPvl);
    BOOL LireNbRelations(int& iNbRelations);
    BOOL LireRelation(long adr,char* chNomRelation,int& iType,int& iNbAttributs,int&
iNrec,char* chNom2,char* chNom3,char* chNom4,int& plibz,int& pliba);
    BOOL LireRelationNom(long adr,char* chNomRelation);
    BOOL LireRelationTypeNbAttributs(long adr,int& iType,int& iNbAttributs);
    BOOL LireAttribut(long adr,char* chNomAttribut,int& iTypeAttribut,int& iNbvalb,int&
iPtrval,int& iNbcar,char* chDescription);
    BOOL LireAttributNbvalbPtrval(long adr,int& iNbvalb,int& iPtrval);
    BOOL EcrireEntete1(int iDefecr,int iPbl,int iVersion,const char* chUtilisateur);
    BOOL EcrireEntete2(const char* chNomBase,int iNbRelations,int iXb,int iYb,int iXh,int
iYh,int iPvl);
    BOOL EcrireUtilisateurAdmi();
    BOOL EcrireUtilisateurNone();
    BOOL EcrireNumero();
    BOOL EcrirePbl(int iPbl);
    BOOL EcrirePvl(int iPvl);
    BOOL EcrireNbRelations(int iNbRelations);
    BOOL EcrireRelation(long adr,const char* chNomRelation,int iTypeRelation,int
iNbAttributs,int iNrec,const char* chNom2,const char* chNom3,const char* chNom4,int plibz,int
pliba);
    BOOL EcrireRelationNom(long adr,const char* chNomRelation);
    BOOL EcrireRelationDirMosaic(long adr,const char* chDirMosaic);
    BOOL EcrireRelationNbAttributsNrec(long adr,int iNbAttributs,int iNrec);
    BOOL EcrireRelationPlibz(long adr,int iPtrLibre);
    BOOL EcrireRelationPliba(long adr,int iPtrLibre);
    BOOL EcrireAttribut(long adr,const char* chNomAttribut,int iTypeAttribut,int iNbvalb,int
iPtrval,int iNbcar,const char* chDescription);
    BOOL EcrireAttributNom(long adr,const char* chNomAttribut);
    BOOL EcrireAttributType(long adr,int iTypeAttribut);
    BOOL EcrireAttributNbvalPtrval(long adr,int iNbvalb,int iPtrval);
    BOOL EcrireAttributDescription(long adr,const char* chDescription);
    BOOL EcrireLigneBlanche(long adr);
    BOOL DecaleMoins(long adrDebut,long adrFin,int iDecalage);
    BOOL DecalePlus(long adrDebut,long adrFin,int iDecalage);
  };
```

Nous avons déjà vu dans le paragraphe précédent (classe `CBase`) comment cette classe est utilisée dans SAVATECA pour charger la structure d'une base de données.

Pour donner un exemple de la façon dont est maintenu le dictionnaire, voici le code de la méthode `EcrireRelation()` qui écrit dans le fichier la description d'une relation :

```
BOOL CDico::EcrireRelation(long adr,const char* chNomRelation,int iTypeRelation,int
iNbAttributs,int iNrec,const char* chNom2,const char* chNom3,const char* chNom4,int plibz,int
pliba)
  {
  if(m_FileBase == NULL)return FALSE;
  memset(m_chBuffer,' ',256);

  char chNomRelationLocal[20];
  char chNom2Local[_MAX_PATH];
  char chNom3Local[_MAX_PATH];
  char chNom4Local[_MAX_PATH];
  strcpy(chNomRelationLocal,chNomRelation);
  strcpy(chNom2Local,chNom2);
  strcpy(chNom3Local,chNom3);
  strcpy(chNom4Local,chNom4);
  if(strlen(chNom2Local) == 0)strcpy(chNom2Local,"aucun");
  if(strlen(chNom3Local) == 0)strcpy(chNom3Local,"aucun");
  if(strlen(chNom4Local) == 0)strcpy(chNom4Local,"aucun");

  int iTaille=strlen(chNomRelationLocal);
  for(int i=iTaille;i < 16;i++)chNomRelationLocal[i]=' ';
  chNomRelationLocal[16]=0;

  iTaille=strlen(chNom2Local);
  for(i=iTaille;i < 55;i++)chNom2Local[i]=' ';
  chNom2Local[55]=0;

  iTaille=strlen(chNom3Local);
  for(i=iTaille;i < 8;i++)chNom3Local[i]=' ';
  chNom3Local[8]=0;

  iTaille=strlen(chNom4Local);
  for(i=iTaille;i < 8;i++)chNom4Local[i]=' ';
  chNom4Local[8]=0;

  if(fseek(m_FileBase,adr,SEEK_SET) != 0)return FALSE;
  if(iTypeRelation == 5)
    {
    int i=sprintf(m_chBuffer,"%-16.16s%1d%4d%4d%-
55.55s",chNomRelationLocal,iTypeRelation,iNbAttributs,iNrec,chNom2Local);
    m_chBuffer[80]=' ';
    if(i != 80)
      {
      Fermer();
      return FALSE;
      }
    }
  else
    {
    chNom2Local[8]=0;
    int i=sprintf(m_chBuffer,"%-16.16s%1d%4d%4d%-8.8s%-8.8s%-
8.8s%8d%8d",chNomRelationLocal,iTypeRelation,iNbAttributs,iNrec,chNom2Local,chNom3Local,chNom4
Local,plibz,pliba);
    m_chBuffer[65]=' ';
    if(i != 65)
      {
      Fermer();
      return FALSE;
      }
    }

  if(fwrite(m_chBuffer,m_iTailleRecord,1,m_FileBase) != 1)
    {
    Fermer();
    return FALSE;
    }
```

```
  return TRUE;
  }
```

De même, la méthode `EcrireAttribut()` écrit dans le fichier la description d'un attribut d'une relation :

```
BOOL CDico::EcrireAttribut(long adr,const char* chNomAttribut,int iTypeAttribut,int
iNbvalb,int iPtrval,int iNbcar,const char* chDescription)
  {
  if(m_FileBase == NULL)return FALSE;
  memset(m_chBuffer,' ',256);

  //--- iNbcar non utilisé pour le moment

  //--- mise en bonne forme des chaines de caracteres
  char chNomAttributLocal[20];
  char chDescriptionLocal[100];
  strcpy(chNomAttributLocal,chNomAttribut);
  strcpy(chDescriptionLocal,chDescription);

  int iTaille=strlen(chNomAttributLocal);
  for(int i=iTaille;i < 16;i++)chNomAttributLocal[i]=' ';
  chNomAttributLocal[16]=0;

  iTaille=strlen(chDescriptionLocal);
  for(i=iTaille;i < 40;i++)chDescriptionLocal[i]=' ';
  chDescriptionLocal[40]=0;

  i=sprintf(m_chBuffer,"%-16.16s%1d%8d%8d",chNomAttributLocal,iTypeAttribut,iNbvalb,iPtrval);
  m_chBuffer[33]=' ';
  if(i != 33)
    {
    Fermer();
    return FALSE;
    }
  memcpy(&m_chBuffer[40],chDescriptionLocal,40);

  if(fseek(m_FileBase,adr,SEEK_SET) != 0)return FALSE;
  if(fwrite(m_chBuffer,m_iTailleRecord,1,m_FileBase) != 1)
    {
    Fermer();
    return FALSE;
    }

  return TRUE;
  }
```

Voici le code utilisé dans SAVATECA pour écrire dans le dictionnaire des données la description d'une nouvelle relation. Le type de la relation correspond au type d'implantation spatiale (zone, ligne, point, pixel, non localisé) :

```
  CDico dico;
  if(dico.Ouvrir(base.m_chNomFichierBase,MODE_READWRITE) == FALSE)return FALSE;
  dico.LirePbl(iPbl);
  dico.LireNbRelations(iNbRelations);

  adr=dico.GetTailleRecord()*(iPbl-1);
  adrRelation=adr;

  strcpy(m_chNomFichierData,"aucun   ");
  strcpy(m_chNomFichierFeuille,"aucun   ");
  strcpy(m_chNomFichierArc,"aucun   ");
  switch(iTypeRelation)
    {
    case 1:
      if(base.NouveauFichier(1,m_chNomFichierData) == FALSE)return FALSE;

  dico.EcrireRelation(adrRelation,chNomRelation,iTypeRelation,m_iNbatt,iNrec,m_chNomFichierFeu
ille,m_chNomFichierData,m_chNomFichierArc,1,1);
      iPbl++;
      iNbRelations++;
      break;

    case 3:
```

```
       if(base.NouveauFichier(1,m_chNomFichierData) == FALSE)return FALSE;
       if(base.NouveauFichier(2,m_chNomFichierFeuille) == FALSE)return FALSE;

  dico.EcrireRelation(adrRelation,chNomRelation,iTypeRelation,m_iNbatt,iNrec,m_chNomFichierFeu
ille,m_chNomFichierData,m_chNomFichierArc,1,1);
       iPbl++;
       iNbRelations++;
       break;

     case 2:
     case 4:
       if(base.NouveauFichier(1,m_chNomFichierData) == FALSE)return FALSE;
       if(base.NouveauFichier(2,m_chNomFichierFeuille) == FALSE)return FALSE;
       if(base.NouveauFichier(3,m_chNomFichierArc) == FALSE)return FALSE;

  dico.EcrireRelation(adrRelation,chNomRelation,iTypeRelation,m_iNbatt,iNrec,m_chNomFichierFeu
ille,m_chNomFichierData,m_chNomFichierArc,1,1);
       iPbl++;
       iNbRelations++;
       break;

     case 5:
     //--- type mosaique
       {
       char chDirMosaic[_MAX_PATH];

       strcpy(chDirMosaic,(LPCTSTR) m_strDirMosaic);
       if(CreerMosaique(chNomRelation,chDirMosaic) == FALSE)
         {
         switch(config.m_iLangage)
           {
           case FRANCAIS:
           default:
             AfxMessageBox("Mosaïque : création impossible !",MB_OK|MB_ICONEXCLAMATION);
             break;
           case ESPAGNOL:
             AfxMessageBox("Mosaico : creación imposible !",MB_OK|MB_ICONEXCLAMATION);
             break;
           case ANGLAIS:
             AfxMessageBox("Mosaïque : creation aborted !",MB_OK|MB_ICONEXCLAMATION);
             break;
           }

         return FALSE;
         }


  dico.EcrireRelation(adrRelation,chNomRelation,iTypeRelation,m_iNbatt,iNrec,chDirMosaic,m_chN
omFichierData,m_chNomFichierArc,1,1);
       iPbl++;
       iNbRelations++;
       }
       break;

     default:
       break;
     }

  dico.EcrirePbl(iPbl);
  dico.EcrireNbRelations(iNbRelations);
  dico.EcrireNumero();
  dico.Fermer();
  …
```

La méthode `EcrireNumero()` permet d'actualiser une variable indiquant la version du dictionnaire, après une modification du schéma par l'administrateur. Tous les programmes client (comme SAVANE, SAVAMER, SAVEDIT) lisent ce numéro dans le fichier *base* à chaque utilisation de la base de données : si elle a été modifiée depuis le dernier accès, le schéma est mis à jour en lisant de nouveau le dictionnaire. Cette procédure est essentielle lorsqu'une même base est utilisée en réseau et est modifiée par l'administrateur pendant sa consultation par des utilisateurs.

A.1.4. La classe CFpacc9

La classe `CFpacc9` permet de manipuler les vues externes. Elles est identique dans l'ensemble des modules du système. Les procédures de lecture sont utilisées par tous les modules qui utilisent les vues externes (SAVANE, SAVAMER, SAVEDIT). La classe permet de gérer des groupes de relations ou d'attributs au niveau des vues externes, de manière à présenter la vue externe avec des niveaux hiérarchiques structurés.

```
class CFpaccV9
  {
  public:
    void EcrireUneVue(FILE* FileAcces,const char* chCode);
    void EcrireEnteteAcces(FILE* FileAcces,const char* chNomBase,int iNbVues);
    void EcrireEnteteVue(FILE* FileAcces,const char* chCode,int
iNbRelations,CGroupesDeRelationsDansVue* pGroupes);
    void EcrireEnteteRelation(FILE* FileAcces,int iNoRelation,int
iNbAttributs,CGroupesDeAttributsDansVue* pGroupes);

  public:

    BOOL Upgrade();
    void LireEnteteAcces(FILE* FileAcces,int& iNbVues);
    void LireEnteteVue(FILE* FileAcces,char* chCode,int&
iNbRelations,CGroupesDeRelationsDansVue* pGroupes);
    void LireEnteteRelation(FILE* FileAcces,int& iNoRelation,int&
iNbAttributs,CGroupesDeAttributsDansVue* pGroupes);

    BOOL LireUneVue(int iNumero,char* chCode,int& iNbRelations);
    BOOL ModifierUneVue(int iNumero);
    BOOL AjouterUneVue(char* chCode);
    BOOL SupprimerUneVue(int iNumero);
  };

class CGroupesDeRelationsDansVue
  {
  private:
  public:
    int m_iNbGroupes;
    char m_ctabArbre[NB_MAX_RELATIONS + 2*NB_MAX_GROUPES_DE_RELATIONS];
    char m_chtabNomGroupe[NB_MAX_GROUPES_DE_RELATIONS][33];

    void Init();
    void SupprimerUneRelation(int iNoRelation);
  };

class CGroupesDeAttributsDansVue
  {
  private:
  public:
    int m_iNbGroupes;
    char m_ctabArbre[NB_MAX_ATTRIBUTS + 2*NB_MAX_GROUPES_DE_ATTRIBUTS];
    char m_chtabNomGroupe[NB_MAX_GROUPES_DE_ATTRIBUTS][33];

    void Init();
    void Copier(CGroupesDeAttributsDansVue* pGroupeInitial);
    void SupprimerUnAttribut(int iNoAttribut);
  };
```

Par exemple, la méthode `LireEnteteVue()` lit dans le fichier `fpacc` le nombre de relations de la vue, le nombre et la description des groupes de relation. Ces deux derniers paramètres sont conservés dans un objet de la classe `CGroupesDeRelationsDansVue`. La description des groupes de relation comprend une liste donnant la place de chaque groupe et de chaque relation dans la vue (`m_ctabArbre`), ainsi que le nom des groupes (`m_chtabNomGroupe`). La liste a la forme suivante : …`bbbbgbbgbbbbfbbfbbb`… (`b` représente une relation, `g` le

début d'un groupe, `f` la fin du groupe). La description des groupes d'attribut pour chaque relation est basé sur le même principe.

```
void CFpaccV9::LireEnteteVue(FILE* FileAcces,char* chCode,int&
iNbRelations,CGroupesDeRelationsDansVue* pGroupes)
  {
  char chBuffer[33];
  memset(chBuffer,' ',32);
  strcpy(chCode,"");
  iNbRelations=0;
  pGroupes->Init();

  if(fread(chBuffer,28,1,FileAcces) == 1)
    {
    chBuffer[28]=0;
    sscanf(chBuffer,"%4s%4d%4d",chCode,&iNbRelations,&pGroupes->m_iNbGroupes);
    chCode[4]=0;

    //--- groupes de relations
    for(int i=0;i < pGroupes->m_iNbGroupes;i++)
      {
      fread(&pGroupes->m_chtabNomGroupe[i],32,1,FileAcces);
      pGroupes->m_chtabNomGroupe[i][32]=0;
      }
    i=iNbRelations + 2*pGroupes->m_iNbGroupes;
    if(i > 0)fread(pGroupes->m_ctabArbre,i,1,FileAcces);
    }
  }
```

A.1.5. La classe CIndexAttribut

La classe `CIndexAttribut` permet d'indexer temporairement les modalités d'un attribut nominal, de manière à retrouver rapidement le code d'une modalité si elle existe dans la base. L'indexation utilise une table d'entrée sur trois caractères à laquelle correspond la fonction de hachage `Paquet()`. Elle est utilisée dans SAVATECA lors de l'intégration descriptive, et dans SAVANE pour trouver le code interne d'une modalité.

```
class CIndexAttribut
  {
  private:
    CBase* m_pBase;
    int m_itabIndex[15626];
    int m_itabNombre[15626];
    unsigned char *m_ctabCle;

    int m_iTypeAttr;
    int m_iNbcarAttr;
    int m_iNbvalb;
    int m_iPtrval;

  public:
    CIndexAttribut(CBase* pBase,int iNoth,int iNoatt);
    ~CIndexAttribut();

    BOOL Init();
    int Recherche(char *chNomValeur);
    void Fermer();
  };
```

L'implémentation des méthodes `Init()` et `Recherche()` permet de montrer la structure de codage des valeurs nominales dans une base SAVANE :

```
BOOL CIndexAttribut::Init()
  {
  //--- allocation mémoire
```

```
  m_ctabCle=(unsigned char *) malloc(m_iNbvalb*(m_iNbcarAttr + sizeof(int))*sizeof(unsigned
char));
  if(m_ctabCle == NULL)
    {
    ::ErreurDeMemoire();
    return FALSE;
    }

  //--- initialisation de m_ctabCle
  memset(m_ctabCle,' ',m_iNbvalb*(m_iNbcarAttr + sizeof(int)));

  //--- initialisation de la table d'index
  int i;
  for(i=0;i < 15626;i++)
    {
    m_itabIndex[i]=0;
    m_itabNombre[i]=0;
    }

  if(m_iNbvalb > 0)
    {
    char chNomValeur[NB_MAX_CHARACTER];
    int iVal=0;
    int iNbRec=m_iNbcarAttr+sizeof(int);

    //--- lecture des valeurs
    FILE* FileValeurs;
    FileValeurs=fopen(m_pBase->m_chNomFichierValeurs,"rb");

    chNomValeur[m_iNbcarAttr]='\0';
    int iPtrval=(m_iPtrval-1)*m_iNbcarAttr;
    fseek(FileValeurs,iPtrval,SEEK_SET);
    for(i=0;i < m_iNbvalb;i++)
      {
      iVal=i+1;
      fread(&chNomValeur[0],m_iNbcarAttr,1,FileValeurs);
      memcpy(&m_ctabCle[iNbRec*i],chNomValeur,m_iNbcarAttr);
      memcpy(&m_ctabCle[iNbRec*i+m_iNbcarAttr],&iVal,sizeof(int));
      }
    fclose(FileValeurs);

    //--- tri des valeurs, ordre donné par paquet
    qsort(m_ctabCle,m_iNbvalb,iNbRec,ComparClePaquet);

    //--- creation de la table d'index
    int irep1,irep1bis;
    int iPtr=0;
    int iNbIndex=0;

    memcpy(chNomValeur,&m_ctabCle[iPtr],m_iNbcarAttr);
    chNomValeur[m_iNbcarAttr]=0;
    irep1=Paquet(chNomValeur);
    m_itabIndex[irep1]=iPtr;
    iPtr+=iNbRec;
    iNbIndex++;
    for(i=1;i < m_iNbvalb;i++)
      {
      memcpy(chNomValeur,&m_ctabCle[iPtr],m_iNbcarAttr);
      chNomValeur[m_iNbcarAttr]=0;
      irep1bis=Paquet(chNomValeur);
      if(irep1bis != irep1)
        {
        m_itabNombre[irep1]=iNbIndex;
        iNbIndex=0;
        irep1=irep1bis;
        m_itabIndex[irep1]=iPtr;
        }
      iNbIndex++;
      iPtr+=iNbRec;
      }
    m_itabNombre[irep1]=iNbIndex;
    }

  return TRUE;
  }

int CIndexAttribut::Recherche(char* chNom)
  {
  int i,j,irep1,iPtr,iNb,iVal;
  char chNomValeur[NB_MAX_CHARACTER];
```

```
   char chNomTableau[NB_MAX_CHARACTER];

   int iNbRec=m_iNbcarAttr+sizeof(int);

   i=0;
   while(i < m_iNbcarAttr)
      {
      if(chNom[i] == 0)break;
      chNomValeur[i]=chNom[i];
      i++;
      }
   while(i < m_iNbcarAttr)
      {
      chNomValeur[i]=' ';
      i++;
      }
   chNomValeur[m_iNbcarAttr]=0;

   irep1=Paquet(chNomValeur);
   iPtr=m_itabIndex[irep1];
   iNb=m_itabNombre[irep1];
   iVal=0;
   for(i=0;i < iNb;i++)
      {
      memcpy(chNomTableau,&m_ctabCle[iPtr],m_iNbcarAttr);
      for(j=0;j < m_iNbcarAttr;j++)
         {
         if(chNomTableau[j] != chNomValeur[j])break;
         }
      if(j == m_iNbcarAttr)
         {
         memcpy(&iVal,&m_ctabCle[iPtr+m_iNbcarAttr],sizeof(int));
         return iVal;
         }
      iPtr+=iNbRec;
      }
   return iVal;
   }

static int __cdecl ComparClePaquet(const void *arg1,const void *arg2)
   {
   g_i1=Paquet((char *)arg1);
   g_i2=Paquet((char *)arg2);
   if(g_i1 < g_i2)return -1;
   if(g_i1 == g_i2)return 0;
   if(g_i1 > g_i2)return 1;
   return 0;
   }

static int Paquet(char* chNom)
   {
   if(chNom[0] <= 47)g_irep1=0;
   else if(chNom[0] <= 57)g_irep1=int(chNom[0])-47;
   else if(chNom[0] <= 65)g_irep1=11;
   else if(chNom[0] <= 70)g_irep1=12;
   else if(chNom[0] <= 75)g_irep1=13;
   else if(chNom[0] <= 80)g_irep1=14;
   else if(chNom[0] <= 85)g_irep1=15;
   else if(chNom[0] <= 90)g_irep1=16;
   else if(chNom[0] <= 95)g_irep1=17;
   else if(chNom[0] <= 100)g_irep1=18;
   else if(chNom[0] <= 105)g_irep1=19;
   else if(chNom[0] <= 110)g_irep1=20;
   else if(chNom[0] <= 115)g_irep1=21;
   else if(chNom[0] <= 120)g_irep1=22;
   else if(chNom[0] <= 135)g_irep1=23;
   else g_irep1=24;

   if(chNom[1] <= 47)g_irep2=0;
   else if(chNom[1] <= 57)g_irep2=int(chNom[1])-47;
   else if(chNom[1] <= 65)g_irep2=11;
   else if(chNom[1] <= 70)g_irep2=12;
   else if(chNom[1] <= 75)g_irep2=13;
   else if(chNom[1] <= 80)g_irep2=14;
   else if(chNom[1] <= 85)g_irep2=15;
   else if(chNom[1] <= 90)g_irep2=16;
   else if(chNom[1] <= 95)g_irep2=17;
   else if(chNom[1] <= 100)g_irep2=18;
   else if(chNom[1] <= 105)g_irep2=19;
   else if(chNom[1] <= 110)g_irep2=20;
```

```
  else if(chNom[1] <= 115)g_irep2=21;
  else if(chNom[1] <= 120)g_irep2=22;
  else if(chNom[1] <= 135)g_irep2=23;
  else g_irep2=24;

  if(chNom[2] <= 47)g_irep3=0;
  else if(chNom[2] <= 57)g_irep3=int(chNom[2])-47;
  else if(chNom[2] <= 65)g_irep3=11;
  else if(chNom[2] <= 70)g_irep3=12;
  else if(chNom[2] <= 75)g_irep3=13;
  else if(chNom[2] <= 80)g_irep3=14;
  else if(chNom[2] <= 85)g_irep3=15;
  else if(chNom[2] <= 90)g_irep3=16;
  else if(chNom[2] <= 95)g_irep3=17;
  else if(chNom[2] <= 100)g_irep3=18;
  else if(chNom[2] <= 105)g_irep3=19;
  else if(chNom[2] <= 110)g_irep3=20;
  else if(chNom[2] <= 115)g_irep3=21;
  else if(chNom[2] <= 120)g_irep3=22;
  else if(chNom[2] <= 135)g_irep3=23;
  else g_irep3=24;

  //--- codage en base 25
  return (g_irep1*625 + g_irep2*25 + g_irep3);
  }
```

A.1.6. La classe CIntegrationObjetsLocalises

La classe `CintegrationObjetLocalisés` regroupe les opérations permettant d'effectuer l'intégration de nouveaux objets dans une relation. Elle est utilisée dans toutes les opérations d'intégration d'objets localisés, à partir des documents de saisie graphique :

```
class CIntegrationObjetsLocalises
  {
  public:
    CIntegrationObjetsLocalises(int iNoth,int iNoatt);
    ~CIntegrationObjetsLocalises();

    int Init(char *chNomDocumentMygale,const char *chRepertoire);
    void MessageErreur(int iNumero);
    BOOL Integrer();

  private:
    int m_iNoth;
    int m_iNoatt;
    int m_iVersionMygale;
    int m_iTypeDocument;
    int m_iTypeSaisie;
    int m_iNbcharCle;
    char m_chRepertoire[_MAX_PATH];
    char m_chNomDocument[_MAX_PATH];
    double m_dXCalageProjection[2];
    double m_dYCalageProjection[2];
    int m_iXCalageTable[2];
    int m_iYCalageTable[2];
    double m_dA,m_dB;

    BOOL IntegrerObjetsZoneVersion1();
    BOOL IntegrerObjetsZoneVersion7();
    BOOL IntegrerObjetsLigneVersion1();
    BOOL IntegrerObjetsLigneVersion7();
    BOOL IntegrerObjetsPointVersion1();
    BOOL IntegrerObjetsPointVersion7();

    void CentreLigne(double* dtabX,double *dtabY,int iNbpt,double &dXSav, double &dYsav);
  };
```

Par exemple, la méthode IntegrerObjetsPointVersion7() permet de créer une nouvelle feuille dans une relation de type point, à partir d'un document de type

point provenant de SAVEDIT. Le processus crée les objets, crée la feuille dans le fichier d'indexation, puis intègre la clé descriptive pour chaque objet.

```
BOOL CIntegrationObjetsLocalises::IntegrerObjetsPointVersion7()
  {
  int iStep,iCompteur;
  int i;
  long adr;
  float fXSav,fYSav;
  float fXminFeuille,fYminFeuille,fXmaxFeuille,fYmaxFeuille;
  double dLong,dColat;
  double dXSav,dYSav;
  double dVal;
  double dXminFeuille,dYminFeuille,dXmaxFeuille,dYmaxFeuille;
  char chTitreProgress[100];
  char chNomFichier[_MAX_PATH];
  char buffer[1000];

  long fXSavUnix,fYSavUnix;

  memset(buffer,0,1000);

  int iSizeIdentifiantTuple=2*SIZECOOR;

  //--- vérification de l'écriture dans le fichier base
  CDico dico;
  if(dico.Ouvrir(base.m_chNomFichierBase,MODE_READWRITE) == FALSE) return FALSE;
  dico.Fermer();

  //--- ouverture des fichiers savedit
  strcpy(chNomFichier,m_chRepertoire);
  strcat(chNomFichier,"\\");
  strcat(chNomFichier,m_chNomDocument);
  strcat(chNomFichier,".pt");
  FILE* FileMygalePoint=fopen(chNomFichier,"rb");
  if(FileMygalePoint == NULL)
    {
    strcpy(chNomFichier,m_chRepertoire);
    strcat(chNomFichier,"\\");
    strcat(chNomFichier,m_chNomDocument);
    strcat(chNomFichier,".PT");
    FileMygalePoint=fopen(chNomFichier,"rb");
    if(FileMygalePoint == NULL)
      {
      ::ErreurDeFichier(chNomFichier);
      return FALSE;
      }
    }

  //--- init fenêtre de feuille
  dXminFeuille=21600.;
  dYminFeuille=10800.;
  dXmaxFeuille=-21600.;
  dYmaxFeuille=-10800.;

  //--- ouverture du fichier savane des tuples
  FILE* FileSavanePoint=fopen(base.m_pRelations[m_iNoth]->m_chFichierZone,"rb+");
  if(FileSavanePoint == NULL)FileSavanePoint=fopen(base.m_pRelations[m_iNoth]-
>m_chFichierZone,"wb");
  if(FileSavanePoint == NULL)
    {
    ::ErreurDeFichierEnEcriture(base.m_pRelations[m_iNoth]->m_chFichierZone);
    fclose(FileMygalePoint);
    return FALSE;
    }

  //--- init des valeurs d'attributs
  long vUnix[NB_MAX_ATTRIBUTS];
  for(i=0;i < NB_MAX_ATTRIBUTS;i++)vUnix[i]=DosFloatToUnix(FINCONNU);
  for(i=1;i <= base.m_pRelations[m_iNoth]->m_iNba;i++)
    {
    if(base.m_pRelations[m_iNoth]->m_pAttributs[i]->m_iType == 1)vUnix[i-
1]=DosFloatToUnix(0.F);
    }

  //--- intégration graphique et descriptive
  //--- init des pointeurs
  int iSavanePtrTupleFeuille=base.m_pRelations[m_iNoth]->m_iPtrLibreTuple;
  int iNba=base.m_pRelations[m_iNoth]->m_iNba;
```

```
   int iNbTupleFeuille=0;
   int iSavanePtrTuple=iSavanePtrTupleFeuille;

   int iTailleRecord=16; //--- pour dX et dY
   if(m_iTypeSaisie == SAISIE_VALEUR || m_iTypeSaisie == SAISIE_CLEVALEUR)iTailleRecord+=8;
   if(m_iTypeSaisie == SAISIE_CLE || m_iTypeSaisie ==
SAISIE_CLEVALEUR)iTailleRecord+=m_iNbcharCle;

   adr=(long) ((iSavanePtrTuple-1)*(iSizeIdentifiantTuple + (iNba*SIZEVAL)));
   fseek(FileSavanePoint,adr,SEEK_SET);
   while(fread(buffer,iTailleRecord,1,FileMygalePoint) == 1)
      {
      memcpy(&dLong,&buffer[0],sizeof(double));
      memcpy(&dColat,&buffer[8],sizeof(double));
      wind.GeoToSav(dLong,dColat,fXSav,fYSav);

      if(m_iNoatt > 0 && (m_iTypeSaisie == SAISIE_VALEUR || m_iTypeSaisie == SAISIE_CLEVALEUR))
         {
         //--- lecture et intégration d'une valeur numérique
         memcpy(&dVal,&buffer[16],sizeof(double));
         vUnix[m_iNoatt-1]=DosFloatToUnix((float) dVal);
         }

      fXSavUnix=DosFloatToUnix(fXSav);
      fYSavUnix=DosFloatToUnix(fYSav);

      //--- fenetre de la feuille
      dXSav=(double) fXSav;
      dYSav=(double) fYSav;
      if(dXSav < dXminFeuille)dXminFeuille=dXSav;
      if(dYSav < dYminFeuille)dYminFeuille=dYSav;
      if(dXSav > dXmaxFeuille)dXmaxFeuille=dXSav;
      if(dYSav > dYmaxFeuille)dYmaxFeuille=dYSav;

      //--- écriture dans le fichier
      fwrite(&fXSavUnix,SIZECOOR,1,FileSavanePoint);
      fwrite(&fYSavUnix,SIZECOOR,1,FileSavanePoint);
      for(i=0;i < iNba;i++)fwrite(&vUnix[i],SIZEVAL,1,FileSavanePoint);
      iNbTupleFeuille++;
      iSavanePtrTuple++;
      }

   if(iNbTupleFeuille == 0)
      {
      fclose(FileMygalePoint);
      fclose(FileSavanePoint);
      return TRUE;
      }

   //--- mise à jour du fichier feuille
   FILE* FileSavaneFeuille=fopen(base.m_pRelations[m_iNoth]->m_chFichierFeuille,"rb+");
   if(FileSavaneFeuille == NULL)FileSavaneFeuille=fopen(base.m_pRelations[m_iNoth]-
>m_chFichierFeuille,"wb");
   if(FileSavaneFeuille == NULL)
      {
      ::ErreurDeFichierEnEcriture(base.m_pRelations[m_iNoth]->m_chFichierFeuille);
      fclose(FileMygalePoint);
      fclose(FileSavanePoint);
      return FALSE;
      }

   fXminFeuille=(float) dXminFeuille;
   fYminFeuille=(float) dYminFeuille;
   fXmaxFeuille=(float) dXmaxFeuille;
   fYmaxFeuille=(float) dYmaxFeuille;

   int iNumeroFeuille=0;
   int iNbArcFeuille=0;
   int iSavanePtrArcFeuille=0;
   while(fgets(buffer,90,FileSavaneFeuille) != 0)iNumeroFeuille++;
   iNumeroFeuille++;
   fprintf(FileSavaneFeuille,"%5d%-12.12s%8.2f%8.2f%8.2f%8.2f%8d%8d%8d%8d\n",
      iNumeroFeuille,m_chNomDocument,
      fXminFeuille,fYminFeuille,fXmaxFeuille,fYmaxFeuille,
      iNbTupleFeuille,iSavanePtrTupleFeuille,
      iNbArcFeuille,iSavanePtrArcFeuille);

   fclose(FileSavaneFeuille);

   //--- mise à jour des pointeurs libres dans le fichier base pour la relation
```

```
if(dico.Ouvrir(base.m_chNomFichierBase,MODE_READWRITE))
   {
   adr=base.m_pRelations[m_iNoth]->m_iPtr;
   dico.EcrireRelationPlibz(adr,iSavanePtrTuple);
   dico.Fermer();

   base.m_pRelations[m_iNoth]->m_iPtrLibreTuple=iSavanePtrTuple;
   base.m_pRelations[m_iNoth]->m_iPtrLibreArc=0;
   }

//--- intégration de l'attribut clé
if(m_iNoatt > 0 && (m_iTypeSaisie == SAISIE_CLE || m_iTypeSaisie == SAISIE_CLEVALEUR))
   {
   char chCle[33],chClebis[33];
   int iTypeAttribut=base.m_pRelations[m_iNoth]->m_pAttributs[m_iNoatt]->m_iType;
   int iNbvalb=base.m_pRelations[m_iNoth]->m_pAttributs[m_iNoatt]->m_iNbvalb;
   int iPtrval=base.m_pRelations[m_iNoth]->m_pAttributs[m_iNoatt]->m_iPtrval;
   int iNbcarSavane=base.m_pRelations[m_iNoth]->m_pAttributs[m_iNoatt]->m_iNbcar;

   int iNbcarMygale=m_iNbcharCle;
   if(iNbcarMygale > iNbcarSavane)iNbcarMygale=iNbcarSavane;

   //--- allocations mémoire pour les tableaux de code par objet
   float *ftabCode=(float *) malloc((iNbTupleFeuille + 1)*sizeof(float));
   char *ctabNewCle=(char *) malloc((iNbTupleFeuille + 1)*iNbcarSavane + 1);
   if(ftabCode == NULL || ctabNewCle == NULL)
      {
      ::ErreurDeMemoire();
      if(ftabCode != NULL)free(ftabCode);
      if(ctabNewCle != NULL)free(ctabNewCle);
      fclose(FileMygalePoint);
      fclose(FileSavanePoint);
      return FALSE;
      }

   for(i=0;i <= iNbTupleFeuille;i++)ftabCode[i]=0.F;
   memset(ctabNewCle,' ',(iNbTupleFeuille+1)*iNbcarSavane);

   //--- emplacement de la clé dans le document mygale
   int iDebutDansBuffer=16;
   if(m_iTypeSaisie == SAISIE_CLE || m_iTypeSaisie == SAISIE_VALEUR)iDebutDansBuffer=16;
   else if(m_iTypeSaisie == SAISIE_CLEVALEUR)iDebutDansBuffer=24;

   pMainFrame->StepProgress();
   int iNbNewValeurs=0;
   int iCode;
   int iNoTuple=0;
   if(iNbvalb > 0)
      {
      //--- indexation des valeurs existentes
      CIndexAttribut index(&base,m_iNoth,m_iNoatt);
      if(index.Init())
         {
         //--- codification des clés
         iNoTuple=0;
         fseek(FileMygalePoint,0L,SEEK_SET);
         while(fread(buffer,iTailleRecord,1,FileMygalePoint) == 1)
            {
            pMainFrame->StepProgress();
            memset(chCle,' ',iNbcarMygale);
            memcpy(chCle,&buffer[iDebutDansBuffer],iNbcarMygale);
            chCle[iNbcarMygale]='\0';
            strstripall(chCle);
            if(strlen(chCle) == 0)strcpy(chCle,"Valeur inconnue");
            iCode=index.Recherche(chCle);

            if(iCode == 0)
               {
               //--- nouvelle valeur
               i=0;
               while(i < iNbNewValeurs && iCode == 0)
                  {
                  memcpy(chClebis,&ctabNewCle[i*iNbcarSavane],iNbcarSavane);
                  chClebis[iNbcarSavane]='\0';
                  strstrip(chClebis);
                  if(strcmp(chCle,chClebis) == 0)iCode=iNbvalb+i+1;
                  i++;
                  }

               if(iCode == 0)
```

```
                {
                i=strlen(chCle);
                memcpy(&ctabNewCle[iNbNewValeurs*iNbcarSavane],chCle,i);
                iNbNewValeurs++;
                iCode=iNbvalb + iNbNewValeurs;
                }
              }
            ftabCode[iNoTuple]=(float) iCode;
            iNoTuple++;
            }
          }
        }
      else
        {
        //--- pas de valeurs déjà introduites, codification directe des clés
        iNoTuple=0;
        fseek(FileMygalePoint,0L,SEEK_SET);
        while(fread(buffer,iTailleRecord,1,FileMygalePoint) == 1)
          {
          memset(chCle,' ',iNbcarMygale);
          memcpy(chCle,&buffer[iDebutDansBuffer],iNbcarMygale);
          chCle[iNbcarMygale]='\0';
          strstripall(chCle);
          if(strlen(chCle) == 0)strcpy(chCle,"Valeur inconnue");

          iCode=0;
          i=0;
          while(i < iNbNewValeurs && iCode == 0)
            {
            memcpy(chClebis,&ctabNewCle[i*iNbcarSavane],iNbcarSavane);
            chClebis[iNbcarSavane]='\0';
            strstrip(chClebis);
            if(strcmp(chCle,chClebis) == 0)iCode=iNbvalb+i+1;
            i++;
            }

          if(iCode == 0)
            {
            i=strlen(chCle);
            memcpy(&ctabNewCle[iNbNewValeurs*iNbcarSavane],chCle,i);
            iNbNewValeurs++;
            iCode=iNbvalb + iNbNewValeurs;
            }
          ftabCode[iNoTuple]=(float) iCode;
          iNoTuple++;
          }
        }

      //--- écriture dans les tuples des relations
      long fValUnix;

      int iPtr=iSavanePtrTupleFeuille;
      iNoTuple=0;
      while(iNoTuple < iNbTupleFeuille)
        {
        pMainFrame->StepProgress();
        fValUnix=DosFloatToUnix(ftabCode[iNoTuple]);
        adr=(long) ((iNba*SIZEVAL + iSizeIdentifiantTuple)*(iPtr - 1) + iSizeIdentifiantTuple +
(m_iNoatt-1)*SIZEVAL);
        fseek(FileSavanePoint,adr,SEEK_SET);
        fwrite(&fValUnix,SIZEVAL,1,FileSavanePoint);

        iPtr++;
        iNoTuple++;
        }

      free(ftabCode);

      //--- insertion des nouvelles valeurs
      base.InsertionNewValeurs(ctabNewCle,iNbNewValeurs,m_iNoth,m_iNoatt);
      free(ctabNewCle);
      }

    fclose(FileMygalePoint);
    fclose(FileSavanePoint);

    return TRUE;
    }
```

A.2. SAVANE

Le module SAVANE contient de nombreuses classes, nous ne présentons ici que les plus importantes pour la compréhension de la structure du programme. Nous avons regroupé les classes en plusieurs sections : gestion de la base de données dans une requête, algorithmique graphique, gestion des changements de système de coordonnées et de projection, et gestion d'un document (cartographie, édition). Enfin, un dernier paragraphe donne des exemples de réalisation d'opérations relationnelles.

A.2.1. La manipulation de la base de données dans un cadre

A.2.1.1. Les classes CRelation et CAttribut

Les classes utilisées dans SAVANE diffèrent légèrement de celles utilisées dans SAVATECA. Elles comportent en particulier chacune une méthode permettant de sauvegarder l'ensemble de leurs paramètres dans le fichier de la carte auquel appartient le cadre.

```
class CAttribut
  {
  public :
    int m_iType;
    int m_iNbvalb;
    int m_iPtrval;
    int m_iNumero;
    int m_iTemporaire;
    int m_iNbcar;
    char m_chNom[20];
  public:
    CAttribut();
    CAttribut(FILE* Fichier);
    void Ecrire(FILE* Fichier);
  };

class CRelation
  {
  private:
    CSchema* m_pSchema;
  public :
    long m_lAdresse;
    int m_iNb0;
    int m_iType;
    int m_iNba;
    int m_iNbObjets;
    char m_chNom[100];
    char m_chFichFeuille[_MAX_PATH];
    char m_chFichZone[_MAX_PATH];
    char m_chFichArc[_MAX_PATH];
    int m_iNoFichDescriptif;
    int m_iNoFichArc;
    int m_iNoFichFeuille;
    int m_iNoFichImage;
    BOOL m_bARasteriser;

    CAttribut* m_pAttributs[NB_MAX_ATTRIBUTS];

  public:
    CRelation(CSchema* pSchema);
    CRelation(CSchema* pSchema,BOOL bMemeBase,FILE* Fichier);
    ~CRelation();
    void Ecrire(FILE* Fichier);
    int NbMaxObjetsParFeuille();
    int NbMaxArcsParFeuille();
    void MiseAJour(int iNoFichier);
```

```
    void CalculDesCentroides();
    unsigned char* ImageRaster(int iNoatt);
    void ExporterUneFeuille(const char* chNomFeuille,const char* chNomAttribut,const char*
chNomFichier);
  };
```

Les méthodes `NbMaxObjetsParFeuille()` et `NbMaxArcsParFeuille()` permettent de connaître de nombre maximum d'objets ou d'arcs dans une feuille. Ces fonctions sont utilisées pour dimensionner l'allocation dynamique de mémoire dans certaines opérations sur la base de données qui utilisent l'indexation géographique.

```
int CRelation::NbMaxObjetsParFeuille()
  {
  int iNbMaxObjetsParFeuille=0;
  FILE* FileFeuille;
  CWind* pWind=m_pSchema->GetCadre()->GetWind();

  //--- traitement
  switch(m_iType)
    {
    case 5:
    case 15:
    case 25:
    case 1:
    case 11:
    default:
      iNbMaxObjetsParFeuille=0;
      break;

    case 4:
    case 3:
    case 2:
      FileFeuille=fopen(m_chFichFeuille,"rb");
      if(FileFeuille != NULL)
        {
        float fXb,fYb,fXh,fYh;
        int iNbObj;
        while(fscanf(FileFeuille,"%*17c%8f%8f%8f%8f%8i%*25c",&fXb,&fYb,&fXh,&fYh,&iNbObj) ==
5)
          {
          if(iNbMaxObjetsParFeuille < iNbObj)iNbMaxObjetsParFeuille=iNbObj;
          }
        fclose(FileFeuille);
        }
      break;

    case 14:
    case 13:
    case 12:
    case 24:
      if(m_iNoFichFeuille == -1)FileFeuille=fopen(m_chFichFeuille,"rb");
      else FileFeuille=fopen(m_pSchema->m_chFprovFeuille[m_iNoFichFeuille],"rb");

      if(FileFeuille != NULL)
        {
        float fXb,fYb,fXh,fYh;
        int iNbObj;
        while(fscanf(FileFeuille,"%*17c%8f%8f%8f%8f%8i%*25c",&fXb,&fYb,&fXh,&fYh,&iNbObj) ==
5)
          {
          if(iNbMaxObjetsParFeuille < iNbObj)iNbMaxObjetsParFeuille=iNbObj;
          }
        fclose(FileFeuille);
        }
      break;
    }
  return iNbMaxObjetsParFeuille;
  }
```

A.2.1.2. La classe CSchema

La classe `CSchema` de SAVANE permet d'encapsuler variables et méthodes concernant la gestion du schéma de la base correspond à une requête dans un cadre. Ce schéma évolue au fur et à mesure de la requête : un objet de cette classe permet de maintenir l'état du schéma de la base au cours de la requête.

```
class CSchema
  {
  private:
    CCadre* m_pCadre;
    int m_iNumero;

  public :
    int m_iNbacc;
    CRelation* m_pRelations[NB_MAX_RELATIONS];

    char m_chFprovDescriptif[NB_MAX_FICHIERS_TEMP][_MAX_PATH];
    char m_chFprovArc[NB_MAX_FICHIERS_TEMP][_MAX_PATH];
    char m_chFprovFeuille[NB_MAX_FICHIERS_TEMP][_MAX_PATH];
    char m_chFprovImage[NB_MAX_FICHIERS_TEMP][_MAX_PATH];

    int m_iFdisDescriptif[NB_MAX_FICHIERS_TEMP];
    int m_iFdisArc[NB_MAX_FICHIERS_TEMP];
    int m_iFdisFeuille[NB_MAX_FICHIERS_TEMP];
    int m_iFdisImage[NB_MAX_FICHIERS_TEMP];

    char m_chNomFtval[_MAX_PATH];
    int m_iPvl;

  public:
    CSchema(CCadre* pCadre);
    CSchema(CCadre* pCadre,BOOL bMemeBase,FILE* Fichier);
    ~CSchema();
    void Ecrire(FILE* Fichier);
    void InitNomFichiers();
    void Alloc();
    void Actualiser();
    void Desalloc();
    int FchoixDescriptif();
    int FchoixImage();
    int FchoixFeuille();
    int FchoixArc();
    int RelRecherche(const char *nom);
    int AttrRecherche(int iNoth,const char *nom);
    CCadre* GetCadre() {return m_pCadre;};

    int NzMax(int iNoth);
    int NombreObjets(int iNoth);
    int NouvelleRelation(int iType,char *chNomRelation,int iNoFichierFeuille,int
iNoFichierDescriptif,int iNoFichierArc);
    int NouvelleRelationMosaique(char *chNomRelation,double dResolution,int
iTailleImagette,const char *chDirMosaique);
    int NouvelAttribut(int iNoth,int iType,int iNbcar,int iNbValb,int iPtrval,char
*chNomAttribut,char *chDescription);

    BOOL ExporterShapeFile(int iNoth,int iNbAttributs,int iTypeExport,int
iTypeCoordonnees,int* itabNoatt,CString strNomFichier);
    BOOL ExporterMosaique(int iNoth,int iNoatt,CString strNomFichier);
    BOOL ExporterDatabase(int iNoth,int iNbAttributs,int iTypeDatabase,int* itabNoatt,CString
strNomFichier);
  };
```

Chaque schéma correspond à un cadre géographique (`m_pCadre`). La méthode `Alloc()` permet de charger le schéma dans un objet de type `CSchema`, à partir du dictionnaire et de la vue externe utilisée. Elle utilise les classes `CDico` et `CFpacc9` :

```
void CSchema::Alloc()
  {
  //--- initialisation des accès aux données suivant la vue externe
  int i,j,iNbrel,iNorel,iNoth,iNbatt,iNovar,iNoatt,iType,iNb0,iNrec,plibz,pliba;
  int iTypeRel[NB_MAX_RELATIONS],iNbattAbs[NB_MAX_RELATIONS];
  long adr;
```

```
long iPtrRel[NB_MAX_RELATIONS];

//--- lecture des relations et attributs
CDico dico;
if(dico.Ouvrir(base.m_chNomFtbase,MODE_READ) == FALSE)return;
dico.LireNumero(m_iNumero);

dico.LireNbRelations(iNbrel);
i=2;
j=1;
while(j <= iNbrel)
   {
   adr=i*dico.GetTailleRecord();
   dico.LireRelationTypeNbAttributs(adr,iType,iNbatt);
   iTypeRel[j]=iType;
   iPtrRel[j]=adr;
   iNbattAbs[j]=iNbatt;
   i+=(iNbatt+1);
   j++;
   }
dico.Fermer();

//--- lecture du fichier des accès et allocation
FILE *FileAcces;
char chBuffer[100];

i=_chdir(base.m_chDirBase);
FileAcces=fopen("fpacc","rb");

CfpaccV9 fpacc;
m_iNbacc=fpacc.RechercheEnteteVue(FileAcces,base.m_chNomVue);

iNoth=1;
while(iNoth <= m_iNbacc)
   {
   fpacc.LireEnteteRelation(FileAcces,iNorel,iNbatt);

   m_pRelations[iNoth]= new CRelation(this);
   m_pRelations[iNoth]->m_lAdresse=iPtrRel[iNorel];
   m_pRelations[iNoth]->m_iNba=iNbatt;

   //--- initialisation des numéros de fichiers temporaires pour les relations
   m_pRelations[iNoth]->m_iNoFichDescriptif=-1;
   m_pRelations[iNoth]->m_iNoFichImage=-1;
   m_pRelations[iNoth]->m_iNoFichFeuille=-1;
   m_pRelations[iNoth]->m_iNoFichArc=-1;

   for(iNoatt=1;iNoatt <= iNbatt;iNoatt++)
      {
      fread(chBuffer,4,1,FileAcces);
      chBuffer[4]=0;
      sscanf(chBuffer,"%4d",&iNovar);

      m_pRelations[iNoth]->m_pAttributs[iNoatt]= new CAttribut();
      m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNumero=iNovar;
      }

   iNoth++;
   }
fclose(FileAcces);
i=_chdir(base.m_chDirSav);

//--- initialisation de la disponibilité des fichiers provisoires
for(i=0;i < NB_MAX_FICHIERS_TEMP;i++)
   {
   m_iFdisDescriptif[i]=0;
   m_iFdisImage[i]=0;
   m_iFdisArc[i]=0;
   m_iFdisFeuille[i]=0;
   }

//--- initialisations des noms de relations et de leurs noms de fichiers permanents
char chNomRelation[100],chNom2[_MAX_PATH],chNom3[100],chNom4[100];

dico.Ouvrir(base.m_chNomFtbase,MODE_READ);
iNoth=1;
while(iNoth <= m_iNbacc)
   {
   adr=m_pRelations[iNoth]->m_lAdresse;
   dico.LireRelation(adr,chNomRelation,iType,iNb0,iNrec,chNom2,chNom3,chNom4,plibz,pliba);
```

```
    strcpy(m_pRelations[iNoth]->m_chNom,chNomRelation);
    m_pRelations[iNoth]->m_iType=iType;
    m_pRelations[iNoth]->m_iNb0=iNb0;
    if(iType == 5)
       {
       strcpy(m_pRelations[iNoth]->m_chFichFeuille,"aucun");
       strcpy(m_pRelations[iNoth]->m_chFichZone,chNom2);
       strcat(m_pRelations[iNoth]->m_chFichZone,"\\");
       strcat(m_pRelations[iNoth]->m_chFichZone,m_pRelations[iNoth]->m_chNom);
       strcpy(m_pRelations[iNoth]->m_chFichArc,"aucun");
       }
    else
       {
       strcpy(m_pRelations[iNoth]->m_chFichFeuille,base.m_chDirBase);
       strcpy(m_pRelations[iNoth]->m_chFichZone,base.m_chDirBase);
       strcpy(m_pRelations[iNoth]->m_chFichArc,base.m_chDirBase);
       strcat(m_pRelations[iNoth]->m_chFichFeuille,"\\");
       strcat(m_pRelations[iNoth]->m_chFichZone,"\\");
       strcat(m_pRelations[iNoth]->m_chFichArc,"\\");
       strcat(m_pRelations[iNoth]->m_chFichFeuille,chNom2);
       strcat(m_pRelations[iNoth]->m_chFichZone,chNom3);
       strcat(m_pRelations[iNoth]->m_chFichArc,chNom4);
       }

    iNoth++;
    }

  //--- initialisations des noms d'attribut
  int iTypeAttr,iNbvalb,iPtrval,iNbcar;
  char chNomAttribut[100],chDescription[128];
  iNoth=1;
  while(iNoth <= m_iNbacc)
     {
     iNoatt=1;
     while(iNoatt <= m_pRelations[iNoth]->m_iNba)
        {
        strcpy(chNomAttribut,"");
        iNovar=m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNumero;
        adr=m_pRelations[iNoth]->m_lAdresse + (iNovar*dico.GetTailleRecord());
        dico.LireAttribut(adr,chNomAttribut,iTypeAttr,iNbvalb,iPtrval,iNbcar,chDescription);

        strcpy(m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_chNom,chNomAttribut);
        m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iType=iTypeAttr;
        m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNbvalb=iNbvalb;
        m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iPtrval=iPtrval;

        iNoatt++;
        }

     iNoth++;
     }

  dico.Fermer();

  m_iPvl=1;
  }
```

La méthode `Actualiser()` permet de recharger le schéma après une modification externe du schéma de la base (avec SAVATECA), tout en conservant l'état des relations temporaires (modifiées par la requête en cours). La variable `m_iNumero` de la classe permet de connaître l'état du schéma et de le comparer avec l'état actuel du dictionnaire.

```
void CSchema::Actualiser()
  {
  //--- vérifier les différences de schéma
  int iNumero;
  CDico dico;
  if(dico.Ouvrir(base.m_chNomFtbase,MODE_READ) == FALSE)return;
  dico.LireNumero(iNumero);
  if(iNumero == m_iNumero)
     {
     dico.Fermer();
     return;
```

```
    }

  //--- vérifier que les relations appartiennent toujours à la base, et mettre à jour les
divers pointeurs
  BOOL bModifie,bTrouve,bTrouveAttribut;
  int i,j,k,iNbRelations,iNoth,iTypeRelation,iNbAttributs,iNrec,plibz,pliba;
  int iNoatt,iTypeAttribut,iNbvalb,iPtrval,iNbcar,iTemporaire;
  long lAdresse,lAdresseAttribut;
  char chNomRelation[64],chNom2[_MAX_PATH],chNom3[100],chNom4[100],chNomFichier[_MAX_PATH];
  char chNomAttribut[64],chDescription[128];

  //--- actualisation
  m_iNumero=iNumero;
  bModifie=FALSE;

  //--- mise à jour des relations
  dico.LireNbRelations(iNbRelations);
  iNoth=1;
  while(iNoth <= m_iNbacc)
    {
    if(m_pRelations[iNoth]->m_lAdresse > 0)
      {
      //--- relation non temporaire, recherche dans la base
      bTrouve=FALSE;
      i=2;
      j=1;
      while(j <= iNbRelations && bTrouve == FALSE)
        {
        lAdresse=i*dico.GetTailleRecord();
             dico.LireRelation(lAdresse,chNomRelation,iTypeRelation,iNbAttributs,iNrec,chNom2
       ,chNom3,chNom4,plibz,pliba);

        if(strcmp(chNomRelation,m_pRelations[iNoth]->m_chNom) == 0)
          {
          //--- on actualise les paramètres de la relation
          bTrouve=TRUE;
          if(m_pRelations[iNoth]->m_lAdresse != lAdresse)
            {
            m_pRelations[iNoth]->m_lAdresse=lAdresse;
            bModifie=TRUE;
            }
          if(m_pRelations[iNoth]->m_iNb0 != iNbAttributs)
            {
            m_pRelations[iNoth]->m_iNb0=iNbAttributs;
            bModifie=TRUE;
            }

          if(m_pRelations[iNoth]->m_iType == 5 || m_pRelations[iNoth]->m_iType == 15)
            {
            //--- cas mosaique
            strcpy(chNomFichier,chNom2);
            strcat(chNomFichier,"\\");
            strcat(chNomFichier,m_pRelations[iNoth]->m_chNom);
            if(strcmp(m_pRelations[iNoth]->m_chFichZone,chNomFichier) != 0)
              {
              strcpy(m_pRelations[iNoth]->m_chFichZone,chNomFichier);
              bModifie=TRUE;
              }
            }
          else
            {
            if(m_pRelations[iNoth]->m_iNoFichFeuille == -1)
              {
              //--- vérifier l'emplacement du fichier feuille d'origine
              strcpy(chNomFichier,base.m_chDirBase);
              strcat(chNomFichier,"\\");
              strcat(chNomFichier,chNom2);
              if(strcmp(m_pRelations[iNoth]->m_chFichFeuille,chNomFichier) != 0)
                {
                strcpy(m_pRelations[iNoth]->m_chFichFeuille,chNomFichier);
                bModifie=TRUE;
                }
              }
            if(m_pRelations[iNoth]->m_iNoFichDescriptif == -1)
              {
              //--- vérifier l'emplacement du fichier descriptif d'origine
              strcpy(chNomFichier,base.m_chDirBase);
              strcat(chNomFichier,"\\");
              strcat(chNomFichier,chNom3);
              if(strcmp(m_pRelations[iNoth]->m_chFichZone,chNomFichier) != 0)
```

```
                  {
                  strcpy(m_pRelations[iNoth]->m_chFichZone,chNomFichier);
                  bModifie=TRUE;
                  }
               }
            if(m_pRelations[iNoth]->m_iNoFichArc == -1)
               {
               //--- vérifier l'emplacement du fichier arc d'origine
               strcpy(chNomFichier,base.m_chDirBase);
               strcat(chNomFichier,"\\");
               strcat(chNomFichier,chNom4);
               if(strcmp(m_pRelations[iNoth]->m_chFichArc,chNomFichier) != 0)
                  {
                  strcpy(m_pRelations[iNoth]->m_chFichArc,chNomFichier);
                  bModifie=TRUE;
                  }
               }
            }

         //--- attributs
         iNoatt=1;
         while(iNoatt <= m_pRelations[iNoth]->m_iNba)
            {
            //--- recherche de l'attribut dans le dico
            bTrouveAttribut=FALSE;
            lAdresseAttribut=m_pRelations[iNoth]->m_lAdresse + dico.GetTailleRecord();
            k=1;
            while(k <= iNbAttributs && bTrouveAttribut == FALSE)
               {
                        dico.LireAttribut(lAdresseAttribut,chNomAttribut,iTypeAttribut,iNb
                     valb,iPtrval,iNbcar,chDescription);
               lAdresseAttribut+=dico.GetTailleRecord();
               if(strcmp(chNomAttribut,m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_chNom)
== 0)
                  {
                  //--- trouve
                  bTrouveAttribut=TRUE;
                  iTemporaire=m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iTemporaire;
                  if(m_pRelations[iNoth]->m_iType < 10 && iTemporaire == 0)
                     {
                     //--- relation de base non modifiée,attribut non temporaire
                     if(m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iType != iTypeAttribut)
                        {
                        m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iType=iTypeAttribut;
                        bModifie=TRUE;
                        }
                     if(m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNumero != k)
                        {
                        m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNumero=k;
                        bModifie=TRUE;
                        }
                     }

                  //--- vérifier les pointeurs des attributs nominaux de base
                  if(m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iType == 1 && iTemporaire
== 0)
                     {
                     if(m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNbvalb != iNbvalb)
                        {
                        m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNbvalb=iNbvalb;
                        bModifie=TRUE;
                        }
                     if(m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iPtrval != iPtrval)
                        {
                        m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iPtrval=iPtrval;
                        bModifie=TRUE;
                        }
                     }
                  }
               k++;
               }

            if(m_pRelations[iNoth]->m_iType < 10 && bTrouveAttribut == FALSE)
               {
               //--- l'attribut iNoatt a été supprimé de la base, le supprimer du schéma
               delete m_pRelations[iNoth]->m_pAttributs[iNoatt];
               for(k=iNoatt;k < m_pRelations[iNoth]->m_iNba;k++)m_pRelations[iNoth]-
>m_pAttributs[k]=m_pRelations[iNoth]->m_pAttributs[k+1];
               m_pRelations[iNoth]->m_pAttributs[m_pRelations[iNoth]->m_iNba]=NULL;
               m_pRelations[iNoth]->m_iNba--;
```

```
                bModifie=TRUE;
                }
              else iNoatt++;
              }
            }
          i+=(iNbAttributs+1);
          j++;
          }

        if(bTrouve == FALSE)
          {
          //--- la relation a disparue depuis la dernière utilisation, il faut la supprimer du
schéma
          CCadre* pCadre=GetCadre();
          if(pCadre != NULL)XRelProj(pCadre,iNoth);
          }
        else iNoth++;
        }
      else iNoth++;
      }

  dico.Fermer();
  if(bModifie)
    {
    carte.ASauver();
    //--- mise à jour des explorateurs
    m_pCadre->MiseAJourExplorateurs();
    }
  i=0;
  }
```

La méthode `NouvelleRelation()` permet de créer une nouvelle relation temporaire dans le schéma :

```
int CSchema::NouvelleRelation(int iType,char *chNomRelation,int iNoFichierFeuille,int
iNoFichierDescriptif,int iNoFichierArc)
  {
  //--- vérifier que la relation n'existe pas déjà (normalement fait dans les dialogues)
  int iNoth=RelRecherche(chNomRelation);
  if(iNoth != 0)
    {
    switch(config.m_iLangage)
      {
      case FRANCAIS:
      default:
        AfxMessageBox("Cette relation existe déjà !",MB_OK|MB_ICONEXCLAMATION);
        break;
      case ESPAGNOL:
        AfxMessageBox("Este relación ya existe !",MB_OK|MB_ICONEXCLAMATION);
        break;
      case ANGLAIS:
        AfxMessageBox("This relation already exist !",MB_OK|MB_ICONEXCLAMATION);
        break;
      }
    return 0;
    }

  m_iNbacc++;

  iNoth=m_iNbacc;
  m_pRelations[iNoth]= new CRelation(this);

  strcpy(m_pRelations[iNoth]->m_chNom,chNomRelation);
  m_pRelations[iNoth]->m_iNoFichFeuille=iNoFichierFeuille;
  m_pRelations[iNoth]->m_iNoFichDescriptif=iNoFichierDescriptif;
  m_pRelations[iNoth]->m_iNoFichArc=iNoFichierArc;
  m_pRelations[iNoth]->m_iType=iType+10;
  m_pRelations[iNoth]->m_lAdresse=0L;

  //--- initialisation des numéros de fichiers temporaires pour les relations
  if(iNoFichierFeuille >= 0)m_iFdisFeuille[iNoFichierFeuille]=1;
  if(iNoFichierDescriptif >= 0)m_iFdisDescriptif[iNoFichierDescriptif]=1;
  if(iNoFichierArc >= 0)m_iFdisArc[iNoFichierArc]=1;

  CCadre* pCadre=GetCadre();
  pCadre->GetCarte()->SauverLesCadres();
```

```
  return iNoth;
  }
```

De même, la méthode `NouvelAttribut()` permet de créer un nouvel attribut temporaire dans le schéma :

```
int CSchema::NouvelAttribut(int iNoth,int iType,int iNbcar,int iNbvalb,int iPtrval,char
*chNomAttribut,char *chDescription)
  {
  //--- creation d'un nouvel attribut dans le schema
  int iNoatt=m_pRelations[iNoth]->m_iNba+1;

  //--- rechercher d'abord pour voir si le nom n'existe pas
  int iTypeRelation=m_pRelations[iNoth]->m_iType;
  if(iTypeRelation != 5 && iTypeRelation != 15 && iTypeRelation != 25)
    {
    char chAjout[5];
    int i=2;
    int iNoattIdem=AttrRecherche(iNoth,chNomAttribut);
    while(iNoattIdem > 0 && i < 10)
      {
      //--- l'attribut existe, on modifie le nom
      chNomAttribut[12]=0;
      sprintf(chAjout," (%1d)",i);
      chAjout[4]=0;
      strcat(chNomAttribut,chAjout);
      iNoattIdem=AttrRecherche(iNoth,chNomAttribut);
      i++;
      }
    }

  m_pRelations[iNoth]->m_iNba++;
  m_pRelations[iNoth]->m_pAttributs[iNoatt]= new CAttribut();
  m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNumero=iNoatt;
  strcpy(m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_chNom,chNomAttribut);
  m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iType=iType;
  m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNbvalb=iNbvalb;
  m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iPtrval=iPtrval;
  m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iTemporaire=1;

  if(iType == 1 || iType == 5)m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNbcar=iNbcar;
  else m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNbcar=0;

  if(strlen(chDescription) == 0)
    {
    switch(config.m_iLangage)
      {
      case FRANCAIS:
      default:
        strcpy(chDescription,"sans description");
        break;
      case ESPAGNOL:
        strcpy(chDescription,"sin descripción");
        break;
      case ANGLAIS:
        strcpy(chDescription,"without description");
        break;
      }
    }

  CCadre* pCadre=GetCadre();
  pCadre->GetCarte()->SauverLesCadres();

  return iNoatt;
  }
```

A.2.1.3. La classe CLecture

La classe CLecture regroupe les opérations de lecture des objets d'une relation. Elle est utilisée dans toutes les opérations pour avoir accès aux données :

```
class CLecture
  {
  private:
    CCadre* m_pCadre;
```

```
    CWind* m_pWind;
    int m_iNoth;
    int m_tabp[NB_MAX_ATTRIBUTS];
    int m_iNb0;
    int m_iNba;
    int m_iNbnew;
    int m_iNbVariables;
    int m_iEcrire;
    int m_iType;
    int m_iNb;
    int m_iNbObj;
    int m_iSizeIdentifiantTuple;

    int nz;
    int m_iArc;
    long m_iPtrarc;
    long m_iPtrTuple;
    long m_iPtrarcFeuille;
    long m_iNbarcFeuille;
    float xc,yc,xb,yb,xh,yh;
    float m_fWindXb;
    float m_fWindYb;
    float m_fWindXh;
    float m_fWindYh;

    FILE* m_FileFeuille;
    FILE* m_FileDescriptif;
    FILE* m_FileArc;
    FILE* m_FileResultat;

  public:
    CLecture();
    ~CLecture();
    BOOL Open(CCadre* pCadre,int iNoth);
    BOOL Open(CCadre* pCadre,int iNoth,int iNoFichier,int iNbnew);
    void Close();
    BOOL Objet(float fXSav,float fYSav,float* v,FILE* FileResultat);
    int Lire(float* v);
    int LireSansTestFenetre(float* v);
    int LireTuple(long ptrFile,float* v);
    void SetFenetreLecture(float fXbSav,float fYbSav,float fXhSav,float fYhSav);
    long GetPointeurTuple() {return m_iPtrTuple;};
    void GetCentroide(float &fXc,float &fYc) {fXc=xc;fYc=yc;};
    int GetPointeurArc() {return m_iPtrarc;};
    int GetNbarcFeuille() {return m_iNbarcFeuille;};
    int GetPointeurArcFeuille() {return m_iPtrarcFeuille;};
    int GetNz() {return nz;};
    FILE* GetFileArc() {return m_FileArc;};
    void GetRectangle(float &fXb,float &fYb,float &fXh,float &fYh)
{fXb=xb;fYb=yb;fXh=xh;fYh=yh;};
    void Ecrire(float* v);
  };
```

Le tableau `m_tabp` contient l'indirection entre les numéros externes et internes pour les attributs de la relation. Il est initialisé, comme toutes les autres variables de la relation, par la méthode `Open()`. Les variables `m_fWind` permettent de définir le rectangle géographique de sélection implicite.

Voici par exemple l'implémentation de la méthode de lecture des objets. On remarquera en particulier les différences de lecture entre les différents types, qu'ils soient de base (5,4,3,2,1) ou temporaires (15,25,14,24,13,12,11). Cette procédure revoie –1 si la fin des objets a été atteinte, 0 si l'objet n'est pas dans la fenêtre du cadre, 1 si un objet a été lu, 2 si l'on change de feuille. Le tableau `v` renvoie les valeurs des attributs de l'objet lu :

```
int CLecture::Lire(float* v)
  {
  int i;
  switch(m_iType)
    {
```

```
case 4:
  if(m_iNb <= m_iNbObj)
    {
    //--- il reste des objets a lire dans la feuille courante
    m_iPtrTuple=ftell(m_FileDescriptif);
    fread(&nz,SIZEINT,1,m_FileDescriptif);
    fread(&xc,SIZECOOR,1,m_FileDescriptif);
    fread(&yc,SIZECOOR,1,m_FileDescriptif);
    fread(&xb,SIZECOOR,1,m_FileDescriptif);
    fread(&yb,SIZECOOR,1,m_FileDescriptif);
    fread(&xh,SIZECOOR,1,m_FileDescriptif);
    fread(&yh,SIZECOOR,1,m_FileDescriptif);
    for(i=1;i <= m_iNb0;i++)fread(&v[i],SIZEVAL,1,m_FileDescriptif);
    m_iNb++;

    nz=UnixToDos(nz);
    xc=UnixToDos(xc);
    yc=UnixToDos(yc);
    xb=UnixToDos(xb);
    yb=UnixToDos(yb);
    xh=UnixToDos(xh);
    yh=UnixToDos(yh);
    if(xh >= m_fWindXb && xb <= m_fWindXh && yh >= m_fWindYb && yb <= m_fWindYh)
      {
      for(i=1;i <= m_iNb0;i++)v[i]=UnixToDos(v[i]);
      return 1;
      }
    return 0;
    }
  else
    {
    if(m_iEcrire && m_iNbObj > 0)
      {
      //--- fin de la feuille precedente
      nz=0;
      Ecrire(v);
      }
    //--- positionne sur le premier élément d'une nouvelle feuille
    int iPtr,iPtrObj,iNbarc,iPtrarc;
    float fXb,fYb,fXh,fYh;

          while(fscanf(m_FileFeuille,"%*17c%8f%8f%8f%8f%8i%8i%8i%8i%*c",&fXb,&fYb,&fXh,&
      fYh,&m_iNbObj,&iPtrObj,&iNbarc,&iPtrarc) == 8)
      {
      if(fXh >= m_fWindXb && fXb <= m_fWindXh && fYh >= m_fWindYb && fYb <= m_fWindYh)
        {
        //--- positionne sur le premier élément a lire, avec m_iNbObj objets a lire
        iPtr=(iPtrObj-1)*(m_iSizeIdentifiantTuple + m_iNb0*SIZEVAL);
        fseek(m_FileDescriptif,iPtr,SEEK_SET);
        m_iNb=1;
        if(m_iEcrire && m_iNbObj > 0)
          {
          fwrite(&iNbarc,SIZEINT,1,m_FileResultat);
          fwrite(&iPtrarc,SIZEINT,1,m_FileResultat);
          }
        m_iNbarcFeuille=iNbarc;
        m_iPtrarcFeuille=iPtrarc;
        return 2;
        }
      }
    //--- fin des feuilles
    if(m_iEcrire)
      {
      iNbarc=0;
      iPtrarc=0;
      fwrite(&iNbarc,SIZEINT,1,m_FileResultat);
      fwrite(&iPtrarc,SIZEINT,1,m_FileResultat);
      m_iNbarcFeuille=0;
      m_iPtrarcFeuille=0;
      }
    return -1;
    }
  break;

case 3:
  if(m_iNb <= m_iNbObj)
    {
    //--- il reste des objets a lire dans la feuille courante
    m_iPtrTuple=ftell(m_FileDescriptif);
    fread(&xc,SIZECOOR,1,m_FileDescriptif);
```

```
        fread(&yc,SIZECOOR,1,m_FileDescriptif);
        for(i=1;i <= m_iNb0;i++)fread(&v[i],SIZEVAL,1,m_FileDescriptif);
        m_iNb++;

        xc=UnixToDos(xc);
        yc=UnixToDos(yc);
        if(xc >= m_fWindXb && xc <= m_fWindXh && yc >= m_fWindYb && yc <= m_fWindYh)
          {
          for(i=1;i <= m_iNb0;i++)v[i]=UnixToDos(v[i]);
          return 1;
          }
        return 0;
        }
      else
        {
        //--- positionne sur le premier élément d'une feuille
        int iPtr,iPtrObj;
        float fXb,fYb,fXh,fYh;


  while(fscanf(m_FileFeuille,"%*17c%8f%8f%8f%8f%8i%8i%*8i%*8i%*c",&fXb,&fYb,&fXh,&fYh,&m_iNbOb
j,&iPtrObj) == 6)
          {
          if(fXh >= m_fWindXb && fXb <= m_fWindXh && fYh >= m_fWindYb && fYb <= m_fWindYh)
            {
            //--- positionne sur le premier élément a lire, avec m_iNbObj objets a lire
            iPtr=(iPtrObj-1)*(m_iSizeIdentifiantTuple + m_iNb0*SIZEVAL);
            fseek(m_FileDescriptif,iPtr,SEEK_SET);
            m_iNb=1;
            return 2;
            }
          }
        //--- fin des feuilles
        return -1;
        }
      break;

    case 2:
      if(m_iNb <= m_iNbObj)
        {
        int iPtr;
        //--- il reste des objets a lire dans la feuille courante
        m_iPtrTuple=ftell(m_FileDescriptif);
        fread(&m_iArc,SIZEINT,1,m_FileDescriptif);
        fread(&m_iPtrarc,SIZEINT,1,m_FileDescriptif);
        fread(&xc,SIZECOOR,1,m_FileDescriptif);
        fread(&yc,SIZECOOR,1,m_FileDescriptif);
        m_iArc=UnixToDos(m_iArc);
        m_iPtrarc=UnixToDos(m_iPtrarc);
        xc=UnixToDos(xc);
        yc=UnixToDos(yc);
        for(i=1;i <= m_iNb0;i++)fread(&v[i],SIZEVAL,1,m_FileDescriptif);

        m_iNb++;

        iPtr=(m_iPtrarc-1)*SIZEREC + 4*SIZECOOR;
        fseek(m_FileArc,iPtr,SEEK_SET);

        fread(&xb,SIZECOOR,1,m_FileArc);
        fread(&yb,SIZECOOR,1,m_FileArc);
        fread(&xh,SIZECOOR,1,m_FileArc);
        fread(&yh,SIZECOOR,1,m_FileArc);
        xb=UnixToDos(xb);
        yb=UnixToDos(yb);
        xh=UnixToDos(xh);
        yh=UnixToDos(yh);
        if(xh >= m_fWindXb && xb <= m_fWindXh && yh >= m_fWindYb && yb <= m_fWindYh)
          {
          for(i=1;i <= m_iNb0;i++)v[i]=UnixToDos(v[i]);
          return 1;
          }
        return 0;
        }
      else
        {
        //--- positionne sur le premier élément d'une feuille
        int iPtr,iPtrObj,iNbarc,iPtrarc;
        float fXb,fYb,fXh,fYh;
```

```
  while(fscanf(m_FileFeuille,"%*17c%8f%8f%8f%8f%8f%8i%8i%8i%8i%*c",&fXb,&fYb,&fXh,&fYh,&m_iNbObj,
&iPtrObj,&iNbarc,&iPtrarc) == 8)
           {
           if(fXh >= m_fWindXb && fXb <= m_fWindXh && fYh >= m_fWindYb && fYb <= m_fWindYh)
             {
             //--- positionne sur le premier élément a lire, avec m_iNbObj objets a lire
             iPtr=(iPtrObj-1)*(m_iNb0*SIZEVAL + m_iSizeIdentifiantTuple);
             fseek(m_FileDescriptif,iPtr,SEEK_SET);
             m_iNb=1;
             m_iNbarcFeuille=iNbarc;
             m_iPtrarcFeuille=iPtrarc;
             return 2;
             }
           }
         //--- fin des feuilles
         m_iNbarcFeuille=0;
         m_iPtrarcFeuille=0;
         return -1;
         }
       break;

     case 1:
       m_iPtrTuple=ftell(m_FileDescriptif);
       for(i=1;i <= m_iNb0;i++)fread(&v[i],SIZEVAL,1,m_FileDescriptif);
       if(!feof(m_FileDescriptif))
         {
         for(i=1;i <= m_iNb0;i++)v[i]=UnixToDos(v[i]);
         return 1;
         }
       else return -1;
       break;

     case 14:
     case 24:
       if(m_iNbObj)
         {
         //--- il reste des objets a lire
         m_iPtrTuple=ftell(m_FileDescriptif);
         fread(&nz,SIZEINT,1,m_FileDescriptif);
         fread(&xc,SIZECOOR,1,m_FileDescriptif);
         fread(&yc,SIZECOOR,1,m_FileDescriptif);
         fread(&xb,SIZECOOR,1,m_FileDescriptif);
         fread(&yb,SIZECOOR,1,m_FileDescriptif);
         fread(&xh,SIZECOOR,1,m_FileDescriptif);
         fread(&yh,SIZECOOR,1,m_FileDescriptif);
         for(i=1;i <= m_iNba;i++)fread(&v[i],SIZEVAL,1,m_FileDescriptif);
         if(nz != 0)return 1;
         else
           {
           if(m_iEcrire)Ecrire(v);
           m_iNbObj=0;
           return 3;
           }
         }
       else
         {
         int iNbarc,iPtrarc;
         fread(&iNbarc,SIZEINT,1,m_FileDescriptif);
         fread(&iPtrarc,SIZEINT,1,m_FileDescriptif);
         if(m_iEcrire)
           {
           fwrite(&iNbarc,SIZEINT,1,m_FileResultat);
           fwrite(&iPtrarc,SIZEINT,1,m_FileResultat);
           }
         if(iNbarc == 0)return -1;

         m_iNbObj=1;
         m_iNbarcFeuille=iNbarc;
         m_iPtrarcFeuille=iPtrarc;
         return 2;
         }

       break;

     case 13:
       m_iPtrTuple=ftell(m_FileDescriptif);
       fread(&xc,SIZECOOR,1,m_FileDescriptif);
       fread(&yc,SIZECOOR,1,m_FileDescriptif);
       for(i=1;i <= m_iNba;i++)fread(&v[i],SIZEVAL,1,m_FileDescriptif);
```

```
      if(!feof(m_FileDescriptif))return 1;
      else return -1;
      break;

    case 12:
      m_iPtrTuple=ftell(m_FileDescriptif);
      fread(&m_iArc,SIZEINT,1,m_FileDescriptif);
      fread(&m_iPtrarc,SIZEINT,1,m_FileDescriptif);
      fread(&xc,SIZECOOR,1,m_FileDescriptif);
      fread(&yc,SIZECOOR,1,m_FileDescriptif);
      for(i=1;i <= m_iNba;i++)fread(&v[i],SIZEVAL,1,m_FileDescriptif);
      if(!feof(m_FileDescriptif))return 1;
      else return -1;
      break;

    case 11:
      m_iPtrTuple=ftell(m_FileDescriptif);
      for(i=1;i <= m_iNba;i++)fread(&v[i],SIZEVAL,1,m_FileDescriptif);
      if(!feof(m_FileDescriptif))return 1;
      else return -1;
      break;

    default: break;
    }

  return -1;
  }
```

Voici l'utilisation typique de cette classe dans une procédure de lecture des valeurs des objets d'une relation :

```
CLecture lecture;
if(lecture.Open(pCadre,iNoth,iNoFichier,0))
  {
  //--- lecture des objets
  float v[NB_MAX_ATTRIBUTS],fResultat;

  i=lecture.Lire(v);
  while(i != -1)
    {
    if(i == 1)
      {
      //--- traitement
      …
      }
    i=lecture.Lire(v);
    }
  lecture.Close();
```

A.2.1.4. La classe CMacro

La classe `CMacro` encapsule les opérations de définition et d'exécution de macro-commandes et de méthodes :

```
class CMacro
  {
  private:
    CCadre* m_pCadre;
    int m_iExec;
    char m_chNomFichierMacro[_MAX_PATH];
    BOOL LireCommande(FILE* FileMacro,char* chCommande);

  public:
    CMacro(CCadre* pCadre);
    ~CMacro();
    BOOL Nouveau(const char* chNomFichierMacro);
    BOOL Ouvrir(const char* chNomFichierMacro);
    BOOL EnregistrerCommande(char* chCommande);
    BOOL Executer(const char* chNomMacro,const char* chNomFichierMacro,BOOL
bEnregistrerDansMacroCadre);
    void Fermer();
    BOOL New();
    int GetMode() {return m_iExec;};
  };
```

La méthode `EnregistrerCommande()` permet d'enregistrer une opération dans une macro-commande, en écrivant les paramètres de l'opération dans le fichier de la macro-commande. Elle est utilisée systématiquement lors du déclenchement d'une commande dans SAVANE, pour inscrire cette commande dans la macro-commande liée au cadre, et dans une macro-commande en cours de définition (si une telle macro-commande a été créée auparavant avec la méthode Nouveau()). Par exemple, voici le code de réponse au menu CRIS-Longueur :

```
void CMainFrame::OnCrisLongueur()
  {
  int iNoCadre;
  if(carte.m_iNbCadres == 1)iNoCadre=0;
  else iNoCadre=selection.m_iNoDuGroupe[0];
  CCadre* pCadre=carte.m_pCadre[iNoCadre];
  if(pCadre != NULL)pCadre->GetSchema()->Actualiser();

  CDialogCrisLongueur dialogue(pCadre->GetSchema());
  if(dialogue.DoModal() == IDOK)
    {
    FILE* FileMenu=fopen(base.m_chNomFtmp,"wb");
    fprintf(FileMenu,"%d\n",dialogue.m_iNoth);
    fprintf(FileMenu,"%d\n",dialogue.m_iUnite);
    fclose(FileMenu);

    //--- calcul du nouvel attribut
    pCadre->GetMacro()->EnregistrerCommande("CRIS_LONGUEUR");
    pCadre->GetMacroCadre()->EnregistrerCommande("CRIS_LONGUEUR");

    XCrisLongueur(pCadre);
    carte.SauverLesCadres();
    pCadre->GetDlgStatExplorateur()->MiseAJour(dialogue.m_iNoth);
    }
  }
```

A.2.1.5. La classe CCalculateur

La classe `CCalculateur` encapsule toutes les variables et méthodes permettant le calcul numérique à partir d'une expression numérique et de la valeur des attributs d'un objet d'une relation :

```
class CCalculateur
  {
  private:
    CSchema* m_pSchema;
    int m_iNoth;
    char* m_chExpressionChaine;
    Symbole m_SymboleCourant;
    double m_dValeurNombre;
    char m_chNomIdentifiant[_MAX_PATH];
    char m_NomNombre[_MAX_PATH];
    float* m_pVariables;
    int m_iNoErreur;

    Symbole LireSymbole();
    double ConditionExpression();
    double ConditionTerme();
    double Expression();
    double Terme();
    double Primaire();
    double ErreurDeCalcul(char* Texte);

    Symbole TestSymbole();
    void TestConditionExpression();
    void TestConditionTerme();
    void TestExpression();
    void TestTerme();
    int TestPrimaire();
    int ErreurDeSyntaxe(char* Texte);

  public:
```

```
    CCalculateur(CSchema* pSchema,int iNoth);
    int Test(char* chFormule);
   double Calcul(char* chFormule, float* dVariables);
    int m_iNbErreurDeCalcul[20];
    int m_iAttributs[NB_MAX_ATTRIBUTS];
};
```

Elle permet de tester les formules (`Test()`), et de calculer le résultat (`Calcul()`). Le calcul est bien sûr récursif :

```
double CCalculateur::Calcul(char* chFormule, float* pVariables)
  {
  m_pVariables=pVariables;
  m_iNoErreur=0;
  m_chExpressionChaine=chFormule;
  m_SymboleCourant=DEBUT;
  LireSymbole();
  double d=ConditionExpression();
  if(m_iNoErreur != 0)d=DINCONNU;
  return d;
  }

Symbole CCalculateur::LireSymbole() // lit un symbole pour le calcul
  {
  char caractere;

  if(m_SymboleCourant == FIN) return m_SymboleCourant;
  do
    {
    caractere=*m_chExpressionChaine;
    if(caractere == '\0') return m_SymboleCourant=FIN;
    m_chExpressionChaine++;
    }
  while (isspace(caractere));

  switch (caractere)
    {
    case ';' : return m_SymboleCourant=FIN;
    case '*': return m_SymboleCourant=MUL;
    case '/': return m_SymboleCourant=DIV;
    case '+': return m_SymboleCourant=PLUS;
    case '-': return m_SymboleCourant=MOINS;
    case '(': return m_SymboleCourant=LP;
    case ')': return m_SymboleCourant=RP;
    case ',': return m_SymboleCourant=VIRGULE;
    case ':': return m_SymboleCourant=DEUXPOINT;
    case '=': return m_SymboleCourant=EGAL;
    case '<':
      caractere=*m_chExpressionChaine++;
      if(caractere == '>')return m_SymboleCourant=DIFFERENT;
      if(caractere == '=')return m_SymboleCourant=INFERIEUREGAL;
      m_chExpressionChaine--;
      return m_SymboleCourant=INFERIEUR;
      break;
    case '>':
      caractere=*m_chExpressionChaine++;
      if(caractere == '=')return m_SymboleCourant=SUPERIEUREGAL;
      m_chExpressionChaine--;
      return m_SymboleCourant=SUPERIEUR;
      break;

    case '0': case '1': case '2': case '3': case '4':
    case '5': case '6': case '7': case '8': case '9': case '.':
      {
      char* p=m_NomNombre;
      *p++=caractere;
      caractere=*m_chExpressionChaine++;
      while(isdigit(caractere) || caractere == '.')
        {
        *p++=caractere;
        caractere=*m_chExpressionChaine++;
        }
      m_chExpressionChaine--;
      *p=0;
      m_dValeurNombre=atof(m_NomNombre);
      return m_SymboleCourant=NOMBRE;
      }
      break;
```

```
    case 'v': // variable
    case 'V':
      {
      char* p=m_chNomIdentifiant;
      caractere=*m_chExpressionChaine++;
      while(isdigit(caractere))
        {
        *p++=caractere;
        caractere=*m_chExpressionChaine++;
        }
      m_chExpressionChaine--;
      *p=0;
      int iIndice=atoi(m_chNomIdentifiant);
      int iNovar=m_pSchema->m_pRelations[m_iNoth]->m_pAttributs[iIndice]->m_iNumero;
      m_dValeurNombre=(double) *(m_pVariables+iNovar);
      return m_SymboleCourant=NOMBRE;
      }
      break;

    default :
      if(isalpha(caractere))
        {
        char* p=m_chNomIdentifiant;
        *p++=(char ) tolower(caractere);
        caractere=*m_chExpressionChaine++;
        while(isalpha(caractere))
          {
          *p++=(char ) tolower(caractere);
          caractere=*m_chExpressionChaine++;
          }
        m_chExpressionChaine--;
        *p=0;
        if(strcmp(m_chNomIdentifiant,"sin") == 0)return m_SymboleCourant=SIN;
        else if(strcmp(m_chNomIdentifiant,"cos") == 0)return m_SymboleCourant=COS;
        else if(strcmp(m_chNomIdentifiant,"tan") == 0)return m_SymboleCourant=TAN;
        else if(strcmp(m_chNomIdentifiant,"asin") == 0)return m_SymboleCourant=ASIN;
        else if(strcmp(m_chNomIdentifiant,"acos") == 0)return m_SymboleCourant=ACOS;
        else if(strcmp(m_chNomIdentifiant,"atan") == 0)return m_SymboleCourant=ATAN;
        else if(strcmp(m_chNomIdentifiant,"abs") == 0)return m_SymboleCourant=ABS;
        else if(strcmp(m_chNomIdentifiant,"exp") == 0)return m_SymboleCourant=EXP;
        else if(strcmp(m_chNomIdentifiant,"log") == 0)return m_SymboleCourant=LOG;
        else if(strcmp(m_chNomIdentifiant,"ln") == 0)return m_SymboleCourant=LN;
        else if(strcmp(m_chNomIdentifiant,"pow") == 0)return m_SymboleCourant=POW;
        else if(strcmp(m_chNomIdentifiant,"mod") == 0)return m_SymboleCourant=MOD;
        else if(strcmp(m_chNomIdentifiant,"sqrt") == 0)return m_SymboleCourant=SQRT;
        else if(strcmp(m_chNomIdentifiant,"int") == 0)return m_SymboleCourant=PARTINT;

        else if(strcmp(m_chNomIdentifiant,"si") == 0)return m_SymboleCourant=SI;
        else if(strcmp(m_chNomIdentifiant,"if") == 0)return m_SymboleCourant=SI;
        else if(strcmp(m_chNomIdentifiant,"et") == 0)return m_SymboleCourant=AND;
        else if(strcmp(m_chNomIdentifiant,"and") == 0)return m_SymboleCourant=AND;
        else if(strcmp(m_chNomIdentifiant,"ou") == 0)return m_SymboleCourant=OR;
        else if(strcmp(m_chNomIdentifiant,"or") == 0)return m_SymboleCourant=OR;

        else if(strcmp(m_chNomIdentifiant,"pi") == 0)
          {
          m_dValeurNombre= PI;
          return m_SymboleCourant=NOMBRE;
          }
        else if(strcmp(m_chNomIdentifiant,"e") == 0)
          {
          m_dValeurNombre= E;
          return m_SymboleCourant=NOMBRE;
          }
        }

      ErreurDeSyntaxe("Erreur de syntaxe : symbole inconnu !");
      return m_SymboleCourant=FIN;
      break;
    }
  }

double CCalculateur::ConditionExpression()
  {
  double dGauche = ConditionTerme();

  for(;;)
  {
    switch (m_SymboleCourant)
```

```
      {
      case AND :
        {
      LireSymbole();
        double dDroit=ConditionTerme();
        if(dGauche != 0. && dDroit != 0.)dGauche=1.; else dGauche=0.;
        }
        break;
    case OR :
        {
      LireSymbole();
        double dDroit=ConditionTerme();
        if(dGauche != 0. || dDroit != 0.)dGauche=1.; else dGauche=0.;
        }
        break;
      default : return dGauche;
      }
  }
  }

double CCalculateur::ConditionTerme()
  {
  double dGauche = Expression();

  switch (m_SymboleCourant)
    {
    case EGAL :
      {
    LireSymbole();
      double dDroit=Expression();
      if(dGauche == dDroit)return 1.; else return 0.;
      }
  case DIFFERENT :
      {
    LireSymbole();
      double dDroit=Expression();
      if(dGauche != dDroit)return 1.; else return 0.;
      }
  case INFERIEUR :
      {
    LireSymbole();
      double dDroit=Expression();
      if(dGauche < dDroit)return 1.; else return 0.;
      }
  case INFERIEUREGAL :
      {
    LireSymbole();
      double dDroit=Expression();
      if(dGauche <= dDroit)return 1.; else return 0.;
      }
  case SUPERIEUR :
      {
    LireSymbole();
      double dDroit=Expression();
      if(dGauche > dDroit)return 1.; else return 0.;
      }
  case SUPERIEUREGAL :
      {
    LireSymbole();
      double dDroit=Expression();
      if(dGauche >= dDroit)return 1.; else return 0.;
      }
  default : return dGauche;
  }
  }

double CCalculateur::Expression()
  {
  double dGauche = Terme();

  for(;;)
  {
  switch (m_SymboleCourant)
    {
    case PLUS :
      LireSymbole();
      dGauche+=Terme();
      break;
    case MOINS :
      LireSymbole();
```

```
        dGauche-=Terme();
        break;
      default : return dGauche;
      }
   }
  }

double CCalculateur::Terme()
   {
   double dGauche = Primaire();

   for(;;)
      {
      switch (m_SymboleCourant)
         {
         case MUL:
           LireSymbole();
           dGauche*=Primaire();
           break;
         case DIV:
           {
           LireSymbole();
           double d=Primaire();
           if(d == 0)return ErreurDeCalcul("division par zéro");
           dGauche/=d;
           }
           break;
         default : return dGauche;
         }
      }
   }

double CCalculateur::Primaire()
   {
   switch (m_SymboleCourant)
      {
      case LP :
        {
        LireSymbole();
        double d=ConditionExpression();
        LireSymbole(); // parenthese fermante
        return d;
        }
      case NOMBRE :
        LireSymbole();
        if(m_dValeurNombre <= DINCONNU)return ErreurDeCalcul("variable : valeur inconnue");
        return m_dValeurNombre;
      case MOINS :
        LireSymbole();
        return -Primaire();
      case PLUS :
        LireSymbole();
        return Primaire();
      case COS :
        LireSymbole();
        return cos(Primaire());
      case ACOS :
        {
        LireSymbole();
        double d=Primaire();
        if(d<-1. || d>1.)return ErreurDeCalcul("acos : erreur de domaine");
        return acos(d);
        }
      case SIN :
        LireSymbole();
        return sin(Primaire());
      case ASIN :
        {
        LireSymbole();
        double d=Primaire();
        if(d<-1. || d>1.)return ErreurDeCalcul("asin : erreur de domaine");
        return asin(d);
        }
      case TAN :
        LireSymbole();
        return tan(Primaire());
      case ATAN :
        LireSymbole();
        return atan(Primaire());
      case EXP :
```

```
      LireSymbole();
      return exp(Primaire());
    case LOG :
      {
      LireSymbole();
      double d= Primaire();
      if(d<=0)return ErreurDeCalcul("log : erreur de domaine");
      return log(d);
      }
    case LN :
      {
      LireSymbole();
      double d= Primaire();
      if(d<=0)return ErreurDeCalcul("ln : erreur de domaine");
      return log10(d);
      }
    case SQRT :
      {
      LireSymbole();
      double d= Primaire();
      if(d<0)return ErreurDeCalcul("sqrt : erreur de domaine");
      return sqrt(d);
      }
    case ABS :
      LireSymbole();
      return fabs(Primaire());
    case PARTINT :
      LireSymbole();
      return floor(Primaire());
    case POW :
      {
      LireSymbole(); // parenthese ouvrante
      LireSymbole();
      double d1= Expression();
      LireSymbole(); // virgule
      double d2= Expression();
      LireSymbole(); // parenthese fermante
      if(d1==0 || d2<0)return ErreurDeCalcul("pow : erreur de domaine");
      double d3=double(int(d2));
      if(d1<0 && d3 != d2)return ErreurDeCalcul("pow : exposant non entier");
      return pow(d1,d2);
      }
    case MOD :
      {
      LireSymbole(); // parenthese ouvrante
      LireSymbole();
      double d1= Expression();
      LireSymbole(); // virgule
      double d2= Expression();
      LireSymbole(); // parenthese fermante
      if(d2 == 0)return ErreurDeCalcul("mod : division par zéro");
      return fmod(d1,d2);
      }
    case SI :
      {
      LireSymbole(); // parenthese ouvrante
      LireSymbole();
      double dCondition= ConditionExpression();
      LireSymbole(); // parenthese fermante
      double dAlors= ConditionExpression();
      LireSymbole(); // deux points
      double dSinon= ConditionExpression();

      if(dCondition == 1.)return dAlors; else return dSinon;
      }
    case FIN :  return 1.;

    default : return 1.;
    }
  }
```

A.2.1.6. La classe CBabel pour l'interpolation

La classe `CBabel` encapsule l'ensemble des opérations de création d'une relation de type mosaïque par interpolation.

```
class CBabel
```

```
  {
  private:
    //--- variables générales
    CCadre* m_pCadre;
    int m_iInterpolation;
    BOOL m_bMasque;
    int  m_iNbcol;
    int  m_iNblig;
    int  m_iMarge;
    int m_iDatasize;
    int m_iTypeAttribut;
    int m_iNbLecture;
    int m_iNbmaxObjets;
    unsigned char* m_ctabImageMasque;
    int  m_iNbVoisins;
    float  m_ftabValeurVoisin[100];
    float  m_ftabContrainteVoisin[100];
    double m_dResolution;
    double m_dtabDistance[100];
    double m_dZmin;
    double m_dZmax;

    BOOL m_bDistanceMax;
    double m_dDistanceMax;

    //--- variables pour l'interpolation sous contrainte
    BOOL m_bContrainte;
    int m_iNothContrainte;
    int m_iNoattContrainte;
    double m_dFacteurContrainte;
    float* m_ftabImagetteContrainte;

    //--- variables pour les objets de base (potentiel)
    float *m_ftabValeur;
    float *m_ftabContrainte;
    double *m_dtabX;
    double *m_dtabY;

    double m_dFlxCalcul;
    double m_dFlyCalcul;
    double m_dFhxCalcul;
    double m_dFhyCalcul;
    double m_dFlxRecherche;
    double m_dFlyRecherche;
    double m_dFhxRecherche;
    double m_dFhyRecherche;
    //--- variables pour le calcul
    int m_iXDebCalcul,m_iYDebCalcul,m_iXFinCalcul,m_iYFinCalcul;
    int m_iXDebRecherche,m_iYDebRecherche,m_iXFinRecherche,m_iYFinRecherche;

    //--- variables pour le lissage
    float *m_fLigne;
    double m_dw1[3];
    double m_dw2[3];
    double m_dp1[20];
    double m_dp2[20];
    double m_dC[20][20];
    int m_iPas;

    void Voisin(float fValeur, float fValeurContrainte,double distance);
    float BaryCalcul(float fValeurContrainte);
    float PotentielCalcul(double dXProj,double dYProj);

  public:
    float *m_ftabImagette;
    int  *m_itabNombre;
    unsigned char* m_ctabData;
    BOOL m_bMoyenne;

    CBabel(CCadre* pCadre,int iInterpolation,int iNbcol,int iNblig,int iMarge,double
dResolution,BOOL bMasque,BOOL bMoyenne,unsigned char* ctabImageMasque);
    ~CBabel();
    BOOL Alloc();
    void SetDistanceMax(BOOL bDistanceMax,double dDistanceMax);
    void SetContrainte(BOOL bContrainte,int iNoth,int iNoatt,double dFacteur);
    void InitFenetre(CWind* pWind,double dFlx,double dFly);
    int Lecture(int iNoImagette,int iNbRelations,int *itabRelation,int *itabAttribut,double
dConstante);
    void LectureContrainte(int iNoImagette);
    BOOL IntersectionMasque();
```

```
    BOOL Barycentre(int iNoImagette,int iPassage);
    BOOL Potentiel();
    BOOL Lissage(int ideb);
    BOOL EcrireImagette(const char* chNomFichier,FILE* FileAttribut);
    BOOL RemplacerImagette(int ptrdeb,FILE* FileAttribut);
    void ProjToImagette(double dXProj,double dYProj,double& dXRas,double& dYRas);

    int GetNblig() {return m_iNblig;};
    int GetNbcol() {return m_iNbcol;};
    int GetMarge() {return m_iMarge;};
    double GetMin() {return m_dZmin;};
    double GetMax() {return m_dZmax;};
    float* GetPtrImagette(int ilig)    {return (float* ) &m_ftabImagette[m_iMarge +
(ilig+m_iMarge)*m_iNbcol];};
  };
```

Par exemple, la méthode `Lecture()` charge les points de références dans la mémoire de l'imagette à créer, à partir des relations et attributs choisis (`itabRelation`, `itabAttribut`) :

```
int CBabel::Lecture(int iNoImagette,int iNbRelations,int *itabRelation,int
*itabAttribut,double dConstante)
  {
  //--- lecture des données
  CSchema* pSchema=m_pCadre->GetSchema();
  int i;
  int iNoth,iNoatt;
  int iNb0,iNba,iNovar;
  float fXSav,fYSav;
  float fWindXbSav,fWindYbSav,fWindXhSav,fWindYhSav;
  float v[NB_MAX_ATTRIBUTS];
  double dXProj,dYProj;

  //--- fenetre de recherche en coordonnees savane
  m_pCadre->ProjToSav(m_dFlxRecherche,m_dFlyRecherche,fWindXbSav,fWindYbSav);
  m_pCadre->ProjToSav(m_dFhxRecherche,m_dFhyRecherche,fWindXhSav,fWindYhSav);
  ::Ordonner(fWindXbSav,fWindXhSav);
  ::Ordonner(fWindYbSav,fWindYhSav);

  //---
  CString strTexte;
  switch(config.m_iLangage)
    {
    case FRANCAIS:
    default:
      strTexte.Format(" Imagette %d : lecture des données...",iNoImagette);
      break;
    case ESPAGNOL:
      strTexte.Format(" Imagette %d :lectura de los datos...",iNoImagette);
      break;
    case ANGLAIS:
      strTexte.Format(" Imagette %d : reading the data...",iNoImagette);
      break;
    }
  pMainFrame->SetStatusTexte(strTexte);
  m_iNbLecture=0;

  switch(m_iInterpolation)
    {
    case INTERPOLATION_BARYCENTRE:
      {
      int iXRas,iYRas;
      double dXRasImagette,dYRasImagette;

      //--- init
      int iNbPixels=m_iNblig*m_iNbcol;
      for(i=0;i < iNbPixels;i++)
        {
        m_itabNombre[i]=0;
        m_ftabImagette[i]=0.F;
        }
      if(m_bMoyenne == FALSE)
        {
        for(i=0;i < iNbPixels;i++)m_ctabData[i]=0;
        }

      //--- écrire dans la mémoire de l'imagette
```

```
int iNoRelation=0;
while(iNoRelation < iNbRelations)
   {
   iNoth=itabRelation[iNoRelation];
   iNoatt=itabAttribut[iNoRelation];

   iNb0=pSchema->m_pRelations[iNoth]->m_iNb0;
   iNba=pSchema->m_pRelations[iNoth]->m_iNba;
   iNovar=pSchema->m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNumero;
   int iTypeRelation=pSchema->m_pRelations[iNoth]->m_iType;

   if(iTypeRelation != 2 && iTypeRelation != 12)
      {
      //--- zone ou point, par centroïde
      CLecture lecture;
      if(lecture.Open(m_pCadre,iNoth))
         {
         lecture.SetFenetreLecture(fWindXbSav,fWindYbSav,fWindXhSav,fWindYhSav);
         i=lecture.Lire(v);
         while(i != -1)
            {
            if(i == 1)
               {
               if(v[iNovar] > FINCONNU)
                  {
                  v[iNovar]*=(float) dConstante;
                  if(v[iNovar] > m_dZmax)m_dZmax=(double) v[iNovar];
                  if(v[iNovar] < m_dZmin)m_dZmin=(double) v[iNovar];
                  lecture.GetCentroide(fXSav,fYSav);
                  m_pCadre->SavToProj(fXSav,fYSav,dXProj,dYProj);
                  ProjToImagette(dXProj,dYProj,dXRasImagette,dYRasImagette);
                  iXRas=(int) (dXRasImagette +0.5);
                  iYRas=(int) (dYRasImagette +0.5);

                  if(iXRas >= 0 && iXRas < m_iNbcol && iYRas >= 0 && iYRas < m_iNblig)
                     {
                     if(m_bMoyenne == FALSE)
                        {
                        if(m_ctabData[iXRas+(iYRas*m_iNbcol)] == 0)
                           {
                           m_itabNombre[iXRas+(iYRas*m_iNbcol)]++;
                           m_ftabImagette[iXRas+(iYRas*m_iNbcol)]+=v[iNovar];
                           m_iNbLecture++;
                           }
                        }
                     else
                        {
                        m_itabNombre[iXRas+(iYRas*m_iNbcol)]++;
                        m_ftabImagette[iXRas+(iYRas*m_iNbcol)]+=v[iNovar];
                        m_iNbLecture++;
                        }
                     }
                  }
               }
            i=lecture.Lire(v);
            }
         lecture.Close();
         }
      }
   else
      {
      int iPtrarc;
      float v[NB_MAX_ATTRIBUTS];
      CArc arc;

      CLecture lecture;
      if(lecture.Open(m_pCadre,iNoth))
         {
         lecture.SetFenetreLecture(fWindXbSav,fWindYbSav,fWindXhSav,fWindYhSav);

         FILE* FileArc=lecture.GetFileArc();
         i=lecture.Lire(v);
         while(i != -1)
            {
            if(i == 1)
               {
               if(v[iNovar] > FINCONNU)
                  {
                  v[iNovar]*=(float) dConstante;
                  if(v[iNovar] > m_dZmax)m_dZmax=(double) v[iNovar];
```

```
                if(v[iNovar] < m_dZmin)m_dZmin=(double) v[iNovar];
                iPtrarc=lecture.GetPointeurArc();
                //--- tester la fenêtre de l'arc

                //--- lecture des points
                arc.LirePoints(FileArc,iPtrarc);
                m_iNbLecture++;
                arc.Draw(m_pCadre,v[iNovar],this);
                }
              }

            i=lecture.Lire(v);
            }
          lecture.Close();
          }
        }

      if(m_bMoyenne == FALSE)
        {
        for(i=0;i < iNbPixels;i++)
          {
          if(m_itabNombre[i] > 0)m_ctabData[i]=1;
          }
        }

      iNoRelation++;
      }

    //--- calcul de la moyenne par pixel a partir de la somme
    for(i=0;i < iNbPixels;i++)
      {
      if(m_itabNombre[i] > 0)m_ftabImagette[i]/=m_itabNombre[i];
      else m_ftabImagette[i]=FINCONNU;
      }
    }
    break;

case INTERPOLATION_POTENTIEL:
case INTERPOLATION_PLUSPROCHEVOISIN:
case INTERPOLATION_SOMME:
case INTERPOLATION_SOMME_DECROISSANTE:
case INTERPOLATION_MOYENNE:
case INTERPOLATION_MOYENNE_DECROISSANTE:
    {
    //--- init
    int iNbPixels=m_iNblig*m_iNbcol;
    for(i=0;i < iNbPixels;i++)
      {
      m_ftabImagette[i]=0.F;
      }

    //--- écrire dans les tableaux
    int iNoRelation=0;
    while(iNoRelation < iNbRelations && m_iNbLecture < m_iNbmaxObjets)
      {
      iNoth=itabRelation[iNoRelation];
      iNoatt=itabAttribut[iNoRelation];

      iNb0=pSchema->m_pRelations[iNoth]->m_iNb0;
      iNba=pSchema->m_pRelations[iNoth]->m_iNba;
      iNovar=pSchema->m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNumero;

      CLecture lecture;
      if(lecture.Open(m_pCadre,iNoth))
        {
        i=lecture.LireSansTestFenetre(v);
        while(i != -1 && m_iNbLecture < m_iNbmaxObjets)
          {
          if(i == 1)
            {
            if(v[iNovar] > FINCONNU)
              {
              v[iNovar]*=(float) dConstante;
              if(v[iNovar] > m_dZmax)m_dZmax=(double) v[iNovar];
              if(v[iNovar] < m_dZmin)m_dZmin=(double) v[iNovar];
              lecture.GetCentroide(fXSav,fYSav);
              m_pCadre->SavToProj(fXSav,fYSav,dXProj,dYProj);

              m_dtabX[m_iNbLecture]=dXProj;
              m_dtabY[m_iNbLecture]=dYProj;
```

```
                m_ftabValeur[m_iNbLecture]=v[iNovar];

                m_iNbLecture++;
                }
              }
            i=lecture.Lire(v);
            }
          lecture.Close();
          }

        iNoRelation++;
        }

      //--- lecture de la contrainte pour tous les points, par appartenance
      if(m_bContrainte)
        {
        FILE *FileMosaic;
        FILE *FileFeuille;
        FILE *FileImage;
        int iMosaiqueNbcol,iMosaiqueNblig,nbcol,nblig;
        int no,ptr,ptrlib,ptrdeb;
        double dXbf,dYbf,dXhf,dYhf,dZmin,dZmax;
        double dMosaiqueResolution;
        char chNomFichierFeuille[_MAX_PATH];
        char chNomFichierMosaique[_MAX_PATH];
        char chNomFichierAttribut[_MAX_PATH];
        char cValeur;
        short int sValeur;
        float fValeur;

        fValeur=0.;
        for(i=0;i < m_iNbLecture;i++)
          {
          m_ftabContrainte[i]=FINCONNU;
          }

        //--- ouverture des fichiers de la mosaïque
        strcpy(chNomFichierMosaique,pSchema->m_pRelations[m_iNothContrainte]->m_chFichZone);
        strcat(chNomFichierMosaique,"\\");
        strcat(chNomFichierMosaique,pSchema->m_pRelations[m_iNothContrainte]->m_chNom);
        strcat(chNomFichierMosaique,"_d");

        //--- lecture des parametres de la mosaique
        FileMosaic=OuvrirMosaique(pSchema,chNomFichierMosaique,m_iNothContrainte);
        if(FileMosaic != NULL)
          {
          fscanf(FileMosaic,"%lf",&dMosaiqueResolution);
          fscanf(FileMosaic,"%d",&iMosaiqueNbcol);
          fscanf(FileMosaic,"%d",&iMosaiqueNblig);
          fclose(FileMosaic);

          int iTypeAttribut=pSchema->m_pRelations[m_iNothContrainte]-
>m_pAttributs[m_iNoattContrainte]->m_iType;
          int iTemporaire=pSchema->m_pRelations[m_iNothContrainte]-
>m_pAttributs[m_iNoattContrainte]->m_iTemporaire;
          if(iTemporaire)
            {
            strcpy(chNomFichierAttribut,carte.m_chNomDir);
            strcat(chNomFichierAttribut,"\\");
            strcat(chNomFichierAttribut,m_pCadre->m_chNom);
            strcat(chNomFichierAttribut,"\\ft");
            strcat(chNomFichierAttribut,pSchema->m_pRelations[m_iNothContrainte]->m_chNom);
            strcat(chNomFichierAttribut,"_");
            strcat(chNomFichierAttribut,pSchema->m_pRelations[m_iNothContrainte]-
>m_pAttributs[m_iNoattContrainte]->m_chNom);
            }
          else
            {
            strcpy(chNomFichierAttribut,pSchema->m_pRelations[m_iNothContrainte]-
>m_chFichZone);
            strcat(chNomFichierAttribut,"\\");
            strcat(chNomFichierAttribut,pSchema->m_pRelations[m_iNothContrainte]-
>m_pAttributs[m_iNoattContrainte]->m_chNom);
            }

          strcpy(chNomFichierFeuille,chNomFichierAttribut);
          strcat(chNomFichierFeuille,"_f");

          FileFeuille=fopen(chNomFichierFeuille,"rb");
          FileImage=fopen(chNomFichierAttribut,"rb");
```

```
            if(FileImage == NULL || FileFeuille == NULL)
               {
               if(FileImage != NULL)fclose(FileImage);
               if(FileFeuille != NULL)fclose(FileFeuille);
               }
            else
               {
               //--- lecture des feuilles
               int iDatasize=1;
               if(iTypeAttribut == 1 || iTypeAttribut == 5 || iTypeAttribut == 6)iDatasize=1;
               else if(iTypeAttribut == 7)iDatasize=2;
               else if(iTypeAttribut == 8)iDatasize=3;
               else if(iTypeAttribut == 9)iDatasize=4;

               fscanf(FileFeuille,"%2d%20d%*c",&i,&ptrlib);
               while(fscanf(FileFeuille,"%2d%20lf%20lf%10d%10d%20lf%20lf%20d%*c",
               &no,&dXbf,&dYbf,&nbcol,&nblig,&dZmin,&dZmax,&ptrdeb) == 8)
                  {
                  if(no != 0)
                     {
                     dXhf=dXbf + (dMosaiqueResolution*(double) iMosaiqueNbcol);
                     dYhf=dYbf + (dMosaiqueResolution*(double) iMosaiqueNblig);

                     for(i=0;i < m_iNbLecture;i++)
                        {
                        if(m_ftabContrainte[i] == FINCONNU)
                           {
                           if(m_dtabX[i] >= dXbf && m_dtabX[i] <= dXhf && m_dtabY[i] >= dYbf &&
m_dtabY[i] <= dYhf)
                              {
                              ptr=ptrdeb + (((int) ((m_dtabY[i]-
dYbf)/dMosaiqueResolution))*iMosaiqueNbcol + (int) ((m_dtabX[i]-
dXbf)/dMosaiqueResolution))*iDatasize;
                              fseek(FileImage,ptr,SEEK_SET);
                              switch(iTypeAttribut)
                                 {
                                 case 1:
                                 case 5:
                                 case 6:
                                    fread(&cValeur,iDatasize,1,FileImage);
                                    if(cValeur != CINCONNU)m_ftabContrainte[i]=(float) cValeur;
                                    break;
                                 case 7:
                                    fread(&sValeur,iDatasize,1,FileImage);
                                    if(sValeur != SINCONNU)m_ftabContrainte[i]=(float) sValeur;
                                    break;
                                 case 9:
                                    fread(&fValeur,iDatasize,1,FileImage);
                                    if(fValeur != FINCONNU)m_ftabContrainte[i]=fValeur;
                                    break;
                                 default: break;
                                 }
                              }
                           }
                        }
                     }
                  }

               fclose(FileImage);
               fclose(FileFeuille);
               }
            }
         }
      }
      break;

   default:
     break;
   }
```

Dans le cas de l'interpolation barycentrique sur les voisins, la méthode `Barycalcul()` effectue le calcul barycentrique sur les voisins qui ont été précédemment déterminés par la méthode `Barycentre()`.

```
float CBabel::BaryCalcul(float fValeurContrainte)
  {
  int k;
```

```
   for(k=0;k < m_iNbVoisins;k++)
      {
      if(m_dtabDistance[k] <= 0.)return m_ftabValeurVoisin[k];
      }

   float fz=0.F;
   double ds1,ds2,du;
   ds1=0.;
   ds2=0.;
   if(m_bContrainte && fValeurContrainte > FINCONNU)
      {
      for(k=0;k < m_iNbVoisins;k++)
         {
         du=1./m_dtabDistance[k];
         ds1+=du;
         if(m_ftabContrainteVoisin[k] > FINCONNU)
            {
            ds2+=(((double)m_ftabValeurVoisin[k] +
m_dFacteurContrainte*(m_ftabContrainteVoisin[k]- fValeurContrainte))*du);
            }
         else ds2+=((double)m_ftabValeurVoisin[k]*du);
         }
      fz=(float) (ds2/ds1);
      }
   else
      {
      for(k=0;k < m_iNbVoisins;k++)
         {

         du=1./m_dtabDistance[k];
         ds1+=du;
         ds2+=((double)m_ftabValeurVoisin[k]*du);
         }
      fz=(float) (ds2/ds1);
      }

   return fz;
   }
```

Dans le cas de l'interpolation sur tous les points sans contrainte, le calcul est effectué par la méthode `PotentielCalcul()`. Les points de référence pris en compte sont filtrés sur la distance au point à interpoler (`m_dDistanceMax`) lorsque cette option a été choisie par l'utilisateur (`m_bDistanceMax`) :

```
float CBabel::PotentielCalcul(double dXProj,double dYProj)
   {
   float fz=0.F;
   switch(m_iInterpolation)
      {
      case INTERPOLATION_POTENTIEL: //--- barycentre sur tous les points
         {
         double ds1,ds2,du,dDistance;
         ds1=0.;
         ds2=0.;
         if(m_bDistanceMax)
            {
            for(int i=0;i < m_iNbLecture;i++)
               {
               dDistance=sqrt((dXProj-m_dtabX[i])*(dXProj-m_dtabX[i]) + (dYProj-
m_dtabY[i])*(dYProj-m_dtabY[i]));
               if(dDistance == 0.)
                  {
                  fz=m_ftabValeur[i];
                  return fz;
                  }

               if(dDistance <= m_dDistanceMax)
                  {
                  du=1./dDistance;
                  ds1+=du;
                  ds2+=((double) m_ftabValeur[i]*du);
                  }
               }
            if(ds1 > 0.)fz=(float) (ds2/ds1);
            else fz=FINCONNU;
            }
```

```
       else
          {
          for(int i=0;i < m_iNbLecture;i++)
             {
             dDistance=sqrt((dXProj-m_dtabX[i])*(dXProj-m_dtabX[i]) + (dYProj-
m_dtabY[i])*(dYProj-m_dtabY[i]));
             if(dDistance == 0.)
                {
                fz=m_ftabValeur[i];
                return fz;
                }
             du=1./dDistance;
             ds1+=du;
             ds2+=((double) m_ftabValeur[i]*du);
             }
          fz=(float) (ds2/ds1);
          }

       return fz;
       }
       break;

    case INTERPOLATION_PLUSPROCHEVOISIN: //--- plus proche voisin
       {
       double dDistance,dDistanceMin;

       fz=m_ftabValeur[0];
       dDistanceMin=(dXProj-m_dtabX[0])*(dXProj-m_dtabX[0]) + (dYProj-m_dtabY[0])*(dYProj-
m_dtabY[0]);
       for(int i=1;i < m_iNbLecture;i++)
          {
          dDistance=(dXProj-m_dtabX[i])*(dXProj-m_dtabX[i]) + (dYProj-m_dtabY[i])*(dYProj-
m_dtabY[i]);
          if(dDistance <= dDistanceMin)
             {
             fz=m_ftabValeur[i];
             dDistanceMin=dDistance;
             }
          }
       return fz;
       }
       break;

    case INTERPOLATION_SOMME:
       {
       int iNb=0;
       float fSomme;
       double dDistance;
       fSomme=0.F;
       for(int i=0;i < m_iNbLecture;i++)
          {
          dDistance=sqrt((dXProj-m_dtabX[i])*(dXProj-m_dtabX[i]) + (dYProj-m_dtabY[i])*(dYProj-
m_dtabY[i]));
          if(dDistance <= m_dDistanceMax)
             {
             fSomme+=m_ftabValeur[i];
             iNb++;
             }
          }
       if(iNb == 0)fSomme=FINCONNU;

       return fSomme;
       }
       break;

    case INTERPOLATION_SOMME_DECROISSANTE:
       {
       int iNb=0;
       float fSomme;
       double dSomme,dDistance;
       dSomme=0.;
       for(int i=0;i < m_iNbLecture;i++)
          {
          dDistance=sqrt((dXProj-m_dtabX[i])*(dXProj-m_dtabX[i]) + (dYProj-m_dtabY[i])*(dYProj-
m_dtabY[i]));
          if(dDistance <= m_dDistanceMax)
             {
             dSomme+=((double)m_ftabValeur[i]*(m_dDistanceMax - dDistance)/m_dDistanceMax);
             iNb++;
             }
```

```
        }
      if(iNb == 0)fSomme=FINCONNU;
      else fSomme=(float) dSomme;

      return fSomme;
      }
      break;

    case INTERPOLATION_MOYENNE:
      {
      int iNb=0;
      float fSomme;
      double dDistance;
      fSomme=0.F;
      for(int i=0;i < m_iNbLecture;i++)
        {
        dDistance=sqrt((dXProj-m_dtabX[i])*(dXProj-m_dtabX[i]) + (dYProj-m_dtabY[i])*(dYProj-
m_dtabY[i]));
        if(dDistance <= m_dDistanceMax)
          {
          fSomme+=m_ftabValeur[i];
          iNb++;
          }
        }
      if(iNb == 0)fSomme=FINCONNU;
      else fSomme/=iNb;

      return fSomme;
      }
      break;

    case INTERPOLATION_MOYENNE_DECROISSANTE:
      {
      int iNb=0;
      float fSomme;
      double dSomme,dDistance;
      dSomme=0.;
      for(int i=0;i < m_iNbLecture;i++)
        {
        dDistance=sqrt((dXProj-m_dtabX[i])*(dXProj-m_dtabX[i]) + (dYProj-m_dtabY[i])*(dYProj-
m_dtabY[i]));
        if(dDistance <= m_dDistanceMax)
          {
          dSomme+=((double)m_ftabValeur[i]*(m_dDistanceMax - dDistance)/m_dDistanceMax);
          iNb++;
          }
        }
      if(iNb == 0)fSomme=FINCONNU;
      else fSomme=(float) (dSomme/iNb);

      return fSomme;
      }
      break;

    default:
      break;
    }

  return fz;
  }
```

A.2.2. Algorithmique graphique

A.2.2.1. La classe CArc

La classe `CArc` représente un objet de type arc; les points sont conservés dans un tableau m_tpoint, alloué dynamiquement lors de la création de l'objet arc :

```
class CArc
  {
  private:
    int m_iNbpt;
    int m_iNzl;
```

```
    int m_iNz2;
    int m_Ptrsuiv;
    float m_fXmin,m_fYmin,m_fXmax,m_fYmax;
    float* m_tpoint;

  public:
    CArc();
    ~CArc();

    int GetPtrSuivant() {return m_Ptrsuiv;};
    int GetNbpt() {return m_iNbpt;};
    void SetNbpt(int iNbpt) {m_iNbpt=iNbpt;};
    void GetPoint(int i,float &fX,float &fY) {fX=m_tpoint[(i-1)*2]; fY=m_tpoint[(i-1)*2+1];};
    void SetPoint(int i,float fX,float fY) {m_tpoint[(i-1)*2]=fX; m_tpoint[(i-1)*2+1]=fY;};

    void LireEntete(FILE* FileArc,int ptr,int& iNz1,int& iNz2,int& ptrSuivant);
    void LireFenetre(FILE* FileArc,int ptr);
    int LireNbpt(FILE* FileArc,int ptr);
    void LirePoints(FILE* FileArc,int ptr);
    void LireBout(FILE* FileArc,int ptr,float &fX1,float &fY1,float &fX2,float &fY2);
    void GetFenetre(FILE* FileArc,int ptr,float& fX1,float& fY1,float& fX2,float& fY2);
    int Ecrire(FILE* FileArc,int iNz1,int iNz2,int iPtr);
    BOOL Intersect(float fX1,float fY1,float fX2,float fY2);
    BOOL IntersectMasque(CCadre* pCadre,unsigned char *itabImage);
    void Filtre(CCadre* pCadre,double dSurfaceMinimum);
    void Douglas(CCadre* pCadre,double dDistanceMinimum,int iNo1,int iNo2);
    void FinDouglas();
    void Tracer(CDC* pDC,CCadre* pCadre);
    void Tracer(CDC* pDC,CCadre* pCadre,int iType);
    void Imprimer(CDC* pDC,CCadre* pCadre,BOOL bSens);
    void Chainer(CDC* pDC,CCadre* pCadre,BOOL bSens,int& iNbptTotal,LPPOINT tabPoint);
    void Chainer(CCadre* pCadre,BOOL bSens,int& iNbptTotal,int& iNbptPolygon,double*
dtabPoints);
    double DistanceAUnPoint(CCadre* pCadre,double dX,double dY);
    double DistanceAUnPoint(float fXSav,float fYSav);
    double Longueur(CCadre* pCadre);
    void CalculCentroide(float &fX,float &fY);
    void CalculBBox(CDC* pDC,CCadre* pCadre,int &iXb,int &iYb,int &iXh,int &iYh);
    void CalculBBox(CCadre* pCadre,double &dXbProj,double &dYbProj,double &dXhProj,double
&dYhProj);
    void Draw(CCadre* pCadre,unsigned char *itabImage,unsigned char iValeur);
    void Draw(CCadre* pCadre,int *itabImage,int iValeur);
    void Draw(CCadre* pCadre,float fValeur,CBabel* pBabel);
    void Draw0(CCadre* pCadre,unsigned char *itabImage,unsigned char iValeur);
    void Draw0(CCadre* pCadre,float *ftabImage,int *itabNombre,float fValeur,int iNbcol, int
iNblig);
  };
```

Voici par exemple la procédure de recherche de l'intersection d'un arc avec un segment. La procédure utilise une autre procédure globale qui recherche l'intersection de deux segments de droite (A.2.2.5) :

```
BOOL CArc::IntersectionAvecUnSegment(double dX1Seg,double dY1Seg,double dX2Seg,double
dY2Seg,double& dXRes,double& dYRes)
  {
  dXRes=0.;
  dYRes=0.;

  //--- tester l'intersection du segment avec la fenêtre de l'arc
  ::Ordonner(dX1Seg,dX2Seg);
  ::Ordonner(dY1Seg,dY2Seg);
  if(Intersect(dX1Seg,dY1Seg,dX2Seg,dY2Seg) == FALSE)return FALSE;

  //--- tester l'intersection de l'arc et du segment
  double dX1,dY1,dX2,dY2;
  for(int i=1;i < m_iNbpt;i++)
    {
    GetPoint(i-1,dX1,dY1);
    GetPoint(i,dX2,dY2);
   if(SegmentIntersection(dX1Seg,dY1Seg,dX2Seg,dY2Seg,dX1,dY1,dX2,dY2,dXRes,dYRes))return
TRUE;
    }
  return FALSE;
  }
```

La méthode `Draw()` permet de dessiner un arc dans un tableau matriciel. Cette procédure est notamment utilisée dans l'interpolation pour charger les points de références dans la matrice de l'imagette à calculer :

```
void CArc::Draw(CCadre* pCadre,float fValeur,CBabel* pBabel)
   {
   int ipt,nbpix,iX,iY,iXp,iYp,nb;
   int iNbcol,iNblig;
   float fX,fY;
   double dCosinus,dSinus,dDistance,dR;
   double dXProj,dYProj,dXa,dYa,dXb,dYb,dX,dY;

   iNbcol=pBabel->GetNbcol();
   iNblig=pBabel->GetNblig();

   fX=m_tpoint[0];
   fY=m_tpoint[1];
   pCadre->SavToProj(fX,fY,dXProj,dYProj);
   pBabel->ProjToImagette(dXProj,dYProj,dXa,dYa);
   iX=(int) dXa;
   iY=(int) dYa;
   if(iX >= 0 && iX < iNbcol && iY >= 0 && iY < iNblig)
      {
      if(pBabel->m_bMoyenne)
         {
         pBabel->m_ftabImagette[iX+(iY*iNbcol)]+=fValeur;
         pBabel->m_itabNombre[iX+(iY*iNbcol)]++;
         }
      else
         {
         if(pBabel->m_ctabData[iX+(iY*iNbcol)] == 0)
            {
            pBabel->m_ftabImagette[iX+(iY*iNbcol)]+=fValeur;
            pBabel->m_itabNombre[iX+(iY*iNbcol)]++;
            }
         }
      }
   iXp=iX;
   iYp=iY;
   ipt=1;
   int j=2;
   while(ipt < m_iNbpt)
      {
      fX=m_tpoint[j];
      fY=m_tpoint[j+1];
      j+=2;
      pCadre->SavToProj(fX,fY,dXProj,dYProj);
      pBabel->ProjToImagette(dXProj,dYProj,dXb,dYb);
      dX=dXb-dXa;
      dY=dYb-dYa;
      dDistance=sqrt(dX*dX + dY*dY);
      if(dDistance > 0.)
         {
         dCosinus=dX/dDistance;
         dSinus=dY/dDistance;
         nbpix=int(dDistance);
         nb=1;
         while(nb < nbpix)
            {
            dR=(double) nb;
            iX=(int) (dXa + dR*dCosinus);
            iY=(int) (dYa + dR*dSinus);
            if(iX != iXp || iY != iYp)
               {
               if(iX >= 0 && iX < iNbcol && iY >= 0 && iY < iNblig)
                  {
                  if(pBabel->m_bMoyenne)
                     {
                     pBabel->m_ftabImagette[iX+(iY*iNbcol)]+=fValeur;
                     pBabel->m_itabNombre[iX+(iY*iNbcol)]++;
                     }
                  else
                     {
                     if(pBabel->m_ctabData[iX+(iY*iNbcol)] == 0)
                        {
                        pBabel->m_ftabImagette[iX+(iY*iNbcol)]+=fValeur;
                        pBabel->m_itabNombre[iX+(iY*iNbcol)]++;
                        }
```

```
                }
              }
            iXp=iX;
            iYp=iY;
            }
          nb++;
          }
        iX=(int) dXb;
        iY=(int) dYb;
        if(iX != iXp || iY != iYp)
          {
          if(iX >= 0 && iX < iNbcol && iY >= 0 && iY < iNblig)
            {
            if(pBabel->m_bMoyenne)
              {
              pBabel->m_ftabImagette[iX+(iY*iNbcol)]+=fValeur;
              pBabel->m_itabNombre[iX+(iY*iNbcol)]++;
              }
            else
              {
              if(pBabel->m_ctabData[iX+(iY*iNbcol)] == 0)
                {
                pBabel->m_ftabImagette[iX+(iY*iNbcol)]+=fValeur;
                pBabel->m_itabNombre[iX+(iY*iNbcol)]++;
                }
              }
            }
          }
        }
      dXa=dXb;
      dYa=dYb;
      ipt++;
      }
  }
```

A.2.2.2. Les classes CCellule et CYX

Les classes `CCellule` et `CYX` sont utilisées pour les procédures de rasterisation. Un objet `CCellule` représente une cellule d'un chaînage, et conserve une coordonnée (`m_iX`), ainsi que les codes à gauche et à droite du point sur la ligne chaînée. Les cellules peuvent être chaînées grâce à la variable `m_ptrSuivant`. La classe CYX contient un tableau de pointeurs de cellules : chaque pointeur correspond à la première cellule d'une ligne horizontale.

```
class CCellule
  {
  public:
    int m_iX;
    int m_iCode1;
    int m_iCode2;
    CCellule* m_ptrSuivant;

  public:
    CCellule();
  };


class CYX
  {
  private:
    CCadre* m_pCadre;
    CCellule* m_Y[NB_MAX_DEFY];
    int m_iNoth;

    void YX(char *chNomFichierImage);
    void Points(FILE* FileArc,int iNbarc,int iPtrarc,int *itabZone,int iNzMax);
    void InsererCellule(int iY,int iX,int iCode1,int iCode2);

  public:
    CYX(CCadre* pCadre);
    ~CYX();
    void Rasteriser(int iNoth);
    void RasteriserParAttribut(int iNoth,int iNoatt,int* itabImage);
    void MasqueRasteriser(char *chNomFichierMasque,char *chNomFichierImage);
```

```
};
```

La procédure `Points()` calcule l'intersection des arcs de la relation avec les lignes horizontales et crée le chaînage des points par ligne. La procédure `YX()` calcule le fichier image à partir du chaînage par ligne. Voici l'implémentation des principales procédures de cette classe :

```
void CYX::Points(FILE* FileArc,int iNbarc,int iPtrarc,int *itabZone,int iNzMax)
  {
  int i;
  int iCode1,iCode2,iX,iXb,iXh,iYb,iYh;
  int ligsup,iNb,iNbpt,nocol,nolig,iNz1,iNz2;
  int ptr,ptrSuivant;
  float fPx1,fPy1,fPx2,fPy2,fP,fX,fY;
  CArc arc;

  int iXMaximum=config.m_iDefX-1;

  ptr=iPtrarc;
  iNb=0;
  while(iNb < iNbarc)
    {
    arc.LireEntete(FileArc,ptr,iNz1,iNz2,ptrSuivant);

    if(iNz1 <= iNzMax)iCode1=itabZone[iNz1]; else iCode1=0;
    if(iNz2 <= iNzMax)iCode2=itabZone[iNz2]; else iCode2=0;
    if(iCode1 != iCode2)
      {
      arc.LirePoints(FileArc,ptr);
      arc.GetPoint(1,fX,fY);
      m_pCadre->SavToRas(fX,fY,fPx1,fPy1);
      iNbpt=arc.GetNbpt();
      i=2;
      while(i <= iNbpt)
        {
        fPx2=fPx1;
        fPy2=fPy1;
        arc.GetPoint(i,fX,fY);
        m_pCadre->SavToRas(fX,fY,fPx1,fPy1);
        if(fPy1 < fPy2)
          {
          iXb=(int) (fPx1+0.5);
          iYb=(int) fPy1;
          iXh=(int) (fPx2+0.5);
          iYh=(int) fPy2;
          }
        else
          {
          iXb=(int) (fPx2+0.5);
          iYb=(int) fPy2;
          iXh=(int) (fPx1+0.5);
          iYh=(int) fPy1;
          }
        if(iYh != iYb)
          {
          if(iYb >= 0 && iYb < config.m_iDefY)
            {
            nocol=__max(0,__min(iXMaximum,iXb));
            InsererCellule(iYb,nocol,iCode1,iCode2);
            }
          fP=(float)(iXh-iXb)/(float)(iYh-iYb);
          fX=(float)iXb+fP;
          nolig=iYb+1;
          ligsup=__min(config.m_iDefY,iYh);
          if(nolig < ligsup)
            {
            while(nolig < 0)
              {
              fX+=fP;
              nolig++;
              }
            while(nolig < ligsup)
              {
              iX=(int) (fX+0.5);
              nocol=__max(0,__min(iXMaximum,iX));
```

```
                InsererCellule(nolig,nocol,iCode1,iCode2);
                nolig++;
                fX+=fP;
                }
              }
            }
          i++;
          }
        }
      ptr=ptrSuivant;
      iNb++;
      }
   }

void CYX::YX(char *chNomFichierImage)
   {
   FILE *FileImage;
   CCellule *ptr;
   int i,iC,recherche;
   int iCode,iCode1,iCode2;
   int iXc,iXp,iXs,iSegment,iNbpt;
   int multiple[1000];

   FileImage=fopen(chNomFichierImage,"wb");
   if(FileImage == NULL)return;

   int iXMaximum=config.m_iDefX-1;

   int ilig=0;
   while(ilig < config.m_iDefY)
     {
     ptr=m_Y[ilig];
     if(ptr == 0)
       {
       //--- ligne vide
       iCode=0;
       fprintf(FileImage,"%d %d\n",iCode,config.m_iDefX);
       }
     else
       {
       //--- calcul des segments
       iCode=0;
       iSegment=0;
       iNbpt=0;
      iXc=ptr->m_iX;
      iCode1=ptr->m_iCode1;
      iCode2=ptr->m_iCode2;
       if(iXc > 0)
          {
          fprintf(FileImage,"%d %d\n",iCode,iXc);
          iNbpt+=iXc;
          }
      iC=1;
      multiple[1]=0;
      iXp=iXc;
       do
          {
          //--- repeat tant qu'il y a des points dans la ligne
          do
             {
             //--- repeat tant qu'il y a des points confondus
             i=1;
             recherche=1;
             while(i <= iC && recherche)
                {
                if(iCode1 != multiple[i])i++;
                else
                   {
                   recherche=0;
                   while(i < iC)
                      {
                      multiple[i]=multiple[i+1];
                      i++;
                      }
                   iC--;
                   }
                }
             if(recherche)
                {
                iC++;
```

```
            multiple[iC]=iCode1;
            }
          i=1;
          recherche=1;
          while(i <= iC && recherche)
            {
            if(iCode2 != multiple[i])i++;
            else
              {
              recherche=0;
              while(i < iC)
                {
                multiple[i]=multiple[i+1];
                i++;
                }
              iC--;
              }
            }
          if(recherche)
            {
            iC++;
            multiple[iC]=iCode2;
            }

          ptr=ptr->m_ptrSuivant;
          if(ptr != 0)
            {
            iXs=ptr->m_iX;
            iCode1=ptr->m_iCode1;
            iCode2=ptr->m_iCode2;
            }
          }
        while(ptr != 0 && iXs == iXc);

        if(ptr != 0)
          {
          //--- determiner le code sortant
          if(iC == 1)
            {
            iCode=multiple[1];
            iSegment=iXs-iXp;
            if(iXs == iXMaximum)iSegment++;
            fprintf(FileImage,"%d %d\n",iCode,iSegment);
            iNbpt+=iSegment;
            iC=1;
            iXp=iXs;
            }
          iXc=iXs;
          }
        }
      while(ptr != 0);

      if(iNbpt < config.m_iDefX)
        {
        iCode=0;
        iSegment=config.m_iDefX-iNbpt;
        fprintf(FileImage,"%d %d\n",iCode,iSegment);
        }
      }
    ilig++;
    }
  fclose(FileImage);
  }

void CYX::InsererCellule(int iY,int iX,int iCode1,int iCode2)
  {
  CCellule* ptrNew;
  CCellule* ptrTete;

  //--- creation d'une cellule
  ptrNew=new CCellule();
  ptrNew->m_iX=iX;
  ptrNew->m_iCode1=iCode1;
  ptrNew->m_iCode2=iCode2;

  ptrTete=m_Y[iY];
  if(ptrTete == 0)
    {
    //--- premier élément de la ligne
    m_Y[iY]=ptrNew;
```

```
    ptrNew->m_ptrSuivant=0;
    }
  else
    {
    //--- insertion
    if(ptrTete->m_iX > iX)
      {
      ptrNew->m_ptrSuivant=ptrTete;
      m_Y[iY]=ptrNew;
      }
    else
      {
      CCellule* ptr;
      CCellule* ptrPred;

      ptrPred=ptrTete;
      ptr=ptrTete->m_ptrSuivant;
      while(ptr > 0)
        {
        if(ptr->m_iX <= iX)
          {
          ptrPred=ptr;
          ptr=ptr->m_ptrSuivant;
          }
        else break;
        }
      //--- chainage
      ptrNew->m_ptrSuivant=ptr;
      ptrPred->m_ptrSuivant=ptrNew;
      }
    }
  }
```

A.2.2.3. la classe CVectoriser

La classe `CVectoriser` permet de réaliser l'opération inverse de la rasterisation. Elle crée des arcs à partir d'une image raster :

```
class CVectoriser
  {
  private:
    CCadre* m_pCadre;
    int m_iNoth;
    int m_iNbObjets;
    int m_iNbArcCree;
    int m_iPtrlib;
    char m_chNomFichierArc[_MAX_PATH];
    int *m_itabImage;
    unsigned char *m_ctabSegment;
    float* m_ligx;
    float* m_ligy;
    int m_iNbpoint;
    FILE* m_FileArc;
    BOOL m_bPasDePixelsIsoles;

    int SegmentType(int iX,int iY);
    void CreationDesArcs(void);
    void EcrireUneLigne(int iNza,int iNzb);

  public:
    CVectoriser(CCadre* pCadre,BOOL bPasDePixelsIsoles);
    ~CVectoriser();
    int GetNbarc() {return m_iNbArcCree;};

    BOOL Relation(int iNoth);
  };
```

Par exemple, la procédure `SegmentType()` permet d'initialiser les valeurs de chaque configuration de quatre pixel adjacents :

```
int CVectoriser::SegmentType(int iX,int iY)
  {
  int iIndice;
  int iType;
  int iNz1,iNz2,iNz3,iNz4;
```

```
iIndice=iY*config.m_iDefX+iX;
iNz1=m_itabImage[iIndice];
iNz2=m_itabImage[iIndice+1];
iIndice+=config.m_iDefX;
iNz3=m_itabImage[iIndice];
iNz4=m_itabImage[iIndice+1];

if(iNz1 == iNz2 && iNz1 == iNz3 && iNz1 == iNz4)iType=0;
else if(iNz1 == iNz2 && iNz1 == iNz4)iType=1;
else if(iNz1 == iNz2 && iNz1 == iNz3)iType=2;
else if(iNz1 == iNz3 && iNz1 == iNz4)iType=3;
else if(iNz2 == iNz3 && iNz2 == iNz4)iType=4;
else if(iNz1 == iNz2 && iNz3 == iNz4)iType=11;
else if(iNz1 == iNz3 && iNz2 == iNz4)iType=12;
else if(iNz1 == iNz4 && iNz2 == iNz3)iType=13;
else if(iNz1 == iNz2)iType=5;
else if(iNz1 == iNz3)iType=6;
else if(iNz1 == iNz4)iType=7;
else if(iNz3 == iNz4)iType=8;
else if(iNz2 == iNz3)iType=9;
else if(iNz2 == iNz4)iType=10;
else iType=13;

return iType;
}
```

Le code de la procédure `CreationDesArcs()` est très long, car le programme teste toutes les configurations possibles. En voici un extrait :

```
int iDefX=config.m_iDefX-1;
int iDefY=config.m_iDefY-1;
ilig=0;
while(ilig < iDefY)
   {
   icol=0;
   while(icol < iDefX)
      {
      iIndiceDebut=icol+(config.m_iDefX*ilig);
      iseg=m_ctabSegment[iIndiceDebut];
      if(iseg != 0)
         {
         //--- un segment non chaine, début d'une ligne, initialisation
         icol1=icol;
         ilig1=ilig;
         icol2=icol;
         ilig2=ilig;
         col=(float)icol;
         lig=(float)ilig;
         icote1=0;
         icote2=0;
         fin1=0;
         fin2=1;
         switch(iseg)
            {
            case 1:
               iType=SegmentType(icol,ilig);
               if(iType == 1 || iType == 7 || iType == 10)
                  {
                  x0=col;
                  y0=lig+0.5F;
                  xc=col+0.5F;
                  yc=lig+1.F;
                  icote1=2;
                  ilig1++;
                  if(ilig1 >= iDefY)fin1=1;
                  fin2=1;
                  iNz1=m_itabImage[iIndiceDebut+config.m_iDefX];
                  iNz2=m_itabImage[iIndiceDebut+config.m_iDefX+1];
                  }
               else if(iType == 5)
                  {
                  x0=col;
                  y0=lig+0.5F;
                  xc=col+0.5F;
                  yc=lig+1.F;
                  fin1=1;
                  iNz1=m_itabImage[iIndiceDebut];
```

```
          iNz2=m_itabImage[iIndiceDebut+config.m_iDefX];
          }
        m_ctabSegment[iIndiceDebut]=0;
        break;
      case 2:
        iType=SegmentType(icol,ilig);
        if(iType == 2 || iType == 9)
          {
          x0=col+0.5F;
          y0=lig+1.F;
          xc=col+1.F;
          yc=lig+0.5F;
          icote1=1;
          icol1++;
          if(icol1 >= iDefX)fin1=1;
           iNz1=m_itabImage[iIndiceDebut+1];
           iNz2=m_itabImage[iIndiceDebut+1+config.m_iDefX];
          icote2=2;
          ilig2++;
          fin2=0;
          if(ilig2 >= iDefY)fin2=1;
          }
        else if(iType == 5)
          {
          x0=col+0.5F;
          y0=lig+1.F;
          xc=col+1.F;
          yc=lig+0.5F;
          icote1=1;
          icol1++;
          if(icol1 >= iDefX)fin1=1;
          iNz1=m_itabImage[iIndiceDebut+1];
          iNz2=m_itabImage[iIndiceDebut+1+config.m_iDefX];
          }
        else if(iType == 6)
          {
          x0=col+1.F;
          y0=lig+0.5F;
          xc=col+0.5F;
          yc=lig+1.F;
         icote1=2;
         ilig1++;
         if(ilig1 >= iDefY)fin1=1;
          iNz1=m_itabImage[iIndiceDebut+config.m_iDefX];
          iNz2=m_itabImage[iIndiceDebut+1+config.m_iDefX];
          }
        m_ctabSegment[iIndiceDebut]=0;
        break;             …
        …
        …
        …
      case 26:
        x0=col+0.5F;
        y0=lig;
        xc=col+0.5F;
        yc=lig+0.5F;
        fin1=1;
        iNz1=m_itabImage[iIndiceDebut];
        iNz2=m_itabImage[iIndiceDebut+1];
        m_ctabSegment[iIndiceDebut]=0;
        break;
      case 27:
        x0=col;
        y0=lig+0.5F;
        xc=col+0.5F;
        yc=lig+0.5F;
        fin1=1;
        iNz1=m_itabImage[iIndiceDebut];
        iNz2=m_itabImage[iIndiceDebut+config.m_iDefX];
        m_ctabSegment[iIndiceDebut]=0;
        break;
      default:
        break;
      }

    iNza=19+iNz1;
    iNzb=19+iNz2;
    nbpt1=0;
    lig1x[nbpt1]=x0;
lig1y[nbpt1]=y0;
```

```
nbpt1++;
//--- fin initialisation de la ligne, debut chainage avant
while(fin1 == 0)
  {
  iIndice=icol1+(ilig1*config.m_iDefX);
  iseg=m_ctabSegment[iIndice];
  if(iseg == 0)fin1=1;
  else
    {
    col=(float)icol1;
    lig=(float)ilig1;
    switch(icote1)
      {
      case 1: //--- icote1
        switch(iseg)
          {
          case 1: //--- iseg
            iType=SegmentType(icol1,ilig1);
            if(iType == 1 || iType == 7)
              {
              m_ctabSegment[iIndice]=0;
              lig1x[nbpt1]=xc;
              lig1y[nbpt1]=yc;
              nbpt1++;
              xc=col+0.5F;
              yc=lig+1.F;
              icote1=2;
              ilig1++;
              }
            else if(iType == 5)
              {
              m_ctabSegment[iIndice]=0;
              lig1x[nbpt1]=xc;
              lig1y[nbpt1]=yc;
              nbpt1++;
              xc=col+0.5F;
              yc=lig+1.F;
              fin1=1;
              }
            else if(iType == 10)fin1=1;
            break;
          case 4: //--- iseg
            iType=SegmentType(icol1,ilig1);
            if(iType == 4 || iType == 9)
              {
              m_ctabSegment[iIndice]=0;
              lig1x[nbpt1]=xc;
              lig1y[nbpt1]=yc;
              nbpt1++;
              xc=col+0.5F;
              yc=lig;
              icote1=4;
              ilig1--;
              }
            else if(iType == 8)
              {
              m_ctabSegment[iIndice]=0;
              lig1x[nbpt1]=xc;
              lig1y[nbpt1]=yc;
              nbpt1++;
              xc=col+0.5F;
              yc=lig;
              fin1=1;
              }
            else if(iType == 10)fin1=1;
            break;
etc….
```

Le programme teste ainsi l'ensemble des configurations possibles en fonction du segment précédent, du segment courant, et du type temporaire de segment. Il avance ainsi de segment en segment ou s'arrête s'il rencontre un nœud. La recherche est effectuée dans les deux sens. Il passe à la création d'un nouvel arc s'il reste encore des segments non chaînés dans l'image.

A.2.2.4. DilaterImage()

Cette fonction globale permet de créer un masque en dilatant une image de `iDistancePixel` pixels :

```
void DilaterImage(int* itabImage,int iDistancePixel,unsigned char* ctabMasque)
  {
  int* itabNewImage=(int *) malloc(config.m_iDefX*config.m_iDefY*sizeof(int));
  if(itabNewImage == NULL)
    {
    ::ErreurDeMemoire();
    return;
    }

  //--- existence du masque
  BOOL bMasque=TRUE;
  if(ctabMasque == NULL)bMasque=FALSE;

  if(iDistancePixel > 200)iDistancePixel=200;

  //--- préparation des pixels de la recherche
  int i,j,ilig,icol,iligAuCarre,iD;
  int iDistancePixelAuCarre=iDistancePixel*iDistancePixel;

  int iNbPixels=(2*iDistancePixel+1)*(2*iDistancePixel+1);
  int iSizeElement=3*sizeof(int);
  unsigned char* ctabElements=(unsigned char *) malloc(iNbPixels*iSizeElement);
  if(ctabElements == NULL)
    {
    ::ErreurDeMemoire();
    free(itabNewImage);
    return;
    }

  //--- tableau non trié des pixels
  int iNbElements=0;
  j=0;
  for(ilig=-iDistancePixel;ilig <= iDistancePixel;ilig++)
    {
    iligAuCarre=ilig*ilig;
    for(icol=-iDistancePixel;icol <= iDistancePixel;icol++)
      {
      iD=iligAuCarre + icol*icol;
      if(iD > 0 && iD <= iDistancePixelAuCarre)
        {
        //--- ajouter au tableau à trier
        memcpy(&ctabElements[j],&iD,sizeof(int));
        j+=sizeof(int);
        memcpy(&ctabElements[j],&icol,sizeof(int));
        j+=sizeof(int);
        memcpy(&ctabElements[j],&ilig,sizeof(int));
        j+=sizeof(int);
        iNbElements++;
        }
      }
    }

  //--- tri sur la distance
  qsort(ctabElements,iNbElements,iSizeElement,comparnumDistance);

  //--- traitement
  int iCode,iCodeARemplir;
  unsigned char cCodeMasque=1;
  int iX,iY,iX0,iY0,iIndice;

  iCodeARemplir=0;
  iY=0;
  while(iY < config.m_iDefY)
    {
    iIndice=iY*config.m_iDefX;
    iX=0;
    while(iX < config.m_iDefX)
      {
      iCode=itabImage[iIndice];
      if(bMasque)cCodeMasque=ctabMasque[iIndice];
      if(iCode == iCodeARemplir && cCodeMasque == 1)
        {
        j=0;
```

```
        for(i=0;i < iNbElements;i++)
          {
          j+=sizeof(int);
          memcpy(&iX0,&ctabElements[j],sizeof(int));
          j+=sizeof(int);
          memcpy(&iY0,&ctabElements[j],sizeof(int));
          j+=sizeof(int);
          icol=iX+iX0;
          ilig=iY+iY0;
          if(ilig >= 0 && ilig < config.m_iDefY && icol >= 0 && icol < config.m_iDefX)
            {
            iCode=itabImage[ilig*config.m_iDefX + icol];
            if(iCode != iCodeARemplir)break;
            }
          }
        }
      itabNewImage[iIndice]=iCode;
      iX++;
      iIndice++;
      }
    iY++;
    }

  iNbPixels=config.m_iDefX*config.m_iDefY;
  memcpy(itabImage,itabNewImage,iNbPixels*sizeof(int));

  free(ctabElements);
  free(itabNewImage);
  }
```

A.2.2.5. SegmentIntersection()

La procédure `SegmentIntersection()` teste l'intersection de deux segments et calcule les coordonnées de l'intersection si elle existe. Elle est utilisée dans de nombreuses procédures faisant intervenir les arcs décrivant des zones ou des lignes. Lorsqu'il y a intersection possible des deux segments, le calcul est effectué en changeant de repère pour aligner l'un des segment avec l'axe des abscisses.

```
BOOL SegmentIntersection(double dX1,double dY1,double dX2,double dY2,double dX3,double
dY3,double dX4,double dY4,double& dXRes,double& dYRes)
  {
  double dPx1,dPx2,dPx3,dPx4;
  double dPy1,dPy2,dPy3,dPy4;
  double dF1xmin,dF1xmax,dF2xmin,dF2xmax,dF1ymin,dF1ymax,dF2ymin,dF2ymax;

  //--- si deux extremités sont égales, retourner FALSE
  //if(dX1 == dX3 && dY1 == dY3)return FALSE;
  //if(dX1 == dX4 && dY1 == dY4)return FALSE;
  //if(dX2 == dX3 && dY2 == dY3)return FALSE;
  //if(dX2 == dX4 && dY2 == dY4)return FALSE;
  if(dX1 == dX2 && dY1 == dY2)return FALSE;
  if(dX3 == dX4 && dY3 == dY4)return FALSE;

  //--- ordonner le premier segment en y et calculer sa fenêtre
  dXRes=0.;
  dYRes=0.;
  if(dY1 < dY2)
    {
    dPx1=dX1;
    dPy1=dY1;
    dPx2=dX2;
    dPy2=dY2;
    dF1ymin=dY1;
    dF1ymax=dY2;
    if(dX1 <= dX2)
      {
      dF1xmin=dX1;
      dF1xmax=dX2;
      }
    else
      {
      dF1xmin=dX2;
      dF1xmax=dX1;
      }
   }
```

```
else if(dY1 > dY2)
   {
  dPx1=dX2;
  dPy1=dY2;
  dPx2=dX1;
  dPy2=dY1;
  dF1ymin=dY2;
  dF1ymax=dY1;
  if(dX1 <= dX2)
     {
    dF1xmin=dX1;
    dF1xmax=dX2;
     }
  else
     {
    dF1xmin=dX2;
    dF1xmax=dX1;
     }
   }
else if(dY1 == dY2)
   {
  if(dX1 <= dX2)
     {
    dPx1=dX1;
    dPy1=dY1;
    dPx2=dX2;
    dPy2=dY2;
    dF1xmin=dX1;
    dF1xmax=dX2;
     }
  else
     {
    dPx1=dX2;
    dPy1=dY2;
    dPx2=dX1;
    dPy2=dY1;
    dF1xmin=dX2;
    dF1xmax=dX1;
     }
  dF1ymin=dY1;
  dF1ymax=dY2;
   }
//--- ordonner le second segment
if(dY3 < dY4)
   {
  dPx3=dX3;
  dPy3=dY3;
  dPx4=dX4;
  dPy4=dY4;
  dF2ymin=dY3;
  dF2ymax=dY4;
  if(dX3 <= dX4)
     {
    dF2xmin=dX3;
    dF2xmax=dX4;
     }
  else
     {
    dF2xmin=dX4;
    dF2xmax=dX3;
     }
   }
else if(dY3 > dY4)
   {
  dPx3=dX4;
  dPy3=dY4;
  dPx4=dX3;
  dPy4=dY3;
  dF2ymin=dY4;
  dF2ymax=dY3;
  if(dX3 <= dX4)
     {
    dF2xmin=dX3;
    dF2xmax=dX4;
     }
  else
     {
    dF2xmin=dX4;
    dF2xmax=dX3;
     }
```

```
    }
  else if(dY3 == dY4)
    {
    if(dX3 <= dX4)
       {
      dPx3=dX3;
      dPy3=dY3;
      dPx4=dX4;
      dPy4=dY4;
      dF2xmin=dX3;
      dF2xmax=dX4;
       }
    else
       {
      dPx3=dX4;
      dPy3=dY4;
      dPx4=dX3;
      dPy4=dY3;
      dF2xmin=dX4;
      dF2xmax=dX3;
       }
    dF2ymin=dY3;
    dF2ymax=dY4;
    }

  if(dF1xmax > dF2xmin && dF2xmax > dF1xmin && dF1ymax > dF2ymin && dF2ymax > dF1ymin)
    {
    //--- test d'intersection
    double dP1P2=sqrt((dPx2-dPx1)*(dPx2-dPx1) + (dPy2-dPy1)*(dPy2-dPy1));
    if(dP1P2 == 0.)return FALSE;
    double cosTeta=(dPx2-dPx1)/dP1P2;
    double sinTeta=(dPy2-dPy1)/dP1P2;

    dX2=(dPx2-dPx1)*cosTeta+(dPy2-dPy1)*sinTeta;

    dX3=(dPx3-dPx1)*cosTeta+(dPy3-dPy1)*sinTeta;
    dY3=-(dPx3-dPx1)*sinTeta+(dPy3-dPy1)*cosTeta;
    dX4=(dPx4-dPx1)*cosTeta+(dPy4-dPy1)*sinTeta;
    dY4=-(dPx4-dPx1)*sinTeta+(dPy4-dPy1)*cosTeta;

    double dYmin,dYmax;
    if(dY3 < dY4)
       {
       dYmin=dY3;
       dYmax=dY4;
       }
    else
       {
       dYmin=dY4;
       dYmax=dY3;
       }
    if(dY3 == dY4)
       {
       if(dY3 == 0)
          {
          Ordonner(dX3,dX4);
          if(dX3 >= dX2)return FALSE;
          if(dX4 <= 0.)return FALSE;
          dXRes=dX3*cosTeta + dPx1;
          dYRes=dX3*sinTeta + dPy1;
          return TRUE;
          }
       else return FALSE;
       }
    if(dYmin >= 0.)return FALSE;
    if(dYmax <= 0.)return FALSE;
    if(dX3 == dX4)
       {
       //--- parallèle à l'axe des Y
       if(dX3 <= 0.)return FALSE;
       if(dX3 >= dX2)return FALSE;
       //--- rotation inverse pour le resultat
       dXRes=dX3*cosTeta + dPx1;
       dYRes=dX3*sinTeta + dPy1;
       return TRUE;
       }
    else
       {
       //--- point d'intersection avec l'axe des X
       double dX=dX3-(dY3*(dX4-dX3)/(dY4-dY3));
```

```
      if(dX <= 0.)return FALSE;
      if(dX >= dX2)return FALSE;
      //--- rotation inverse pour le resultat
      dXRes=dX*cosTeta + dPx1;
      dYRes=dX*sinTeta + dPy1;
      return TRUE;
      }
    }
  return FALSE;
  }
```

A.2.2.6. Les classes CZone et CTache

Les classes `CZone` et `CTache` permettent, pour les zones, de passer d'une représentation par arc frontière à une représentation par arcs chaînés. Une tache représente le contour d'un polygone. Une zone est un ensemble de polygones. Les taches d'une même zone peuvent être imbriquées et la principale difficulté du chaînage consiste à déterminer cette imbrication uniquement à partir des arcs frontières.

```
class CTache
  {
  private:
    int m_iNbpt;
    int m_iIndex;
    int m_iPere;
    double m_dSurface;
    double m_dXMin;
    double m_dYMin;
    double m_dXMax;
    double m_dYMax;
    CZone* m_pZone;

  public:
    CTache(CZone* pZone,int iIndex,int iNbpt);
    BOOL BoundingBox(double& dXMin,double& dYMin,double& dXMax,double& dYMax);
    BOOL IsPointInside(double dX,double dY);
    void CalculCentroide(double &dX,double &dY);
    BOOL GetPointInterieur(double &dX,double &dY);
    void SetPere(int iPere) {m_iPere=iPere;};
    int GetNbpt() {return m_iNbpt;};
    int GetIndex() {return m_iIndex;};
    int GetPere() {return m_iPere;};
    double GetSurface() {return m_dSurface;};
    void GetBB(double& dXMin,double& dYMin,double& dXMax,double& dYMax)
      {
      dXMin=m_dXMin;
      dYMin=m_dYMin;
      dXMax=m_dXMax;
      dYMax=m_dYMax;
      return;
      };
    void ChangerLeSens();
    double Surface();
  };

class CZone
  {
  private:
    FILE* m_FileArc;
    CCadre* m_pCadre;
    int m_iNbArc;
    int* m_itabPtrArc;

    int m_iNz;
    int m_iNbTaches;
    int m_iNbptTotal;

  public:
    CTache* m_ptabTache[NB_MAX_TACHES];
    double *m_dtabPoints;
  private:

  public:
```

```
    CZone();
    ~CZone();

    BOOL Init(CCadre* pCadre,FILE* FileArc,int iNbarcDeLaZone,int* itabPtr);
    BOOL BoundingBox(double& dXMin,double& dYmin,double& dXMax,double& dYmax);
    BOOL Chainer();
    int NbptTotal();
    int GetNbTaches() {return m_iNbTaches;};
    void Desalloc();
    };
```

Voici l'implémentation de la procédure Chainer() qui crée le chaînage des arcs appartenant à une zone, à partir de l'ensemble des arcs de la relation. Cette procédure est notamment utilisée pour exporter en format ShapeFile (ArcView) des données graphiques SAVANE.

```
BOOL CZone::Chainer()
  {
  m_iNbTaches=0;
  m_iNbptTotal=0;
  //--- chainage par tache
  double* dtabX1=(double *) malloc(sizeof(double)*m_iNbArcZone);
  double* dtabY1=(double *) malloc(sizeof(double)*m_iNbArcZone);
  double* dtabX2=(double *) malloc(sizeof(double)*m_iNbArcZone);
  double* dtabY2=(double *) malloc(sizeof(double)*m_iNbArcZone);

  BOOL *btabLibre=(BOOL *) malloc(sizeof(BOOL)*m_iNbArcZone);
  BOOL *btabSens=(BOOL *) malloc(sizeof(int)*m_iNbArcZone);
  int *itabArcDeLaTache=(int *) malloc(sizeof(int)*m_iNbArcZone);

  //--- vérification des allocations mémoire
  if(dtabX1 == NULL || dtabY1 == NULL || dtabX2 == NULL || dtabY2 == NULL || btabLibre == NULL
|| btabSens == NULL || itabArcDeLaTache == NULL)
    {
    if(dtabX1 != NULL)free(dtabX1);
    if(dtabY1 != NULL)free(dtabY1);
    if(dtabX2 != NULL)free(dtabX2);
    if(dtabY2 != NULL)free(dtabY2);
    if(btabLibre != NULL)free(btabLibre);
    if(btabSens != NULL)free(btabSens);
    if(itabArcDeLaTache != NULL)free(itabArcDeLaTache);
    return FALSE;
    }

  //--- initialisations
  int i,iArc,iNbpt;
  CArcMygale* pArc;
  double dX1,dY1,dX2,dY2;

  for(iArc=0;iArc < m_iNbArcZone;iArc++)
    {
    pArc=m_ptabArc[iArc];
    if(pArc != NULL)
      {
      iNbpt=pArc->GetNbpt();
      pArc->GetPoint(0,dX1,dY1);
      pArc->GetPoint(iNbpt-1,dX2,dY2);
      dtabX1[iArc]=dX1;
      dtabY1[iArc]=dY1;
      dtabX2[iArc]=dX2;
      dtabY2[iArc]=dY2;
      btabLibre[iArc]=TRUE;
      }
    }

  //--- chainage par tâche
  double dXDebut,dYDebut,dXFin,dYFin;
  int iNbTachesNonFermees=0;
  CTache* pTache=NULL;

  iArc=0;
  while(iArc < m_iNbArcZone && m_iNbTaches < NB_MAX_TACHES)
    {
    //--- chainage par tache
    while(iArc < m_iNbArcZone && btabLibre[iArc] == FALSE)iArc++;
```

```
  if(iArc == m_iNbArcZone)break;

  //--- début du ring
  dXDebut=dtabX1[iArc];
  dYDebut=dtabY1[iArc];
  dXFin=dtabX2[iArc];
  dYFin=dtabY2[iArc];

  int iNbarcTache=0;
  itabArcDeLaTache[iNbarcTache]=iArc;
  btabSens[iNbarcTache]=0;
  iNbarcTache++;
  btabLibre[iArc]=FALSE;
  iArc++;

  BOOL bFerme=FALSE;
  if(dXDebut == dXFin && dYDebut == dYFin)bFerme=TRUE;
  int jArc=iArc;
  while(jArc < m_iNbArcZone && bFerme == FALSE)
    {
    if(btabLibre[jArc] == TRUE)
      {
      if(dtabX1[jArc] == dXFin && dtabY1[jArc] == dYFin)
        {
        itabArcDeLaTache[iNbarcTache]=jArc;
        btabSens[iNbarcTache]=0;
        iNbarcTache++;
        dXFin=dtabX2[jArc];
        dYFin=dtabY2[jArc];
        btabLibre[jArc]=FALSE;
        jArc=iArc;
        if(dXFin == dXDebut && dYFin == dYDebut)bFerme=TRUE;
        }
      else if(dtabX2[jArc] == dXFin && dtabY2[jArc] == dYFin)
        {
        itabArcDeLaTache[iNbarcTache]=jArc;
        btabSens[iNbarcTache]=1;
        iNbarcTache++;
        dXFin=dtabX1[jArc];
        dYFin=dtabY1[jArc];
        btabLibre[jArc]=FALSE;
        jArc=iArc;
        if(dXFin == dXDebut && dYFin == dYDebut)bFerme=TRUE;
        }
      else jArc++;
      }
    else jArc++;
    }

  //--- fin du chainage d'une tâche, écriture du polygone
  if(bFerme)
    {
    //--- chainage dans m_dtabPoints
    int iIndex=m_iNbptTotal;
    int iNbptPolygon=0;
    for(i=0;i < iNbarcTache;i++)
      {
      pArc=m_ptabArc[itabArcDeLaTache[i]];
      pArc->Chainer(btabSens[i],m_iTypeCoordonnees,m_iNbptTotal,iNbptPolygon,m_dtabPoints);
      }
    if(iNbptPolygon > 3)
      {
      //--- trois points ou moins : surface nulle !
      pTache=new CTache(this,iIndex,iNbptPolygon);
      if(pTache != NULL)
        {
        m_ptabTache[m_iNbTaches]=pTache;
        m_iNbTaches++;
        }
      }
    }
  else iNbTachesNonFermees++;
  }

free(dtabX1);
free(dtabY1);
free(dtabX2);
free(dtabY2);
free(btabLibre);
free(btabSens);
```

```
free(itabArcDeLaTache);

if(iNbTachesNonFermees > 0)return FALSE;

//--- calcul du sens pour chaque tâche
if(m_iNbTaches == 0)return FALSE;
else if(m_iNbTaches == 1)
   {
   double dSurface=m_ptabTache[0]->Surface();
   if(dSurface > 0.)m_ptabTache[0]->ChangerLeSens();
   }
else if(m_iNbTaches == 2)
   {
   //--- zone de deux tâches
   double dX0b,dY0b,dX0h,dY0h;
   double dX1b,dY1b,dX1h,dY1h;
   double dXInterieur,dYInterieur;
   m_ptabTache[0]->BoundingBox(dX0b,dY0b,dX0h,dY0h);
   m_ptabTache[1]->BoundingBox(dX1b,dY1b,dX1h,dY1h);
   double dSurface0=m_ptabTache[0]->Surface();
   double dSurface1=m_ptabTache[1]->Surface();
   if(fabs(dSurface0) > fabs(dSurface1))
      {
      if(dSurface0 > 0.)m_ptabTache[0]->ChangerLeSens();
      //--- tester si 1 est inclus dans 0
      if(dSurface1 != 0. && dX1b >= dX0b && dX1h <= dX0h && dY1b >= dY0b && dY1h <= dY0h)
         {
         //--- tester l'appartenance d'un point de la 1 dans la 0
         m_ptabTache[1]->GetPointInterieur(dXInterieur,dYInterieur);

         if(m_ptabTache[0]->IsPointInside(dXInterieur,dYInterieur))
            if(dSurface1 < 0.)m_ptabTache[1]->ChangerLeSens();
         else
            if(dSurface1 > 0.)m_ptabTache[1]->ChangerLeSens();
         }
      else
         {
         //--- on est sûr que la tache 1 n'est pas incluse
         if(dSurface1 > 0.)m_ptabTache[1]->ChangerLeSens();
         }
      }
   else
      {
      if(dSurface1 > 0.)m_ptabTache[1]->ChangerLeSens();
      //--- tester si 0 est inclus dans 1
      if(dSurface0 != 0. && dX0b >= dX1b && dX0h <= dX1h && dY0b >= dY1b && dY0h <= dY1h)
         {
         //--- tester l'appartenance d'un point de la 0 dans la 1
         m_ptabTache[0]->GetPointInterieur(dXInterieur,dYInterieur);

         if(m_ptabTache[1]->IsPointInside(dXInterieur,dYInterieur))
            if(dSurface0 < 0.)m_ptabTache[0]->ChangerLeSens();
         else
            if(dSurface0 > 0.)m_ptabTache[0]->ChangerLeSens();
         }
      else
         {
         //--- on est sûr que la tache 0 n'est pas incluse
         if(dSurface0 > 0.)m_ptabTache[0]->ChangerLeSens();
         }
      }
   }
else
   {
   //--- zones de plusieurs taches
   BOOL bIncluse;
   int j,iIndice,iNoTache,iNoTacheBis,iSize,iNbTachesNonNulles;
   double dSurface,dXInterieur,dYInterieur;
   double dX0b,dY0b,dX0h,dY0h;
   double dX1b,dY1b,dX1h,dY1h;

   iSize=sizeof(double) + sizeof(int);
   unsigned char *ctabSurface=(unsigned char *) malloc(m_iNbTaches*iSize);
   int *itabTri=(int *) malloc(m_iNbTaches*sizeof(int));

   if(ctabSurface == NULL || itabTri == NULL)
      {
      ::ErreurDeMemoire();
      if(ctabSurface != NULL)free(ctabSurface);
      if(itabTri != NULL)free(itabTri);
```

```
      return FALSE;
      }

    //--- trier les taches sur leur surface, établir le tableau des indices des taches triées
    iNbTachesNonNulles=0;
    iIndice=0;
    for(i=0;i < m_iNbTaches;i++)
      {
      m_ptabTache[i]->BoundingBox(dX0b,dY0b,dX0h,dY0h);
      dSurface=fabs(m_ptabTache[i]->Surface());
      if(dSurface > 0.)
        {
        memcpy(&ctabSurface[iIndice],&dSurface,sizeof(double));
        memcpy(&ctabSurface[iIndice + sizeof(double)],&i,sizeof(int));
        iIndice+=iSize;
        iNbTachesNonNulles++;
        }
      }
    qsort(ctabSurface,iNbTachesNonNulles,iSize,compareDoubleAsc);
    iIndice=0;
    for(i=0;i < iNbTachesNonNulles;i++)
      {
      memcpy(&iNoTache,&ctabSurface[iIndice + sizeof(double)],sizeof(int));
      itabTri[i]=iNoTache;
      iIndice+=iSize;
      }

    //--- en partant de la plus petite tache, on cherche le premier incluant, pour chaque
tache
    for(i=0;i < iNbTachesNonNulles;i++)
      {
      iNoTache=itabTri[i];
      m_ptabTache[iNoTache]->SetPere(-2);
      m_ptabTache[iNoTache]->GetBB(dX0b,dY0b,dX0h,dY0h);
      m_ptabTache[iNoTache]->GetPointInterieur(dXInterieur,dYInterieur);
      for(j=i+1;j < iNbTachesNonNulles;j++)
        {
        iNoTacheBis=itabTri[j];
        m_ptabTache[iNoTacheBis]->GetBB(dX1b,dY1b,dX1h,dY1h);
        if(dX0b >= dX1b && dX0h <= dX1h && dY0b >= dY1b && dY0h <= dY1h)
          {
          //--- vérifier l'inclusion par un point
          bIncluse=m_ptabTache[iNoTacheBis]->IsPointInside(dXInterieur,dYInterieur);
          if(bIncluse)
            {
            m_ptabTache[iNoTache]->SetPere(iNoTacheBis);
            break;
            }
          }
        }
      }

    //--- en partant du plus grand, on ecrit la tache (clockwise) et tous les fils
(counterclock) jusqu'a ecrire toutes les taches
    for(i=iNbTachesNonNulles-1;i >= 0;i--)
      {
      iNoTache=itabTri[i];
      if(m_ptabTache[iNoTache]->GetPere() != -2)
        {
        //--- ecriture de la tache iNoTache clockwise
        if(m_ptabTache[iNoTache]->GetSurface() > 0.)m_ptabTache[iNoTache]->ChangerLeSens();
        m_ptabTache[iNoTache]->SetPere(-2);

        for(j=i-1;j >= 0;j--)
          {
          iNoTacheBis=itabTri[j];
          if(m_ptabTache[iNoTacheBis]->GetPere() == iNoTache)
            {
            //--- ecriture de la tache iNoTacheBis counterclockwise
            if(m_ptabTache[iNoTacheBis]->GetSurface() < 0.)m_ptabTache[iNoTacheBis]-
>ChangerLeSens();
            m_ptabTache[iNoTacheBis]->SetPere(-2);
            }
          }

        }
      }

    free(ctabSurface);
    free(itabTri);
```

```
    }

  return TRUE;
  }
```

A.2.3. Fenêtre et projections

A.2.3.1. La classe CWind

La classe `CWind` encapsule données et méthodes permettant de manipuler la fenêtre de sélection géographique dans un cadre. Chaque cadre contient un objet de type `CWind`.

```
class CWind
  {
  public:

    CCadre* m_pCadre;
    double m_dFi1;
    double m_dFi2;
    double m_dFi3;
    double m_dFi4;
    float m_fFx1;
    float m_fFy1;
    float m_fFx2;
    float m_fFy2;
    double m_dFlx;
    double m_dFly;
    double m_dFhx;
    double m_dFhy;
    double m_dCoefRas;
    double m_dFpixRas; // taille en metre du pixel rasterisation savane

    int m_iNbWind; // pour l'historique des fenetres
    float m_fFx1Undo[NB_MAX_WIND];
    float m_fFy1Undo[NB_MAX_WIND];
    float m_fFx2Undo[NB_MAX_WIND];
    float m_fFy2Undo[NB_MAX_WIND];

  public:

    CWind(CCadre* pCadre);
    CWind(CCadre* pCadre,FILE* Fichier);
    void Ecrire(FILE* Fichier);
    void Copier(CWind* pWindInit);
    void ProjToRas(double dProjX,double dProjY,float &fRasX,float &fRasY);
    void RasToProj(float fRasX,float fRasY,double &dProjX,double &dProjY);
    void NouvelleFenetre(float fFx1,float fFy1,float fFx2,float fFy2);
    void NouvelleFenetre(double dFlx,double dFly,double dFpixRas);
    void NouvelleFenetre(double dFlx,double dFly,double dFhx,double dFhy);
    void NouvelleFenetre(double dFlx,double dFly,double dFhx,double dFhy,double dFpixRas);
    void Undo();
    void NouvelleProjection();
  };
```

Par exemple, voici le code de la procédure permettant de modifier la fenêtre par coordonnées de projection :

```
void CWind::NouvelleFenetre(double dFlx,double dFly,double dFhx,double dFhy)
  {
  //--- mise a jour du undo
  if(m_iNbWind < NB_MAX_WIND)
    {
    m_fFx1Undo[m_iNbWind]=m_fFx1;
    m_fFy1Undo[m_iNbWind]=m_fFy1;
    m_fFx2Undo[m_iNbWind]=m_fFx2;
    m_fFy2Undo[m_iNbWind]=m_fFy2;
    m_iNbWind++;
    }
  else
    {
```

```
   for(int i=1;i < m_iNbWind;i++)
      {
      m_fFx1Undo[i-1]=m_fFx1Undo[i];
      m_fFy1Undo[i-1]=m_fFy1Undo[i];
      m_fFx2Undo[i-1]=m_fFx2Undo[i];
      m_fFy2Undo[i-1]=m_fFy2Undo[i];
      }
   m_fFx1Undo[m_iNbWind-1]=m_fFx1;
   m_fFy1Undo[m_iNbWind-1]=m_fFy1;
   m_fFx2Undo[m_iNbWind-1]=m_fFx2;
   m_fFy2Undo[m_iNbWind-1]=m_fFy2;
   }

//--- nouvelle fenètre par coordonnées de projection
m_dFlx=dFlx;
m_dFly=dFly;
m_dFhx=dFhx;
m_dFhy=dFhy;

//--- calcul des coefficients pour le raster Savane
if((m_dFhx-m_dFlx) > 0. && (m_dFhy-m_dFly) > 0.)
   {
   double d1=(double)(config.m_iDefX)/(m_dFhx-m_dFlx);
   double d2=(double)(config.m_iDefY)/(m_dFhy-m_dFly);
   if(d1 < d2)m_dCoefRas=d1; else m_dCoefRas=d2;
   m_dFpixRas=1./m_dCoefRas;
   }

//--- calcul de Fx1,Fy1,Fx2,Fy2
m_pCadre->ProjToSav(dFlx,dFly,m_fFx1,m_fFy1);
m_pCadre->ProjToSav(dFhx,dFhy,m_fFx2,m_fFy2);

//--- réinitialisation des rasterisations pour le cadre
CSchema* pSchema=m_pCadre->GetSchema();
int iNoth=1;
while(iNoth <= pSchema->m_iNbacc)
   {
   pSchema->m_pRelations[iNoth]->m_iNbObjets=-1;
   int iTypeRel=pSchema->m_pRelations[iNoth]->m_iType;
   if(iTypeRel == 24)
      {
      pSchema->m_pRelations[iNoth]->m_iType=14;
      int iFim=pSchema->m_pRelations[iNoth]->m_iNoFichImage;
      pSchema->m_iFdisImage[iFim]=0;
      }
   iNoth++;
   }

//--- suppression de la relation Raster
int iNothRaster=pSchema->RelRecherche("Raster");
if(iNothRaster > 0)XRelProj(m_pCadre,iNothRaster);

//--- dimensionner le cadre à la nouvelle fenetre
m_pCadre->ReAjuster();
groupes.TailleDesCadres();
}
```

B.2.3.2. La classe CProjection

La classe `CProjection` se charge de la gestion et du calcul des projection géographique dans un cadre. Chaque cadre contient un objet de type `CProjection` :

```
class CProjection
  {
  private:

  //--- projection courante
  int m_iProjection;
  CString m_strNomProjection;

  //--- description de l'ellipsoïde
  double m_dEllipseA;
  double m_dEllipseE;
  double m_dEllipseE2;
```

```
double m_dEllipseEprim;
double m_dEllipseEprim2;
double m_dEllipseEsur2;

//--- description de la projection
double m_dXOrigine;
double m_dYOrigine;
int m_iZoneUtm;
double m_dFacteur;

int m_iMeridienCentralD;
int m_iMeridienCentralM;
double m_dMeridienCentralS;
int m_iMeridienCentralHem;

int m_iParalleleOrigineD;
int m_iParalleleOrigineM;
double m_dParalleleOrigineS;
int m_iParalleleOrigineHem;

int m_iParallele1D;
int m_iParallele1M;
double m_dParallele1S;
int m_iParallele1Hem;

int m_iParallele2D;
int m_iParallele2M;
double m_dParallele2S;
int m_iParallele2Hem;

//--- membres pour les calculs
double m_dPi;
double m_dPisur2;
double m_dPisur4;
double m_dMinuteToRadian;
double m_dRadianToMinute;
double m_dMeridienCentral;
double m_dParalleleOrigine;
double m_dParallele1;
double m_dParallele2;
double m_dGelMer[5];
double m_dXs;
double m_dYs;
double m_dN;
double m_dC;
double m_dRh;
double m_dSin0;
double m_dCos0;

//--- procedures d'initialisation
void LambertTangenteInit();
void LambertSecanteInit();
void StereographicInit();
void OrthographicInit();
void AlbertEquivInit();

//--- procedures de calcul
void GeographicToGeo(double dX,double dY,double &dLongitude,double &dColatitude);
void GeoToGeographic(double dLongitude,double dColatitude,double &dX,double &dY);
void UtmToGeo(double dX,double dY,double &dLongitude,double &dColatitude);
void GeoToUtm(double dLongitude,double dColatitude,double &dX,double &dY);
void LambertToGeo(double dX,double dY,double &dLongitude,double &dColatitude);
void GeoToLambert(double dLongitude,double dColatitude,double &dX,double &dY);
void MercatorToGeo(double dX,double dY,double &dLongitude,double &dColatitude);
void GeoToMercator(double dLongitude,double dColatitude,double &dX,double &dY);
void StereographicToGeo(double dX,double dY,double &dLongitude,double &dColatitude);
void GeoToStereographic(double dLongitude,double dColatitude,double &dX,double &dY);
void OrthographicToGeo(double dX,double dY,double &dLongitude,double &dColatitude);
void GeoToOrthographic(double dLongitude,double dColatitude,double &dX,double &dY);
void AlbertEquivToGeo(double dX,double dY,double &dLongitude,double &dColatitude);
void GeoToAlbertEquiv(double dLongitude,double dColatitude,double &dX,double &dY);
void GeodegreToGeo(double dX,double dY,double &dLongitude,double &dColatitude);
void GeoToGeodegre(double dLongitude,double dColatitude,double &dX,double &dY);

//--- procedures de calcul
double Msfnz(double sinphi,double cosphi);
double Qsfnz(double sinphi,double cosphi);
double Phi1z(double qs);

public:
```

```
  CProjection();
  CProjection(FILE* Fichier);

  void Ecrire(FILE* Fichier);
  BOOL SetProjection(int iProjection);
  BOOL SetProjection(double dProj[20]);
  int GetProjection();
  void GetProjection(double dProj[20]);
  CString GetNomProjection();
  CString GetNomProjection(int iProjection);

  BOOL SetEllipse(double dA,double dE2);
  void GetEllipse(double &dA,double &dE2);
  BOOL SetFacteur(double dFacteur);
  double GetFacteur();
  BOOL SetOrigine(double dX,double dY);
  void GetOrigine(double &dX,double &dY);
  void SetZoneUtm(int iZone);
  int GetZoneUtm();
  void SetMeridienCentral(class CLongitude);
  void SetMeridienCentral(int iDegre,int iMinute,double dSeconde,int iHem);
  BOOL SetMeridienCentral(double dMeridienCentral);
  class CLongitude GetMeridienCentral();
  void SetParalleleOrigine(class CLatitude);
  void SetParalleleOrigine(int iDegre,int iMinute,double dSeconde,int iHem);
  BOOL SetParalleleOrigine(double dColatitude);
  class CLatitude GetParalleleOrigine();
  void SetParallele1(class CLatitude);
  void SetParallele1(int iDegre,int iMinute,double dSeconde,int iHem);
  BOOL SetParallele1(double dColatitude);
  class CLatitude GetParallele1();
  void SetParallele2(class CLatitude);
  void SetParallele2(int iDegre,int iMinute,double dSeconde,int iHem);
  BOOL SetParallele2(double dColatitude);
  class CLatitude GetParallele2();

  void Init();
  void GeoToProj(double dLongitude,double dColatitude,double &dX,double &dY);
  void ProjToGeo(double dX,double dY,double &dLongitude,double &dColatitude);
  };
```

Les coordonnées géographiques (dLongitude, dColatitude) sont toujours exprimées en minutes décimales dans la classe CProjection. Voici par exemple le code permettant de calculer les transformations de coordonnées pour la projection Mercator :

```
void CProjection::MercatorToGeo(double dX,double dY,double &dLongitude,double &dColatitude)
  {
  int i,iHem;
  double v,phibis,phi,u,puissance,dLatitude;

  dLongitude=m_dMeridienCentral+((dX/m_dEllipseA)*m_dRadianToMinute);
  if(dLongitude < 0.)dLongitude+=21600.;
  else if(dLongitude >= 21600.)dLongitude-=21600.;

  if(dY <= 0.)
    {
    iHem=SUD;
    dY=-dY;
    }
  else iHem=NORD;

  v=exp(dY/m_dEllipseA);
  phibis=2.*atan(v)-m_dPisur2;
  phi=phibis;
  u=m_dEllipseE*sin(phi);
  puissance=pow(((1.+u)/(1.-u)),m_dEllipseEsur2);
  phibis=2.*(atan(v*puissance))-m_dPisur2;
  i=1;
  while(phibis != phi && i < 50)
    {
    phi=phibis;
    u=m_dEllipseE*sin(phi);
    puissance=pow(((1.+u)/(1.-u)),m_dEllipseEsur2);
```

```
    phibis=2.*(atan(v*puissance))-m_dPisur2;
    i++;
    }

  dLatitude=phi*m_dRadianToMinute;
  if(iHem==NORD)dColatitude=5400.-dLatitude;
  else dColatitude=5400.+dLatitude;
  }

void CProjection::GeoToMercator(double dLongitude,double dColatitude,double &dX,double &dY)
  {
  int iLongitudeHem;
  double rl,dLatitude,phi,u;

  if(dLongitude < 0.)dLongitude+=21600.;
  else if(dLongitude >= 21600.)dLongitude-=21600.;

  if(dLongitude < 10800.)iLongitudeHem=EST;
  else iLongitudeHem=OUEST;

  if(m_iMeridienCentralHem == iLongitudeHem)rl=dLongitude-m_dMeridienCentral;
  else if(m_iMeridienCentralHem == EST && iLongitudeHem == OUEST)
    {
    rl=dLongitude-m_dMeridienCentral;
    if(rl > 10800.)rl-= 21600.;
    }
  else if(m_iMeridienCentralHem == OUEST && iLongitudeHem == EST)
    {
    rl=dLongitude+21600.-m_dMeridienCentral;
    if(rl > 10800.)rl-= 21600.;
    }

  if(dColatitude > 5400.)dLatitude=dColatitude-5400.;
  else dLatitude=5400.-dColatitude;

  dX=(rl*m_dMinuteToRadian)*m_dEllipseA;

  phi=dLatitude*m_dMinuteToRadian;
  u=m_dEllipseE*sin(phi);
  dY=m_dEllipseA*(log(tan(m_dPisur4+(phi/2.)))-(m_dEllipseEsur2*log((1.+u)/(1.-u))));
  if(dColatitude > 5400.)dY=-dY;
  }
```

La projection UTM nécessite de nombreux paramètres, dont ceux permettant de calculer la longueur d'un arc de méridien sur l'ellipsoïde. Les paramètres sont calculés lors du chargement ou de la modification de l'ellipsoïde :

```
BOOL CProjection::SetEllipse(double dA,double dE2)
  {
  //--- indique en retour si l'ellipsoïde a changé
  if(m_dEllipseA == dA && m_dEllipseE2 == dE2)return FALSE;
  m_dEllipseA=dA;
  m_dEllipseE2=dE2;

  //--- calcul de e, e'2 et e/2
  m_dEllipseE=sqrt(dE2);
  m_dEllipseEsur2=m_dEllipseE/2.;
  m_dEllipseEprim2=dE2/(1.-dE2);
  m_dEllipseEprim=sqrt(m_dEllipseEprim2);
  //--- calcul du nouveau gelmer pour UTM
  m_dGelMer[0]=1.+ dE2*(3./4.+dE2*(45./64.+dE2*(175./256.+dE2*11025./16384.)));
  m_dGelMer[1]=dE2*(3./4.+dE2*(15./16.+dE2*(525./512.+dE2*2205./2048.)));
  m_dGelMer[2]=dE2*dE2*(15./64.+dE2*(105./256.+dE2*2205./4096.));
  m_dGelMer[3]=dE2*dE2*dE2*(35./512.+dE2*315./2048.);
  m_dGelMer[4]=dE2*dE2*dE2*dE2*315./16384.;

  return TRUE;
  }
```

Voici les calculs de transformations de coordonnées pour la projection UTM :

```
void CProjection::UtmToGeo(double dX,double dY,double &dLongitude,double &dColatitude)
  {
  int iLatitudeHem;
  double y1,x1,rp,b,db,gelmer,x11,x12,rl,rp1,u,dLatitude;
```

```
   y1=dY-10000000.;
   x1=dX-500000.;
   if(y1 < 0.)
      {
      iLatitudeHem=SUD;
      y1=-y1;
      }
   else iLatitudeHem=NORD;

   x1=x1/m_dFacteur;
   y1=y1/m_dFacteur;

   rp=m_dPi*y1/2.e+07;

   b=m_dGelMer[0]*rp;
   db=m_dGelMer[1]*sin(2.*rp)/2.;
   b=b-db;
   db=m_dGelMer[2]*sin(4.*rp)/4.;
   b=b+db;
   db=m_dGelMer[3]*sin(6.*rp)/6.;
   b=b-db;
   db=m_dGelMer[4]*sin(8.*rp)/8.;
   b=b+db;
   gelmer=b*m_dEllipseA*(1.- m_dEllipseE2);

   y1=y1-gelmer;
   u=sin(rp);
   u=u*u;
   x11=m_dEllipseA/sqrt(1.-m_dEllipseE2*u);
   x12=m_dEllipseE2*(1.-u)/6./(1.- m_dEllipseE2);
   y1=y1*(1.-3.*x12*x1*x1/x11/x11)/x11;
   x1=x1*(1.-x12*x1*x1/x11/x11)/x11;
   y1=y1+rp;
   x1=2.*atan(exp(x1))-m_dPisur2;
   rl=atan(tan(x1)/cos(y1));

   dLongitude=m_dMeridienCentral+rl*m_dRadianToMinute;
   if(dLongitude < 0.)dLongitude+=21600.;
   else if(dLongitude >= 21600.)dLongitude-=21600.;

   rp1=asin(sin(y1)*cos(x1));
   rp=rp+(1.-m_dEllipseE2*sin(rp)*sin(rp))/(1.-m_dEllipseE2)*(rp1-rp)*(1.-
1.5*m_dEllipseEprim2*sin(rp)*cos(rp)*(rp1-rp));

   dLatitude=rp*m_dRadianToMinute;
   if(iLatitudeHem == NORD) dColatitude=5400.-dLatitude;
   else dColatitude=5400.+dLatitude;
   }

void CProjection::GeoToUtm(double dLongitude,double dColatitude,double &dX,double &dY)
   {
   int iLongitudeHem,iLatitudeHem;
   double rl,rp,dLatitude;
   double cpsi,spsi,tpsi,tpsi2,tpsi4,nu,nu2,N,u,u2,u3,x;
   double y,b,db,gelmer;

   if(dLongitude < 0.)dLongitude+=21600.;
   else if(dLongitude >= 21600.)dLongitude-=21600.;

   if(dLongitude < 10800.)iLongitudeHem=EST;
   else iLongitudeHem=OUEST;

   if(m_iMeridienCentralHem == iLongitudeHem)rl=dLongitude-m_dMeridienCentral;
   else if(m_iMeridienCentralHem == EST && iLongitudeHem == OUEST)
      {
      rl=dLongitude-m_dMeridienCentral;
      if(rl > 10800.)rl-=21600.;
      }
   else if(m_iMeridienCentralHem == OUEST && iLongitudeHem == EST)
      {
      rl=dLongitude+21600.-m_dMeridienCentral;
      if(rl > 10800.)rl-=21600.;
      }

   if(dColatitude > 5400.)
      {
      iLatitudeHem=SUD;
      dLatitude=dColatitude-5400.;
      }
```

```
  else
     {
     iLatitudeHem=NORD;
     dLatitude=5400.-dColatitude;
     }

  rl=rl*m_dMinuteToRadian;
  rp=dLatitude*m_dMinuteToRadian;

  cpsi=cos(rp);
  spsi=sin(rp);
  tpsi=spsi/cpsi;
  tpsi2=tpsi*tpsi;
  tpsi4=tpsi2*tpsi2;
  nu=cpsi*m_dEllipseEprim;
  nu2=nu*nu;
  N=m_dEllipseA/sqrt(1.- m_dEllipseE2*spsi*spsi);
  u=rl*cpsi;
  u2=u*u;
  u3=u2*u;
  //--- calcul de x
  x=N*u+(N*u3*(1.-tpsi2+nu2)/6.);
  x=x+(N*u2*u3*(5.-(18.*tpsi2)+tpsi4+(14.*nu2)-(58.*tpsi2*nu2))/120.);
  dX=500000.+ m_dFacteur*x;

  //--- calcul de la longueur de l'arc de meridien central
  b=m_dGelMer[0]*rp;
  db=m_dGelMer[1]*sin(2.*rp)/2.;
  b=b-db;
  db=m_dGelMer[2]*sin(4.*rp)/4.;
  b=b+db;
  db=m_dGelMer[3]*sin(6.*rp)/6.;
  b=b-db;
  db=m_dGelMer[4]*sin(8.*rp)/8.;
  b=b+db;
  gelmer=b*m_dEllipseA*(1.- m_dEllipseE2);
  //--- termes du developpement pour la projection
  y=gelmer+(N*u2*tpsi/2.);
  y=y+(N*u2*u2*tpsi*(5.-tpsi2+(9.*nu2)+(4.*nu2*nu2))/24.);
  y=y+(N*u3*u3*tpsi*(61.-(58.*tpsi2)+tpsi4+(270.*nu2)-(330.*nu2*tpsi2))/720.);
  y=m_dFacteur*y;

  if(iLatitudeHem == SUD)dY=10000000.-y;
  else dY=y+10000000.;
  }
```

La projection Lambert est une conique. Les paramètres sont initialisés lors de la définition de la projection dans le cadre :

```
void CProjection::LambertTangenteInit()
  {
  double dPar,phi0,W0,N0,u,L0,R0;

  if(m_dParalleleOrigine>5400.)dPar=m_dParalleleOrigine-5400.;
  else dPar=5400.-m_dParalleleOrigine;

  phi0=dPar*m_dMinuteToRadian;

  m_dN=sin(phi0);
  W0=sqrt((1.-(m_dEllipseE2*m_dN*m_dN)));
  N0=m_dEllipseA/W0;
  u=m_dEllipseE*m_dN;
  L0=log(tan(m_dPisur4+(phi0/2.)))-(m_dEllipseEsur2*log((1.+u)/(1.-u)));
  R0=m_dFacteur*(N0/tan(phi0));
  m_dC=R0*exp(m_dN*L0);
  m_dXs=m_dXOrigine;
  m_dYs=m_dYOrigine+R0;
  }

void CProjection::LambertSecanteInit()
  {
  double dPar,phi0,phi1,phi2,u,w1,n1,l1,w2,n2,l2,l0,r0;

  if(m_dParalleleOrigine > 5400.)dPar=m_dParalleleOrigine-5400.;
  else dPar=5400.-m_dParalleleOrigine;
  phi0=dPar*m_dMinuteToRadian;

  if(m_dParallele1 > 5400.)dPar=m_dParallele1-5400.;
```

```
  else dPar=5400.-m_dParallele1;
  phi1=dPar*m_dMinuteToRadian;

  if(m_dParallele2 > 5400.)dPar=m_dParallele2-5400.;
  else dPar=5400.-m_dParallele2;
  phi2=dPar*m_dMinuteToRadian;

  u=sin(phi1);
  w1=sqrt(1.-(m_dEllipseE2*u*u));
  n1=m_dEllipseA/w1;
  u=m_dEllipseE*u;
  l1=log(tan(m_dPisur4+(phi1/2.)))-(m_dEllipseEsur2*log((1.+u)/(1.-u)));
  u=sin(phi2);
  w2=sqrt(1.-(m_dEllipseE2*u*u));
  n2=m_dEllipseA/w2;
  u=m_dEllipseE*u;
  l2=log(tan(m_dPisur4+(phi2/2.)))-(m_dEllipseEsur2*log((1.+u)/(1.-u)));
  m_dN=(log((n2*cos(phi2))/(n1*cos(phi1))))/(l1-l2);
  m_dC=((n1*cos(phi1))/m_dN)*exp(m_dN*l1);
  u=m_dEllipseE*sin(phi0);
  l0=log(tan(m_dPisur4+(phi0/2.)))-(m_dEllipseEsur2*log((1.+u)/(1.-u)));
  r0=m_dC*exp(-m_dN*l0);
  m_dXs=m_dXOrigine;
  m_dYs=m_dYOrigine+r0;
  }

void CProjection::GeoToLambert(double dLongitude,double dColatitude,double &dX,double &dY)
  {
  int iLongitudeHem,iLatitudeHem;
  double rl,dLatitude,phi,u,L,gama,r;

  if(dLongitude < 0.)dLongitude+=21600.;
  else if(dLongitude >= 21600.)dLongitude-=21600.;

  if(dLongitude < 10800.)iLongitudeHem=EST;
  else iLongitudeHem=OUEST;

  if(m_iMeridienCentralHem == iLongitudeHem)rl=dLongitude-m_dMeridienCentral;
  else if(m_iMeridienCentralHem == EST && iLongitudeHem == OUEST)
    {
    rl=dLongitude-m_dMeridienCentral;
    if(rl > 10800.)rl-=21600.;
    }
  else if(m_iMeridienCentralHem == OUEST && iLongitudeHem == EST)
    {
    rl=dLongitude+21600.-m_dMeridienCentral;
    if(rl > 10800.)rl-=21600.;
    }

  if(dColatitude > 5400.)
    {
    iLatitudeHem=SUD;
    dLatitude=dColatitude-5400.;
    }
  else
    {
    iLatitudeHem=NORD;
    dLatitude=5400.-dColatitude;
    }

  if(iLatitudeHem != m_iParalleleOrigineHem)dLatitude=-dLatitude;

  phi=dLatitude*m_dMinuteToRadian;
  rl*=m_dMinuteToRadian;
  u=m_dEllipseE*sin(phi);
  L=log(tan(m_dPisur4+(phi/2.)))-(m_dEllipseEsur2*log((1.+u)/(1.-u)));
  gama=m_dN*rl;
  r=m_dC*exp(-m_dN*L);

  //--- traitement en fonction de l'hémisphère
  if(m_iParalleleOrigineHem == NORD)
    {
    dX=m_dXs+(r*sin(gama));
    dY=m_dYs-(r*cos(gama));
    }
  else
    {
    dX=m_dXs-(r*sin(gama));
    dY=-m_dYs+(r*cos(gama));
    }
```

```
   }

void CProjection::LambertToGeo(double dX,double dY,double &dLongitude,double &dColatitude)
   {
   int i;
   double u,v,r,gama,l,phibis,phi,puissance,dLatitude;

   //--- calcul de la longitude
   u=dX-m_dXs;
   v=dY-m_dYs;
   r=sqrt(u*u+v*v);
   gama=atan(-(u/v));
   dLongitude=(gama/m_dN);
   dLongitude=m_dMeridienCentral+(dLongitude*m_dRadianToMinute);
   if(dLongitude < 0.)dLongitude+=21600.;
   else if(dLongitude >= 21600.)dLongitude-=21600.;

   //--- calcul de la colatitude
   l=-(log(r/m_dC))/m_dN;
   phibis=l;
   v=exp(l);
   phi=phibis;
   u=m_dEllipseE*sin(phi);
   puissance=pow(((1.+u)/(1.-u)),m_dEllipseEsur2);
   phibis=2.*(atan(v*puissance))-m_dPisur2;
   i=1;
   while(phibis != phi && i<50)
      {
      phi=phibis;
      u=m_dEllipseE*sin(phi);
      puissance=pow(((1.+u)/(1.-u)),m_dEllipseEsur2);
      phibis=2.*(atan(v*puissance))-m_dPisur2;
      i++;
      }
   dLatitude=phi*m_dRadianToMinute;

   if(m_iParalleleOrigineHem == NORD)dColatitude=5400.-dLatitude;
   else dColatitude=5400.+dLatitude;
   }
```

A.2.3.3. La classe CMolodensky

La classe `CModolensky` permet de calculer des changements de datum. Le calcul utilise les formules de Molodensky. La précision absolue de ces formules est d'environ deux mètres. Dans SAVANE, toutes les données d'une même base doivent être dans le même *datum*. Le changement de *datum* dans SAVEDIT concerne le seul document de digitalisation ouvert. Le changement de *datum* dans le module SAVANE concerne l'ensemble des objets de la base de données.

```
class CMolodensky
   {
   private:
      double m_dA;
      double m_dF;
      double m_dDeltaX;
      double m_dDeltaY;
      double m_dDeltaZ;
      double m_dDeltaA;
      double m_dDeltaF;
      double m_dBsurA;
      double m_dAsurB;
      double m_dE2;

   public:
   CMolodensky();
   void Init(double dA,double dE2,double dDeltaX,double dDeltaY,double dDeltaZ,double
dDeltaA,double dDeltaF);
   void Calcul(double dLongitude,double dColatitude,double dHauteur,double& dLongRes,double&
dLatRes,double& dHauteurRes);
   };

void CMolodensky::Init(double dA,double dE2,double dDeltaX,double dDeltaY,double
dDeltaZ,double dDeltaA,double dDeltaF)
```

```
  {
  m_dA=dA;
  m_dE2=dE2;
  m_dDeltaX=dDeltaX;
  m_dDeltaY=dDeltaY;
  m_dDeltaZ=dDeltaZ;
  m_dDeltaA=dDeltaA;
  m_dDeltaF=dDeltaF;

  m_dF=1.- sqrt(1.-m_dE2);
  m_dBsurA=1.-m_dF;
  m_dAsurB=1./m_dBsurA;
  }

void CMolodensky::Calcul(double dLongitude,double dColatitude,double dHauteur,double&
dLongitudeRes,double& dColatitudeRes,double& dHauteurRes)
  {
  double dMinuteToRadian=acos(-1.)/10800.;
  //--- longitude
  double dLong=dLongitude;
  if(dLong > 10800.)dLong=21600.-dLong;
  dLong*=dMinuteToRadian;
  double dSinLong=sin(dLong);
  double dCosLong=cos(dLong);
  //--- latitude
  double dLat=dColatitude;
  dLat=5400.-dLat;
  dLat*=dMinuteToRadian;
  double dSinLat=sin(dLat);
  double dCosLat=cos(dLat);

  double dRn=m_dA/sqrt(1.-m_dE2*dSinLat*dSinLat);
  double dRm=m_dA*(1.-m_dE2)/pow(1.-m_dE2*dSinLat*dSinLat,1.5);

  double dDeltaLong=(-m_dDeltaX*dSinLong + m_dDeltaY*dCosLong)/((dRn+dHauteur)*dCosLat);
  dDeltaLong/=dMinuteToRadian;

  double dDeltaLat=-m_dDeltaX*dSinLat*dCosLong-m_dDeltaY*dSinLat*dSinLong+m_dDeltaZ*dCosLat;
  dDeltaLat=dDeltaLat + m_dDeltaA*(dRn*m_dE2*dSinLat*dCosLat)/m_dA;
  dDeltaLat=dDeltaLat + m_dDeltaF*(dRm*m_dAsurB + dRn*m_dBsurA)*dSinLat*dCosLat;
  dDeltaLat=dDeltaLat/(dRm + dHauteur);
  dDeltaLat/=dMinuteToRadian;

  double dDeltaH=m_dDeltaX*dCosLat*dCosLong+m_dDeltaY*dCosLat*dSinLong+m_dDeltaZ*dSinLat;
  dDeltaH=dDeltaH-m_dDeltaA*sqrt(1.- m_dE2*dSinLat*dSinLat)
  + m_dDeltaF*m_dBsurA*dRn*dSinLat*dSinLat;

  if(dLongitude < 0. || dLongitude > 10800.)dLongitudeRes=dLongitude-dDeltaLong;
  else dLongitudeRes=dLongitude+dDeltaLong;
  dColatitudeRes=dColatitude-dDeltaLat;
  dHauteurRes=dHauteur+dDeltaH;
  }
```

dA, *dE2* sont les paramètres de l'ellipsoïde de départ ; *dDeltaX,dDeltaY,dDeltaZ* sont les différences de position des centres des datum ; *dDeltaA* et *dDeltaF* sont les différences de grand coté et d'aplatissement entre les deux ellipsoïdes.

A.2.4. Documents et cartographie

A.2.4.1. La classe CCarte

La classe CCarte encapsule données et méthodes pour le document de base du module SAVANE. Le module SAVANE ne contient qu'une seule instance de cette classe.

```
class CCarte
   {
   //--- attributs
   private:
      BOOL m_bASauver;
      int m_iVersion;
      long m_ptrlib;
      void DeleteObjets();
   public:
      char m_chNom[_MAX_PATH];
      char m_chNomDir[_MAX_PATH];
      int m_iLargeur; //--- largeur de la feuille (mm/100)
      int m_iHauteur; //--- hauteur de la feuille (mm/100)
      int m_iTaille; //--- taille de la feuille, codé
      int m_iOrientation; //--- orientation de la feuille (0 portrait,1 paysage)
      int m_iNbCadres; //--- nombre de cadres géographiques

      //--- pointeur des objets cadres
      CCadre* m_pCadre[NB_MAX_CADRES];

      int m_iXVueDansCarte; //--- position du point (0,0) de la vue (espace de visualisation)
dans la carte (mm)
      int m_iYVueDansCarte;
      double m_dFacteur; //--- facteur de zoom de tracé pour la carte
      double m_dCoef; //--- coefficient de zoom pour la carte
      double m_dMMToPrint; //--- pour passer des coordonnées en MM/100 aux coordonnées du
DCPrint

      int m_iRegle; //--- active ou desactive le tracé des règles
      int m_iGrille; //--- active ou desactive les repères magnétiques
      int m_iGapPixel; //--- taille de la grille magnetique en pixel ecran
      int m_iGapMM; //--- taille de la grille magnetique en mm/10

      int m_iGuideCarteX[NB_MAX_GUIDES]; //--- position des guides de tracé
      int m_iGuideCarteY[NB_MAX_GUIDES];
      int m_iGuideVueX[NB_MAX_GUIDES]; //--- position des guides de tracé dans la vue
      int m_iGuideVueY[NB_MAX_GUIDES];

      int m_iNbObjets;
      CODD m_tabObjets[NB_MAX_OBJET]; //--- tableau des objets à tracer dans la carte

      //--- mémoire des zoom successifs
      int m_iMemoireNbZoom;
      int m_itabMemoireXVueDansCarte[10];
      int m_itabMemoireYVueDansCarte[10];
      double m_dtabMemoireFacteur[10];

   //--- implémentation
   public:
      CCarte();
      BOOL Nouveau(void);
      BOOL Ouvrir(char* chNomCarte);
      BOOL Fermer(void);
      BOOL Sauver(void);
      BOOL SauverSous(void);
      BOOL ChargerLesCadres();
      BOOL SauverLesCadres();
      BOOL ChargerLesObjets();
      BOOL SauverLesObjets();
      void ReInit();

      int QuelCadre(CPoint point);
      void ASauver() {m_bASauver=TRUE;};
      BOOL IsASauver() {return m_bASauver;};
      BOOL IsPointDansCadre(CCadre* pCadre,int iXVue,int iYVue);

      void TailleDesGroupes(void);
```

```
    void ModifierLaSelection(int iMode,int iX,int iY);
    void SauverLeZoom();
    void ZoomDansCarte();
    void ZoomDansCadre(int iNoCadre);
    void EffacerPixmap();
    void ToutTracer();
    void TracerUnCadre(CCadre* pCadre);
    void TracerLePapier();
    void TracerLePapier(int iXbas,int iYbas,int iXhaut,int iYHaut);
    void TracerLesObjets(CDC* pDC);
    void TracerTexte(CDC* pDC,int iXbCarte,int iYbCarte,int iXhCarte,int iYhCarte,int
iSizePoint,COLORREF Couleur,LOGFONT* pLogfont,const char* chTexte);
    int EpaisseurMMToVue(double dEpaisseur,int iPrint);
    int MMToVue(double dMM,int iPrint);
  };
```

Les seuls membres liées à la base de données sont les pointeurs des objets de type `CCadre` (`CCadre* m_pCadre[NB_MAX_CADRES]`). Tous les autres membres et méthodes de cette classe sont liées à la représentation graphique de la carte, sur l'écran ou sur imprimante. Outre des cadres, une carte peut contenir des objets graphiques provenant du dessin direct effectué l'écran. Ces objets sont gérés par la classe CODD (B.2.4.2). Les ODD représentent des classes propres à chaque type d'objet graphique (texte, ligne, polyligne, polygone, rectangle, rectangle dégradé, cercle, ellipse, symbole, échelle graphique, bitmap). Les objets graphiques sont conservés dans la carte par le tableau des pointeurs des objets de type CODD correspondants.

A.2.4.2. Les classes CODD

Les ODD représentent des classes propres à chaque type d'objet graphique pouvant être dessiné à l'écran par l'utilisateur du système (texte, ligne, polyligne, polygone, rectangle, rectangle dégradé, cercle, ellipse, symbole, échelle graphique, bitmap). Les classe CODD_ encapsulent toutes les informations nécessaires à la manipulation des objets graphiques, selon leur type. Voici par exemple les classes `CODD_Symbole` et `CODD_Texte` :

```
class CODD_Symbole
  {
  public:
    int m_iNumero;
    int m_iXCarte;
    int m_iYCarte;
    int m_iTaille; //--- taille en mm/100
    double m_dEpaisseur; //--- epaisseur en mm/100
    int m_iDetourage;
    int m_iTrame;
    COLORREF m_CouleurInt;
    COLORREF m_CouleurExt;
  };

class CODD_Texte
  {
  public:
    int m_itabX[6];
    int m_itabY[6];
    int m_iSizePoint;
    COLORREF m_Couleur;
    LOGFONT m_logfont;
    char m_chTexte[500];

  public:
    CODD_Texte();
  };
```

Ces classes ne contiennent pas de méthodes. La classe CODD regroupe l'ensemble des méthodes de manipulation des objets graphiques, quelque soit leur type :

```
class CODD
    {
    public:
     BOOL m_bValid;
     int m_iType; //--- type (texte, rectangle, ...)
     int m_iGroupe;
     int m_iXbCarte; //--- position de l'objet dans la carte
     int m_iYbCarte;
     int m_iXhCarte;
     int m_iYhCarte;
     double m_dAngle; //--- angle de rotation,sens trigo, en radian
     LPVOID m_lpODD; //--- pointeur vers un ODD

  public:
     CODD();
     void Deplacer(int iX,int iY);
     int Dupliquer(int iX,int iY);
     void Modifier(int iAction,int iX,int iY);
     void Symetrie(BOOL bVertical=TRUE);
     void Rotation(double dAngle);
     void Couleur(COLORREF Couleur,BOOL bInterieur=TRUE);
     void Trait(int iType,double dEpaisseur,COLORREF Couleur);
     void Taille(int iX1,int iY1,int iX2,int iY2);
     int Lire(FILE* Fichier);
     BOOL Ecrire(FILE* Fichier);
  };
```

A.2.4.3. Les classes CGroupe et CSelection

Les classes `CGroupe` et `CSelection` permettent de gérer le groupement d'objets graphiques et de manipuler les objets graphiques sur l'écran.

```
class CGroupes
  {
  private:
     BOOL m_bOccupe[NB_MAX_GROUP];

  public:
     int m_iXbCarte[NB_MAX_GROUP];
     int m_iYbCarte[NB_MAX_GROUP];
     int m_iXhCarte[NB_MAX_GROUP];
     int m_iYhCarte[NB_MAX_GROUP];

  public:
     void Init(void);
     int Nouveau(void);
     int NbObjets(int iNoGroupe);
     BOOL IsOccupe(int iNoGroupe) {return m_bOccupe[iNoGroupe];};
     void TailleDesCadres();
     void TailleDesObjets();
  };
```

La variable `m_iGroupe` de CODD donne le numéro du groupe auquel appartient l'objet.

```
class CSelection
  {
  public:
     int m_iNb;
     int m_iNoDuGroupe[NB_MAX_SELECTION];
     int m_iFirst;

  public:
     BOOL IsPointInside(int iXVue,int iYVue);
     void RectangleCarte(int& iXb,int& iYb,int& iXh,int& iYh);
     CRect RectangleVue();
  };
```

La variable `m_iNoDuGroupe` est un tableau qui donne l'ensemble des groupes d'objets graphiques sélectionnés sur l'écran.

A.2.4.4. La classe CCadre

La classe `CCadre` est une des plus importantes du module SAVANE. Elle décrit un cadre géographique. Elle contient des variables décrivant l'apparence graphique du cadre (position dans la carte, type bord, échelle, amorces géographiques, etc.), et des variables décrivant la requête sur la base de données (`m_pWind`, `m_pSchema`, etc.) :

```
class CCadre
  {
  private:
    //--- carte propriétaire
    CCarte* m_pCarte;
    //--- wind et projection, schéma d'interrogation SGBD, macro, explorateur
    CProjection* m_pProjection;
    CWind* m_pWind;
    CSchema* m_pSchema;
    CMacro* m_pMacro;
    CMacro* m_pMacroCadre;
    CCartExplorateur* m_pExplorateur;
    CDialogStatExplorateur* m_pDlgStatExplorateur;
  public:
    //--- fenetre du cadre en projection, normalement identique à la fenetre wind
    double m_dFgxb;
    double m_dFgyb;
    double m_dFgxh;
    double m_dFgyh;

    //--- propriétés de dessin du cadre
    int m_iType;
    int m_iXDansCarte;
    int m_iYDansCarte;
    double m_dEchelle; //--- echelle en metre du cadre geo dans la carte
    double m_dEchelleMM; //--- echelle en mm/100 du cadre geo dans la carte
    double m_dUnite; //--- unite  pour le bord et la legende (metres)

    //--- amorces de projection
    BOOL m_bAmorcesProj;
    double m_dAmorcesProjIntervalle;
    int m_iAmorcesProjLignes;
    double m_dAmorcesProjEpaisseur;
    COLORREF m_couleurAmorcesProj;
    BOOL m_bAmorcesProjTexte;
    LOGFONT m_logfontAmorcesProj;
    COLORREF m_couleurAmorcesProjTexte;
    int m_iAmorcesProjSizeFont;

    //--- amorces geographiques
    BOOL m_bAmorcesGeo;
    int m_iAmorcesGeoDeg;
    int m_iAmorcesGeoMin;
    int m_iAmorcesGeoSec;
    int m_iAmorcesGeoLignes;
    double m_dAmorcesGeoEpaisseur;
    COLORREF m_couleurAmorcesGeo;
    BOOL m_bAmorcesGeoTexte;
    LOGFONT m_logfontAmorcesGeo;
    COLORREF m_couleurAmorcesGeoTexte;
    int m_iAmorcesGeoSizeFont;

    //--- amorces de coin
    BOOL m_bAmorcesCoin;
    LOGFONT m_logfontAmorcesCoin;
    COLORREF m_couleurAmorcesCoin;
    int m_iAmorcesCoinSizeFont;

  public:
    char m_chNom[_MAX_PATH]; //--- nom du répertoire de sauvegarde pour le cadre

  public:
```

```
    CCadre(CCarte* pCarte,int iNoCadre);
    CCadre(CCarte* pCarte,int iNoCadre,BOOL bMemeBase,FILE* Fichier);
    ~CCadre();

    CCarte* GetCarte() {return m_pCarte;};
    CProjection* GetProjection() {return m_pProjection;};
    CWind* GetWind() {return m_pWind;};
    CSchema* GetSchema() {return m_pSchema;};
    CMacro* GetMacro() {return m_pMacro;};
    CMacro* GetMacroCadre() {return m_pMacroCadre;};
    CCartExplorateur* GetExplorateur() {return m_pExplorateur;};
    CDialogStatExplorateur* GetDlgStatExplorateur() {return m_pDlgStatExplorateur;};

    //--- procédures de changement de repère
    void SavToProj(float fSavX,float fSavY,double& dProjX,double& dProjY);
    void SavToRas(float fSavX,float fSavY,float& fRasX,float& fRasY);
    void SavToRas(float fXSav,float fYSav,int& iXRas,int& iYRas);
    void SavToCarte(float fSavX,float fSavY,int& iCarteX,int& iCarteY);
    void SavToVue(float fSavX,float fSavY,int& iVueX,int& iVueY,int iPrint);

    void ProjToSav(double dProjX,double dProjY,float& fSavX,float& fSavY);
    void ProjToCarte(double dProjX,double dProjY,int& iCarteX,int& iCarteY);
    void ProjToVue(double dProjX,double dProjY,int& iVueX,int& iVueY);
    void ProjToVue(double dProjX,double dProjY,int& iVueX,int& iVueY,int iPrint);

    void VueToSav(int iVueX,int iVueY,float& fSavX,float& fSavY);
    void VueToProj(int iVueX,int iVueY,double& dProjX,double& dProjY);
    void VueToRas(int iVueX,int iVueY,float& fRasX,float& fRasY);

    void RasToVue(float fRasX,float fRasY,int& iVueX,int& iVueY);
    void RasToSav(float fRasX,float fRasY,float& fSavX,float& fSavY);

    void Unite();
    void ReAjuster();
    void Effacer();
    void ToutEffacer();
    void TracerBord(CDC* pDC);
    void CoordonneesDansLaVue(int& iXb,int& iYb,int& iXh,int& iYh,int iPrint=0);
  };
```

La classe contient des méthodes permettant de changer de repère, entre coordonnées géographiques, coordonnées de projection, coordonnées carte, coordonnées écran ou imprimante. Par exemple, voici l'implémentation de la fonction `GeoToVue`, permettant de passer des coordonnées géographiques d'un point à sa position dans la vue (écran ou imprimante) :

```
void CCadre::GeoToVue(double dXGeo,double dYGeo,int& iVueX,int& iVueY,int iPrint)
  {
  double dX,dY;
  GeoToProj(dXGeo,dYGeo,dX,dY);

  if(iPrint)
    {
    dX=((double)m_iXDansCarte)+((dX-m_dFgxb)*m_dEchelleMM);
    dY=((double)m_iYDansCarte)+((dY-m_dFgyb)*m_dEchelleMM);
    iVueX=(int) (dX*carte.m_dMMToPrint);
    iVueY=(int) ((dY - carte.m_iHauteur)*carte.m_dMMToPrint);

    if(config.m_bMetaFile)iVueY=-iVueY;

    if(iVueX > INFINI)iVueX=INFINI;else if(iVueX < -INFINI)iVueX=-INFINI;
    if(iVueY > INFINI)iVueY=INFINI;else if(iVueY < -INFINI)iVueY=-INFINI;
    }
  else
    {
    dX=((double)m_iXDansCarte)+((dX-m_dFgxb)*m_dEchelleMM);
    dY=((double)m_iYDansCarte)+((dY-m_dFgyb)*m_dEchelleMM);
    iVueX= (int) ((dX - (double)carte.m_iXVueDansCarte)*carte.m_dCoef);
    iVueY=config.m_iVueHauteurRes-1 -(int) ((dY-
(double)carte.m_iYVueDansCarte)*carte.m_dCoef);
    if(iVueX > INFINI)iVueX=INFINI;else if(iVueX < -INFINI)iVueX=-INFINI;
    if(iVueY > INFINI)iVueY=INFINI;else if(iVueY < -INFINI)iVueY=-INFINI;
    }
  }
```

Un certain nombre de procédures globales permettent de passer d'un système de coordonnées à un autre. Par exemple, la procédure globale permettant de passer des coordonnées « carte » aux coordonnées « vue » est la suivante :

```
void CarteToVue(int iCarteX,int iCarteY,int& iVueX,int& iVueY,int iPrint)
   {
   if(iPrint)
      {
      iVueX=(int) (iCarteX*carte.m_dMMToPrint);
      iVueY=(int) ((iCarteY - carte.m_iHauteur)*carte.m_dMMToPrint);
      }
   else
      {
      iVueX= (int) ((double)(iCarteX - carte.m_iXVueDansCarte)*carte.m_dCoef);
      iVueY=config.m_iVueHauteurRes-1 - (int) ((double)(iCarteY -
carte.m_iYVueDansCarte)*carte.m_dCoef);
      }
   if(iVueX > INFINI)iVueX=INFINI;else if(iVueX < -INFINI)iVueX=-INFINI;
   if(iVueY > INFINI)iVueY=INFINI;else if(iVueY < -INFINI)iVueY=-INFINI;
   }
```

A.2.4.5. La classe CCartExplorateur

La classe `CCartExplorateur` correspond à l'explorateur cartographique d'un cadre. Elle regroupe l'ensemble des informations permettant de dessiner dans un cadre à partir des objets de la base de données. Les paramètres de dessin sont contenus dans les variables membres de type `CDessin`, que nous allons décrire au paragraphe suivant. La classe `CCartExplorateur` permet surtout de gérer la liste des relations, masques, ou fonds graphiques à dessiner. A chaque entrée correspondent un objet de chaque type d'implantation graphique (`CDessinPixel`, `CDessinZone`, …) contenant les paramètres reliant base de données et variables visuelles, et que nous présenterons dans le paragraphe suivant.

```
class CCartExplorateur
   {
   public:
      CDessinPixel* m_pDessinPixel[NB_MAX_DESSIN];
      CDessinZone* m_pDessinZone[NB_MAX_DESSIN];
      CDessinLigne* m_pDessinLigne[NB_MAX_DESSIN];
      CDessinSymbole* m_pDessinSymbole[NB_MAX_DESSIN];
      CDessinMasque* m_pDessinMasque[NB_MAX_DESSIN];
      CDessinFond* m_pDessinFond[NB_MAX_DESSIN];
      int m_iType[NB_MAX_DESSIN];
      char m_chNom[NB_MAX_DESSIN][100];
      BOOL m_btabTracer[NB_MAX_DESSIN];

   private:
      CCadre* m_pCadre;
      int m_iNbEntree;

   public:
      CCartExplorateur(CCadre* pCadre);
      CCartExplorateur(CCadre* pCadre,FILE* Fichier);
      ~CCartExplorateur();
      void Ecrire(FILE* Fichier);
      void AjouterUneRelation(int iNoth);
      void AjouterUnMasque(const char *chNomMasque);
      void AjouterUnFond(const char *chNomFond);
      void SupprimerUneEntree(int iIndex);
      int GetNbEntree() {return m_iNbEntree;};
      CCadre* GetCadre() {return m_pCadre;};
      BOOL IsRelationInside(const char* chNomRelation);
      BOOL IsRelationInside(int iNoth);
      void MiseAJour();

      void TracerUneEntree(CDC* pDC,int iIndex);
```

```
    void ExporterUneEntreeEnBMP(int iIndex,const char* chNomFichierBMP);
    void ToutTracer(CDC* pDC);
  };
```

A.2.4.6. Les classes CDessin

Les classes `CDessin` correspondent aux différentes implantations graphiques en cartographie (par pixel, par zone, par ligne, par point). Chaque objet contient l'ensemble des paramètres permettant de relier les valeurs des objets de la relation à dessiner et les variables visuelles choisies par l'utilisateur. Ces classes contiennent donc de très nombreuses variables. Elles contiennent également les méthodes de tracé ou d'impression, ainsi que les procédures de sérialisation de l'ensemble des paramètres.

```
class CDessinPixel
  {
  //--- classe pour le dessin de pixel
  private:
    CCadre* m_pCadre;
  public:
    char m_chNomRelation[100];

    //--- type de dessin (palette, valeurs, illumination, rvb)
    int m_iDessin;
    //--- attribut à tracer
    int m_iNoatt;
    char m_chNomAttribut[100];
    //--- tracé par palette
    int m_iPalNbValeurs;
    char m_chPalValeurs[TAILLE_PALETTE_VALEUR][NB_MAX_CHARACTER];
    int m_iPalTrame[TAILLE_PALETTE_VALEUR];
    int m_iPalCouleur[TAILLE_PALETTE_VALEUR];
    //--- tracé par valeur
    float m_fA;
    float m_fB;
    int m_iIndexDeCouleur1;
    int m_iIndexDeCouleur2;
    int m_iIndexDeTrame;
    int m_iEtalement;
    //--- tracé par illumination
    int m_iIllumOrientation; //--- en degré
    int m_iIllumInclinaison;
    double m_dIllumFacteur;

    //--- tracé avec estompage
    int m_iEstompage;
    char m_chNomRelationEstompage[100];
    char m_chNomAttributEstompage[100];
    int m_iInclinaisonEstompage;
    int m_iOrientationEstompage;
    double m_dFacteurEstompage;

  private:
    void TracerPalette(CDC* pDC);
    void TracerValeur(CDC* pDC);
    void TracerIllum(CDC* pDC);
    void TracerRVB(CDC* pDC);
    void ImprimerPalette(CDC* pDC);
    void ImprimerValeur(CDC* pDC);
    void ImprimerIllum(CDC* pDC);
    void ImprimerRVB(CDC* pDC);
    void TracerPaletteRaster(CDC* pDC);
    void TracerValeurRaster(CDC* pDC);
    void TracerIllumRaster(CDC* pDC);
    void TracerRVBRaster(CDC* pDC);
    void ExporterPalette(const char* chNomFichier);
    void ExporterValeur(const char* chNomFichier);
    void ExporterIllum(const char* chNomFichier);
    void ExporterRVB(const char* chNomFichier);

  public:
    CDessinPixel(CCadre* pCadre);
    void Lire(FILE* Fichier);
```

```
    void Ecrire(FILE* Fichier);
    void Tracer(CDC* pDC);
    void TracerRaster(CDC* pDC);
    void Exporter(const char* chNomFichierBmp);
    CCadre* GetCadre() {return m_pCadre;};
    BOOL MiseAJour();
  };

class CDessinZone
  {
  //--- classe pour le dessin de zones et contours de zones
  private:
    CCadre* m_pCadre;
  public:
    //--- attributs généraux
    char m_chNomRelation[100];

    //--- attributs pour le tracé de zones par plages
    //--- type de dessin (palette, valeurs)
    int m_iDessinPlage;
    int m_iTypeDessinPlage;
    int m_iNoattPlage;
    char m_chNomAttributPlage[100];

    //--- tracé par palette
    int m_iPalNbValeursPlage;
    char m_chPalValeursPlage[TAILLE_PALETTE_VALEUR][NB_MAX_CHARACTER];
    int m_iPalTramePlage[TAILLE_PALETTE_VALEUR];
    int m_iPalCouleurPlage[TAILLE_PALETTE_VALEUR];
    //--- tracé par valeurs
    float m_fAPlage;
    float m_fBPlage;
    int m_iIndexDeCouleur1Plage;
    int m_iIndexDeCouleur2Plage;
    int m_iIndexDeTrame1Plage;
    int m_iIndexDeTrame2Plage;
    int m_iEtalementPlage;
    //--- tracé par trame
    float m_fATrame;
    float m_fBTrame;
    int m_iMotifTrame;
    int m_iIntervalle1Trame;
    int m_iEpaisseur1Trame;
    int m_iIntervalle2Trame;
    int m_iEpaisseur2Trame;
    COLORREF m_CouleurTrame;

    //--- tracé avec estompage
    int m_iEstompage;
    char m_chNomRelationEstompage[100];
    char m_chNomAttributEstompage[100];
    int m_iInclinaisonEstompage;
    int m_iOrientationEstompage;
    double m_dFacteurEstompage;

    //--- attributs pour le tracé des contours des zones
    //--- type de dessin (rien, tout, attribut)
    int m_iDessinContour;

    int m_iIndexDeCouleurContour;
    double m_dEpaisseurContour; //--- en mm/100
    int m_iTypeDeTraitContour;

    int m_iNoattContour;
    char m_chNomAttributContour[100];
    int m_bDessinArcEgalite;
    int m_bDessinArcDifference;
    int m_iIndexDeCouleurContourEgalite;
    double m_dEpaisseurContourEgalite;
    int m_iTypeDeTraitContourEgalite;
    int m_iIndexDeCouleurContourDifference;
    double m_dEpaisseurContourDifference;
    int m_iTypeDeTraitContourDifference;

  private:
    void TracerPlage(CDC* pDC);
    void TracerPlageRaster(CDC* pDC,int iModeSuperposition,CPerspective* pPerspective);
    void TracerParEstompage(CDC* pDC);
    void TracerContours(CDC* pDC);
```

```
    void ImprimerPlage(CDC* pDC);
    void ImprimerParEstompage(CDC* pDC);
    void ImprimerZones(CDC* pDC);
    BOOL ImprimerUneZone(CDC* pDC,FILE* FileArc,COLORREF couleur,int iTrame,int iNbarc,int*
itabPtrarc);

  public:
    CDessinZone(CCadre* pCadre);
    void Lire(FILE* Fichier);
    void Ecrire(FILE* Fichier);
    void Tracer(CDC* pDC);
    void TracerRaster(CDC* pDC,int iModeSuperposition,CPerspective* pPerspective);
    CCadre* GetCadre() {return m_pCadre;};
    BOOL MiseAJour();
    void ExporterParEstompage(const char* chNomFichier);
  };


class CDessinLigne
  {
  //--- classe pour le dessin de ligne
  private:
    CCadre* m_pCadre;
  public:
    //--- attributs généraux
    char m_chNomRelation[100];

    int m_iDessin;
    //--- tous les objets identiques
    COLORREF m_Couleur;
    int m_iTypeDeTrait;
    double m_dEpaisseur;
    //--- courbes de niveaux
    int m_iTrace1Courbe;
    int m_iTrace2Courbe;
    double m_dIntervalle1Courbe;
    double m_dIntervalle2Courbe;
    double m_dEpaisseur1Courbe; //--- en mm/100
    double m_dEpaisseur2Courbe; //--- en mm/100
    COLORREF m_Couleur1Courbe;
    COLORREF m_Couleur2Courbe;
    int m_iTraceDesValeurs1Courbe;
    int m_iTraceDesValeurs2Courbe;
    COLORREF m_CouleurTexte1Courbe;
    COLORREF m_CouleurTexte2Courbe;
    LOGFONT m_logfontTexte1Courbe;
    LOGFONT m_logfontTexte2Courbe;
    int m_iSizePointTexte1Courbe; //--- taille en points
    int m_iSizePointTexte2Courbe; //--- taille en points
    //--- épaisseur par attribut numérique, couleur et type constant (ou venant du nominal si
existe)
    int m_iNoattNum;
    char m_chNomAttributNum[100];
    float m_fA;
    float m_fB;
    int m_iEtalement;
    double m_dEpaisseur1; //--- en mm/100
    double m_dEpaisseur2; //--- en mm/100
    //--- couleur, type, epaisseur par attribut nominal (epaisseur venant du numerique si
existe)
    int m_iNoattNom;
    char m_chNomAttributNom[100];
    int m_iPalNbValeurs;
    char m_chtabPalValeurs[TAILLE_PALETTE_VALEUR][NB_MAX_CHARACTER];
    int m_tabPalCouleur[TAILLE_PALETTE_VALEUR];
    int m_itabPalType[TAILLE_PALETTE_VALEUR];
    double m_dtabPalEpaisseur[TAILLE_PALETTE_VALEUR];

  public:
    CDessinLigne(CCadre* pCadre);
    void Lire(FILE* Fichier);
    void Ecrire(FILE* Fichier);
    void Tracer(CDC* pDC);
    CCadre* GetCadre() {return m_pCadre;};
    BOOL MiseAJour();
  };


class CDessinSymbole
  {
  //--- classe pour le dessin de symboles et de valeurs
```

```
private:
   CCadre* m_pCadre;

   void TracerSymboles(CDC* pDC);
   void TracerValeurs(CDC* pDC);
   void TracerDiagrammes(CDC* pDC);
public:
   char m_chNomRelation[100];
   //--- attributs généraux pour les symboles
   int m_iNoSymbole;
   double m_dEpaisseurSymbole;
   double m_dTailleSymbole;
   int m_iTrameSymbole;
   COLORREF m_CouleurIntSymbole;
   COLORREF m_CouleurExtSymbole;
   int m_iDetourageSymbole;
   int m_iTriSymbole;

   //--- attributs pour le dessin de symboles
   int m_iDessinSymbole;
   int m_iNoattNomSymbole;
   int m_iNoattNumSymbole;
   char m_chNomAttributNomSymbole[100];
   char m_chNomAttributNumSymbole[100];
   //--- tracé par palette de valeur pour l'attribut nominal ou l'attribut nominal associé
   int m_iPalNbValeurs;
   char m_chtabPalValeurs[TAILLE_PALETTE_VALEUR][NB_MAX_CHARACTER];
   int m_itabPalNoSymbole[TAILLE_PALETTE_VALEUR];
   double m_dtabPalEpaisseur[TAILLE_PALETTE_VALEUR]; //--- taille en mm/100
   double m_dtabPalTaille[TAILLE_PALETTE_VALEUR]; //--- taille en mm/100
   int m_itabPalTrame[TAILLE_PALETTE_VALEUR];
   COLORREF m_tabPalCouleurInt[TAILLE_PALETTE_VALEUR];
   COLORREF m_tabPalCouleurExt[TAILLE_PALETTE_VALEUR];
   //--- tracé par valeurs pour attribut numérique
   int m_iEtalementSymbole;
   int m_iProportionaliteSymbole;
   float m_fASymbole;
   float m_fBSymbole;
   float m_fCSymbole;
   double m_dTailleMinSymbole;
   double m_dTailleMaxSymbole;
   double m_dTaillePourCSymbole;

   //--- attributs pour le dessin de valeurs
   int m_iDessinTexte;
   int m_iNoattTexte;
   char m_chNomAttributTexte[100];
   COLORREF m_CouleurTexte;
   LOGFONT m_logfontTexte;
   int m_iSizePointTexte; //--- taille en points
   int m_iXDeplacementTexte;
   int m_iYDeplacementTexte;
   int m_iNbCaracteresTexte;
   int m_iDebutTexte;
   int m_iNbDecimalesTexte;
   BOOL m_bConditionTexte;
   char m_chConditionTexte[100];

   //--- Attributs pour le dessin de diagrammes sectoriels
   int m_iDessinDiagrammes;
   int m_iNbAttributDiagrammes;
   char m_chtabNomAttributDiagrammes[NB_MAX_ATTRIBUTS][100];
   COLORREF m_tabCouleurDiagrammes[NB_MAX_ATTRIBUTS];
   COLORREF m_CouleurDuBordDiagrammes;
   int m_iTypeDeTailleDiagrammes;
   char m_chNomAttributTailleDiagrammes[100];
   float m_fCDiagrammes;
   double m_dTaillePourCDiagrammes;
   int m_iEtalementValeursDiagrammes;
   int m_iEtalementTailleDiagrammes;

public:
   CDessinSymbole(CCadre* pCadre);
   void Lire(FILE* Fichier);
   void Ecrire(FILE* Fichier);
   void Tracer(CDC* pDC);
   CCadre* GetCadre() {return m_pCadre;};
   BOOL MiseAJour();
};
```

```
class CDessinMasque
  {
  //--- classe pour le dessin de masque
  private:
    CCadre* m_pCadre;
    BOOL ImprimerUneZone(CDC* pDC,FILE* FileArc,COLORREF couleur,int iTrame,int iNbarc,int*
itabPtrarc);

  public:
    //--- attributs généraux
    char m_chNom[100];
    int m_bContour;
    int m_bInterieur;
    int m_bExterieur;
    int m_iIndexDeCouleurTrait;
    int m_iTypeDeTrait;
    double m_dEpaisseurTrait;
    int m_iIndexDeCouleurInterieur;
    int m_iTrameInterieur;
    int m_iIndexDeCouleurExterieur;
    int m_iTrameExterieur;

  public:
    CDessinMasque(CCadre* pCadre);
    void Lire(FILE* Fichier);
    void Ecrire(FILE* Fichier);
    void Tracer(CDC *pDC);
    void TracerPlage(CDC *pDC);
    void ImprimerPlage(CDC *pDC);
    void TracerRaster(CDC *pDC,int iModeSuperposition,CPerspective* pPerspective);
    CCadre* GetCadre() {return m_pCadre;};
    BOOL MiseAJour();
  };


class CDessinFond
  {
  //--- classe pour le dessin de fond graphique
  private:
    CCadre* m_pCadre;
  public:
    //--- attributs généraux
    char m_chNom[100];
    int m_bNiveau[256];

  public:
    CDessinFond(CCadre* pCadre);
    void Lire(FILE* Fichier);
    void Ecrire(FILE* Fichier);
    void Tracer(CDC *pDC);
    CCadre* GetCadre() {return m_pCadre;};
    BOOL MiseAJour();
  };
```

Par exemple, la classe `CDessinPixel` permet de tracer ou d'imprimer une relation de type mosaïque. Les procédures de tracer sur écran sont quelque peu différentes des procédures de tracé sur imprimante : le tracé sur écran se fait pixel par pixel, alors que l'impression utilise des procédures de la MFC pour la gestion des DIB (`StretchDibBits()`). Les procédures de tracé ou d'impression sont différentes selon le type de l'attribut à représenter : si l'attribut à tracer est nominal, la procédure `TracerPalette()` sera utilisée pour affecter à chaque pixel la couleur et la trame définies par l'utilisateur dans la palette de valeur. `TracerValeur()` est utilisée pour le tracé d'un attribut numérique (sans palette de valeurs), `TracerIllum()` pour tracer un attribut numérique par illumination, `TracerRVB()` pour tracer un attribut de type RVB. Par exemple, voici le code de la procédure `TracerIllum()` :

```
void CDessinPixel::TracerIllum(CDC* pDC)
   {
   //--- paramètres
   double dPI=acos(-1.);
   int iPrint=config.IsPrinting(pDC);
   CSchema* pSchema=m_pCadre->GetSchema();
   int iNoth=pSchema->RelRecherche(m_chNomRelation);
   if(iNoth == 0)return;
   int iNoatt=pSchema->AttrRecherche(iNoth,m_chNomAttribut);
   if(iNoatt == 0)return;

   FILE *FileMosaic;
   FILE *FileFeuille;
   FILE *FileImage;
   int iMosaiqueNbcol,iMosaiqueNblig;
   double dMosaiqueResolution;
   int iTypeAttribut,iDatasize;
   int iLargeur,iHauteur,iX,iY;
   int iVueX,iVueY;
   int iImageX,iImageY,iPrecedentY;
   int i,iColor,no,nbcol,nblig;
   int ptr,ptrdeb,ptrlib;
   double dAvanceX,dAvanceY,dFacteur;
   double d1,d2,dZmin,dZmax;
   double dXbf,dYbf,dXhf,dYhf;
   char chNomFichierFeuille[_MAX_PATH];
   char chNomFichierMosaique[_MAX_PATH];
   char chNomFichierAttribut[_MAX_PATH];
   BYTE bColor;

   unsigned char cval1,cval2,cval3,cval4;
   short int sval1,sval2,sval3,sval4;
   float fval1,fval2,fval3,fval4;

   double dCol,dSol,dCil,dSil,dNxl,dNyl,dNzl;
   double dA,dB,dNx,dNy,dNz,dNz2,dTeta,dNorme,dAngle;

   unsigned char *bufima1,*bufima2;

   //--- coordonnees du cadre
   double dCadreXbProj,dCadreYbProj,dCadreXhProj,dCadreYhProj;
   int iCadreXbVue,iCadreYbVue,iCadreXhVue,iCadreYhVue;
   //--- coordonnees de l'imagette
   double dImagetteXbProj,dImagetteYbProj,dImagetteXhProj,dImagetteYhProj;
   int iImagetteXbVue,iImagetteYbVue,iImagetteXhVue,iImagetteYhVue;
   double dImagetteXbImage,dImagetteYbImage;
   int iImagetteXbImage,iImagetteYbImage;

   strcpy(chNomFichierMosaique,pSchema->m_pRelations[iNoth]->m_chFichZone);
   strcat(chNomFichierMosaique,"\\");
   strcat(chNomFichierMosaique,pSchema->m_pRelations[iNoth]->m_chNom);
   strcat(chNomFichierMosaique,"_d");

   //--- lecture des parametres de la mosaique
   FileMosaic=OuvrirMosaique(pSchema,chNomFichierMosaique,iNoth);
   ::SetCursor(LoadCursor(NULL, IDC_WAIT));
   if(FileMosaic != NULL)
      {
      fscanf(FileMosaic,"%lf",&dMosaiqueResolution);
      fscanf(FileMosaic,"%d",&iMosaiqueNbcol);
      fscanf(FileMosaic,"%d",&iMosaiqueNblig);
      fclose(FileMosaic);
      }
   else return;

   int iTemporaire=pSchema->m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iTemporaire;
   if(iTemporaire)
      {
      strcpy(chNomFichierAttribut,carte.m_chNomDir);
      strcat(chNomFichierAttribut,"\\");
      strcat(chNomFichierAttribut,m_pCadre->m_chNom);
      strcat(chNomFichierAttribut,"\\ft");
      strcat(chNomFichierAttribut,pSchema->m_pRelations[iNoth]->m_chNom);
      strcat(chNomFichierAttribut,"_");
      strcat(chNomFichierAttribut,pSchema->m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_chNom);
      }
   else
      {
      strcpy(chNomFichierAttribut,pSchema->m_pRelations[iNoth]->m_chFichZone);
```

```
     strcat(chNomFichierAttribut,"\\");
     strcat(chNomFichierAttribut,pSchema->m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_chNom);
     }

   strcpy(chNomFichierFeuille,chNomFichierAttribut);
   strcat(chNomFichierFeuille,"_f");

   FileFeuille=fopen(chNomFichierFeuille,"rb");
   FileImage=fopen(chNomFichierAttribut,"rb");
   if(FileImage == NULL || FileFeuille == NULL)
     {
     if(FileImage != NULL)fclose(FileImage);
     if(FileFeuille != NULL)fclose(FileFeuille);
     return;
     }

   //--- caracteristiques du cadre dans la vue
   dCadreXbProj=m_pCadre->m_dFgxb;
   dCadreYbProj=m_pCadre->m_dFgyb;
   dCadreXhProj=m_pCadre->m_dFgxh;
   dCadreYhProj=m_pCadre->m_dFgyh;

   iLargeur=(int) ((dCadreXhProj-dCadreXbProj)*m_pCadre->m_dEchelleMM);
   iHauteur=(int) ((dCadreYhProj-dCadreYbProj)*m_pCadre->m_dEchelleMM);

   ::CarteToVue(m_pCadre->m_iXDansCarte,m_pCadre-
>m_iYDansCarte,iCadreXbVue,iCadreYbVue,iPrint);
   iX=m_pCadre->m_iXDansCarte + iLargeur;
   iY=m_pCadre->m_iYDansCarte + iHauteur;
   ::CarteToVue(iX,iY,iCadreXhVue,iCadreYhVue,iPrint);
   ::Ordonner(iCadreYbVue,iCadreYhVue);

   if(iCadreXhVue < 0 || iCadreXbVue >= config.m_iVueLargeurRes || iCadreYhVue < 0 ||
iCadreYbVue >= config.m_iVueHauteurRes)
     {
     fclose(FileImage);
     fclose(FileFeuille);
     return;
     }

   if((iCadreXhVue-iCadreXbVue) == 0 || (iCadreYhVue-iCadreYbVue) == 0)
     {
     fclose(FileImage);
     fclose(FileFeuille);
     return;
     }

   //--- allocation memoire
   iDatasize=1;
   iTypeAttribut=pSchema->m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iType;
   if(iTypeAttribut == 1 || iTypeAttribut == 5 || iTypeAttribut == 6)iDatasize=1;
   else if(iTypeAttribut == 7)iDatasize=2;
   else if(iTypeAttribut == 8)iDatasize=3;
   else if(iTypeAttribut == 9)iDatasize=4;

   bufima1=(unsigned char *) malloc(iMosaiqueNbcol*iDatasize);
   bufima2=(unsigned char *) malloc(iMosaiqueNbcol*iDatasize);
   if(bufima1 == NULL || bufima2 == NULL)
     {
     ::ErreurDeMemoire();
     if(bufima1 != NULL)free(bufima1);
     if(bufima2 != NULL)free(bufima2);
     fclose(FileImage);
     fclose(FileFeuille);
     return;
     }

   //--- facteur de lecture
   double dFpix;
   d1=(dCadreXhProj-dCadreXbProj)/((double)(iCadreXhVue-iCadreXbVue));
   d2=(dCadreYhProj-dCadreYbProj)/((double)(iCadreYhVue-iCadreYbVue));
   if(d1 < d2)dFpix=d2; else dFpix=d1;
   dFacteur=dFpix/dMosaiqueResolution;

   //--- parametres de visualisation
   dAngle=(double) m_iIllumOrientation;
   dAngle=dAngle+90.; //--- pour rattraper mon erreur...
   if(dAngle > 360.)dAngle=dAngle-360.;
   dAngle = (dAngle/180.)*dPI;
   dCol= cos(dAngle);
```

```
  dSol= sin(dAngle);
  dAngle = (double) (m_iIllumInclinaison/180.)*dPI;
  dCil = cos(dAngle);
  dSil = sin(dAngle);
  dNxl=dCil*dSol;
  dNyl=(-1.)*dCil*dCol;
  dNzl=dSil;
  dNz= 2.*dMosaiqueResolution/m_dIllumFacteur;
  dNz2=dNz*dNz;

  //--- lecture des feuilles
  BOOL bInconnu;

  fscanf(FileFeuille,"%2d%20d%*c",&i,&ptrlib);
 while(fscanf(FileFeuille,"%2d%20lf%20lf%10d%10d%20lf%20lf%20d%*c",
  &no,&dXbf,&dYbf,&nbcol,&nblig,&dZmin,&dZmax,&ptrdeb) == 8)
    {
    if(no != 0)
      {
      dXhf=dXbf + (dMosaiqueResolution*(double) iMosaiqueNbcol);
      dYhf=dYbf + (dMosaiqueResolution*(double) iMosaiqueNblig);
      if(dXhf >= dCadreXbProj && dXbf <= dCadreXhProj && dYhf >= dCadreYbProj && dYbf <=
dCadreYhProj)
        {
        //--- au moins un bout de l'imagette dans le cadre
        m_pCadre->ProjToVue(dXbf,dYbf,iImagetteXbVue,iImagetteYbVue);
        m_pCadre->ProjToVue(dXhf,dYhf,iImagetteXhVue,iImagetteYhVue);
        ::Ordonner(iImagetteYbVue,iImagetteYhVue);

        if(iImagetteXbVue < iCadreXbVue)iImagetteXbVue=iCadreXbVue;
        if(iImagetteXhVue > iCadreXhVue)iImagetteXhVue=iCadreXhVue;
        if(iImagetteYbVue < iCadreYbVue)iImagetteYbVue=iCadreYbVue;
        if(iImagetteYhVue > iCadreYhVue)iImagetteYhVue=iCadreYhVue;

        if(iImagetteXhVue >= 0 && iImagetteXbVue < config.m_iVueLargeurRes && iImagetteYhVue
>= 0 && iImagetteYbVue < config.m_iVueHauteurRes)
          {
          //--- au moins un bout de l'imagette dans la vue
          if(iImagetteXbVue < 0)iImagetteXbVue=0;
          if(iImagetteXhVue >=
config.m_iVueLargeurRes)iImagetteXhVue=config.m_iVueLargeurRes-1;
          if(iImagetteYbVue < 0)iImagetteYbVue=0;
          if(iImagetteYhVue >=
config.m_iVueHauteurRes)iImagetteYhVue=config.m_iVueHauteurRes-1;

          //--- coordonnées du tracé en projection
          m_pCadre-
>VueToProj(iImagetteXbVue,iImagetteYhVue,dImagetteXbProj,dImagetteYbProj);
          m_pCadre-
>VueToProj(iImagetteXhVue,iImagetteYhVue,dImagetteXhProj,dImagetteYhProj);
          ::Ordonner(dImagetteYbProj,dImagetteYhProj);

          //--- coordonnées du tracé dans l'image
          if(dXbf >= dImagetteXbProj)dImagetteXbImage=0.;
          else dImagetteXbImage=(dImagetteXbProj - dXbf)/dMosaiqueResolution;
          if(dYbf >= dImagetteYbProj)dImagetteYbImage=0.;
          else dImagetteYbImage=(dImagetteYbProj - dYbf)/dMosaiqueResolution;

          iImagetteXbImage=(int) (dImagetteXbImage);
          iImagetteYbImage=(int) (dImagetteYbImage);

          //--- tracé de l'image
          dAvanceY=dImagetteYbImage - (double)iImagetteYbImage;
          iImageY=0;
          iPrecedentY=-1;

          int iNbcolALire=nbcol-iImagetteXbImage;
          iVueY=iImagetteYhVue;
          while(iVueY >= iImagetteYbVue && iImageY < nblig)
            {
            //--- lire la ligne
            if(iImageY != iPrecedentY)
              {
              if(iImageY < nblig-1)
                {
                ptr=ptrdeb+(iImagetteXbImage+(iImageY*nbcol))*iDatasize;
                fseek(FileImage,ptr,SEEK_SET);
                fread(bufima1,iNbcolALire,iDatasize,FileImage);

                ptr=ptrdeb+(iImagetteXbImage+((iImageY+1)*nbcol))*iDatasize;
```

```
                  fseek(FileImage,ptr,SEEK_SET);
                  fread(bufima2,iNbcolALire,iDatasize,FileImage);
                  }
               else
                  {
                  ptr=ptrdeb+(iImagetteXbImage+((iImageY-1)*nbcol))*iDatasize;
                  fseek(FileImage,ptr,SEEK_SET);
                  fread(bufima1,iNbcolALire,iDatasize,FileImage);

                  ptr=ptrdeb+(iImagetteXbImage+(iImageY*nbcol))*iDatasize;
                  fseek(FileImage,ptr,SEEK_SET);
                  fread(bufima2,iNbcolALire,iDatasize,FileImage);
                  }
               }

            dAvanceX=dImagetteXbImage-(double)iImagetteXbImage;
            iImageX=0;
            iVueX=iImagetteXbVue;
            while(iVueX < iImagetteXhVue && iImageX < iNbcolALire-1)
               {
               bInconnu=FALSE;
               if(iTypeAttribut == 6)
                  {
                  memcpy(&cval1,&bufima1[iImageX],1);
                  memcpy(&cval2,&bufima1[iImageX+1],1);
                  memcpy(&cval3,&bufima2[iImageX],1);
                  memcpy(&cval4,&bufima2[iImageX+1],1);
                  if(cval1 == CINCONNU || cval2 == CINCONNU || cval3 == CINCONNU || cval4 ==
CINCONNU)bInconnu=TRUE;
                  else
                     {
                     fval1=(float) cval1;
                     fval2=(float) cval2;
                     fval3=(float) cval3;
                     fval4=(float) cval4;
                     }
                  }
               else if(iTypeAttribut == 7)
                  {
                  memcpy(&sval1,&bufima1[iImageX*iDatasize],iDatasize);
                  memcpy(&sval2,&bufima1[(iImageX+1)*iDatasize],iDatasize);
                  memcpy(&sval3,&bufima2[iImageX*iDatasize],iDatasize);
                  memcpy(&sval4,&bufima2[(iImageX+1)*iDatasize],iDatasize);
                  if(sval1 == SINCONNU || sval2 == SINCONNU || sval3 == SINCONNU || sval4 ==
SINCONNU)bInconnu=TRUE;
                  else
                     {
                     fval1=(float) sval1;
                     fval2=(float) sval2;
                     fval3=(float) sval3;
                     fval4=(float) sval4;
                     }
                  }
               else if(iTypeAttribut == 8)
                  {
                  bInconnu=TRUE;
                  }
               else if(iTypeAttribut == 9)
                  {
                  memcpy(&fval1,&bufima1[iImageX*iDatasize],iDatasize);
                  memcpy(&fval2,&bufima1[(iImageX+1)*iDatasize],iDatasize);
                  memcpy(&fval3,&bufima2[iImageX*iDatasize],iDatasize);
                  memcpy(&fval4,&bufima2[(iImageX+1)*iDatasize],iDatasize);
                  if(fval1 <= FINCONNU || fval2 <= FINCONNU || fval3 <= FINCONNU || fval4 <=
FINCONNU)bInconnu=TRUE;
                  }

               if(bInconnu == FALSE)
                  {
                  dA=(double) (fval4-fval1);
                  dB=(double) (fval2-fval3);
                  dNx= (dB-dA);
                  dNy= -(dA+dB);
                  dNorme= sqrt(dNx*dNx + dNy*dNy + dNz2);
                  dTeta= dNx*dNxl + dNy*dNyl + dNz*dNzl;
                  dTeta= dTeta/dNorme;
                  if(config.m_iPalette == 0)
                     {
                     //--- pas de palette de couleur, valeur directe de gris
                     bColor = (BYTE) (127.*dTeta + 128.);
```

```
                    pDC->SetPixelV(iVueX,iVueY,RGB(bColor,bColor,bColor));
                    }
                  else
                    {
                    iColor = (int) (PALETTE_IMAGE_MIN + 60.*dTeta +61.);
                    pDC->SetPixelV(iVueX,iVueY,PALETTEINDEX((WORD)iColor));
                    }
                  }
               dAvanceX+=dFacteur;
               iImageX=(int)dAvanceX;
               iVueX++;
               }

            while(iVueX < iImagetteXhVue)
               {
               pDC->SetPixelV(iVueX,iVueY,PALETTEINDEX((WORD)iColor));
               iVueX++;
               }

            iPrecedentY=iImageY;
            dAvanceY+=dFacteur;
            iImageY=iImagetteYbImage + (int)dAvanceY;
            iVueY--;
            }
         }
      }
   }
  free(bufima1);
  free(bufima2);

  fclose(FileImage);
  fclose(FileFeuille);
  }
```

Le tracé de zones sur l'écran utilise l'image matricielle interne de la relation. La couleur et la trame de chaque pixel sont déterminées par la valeur de la zone correspondante. La vitesse du tracé est ainsi très rapide. Par contre, l'impression utilise les arcs constituant le contour des objets pour conserver une précision optimale, mais l'algorithme doit recréer le contour de chaque zone :

```
void CDessinZone::ImprimerZones(CDC* pDC)
  {
  if(m_iDessinPlage == 0)return;
  int iPrint=config.IsPrinting(pDC);
  if(iPrint == 0)return;

  if(m_iEstompage != 0 && strlen(m_chNomRelationEstompage) != 0 &&
strlen(m_chNomAttributEstompage) != 0)
    {
    ImprimerParEstompage(pDC);
    return;
    }

  pDC->SetPolyFillMode(ALTERNATE);

  //--- paramètres généraux
  int iXb,iYb,iXh,iYh;
  int i,iNoth,iNoatt,iX,iY,iLargeur,iHauteur;
  int iCadreXbVue,iCadreYbVue,iCadreXhVue,iCadreYhVue;
  double dCadreXbProj,dCadreYbProj,dCadreXhProj,dCadreYhProj;

  CSchema* pSchema=m_pCadre->GetSchema();
  CWind* pWind=m_pCadre->GetWind();

  iNoth=pSchema->RelRecherche(m_chNomRelation);
  if(iNoth == 0)return;
  iNoatt=pSchema->AttrRecherche(iNoth,m_chNomAttributPlage);
  if(iNoatt == 0)return;

  //--- caracteristiques du cadre
  dCadreXbProj=m_pCadre->m_dFgxb;
  dCadreYbProj=m_pCadre->m_dFgyb;
  dCadreXhProj=m_pCadre->m_dFgxh;
  dCadreYhProj=m_pCadre->m_dFgyh;
```

```
   iLargeur=(int) ((dCadreXhProj-dCadreXbProj)*m_pCadre->m_dEchelleMM);
   iHauteur=(int) ((dCadreYhProj-dCadreYbProj)*m_pCadre->m_dEchelleMM);

   ::CarteToVue(m_pCadre->m_iXDansCarte,m_pCadre-
>m_iYDansCarte,iCadreXbVue,iCadreYbVue,iPrint);
   iX=m_pCadre->m_iXDansCarte + iLargeur;
   iY=m_pCadre->m_iYDansCarte + iHauteur;
   ::CarteToVue(iX,iY,iCadreXhVue,iCadreYhVue,iPrint);
   ::Ordonner(iCadreYbVue,iCadreYhVue);

   ::CarteToVue(0,0,iXb,iYb,iPrint);
   ::CarteToVue(carte.m_iLargeur,carte.m_iHauteur,iXh,iYh,iPrint);
   ::Ordonner(iYb,iYh);

   //--- le cadre est-il dans la carte ?
   if(iCadreXhVue < iXb || iCadreXbVue >= iXh || iCadreYhVue < iYb || iCadreYbVue >=
iYh)return;

   //--- la fenetre d'etude est-elle dans le cadre ?
   if(pWind->m_dFhx < dCadreXbProj || pWind->m_dFlx >= dCadreXhProj || pWind->m_dFhy <
dCadreYbProj || pWind->m_dFly >= dCadreYhProj)return;

   int iNbMaxArcParFeuille=pSchema->m_pRelations[iNoth]->NbMaxArcsParFeuille();

   int* itabNz1=(int *) malloc(iNbMaxArcParFeuille*sizeof(int));
   int* itabNz2=(int *) malloc(iNbMaxArcParFeuille*sizeof(int));
   int* itabPtrarc=(int *) malloc(iNbMaxArcParFeuille*sizeof(int));
   int* itabPtr=(int *) malloc(iNbMaxArcParFeuille*sizeof(int));
   if(itabNz1 == NULL || itabNz2 == NULL || itabPtrarc == NULL || itabPtr == NULL)
     {
     ::ErreurDeMemoire();

     if(itabNz1 != NULL)free(itabNz1);
     if(itabNz2 != NULL)free(itabNz2);
     if(itabPtrarc != NULL)free(itabPtrarc);
     if(itabPtr != NULL)free(itabPtr);
     return;
     }

   BOOL bTrouve;
   int iSizeIdentifiantTuple;
   int iIndexDeCouleur,iTrame,iValeur;
   int nz,iNz1,iNz2;
   int iObjet,iNbObj,iPtrObj,iNbarc,iPtr,iArc,iPtrarc,iPtrsuiv;
   int iIndexMin;
   long lAdresse;
   float fU;
   float xc,yc,xb,yb,xh,yh;
   float fXb,fYb,fXh,fYh;
   float fValeur,v[NB_MAX_ATTRIBUTS];
   char chNomValeur[128];
   FILE* FileValeurs;
   FILE* FileFeuille;
   FILE* FileDescriptif;
   FILE* FileArc;
   CArc arc;

   int iTypeRelation=pSchema->m_pRelations[iNoth]->m_iType;
   int iTypeAttribut=pSchema->m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iType;
   int iNb0=pSchema->m_pRelations[iNoth]->m_iNb0;
   int iNba=pSchema->m_pRelations[iNoth]->m_iNba;
   int iNovar=pSchema->m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNumero;
   int iNbcarAttr=pSchema->m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNbcar;
   int iNbvalb=pSchema->m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNbvalb;
   int iPtrval=pSchema->m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iPtrval;

   FileValeurs=NULL;
   if(iTypeAttribut == 4)
     {
     //--- trace par valeurs numeriques
     iIndexMin=0;
     if(m_iIndexDeCouleur2Plage >= m_iIndexDeCouleur1Plage)
       {
       switch(m_iEtalementPlage)
         {
         case 0:
         default:
           fU=((float)(m_iIndexDeCouleur2Plage-m_iIndexDeCouleur1Plage))/(m_fBPlage-
m_fAPlage);
```

```
           break;
         case 1:
           fU=((float)(m_iIndexDeCouleur2Plage-
m_iIndexDeCouleur1Plage))/(float)log(1.+m_fBPlage-m_fAPlage);
           break;
         case 2:
           fU=((float)(m_iIndexDeCouleur2Plage-
m_iIndexDeCouleur1Plage))/(float)sqrt(m_fBPlage-m_fAPlage);
           break;
         }

       iIndexMin=m_iIndexDeCouleur1Plage;
       }
     else
       {
       switch(m_iEtalementPlage)
         {
         case 0:
         default:
           fU=((float)(m_iIndexDeCouleur1Plage-m_iIndexDeCouleur2Plage))/(m_fBPlage-
m_fAPlage);
           break;
         case 1:
           fU=((float)(m_iIndexDeCouleur1Plage-
m_iIndexDeCouleur2Plage))/(float)log(1.+m_fBPlage-m_fAPlage);
           break;
         case 2:
           fU=((float)(m_iIndexDeCouleur1Plage-
m_iIndexDeCouleur2Plage))/(float)sqrt(m_fBPlage-m_fAPlage);
           break;
         }

       iIndexMin=m_iIndexDeCouleur2Plage;
       }
     }

  else if(iTypeAttribut == 1 || iTypeAttribut == 5)
     {
     //--- attribut nominal, tracé par palette de valeurs
     if(iTypeAttribut == 1)FileValeurs=fopen(base.m_chNomFichierValeurs,"rb");
     else FileValeurs=fopen(pSchema->m_chNomFtval,"rb");
     }

  //--- début
  iSizeIdentifiantTuple=SIZEINT + 6*SIZECOOR;
  if(iTypeRelation == 4)
     {
     FileFeuille=fopen(pSchema->m_pRelations[iNoth]->m_chFichFeuille,"rb");
     FileDescriptif=fopen(pSchema->m_pRelations[iNoth]->m_chFichZone,"rb");
     FileArc=fopen(pSchema->m_pRelations[iNoth]->m_chFichArc,"rb");


  while(fscanf(FileFeuille,"%*17c%8f%8f%8f%8f%8i%8i%8i%8i%*c",&fXb,&fYb,&fXh,&fYh,&iNbObj,&iPt
rObj,&iNbarc,&iPtrarc) == 8)
       {
       if(fXh >= pWind->m_fFx1 && fXb <= pWind->m_fFx2 && fYh >= pWind->m_fFy1 && fYb <=
pWind->m_fFy2)
         {
         iPtr=iPtrarc;
         iArc=0;
         while(iArc < iNbarc)
           {
           arc.LireEntete(FileArc,iPtr,iNz1,iNz2,iPtrsuiv);
           itabNz1[iArc]=iNz1;
           itabNz2[iArc]=iNz2;
           itabPtrarc[iArc]=iPtr;
           iArc++;
           iPtr=iPtrsuiv;
           }

         iPtr=(iPtrObj-1)*(iNb0*SIZEVAL + iSizeIdentifiantTuple);
         fseek(FileDescriptif,iPtr,SEEK_SET);
         iObjet=0;
         while(iObjet < iNbObj)
           {
           fread(&nz,SIZEINT,1,FileDescriptif);
           fread(&xc,SIZECOOR,1,FileDescriptif);
           fread(&yc,SIZECOOR,1,FileDescriptif);
           fread(&xb,SIZECOOR,1,FileDescriptif);
           fread(&yb,SIZECOOR,1,FileDescriptif);
```

```
            fread(&xh,SIZECOOR,1,FileDescriptif);
            fread(&yh,SIZECOOR,1,FileDescriptif);
            for(i=1;i <= iNb0;i++)fread(&v[i],SIZEVAL,1,FileDescriptif);
            iObjet++;

            nz=UnixToDos(nz);
            xc=UnixToDos(xc);
            yc=UnixToDos(yc);
            xb=UnixToDos(xb);
            yb=UnixToDos(yb);
            xh=UnixToDos(xh);
            yh=UnixToDos(yh);
            if(xh >= pWind->m_fFx1 && xb <= pWind->m_fFx2 && yh >= pWind->m_fFy1 && yb <=
pWind->m_fFy2)
               {
               for(i=1;i <= iNb0;i++)v[i]=UnixToDos(v[i]);

               //--- couleur et trame de la zone nz
               iIndexDeCouleur=-1;
               iTrame=m_iIndexDeTrame1Plage;

               if(iTypeAttribut == 4)
                  {
                  //--- attribut numérique
                  fValeur=v[iNovar];
                  if(fValeur > FINCONNU && fValeur >= m_fAPlage && fValeur <= m_fBPlage)
                     {
                     switch(m_iEtalementPlage)
                        {
                        case 0:
                        default:
                           iIndexDeCouleur=int((fU*(fValeur-m_fAPlage)) + iIndexMin);
                           break;
                        case 1:
                           iIndexDeCouleur=int((fU*(float)log(1+fValeur-m_fAPlage)) + iIndexMin);
                           break;
                        case 2:
                           iIndexDeCouleur=int((fU*(float)sqrt(fValeur-m_fAPlage)) + iIndexMin);
                           break;
                        }
                     }
                  }
               else if(iTypeAttribut == 1 || iTypeAttribut == 5)
                  {
                  //--- attribut nominal, tracé par palette de valeurs
                  iValeur= (int) (v[iNovar] + 0.1);
                  if(iValeur > 0)
                     {
                     lAdresse=(iPtrval+iValeur-2)*iNbcarAttr;
                     fseek(FileValeurs,lAdresse,SEEK_SET);
                     fread(chNomValeur,iNbcarAttr,1,FileValeurs);
                     chNomValeur[iNbcarAttr]=0;
                     strstrip(chNomValeur);
                     }
                  else
                     {
                     switch(config.m_iLangage)
                        {
                        case FRANCAIS:
                        default:
                           strcpy(chNomValeur,"valeur manquante");
                           break;
                        case ESPAGNOL:
                           strcpy(chNomValeur,"valor faltante");
                           break;
                        case ANGLAIS:
                           strcpy(chNomValeur,"unknow value");
                           break;
                        }
                     }

                  bTrouve=FALSE;
                  i=0;
                  while(i < m_iPalNbValeursPlage && bTrouve == FALSE)
                     {
                     if(strcmp(m_chPalValeursPlage[i],chNomValeur) == 0)
                        {
                        bTrouve=TRUE;
                        iIndexDeCouleur=m_iPalCouleurPlage[i];
                        iTrame=m_iPalTramePlage[i];
```

```
                  }
                i++;
                }
              }

            if(iIndexDeCouleur != -1)
              {
              //--- chainage de la zone en tâches et impression
              int iNbarcDeLaZone=0;
              for(iArc=0;iArc < iNbarc;iArc++)
                {
                if(itabNz1[iArc] == nz || itabNz2[iArc] == nz)
                  {
                  itabPtr[iNbarcDeLaZone]=itabPtrarc[iArc];
                  iNbarcDeLaZone++;
                  }
                }
              COLORREF
Couleur=PALETTERGB(palette.rouge[iIndexDeCouleur],palette.vert[iIndexDeCouleur],palette.bleu[i
IndexDeCouleur]);
              ImprimerUneZone(pDC,FileArc,Couleur,iTrame,iNbarcDeLaZone,itabPtr);
              }
            }
          }
        }
      }
    fclose(FileFeuille);
    fclose(FileDescriptif);
    fclose(FileArc);
    }

  else if(iTypeRelation == 14 || iTypeRelation == 24)
    {
    int iNo=pSchema->m_pRelations[iNoth]->m_iNoFichDescriptif;
    FileDescriptif=fopen(pSchema->m_chFprovDescriptif[iNo],"rb");

    iNo=pSchema->m_pRelations[iNoth]->m_iNoFichArc;
    if(iNo == -1)FileArc=fopen(pSchema->m_pRelations[iNoth]->m_chFichArc,"rb");
    else FileArc=fopen(pSchema->m_chFprovArc[iNo],"rb");

    fread(&iNbarc,SIZEINT,1,FileDescriptif);
    fread(&iPtrarc,SIZEINT,1,FileDescriptif);
    while(iNbarc > 0)
      {
      //--- lire les arcs de la feuille
      iPtr=iPtrarc;
      iArc=0;
      while(iArc < iNbarc)
        {
        arc.LireEntete(FileArc,iPtr,iNz1,iNz2,iPtrsuiv);
        itabNz1[iArc]=iNz1;
        itabNz2[iArc]=iNz2;
        itabPtrarc[iArc]=iPtr;
        iPtr=iPtrsuiv;
        iArc++;
        }

      fread(&nz,SIZEINT,1,FileDescriptif);
      fread(&xc,SIZECOOR,1,FileDescriptif);
      fread(&yc,SIZECOOR,1,FileDescriptif);
      fread(&xb,SIZECOOR,1,FileDescriptif);
      fread(&yb,SIZECOOR,1,FileDescriptif);
      fread(&xh,SIZECOOR,1,FileDescriptif);
      fread(&yh,SIZECOOR,1,FileDescriptif);
      for(i=1;i <= iNba;i++)fread(&v[i],SIZEVAL,1,FileDescriptif);
      while(nz > 0)
        {
        if(xh >= pWind->m_fFx1 && xb <= pWind->m_fFx2 && yh >= pWind->m_fFy1 && yb <= pWind-
>m_fFy2)
          {
          //--- couleur et trame de la zone nz
          iIndexDeCouleur=-1;
          iTrame=m_iIndexDeTrame1Plage;

          if(iTypeAttribut == 4)
            {
            //--- attribut numérique
            fValeur=v[iNovar];
            if(fValeur > FINCONNU && fValeur >= m_fAPlage && fValeur <= m_fBPlage)
              {
```

```
                switch(m_iEtalementPlage)
                  {
                  case 0:
                  default:
                    iIndexDeCouleur=int((fU*(fValeur-m_fAPlage)) + iIndexMin);
                    break;
                  case 1:
                    iIndexDeCouleur=int((fU*(float)log(1+fValeur-m_fAPlage)) + iIndexMin);
                    break;
                  case 2:
                    iIndexDeCouleur=int((fU*(float)sqrt(fValeur-m_fAPlage)) + iIndexMin);
                    break;
                  }
                }
              }

            else if(iTypeAttribut == 1 || iTypeAttribut == 5)
              {
              //--- attribut nominal, tracé par palette de valeurs
              iValeur= (int) (v[iNovar] + 0.1);
              if(iValeur > 0)
                {
                lAdresse=(iPtrval+iValeur-2)*iNbcarAttr;
                fseek(FileValeurs,lAdresse,SEEK_SET);
                fread(chNomValeur,iNbcarAttr,1,FileValeurs);
                chNomValeur[iNbcarAttr]=0;
                strstrip(chNomValeur);
                }
              else
                {
                switch(config.m_iLangage)
                  {
                  case FRANCAIS:
                  default:
                    strcpy(chNomValeur,"valeur manquante");
                    break;
                  case ESPAGNOL:
                    strcpy(chNomValeur,"valor faltante");
                    break;
                  case ANGLAIS:
                    strcpy(chNomValeur,"unknow value");
                    break;
                  }
                }

              bTrouve=FALSE;
              i=0;
              while(i < m_iPalNbValeursPlage && bTrouve == FALSE)
                {
                if(strcmp(m_chPalValeursPlage[i],chNomValeur) == 0)
                  {
                  bTrouve=TRUE;
                  iIndexDeCouleur=m_iPalCouleurPlage[i];
                  iTrame=m_iPalTramePlage[i];
                  }
                i++;
                }
              }

            if(iIndexDeCouleur != -1)
              {
              //--- chainage de la zone en tâches et impression
              int iNbarcDeLaZone=0;
              for(iArc=0;iArc < iNbarc;iArc++)
                {
                if(itabNz1[iArc] == nz || itabNz2[iArc] == nz)
                  {
                  itabPtr[iNbarcDeLaZone]=itabPtrarc[iArc];
                  iNbarcDeLaZone++;
                  }
                }
              COLORREF
Couleur=PALETTERGB(palette.rouge[iIndexDeCouleur],palette.vert[iIndexDeCouleur],palette.bleu[i
IndexDeCouleur]);
              ImprimerUneZone(pDC,FileArc,Couleur,iTrame,iNbarcDeLaZone,itabPtr);
              }
            }

          fread(&nz,SIZEINT,1,FileDescriptif);
          fread(&xc,SIZECOOR,1,FileDescriptif);
```

```
         fread(&yc,SIZECOOR,1,FileDescriptif);
         fread(&xb,SIZECOOR,1,FileDescriptif);
         fread(&yb,SIZECOOR,1,FileDescriptif);
         fread(&xh,SIZECOOR,1,FileDescriptif);
         fread(&yh,SIZECOOR,1,FileDescriptif);
         for(i=1;i <= iNba;i++)fread(&v[i],SIZEVAL,1,FileDescriptif);
         }

       fread(&iNbarc,SIZEINT,1,FileDescriptif);
       fread(&iPtrarc,SIZEINT,1,FileDescriptif);
       }

     fclose(FileDescriptif);
     fclose(FileArc);
     }

  if(FileValeurs != NULL)fclose(FileValeurs);
  free(itabNz1);
  free(itabNz2);
  free(itabPtrarc);
  free(itabPtr);
  }

BOOL CDessinZone::ImprimerUneZone(CDC* pDC,FILE* FileArc,COLORREF Couleur,int iTrame,int
iNbArc,int* itabPtrarc)
  {
  //--- impression d'une zone avec chainage par tâche
  double* dtabX1=(double *) malloc(sizeof(double)*iNbArc);
  double* dtabY1=(double *) malloc(sizeof(double)*iNbArc);
  double* dtabX2=(double *) malloc(sizeof(double)*iNbArc);
  double* dtabY2=(double *) malloc(sizeof(double)*iNbArc);

  BOOL *btabLibre=(BOOL *) malloc(sizeof(BOOL)*iNbArc);
  BOOL *btabSens=(BOOL *) malloc(sizeof(int)*iNbArc);
  int *itabPtrTache=(int *) malloc(sizeof(int)*iNbArc);

  //--- vérification des allocations mémoire
  if(dtabX1 == NULL || dtabY1 == NULL || dtabX2 == NULL || dtabY2 == NULL || btabLibre == NULL
|| btabSens == NULL || itabPtrTache == NULL)
    {
    if(dtabX1 != NULL)free(dtabX1);
    if(dtabY1 != NULL)free(dtabY1);
    if(dtabX2 != NULL)free(dtabX2);
    if(dtabY2 != NULL)free(dtabY2);
    if(btabLibre != NULL)free(btabLibre);
    if(btabSens != NULL)free(btabSens);
    if(itabPtrTache != NULL)free(itabPtrTache);
    return FALSE;
    }

  //--- initialisations
  CArc arc;
  float fX1,fY1,fX2,fY2;
  for(int iArc=0;iArc < iNbArc;iArc++)
    {
    arc.LireBout(FileArc,itabPtrarc[iArc],fX1,fY1,fX2,fY2);
    dtabX1[iArc]=fX1;
    dtabY1[iArc]=fY1;
    dtabX2[iArc]=fX2;
    dtabY2[iArc]=fY2;
    btabLibre[iArc]=TRUE;
    }

  //--- bounding box de la zone
  int iXbZone,iYbZone,iXhZone,iYhZone;
  int iXbArc,iYbArc,iXhArc,iYhArc;
  if(iTrame > 1)
    {
    for(iArc=0;iArc < iNbArc;iArc++)
      {
      arc.LirePoints(FileArc,itabPtrarc[iArc]);
      arc.CalculBBox(pDC,m_pCadre,iXbArc,iYbArc,iXhArc,iYhArc);
      if(iArc == 0)
        {
        iXbZone=iXbArc;
        iYbZone=iYbArc;
        iXhZone=iXhArc;
        iYhZone=iYhArc;
        }
      else
```

```
        {
        if(iXbArc < iXbZone)iXbZone=iXbArc;
        if(iXhArc > iXhZone)iXhZone=iXhArc;
        if(iYbArc < iYbZone)iYbZone=iYbArc;
        if(iYhArc > iYhZone)iYhZone=iYhArc;
        }
      }
    }

  pDC->BeginPath();
  //--- chainage par tâche
  double dXDebut,dYDebut,dXFin,dYFin;
  int iNbTachesNonFermees=0;
  CTrame trame;
  trame.Set(iTrame);

  iArc=0;
  while(iArc < iNbArc)
    {
    //--- chainage par ring
    while(iArc < iNbArc && btabLibre[iArc] == FALSE)iArc++;
    if(iArc == iNbArc)break;

    //--- début du ring
    dXDebut=dtabX1[iArc];
    dYDebut=dtabY1[iArc];
    dXFin=dtabX2[iArc];
    dYFin=dtabY2[iArc];

    int iNbarcTache=0;
    itabPtrTache[iNbarcTache]=itabPtrarc[iArc];
    btabSens[iNbarcTache]=0;
    iNbarcTache++;
    btabLibre[iArc]=FALSE;
    iArc++;
    BOOL bFerme=FALSE;
    if(dXDebut == dXFin && dYDebut == dYFin)bFerme=TRUE;
    int jArc=iArc;
    while(jArc < iNbArc && bFerme == FALSE)
      {
      if(btabLibre[jArc] == TRUE)
        {
        if(dtabX1[jArc] == dXFin && dtabY1[jArc] == dYFin)
          {
          itabPtrTache[iNbarcTache]=itabPtrarc[jArc];
          btabSens[iNbarcTache]=0;
          iNbarcTache++;
          dXFin=dtabX2[jArc];
          dYFin=dtabY2[jArc];
          btabLibre[jArc]=FALSE;
          jArc=iArc;
          if(dXFin == dXDebut && dYFin == dYDebut)bFerme=TRUE;
          }
        else if(dtabX2[jArc] == dXFin && dtabY2[jArc] == dYFin)
          {
          itabPtrTache[iNbarcTache]=itabPtrarc[jArc];
          btabSens[iNbarcTache]=1;
          iNbarcTache++;
          dXFin=dtabX1[jArc];
          dYFin=dtabY1[jArc];
          btabLibre[jArc]=FALSE;
          jArc=iArc;
          if(dXFin == dXDebut && dYFin == dYDebut)bFerme=TRUE;
          }
        else jArc++;
        }
      else jArc++;
      }
    //--- fin du chainage d'une tâche, écriture du polygone dans le path
    if(bFerme)
      {
      //--- allocation mémoire pour le bigarc
      int iNbpt=0;
      int iNbptTotal=0;
      for(int i=0;i < iNbarcTache;i++)
        {
        iNbpt=arc.LireNbpt(FileArc,itabPtrTache[i]);
        iNbptTotal+=iNbpt;
        }
```

```
      CPoint* tabPoint=(CPoint *) malloc(sizeof(CPoint)*iNbptTotal);
      if(tabPoint == NULL)return FALSE;
      //--- chainage dans le bigarc
      int iNbptPolygon=0;
      for(i=0;i < iNbarcTache;i++)
        {
        arc.LirePoints(FileArc,itabPtrTache[i]);
        arc.Chainer(pDC,m_pCadre,btabSens[i],iNbptPolygon,tabPoint);
        }
      //--- tracer dans le path
      pDC->Polygon(tabPoint,iNbptPolygon);
      free(tabPoint);
      }
    else iNbTachesNonFermees++;
    }
  pDC->EndPath();

  free(dtabX1);
  free(dtabY1);
  free(dtabX2);
  free(dtabY2);
  free(btabLibre);
  free(btabSens);
  free(itabPtrTache);

  if(iNbTachesNonFermees == 0)
    {
    //--- remplir avec la couleur
    CBrush brosse,*pBrosseOld;
    if(iTrame == 1)
      {
      brosse.CreateSolidBrush(Couleur);
      pBrosseOld=(CBrush *) pDC->SelectObject(&brosse);
      pDC->FillPath();
      pDC->SelectObject(pBrosseOld);
      }
    else
      {
      //--- clipping dans le polygone
      int iSave=pDC->SaveDC();
      if(iSave != 0)
        {
        pDC->SelectClipPath(RGN_AND);
        //--- tracer le rectangle avec la trame choisie
        trame.Rectangle(pDC,iXbZone,iYbZone,iXhZone,iYhZone,Couleur);

        pDC->RestoreDC(iSave);
        }
      }
    return TRUE;
    }
  return FALSE;
  }
```

A.2.4.7. Les classes CLegende

Les classes `CLegende` encapsulent toutes les opérations de définition et de tracé de légendes, à partir des données de l'explorateur cartographique du cadre. La légende dépend bien sûr du type des objets représentés et du type des variables visuelles utilisées. Différentes classes permettent de gérer les différents types de légendes : légendes par caissons, légendes de symboles, de diagrammes, etc. :

```
class CLegendeCaisson
  {
  private:
    int m_iNbCaissons;
    char m_chTexte[TAILLE_PALETTE_VALEUR][NB_MAX_CHARACTER];
    CSize m_tabSizeTexte[TAILLE_PALETTE_VALEUR];
    int m_itabXbCaisson[TAILLE_PALETTE_VALEUR];
    int m_itabYbCaisson[TAILLE_PALETTE_VALEUR];
    int m_itabXhCaisson[TAILLE_PALETTE_VALEUR];
    int m_itabYhCaisson[TAILLE_PALETTE_VALEUR];
    int m_itabXbTexte[TAILLE_PALETTE_VALEUR];
    int m_itabYbTexte[TAILLE_PALETTE_VALEUR];
    int m_itabXhTexte[TAILLE_PALETTE_VALEUR];
```

```
    int m_itabYhTexte[TAILLE_PALETTE_VALEUR];
    char m_chTitre[80];

  public:
    CLegendeCaisson();
    void Init(CDC* pDC);
    void TraceRapide(CDC* pDC,int iXVue,int iYVue);
    void TraceDansCarte(CDC* pDC,int iXVue,int iYVue);
  };

class CLegendeDegrade
  {
  private:
    int m_iNbCaissons;
    char m_chTexte[TAILLE_PALETTE_VALEUR][NB_MAX_CHARACTER];
    CSize m_tabSizeTexte[TAILLE_PALETTE_VALEUR];
    int m_itabXbCaisson[TAILLE_PALETTE_VALEUR];
    int m_itabYbCaisson[TAILLE_PALETTE_VALEUR];
    int m_itabXhCaisson[TAILLE_PALETTE_VALEUR];
    int m_itabYhCaisson[TAILLE_PALETTE_VALEUR];
    int m_itabXbTexte[TAILLE_PALETTE_VALEUR];
    int m_itabYbTexte[TAILLE_PALETTE_VALEUR];
    int m_itabXhTexte[TAILLE_PALETTE_VALEUR];
    int m_itabYhTexte[TAILLE_PALETTE_VALEUR];
    char m_chTitre[80];

  public:
    CLegendeDegrade();
    void Init(CDC* pDC);
    void TraceRapide(CDC* pDC,int iXVue,int iYVue);
    void TraceDansCarte(CDC* pDC,int iXVue,int iYVue);
  };

class CLegendeSymbole
  {
  private:
    int m_iLargeurTexte;
    int m_iHauteurTexte;
    int m_iNoSymbole;
    double m_dTailleSymbole;
    char m_chTexte[100];

  public:
    CLegendeSymbole();
    void Init(CDC* pDC);
    void TraceRapide(CDC* pDC,int iXVue,int iYVue);
    void TraceDansCarte(CDC* pDC,int iXVue,int iYVue);
  };

class CLegendeSymboleParTaille
  {
  private:
    int m_iDisposition;
    int m_iNoSymbole;
    int m_iNbSymboles;
    int m_iSizePoint;
    COLORREF m_CouleurTexte;
    LOGFONT m_logfont;
    double m_dtabTailleSymbole[20];
    char m_chTexte[20][100];
    char m_chTitre[80];

  public:
    CLegendeSymboleParTaille();
    void Init();
    void TraceRapide(CDC* pDC,int iXVue,int iYVue);
    void TraceDansCarte(CDC* pDC,int iXVue,int iYVue);
  };

class CLegendeSymboleParValeur
  {
  private:
    int m_iNbSymboles;
    int m_itabNoSymbole[TAILLE_PALETTE_VALEUR];
    int m_itabTailleSymbole[TAILLE_PALETTE_VALEUR];
    int m_itabXSymbole[TAILLE_PALETTE_VALEUR];
    int m_itabYSymbole[TAILLE_PALETTE_VALEUR];
    int m_itabXTexte[TAILLE_PALETTE_VALEUR];
    int m_itabYTexte[TAILLE_PALETTE_VALEUR];
    int m_itabLargeurTexte[TAILLE_PALETTE_VALEUR];
```

```
    int m_itabHauteurTexte[TAILLE_PALETTE_VALEUR];
    char m_chTitre[80];
    int m_iXbTitre;
    int m_iYbTitre;
    int m_iXhTitre;
    int m_iYhTitre;

  public:
    CLegendeSymboleParValeur();
    void Init(CDC* pDC);
    void TraceRapide(CDC* pDC,int iXVue,int iYVue);
    void TraceDansCarte(CDC* pDC,int iXVue,int iYVue);
  };

class CLegendeDiagrammeCaisson
  {
  private:
    int m_iNbCaissons;
    char m_chTexte[TAILLE_PALETTE_VALEUR][NB_MAX_CHARACTER];
    CSize m_tabSizeTexte[TAILLE_PALETTE_VALEUR];
    int m_itabXbCaisson[TAILLE_PALETTE_VALEUR];
    int m_itabYbCaisson[TAILLE_PALETTE_VALEUR];
    int m_itabXhCaisson[TAILLE_PALETTE_VALEUR];
    int m_itabYhCaisson[TAILLE_PALETTE_VALEUR];
    int m_itabXbTexte[TAILLE_PALETTE_VALEUR];
    int m_itabYbTexte[TAILLE_PALETTE_VALEUR];
    int m_itabXhTexte[TAILLE_PALETTE_VALEUR];
    int m_itabYhTexte[TAILLE_PALETTE_VALEUR];

  public:
    CLegendeDiagrammeCaisson();
    void Init(CDC* pDC);
    void TraceRapide(CDC* pDC,int iXVue,int iYVue);
    void TraceDansCarte(CDC* pDC,int iXVue,int iYVue);
  };

class CLegendeDiagrammeParTaille
  {
  private:
    int m_iDisposition;
    int m_iNbSymboles;
    int m_iSizePoint;
    COLORREF m_CouleurTexte;
    LOGFONT m_logfont;
    double m_dtabTailleSymbole[20];
    char m_chtabTexte[20][100];
    char m_chTitre[80];

  public:
    CLegendeDiagrammeParTaille();
    void Init();
    void TraceRapide(CDC* pDC,int iXVue,int iYVue);
    void TraceDansCarte(CDC* pDC,int iXVue,int iYVue);
  };
```

Les procédures de tracé rapide (`TraceRapide()`) permettent de tracé rapidement sur l'écran le contour des objets graphiques constituant la légende, pour la positionner dans la carte de manière interactive. Les procédures `TraceDansCarte()` inscrivent les objets graphiques dans la carte, sous forme d'objets de type `CODD`.

A.2.4.8. La classe CPaletteSavane

La classe `CPaletteSavane` est dérivée de la classe `CPalette` de la MFC. Elle ajoute des fonctionnalités permettant de gérer les deux ensembles de couleurs (thématique et imagerie), de charger des palettes standards, de gérer le passage couleur-index. La gestion de la couleur dépend du mode d'affichage de la carte graphique de l'ordinateur (8 bits, 16 bits, 24 ou 32 bits). La classe `CPaletteSavane` permet de maintenir la gestion d'une palette dans le mode 24 ou

32 bits, mode où Windows permet d'afficher directement les couleurs sans l'indirection d'une palette.

```
class CPaletteSavane : public CPalette
  {
  public :
    CPaletteSavane();

    int rouge[PALETTE_SIZE];
    int vert[PALETTE_SIZE];
    int bleu[PALETTE_SIZE];
    int rouge_tmp[PALETTE_SIZE];
    int vert_tmp[PALETTE_SIZE];
    int bleu_tmp[PALETTE_SIZE];
    int rouge_image[256];
    int vert_image[256];
    int bleu_image[256];

    int m_iCouleur;
    int m_iColor1;
    int m_iColor2;
    int m_iCouleurTheme;
    int m_iCouleurImage;
    int m_iIndex;
    int m_iIndexOld;
    int m_iTrame;
    int m_iMini;
    int m_iMaxi;
    int m_iTheme;

    void SetThematique();
    void SetImagerie();
    void CouleursSysteme();
    int GetExactPaletteIndex(int iDebut,COLORREF couleur);
    COLORREF RGBColorFromIndex(int iIndex) {return
PALETTERGB(rouge[iIndex],vert[iIndex],bleu[iIndex]);};
    void Tetascan();
    void Gris();
    void Ntsc();
    void Pseudocolor();
    void Telecolor();
    void Compocolor();

    void Animer();
    void SetColorInPaletteLogique(int iIndex,COLORREF Couleur);
  };
```

A.2.4.9. La classe CPerspective

La classe `CPerspective` contient méthodes et variables permettant de tracer une mosaïque en perspective. Un attribut d'une autre relation peut être superposé à la mosaïque.

```
class CPerspective
  {
  private:
    CCadre* m_pCadre;
    CImage* m_pImage;
    double* m_dtabZBuffer;
    int m_iLargeurVisu;
    int m_iHauteurVisu;
    int m_iNbcolImage;
    int m_iNbligImage;
    int m_iDatasize;
    int m_iTypeImage;
    double m_dFpix;
    double m_dRapportVisu;

    double m_dPi;
    double m_dDegreToRadian;
    //--- lumiere
    double m_dCol;
    double m_dSol;
    double m_dCil;
    double m_dSil;
```

```
    double m_dNxl;
    double m_dNyl;
    double m_dNzl;

    //--- camera intrinsic
    int m_iCameraLargeurVisuMM;
    int m_iCameraHauteurVisuMM;
    int m_iCameraLargeurVisuPixel;
    int m_iCameraHauteurVisuPixel;
    int m_iCameraFocaleMM;
    //--- camera extrinsic
    int m_iCameraOrientationDegre;
    int m_iCameraInclinaisonDegre;
    int m_iCameraAssietteDegre;
    double m_dCameraLongitude;
    double m_dCameraColatitude;
    double m_dCameraAltitudeMetre;

    double m_dCoo;
    double m_dSoo;
    double m_dCio;
    double m_dSio;
    CBrush m_tabBrosse[256];

  private:
    int m_iMargeXBas;
    int m_iMargeYBas;
    int m_iMargeXHaut;
    int m_iMargeYHaut;
    int m_iMargeXBasVisu;
    int m_iMargeYBasVisu;
    int m_iMargeXHautVisu;
    int m_iMargeYHautVisu;
    void SauverEnBmp(CDC* pDC,char* chNomFichierBmp);
    //--- pour la perspective conique
    double m_dX0;
    double m_dY0;
    double m_dZ0;
    double m_dLG;
    double m_dD;
    double m_dXC;
    double m_dYC;
    double m_dCouleurMinimum;
    double m_dCouleurLargeur;
    int m_iNbSegment;
    POINT3D m_dtabMaille[4];
    POINT3D m_dPolygone[80];

    unsigned char EclairMaille();
    void AffecteMaille(int i,int j,int iCadran);
    void ChangeRepere(POINT3D &point);
    BOOL TestVisibilite(POINT3D point1,POINT3D point2);
    BOOL CalculIntersecPlan(POINT3D point1,POINT3D point2,POINT3D &point_inter);
    BOOL PointVisual(POINT3D point);

    void InitConique(int iPas);
    BOOL Conique(int i,int j,int &iX,int &iY);
    void TracerConiqueFilaire(CDC* pDC);

  public:
    double m_dZmin;
  private:
    //--- pour la perspective cavaliere
    int m_iPas;
    double m_dA;
    double m_dB;
    double m_dU;
    double m_dV;
    double m_dXmin;
    double m_dYmin;
    double m_dCoef;
    double m_dCtx;
    double m_dCty;

    void InitCavaliere(int iPas);
    BOOL Cavaliere(int i,int j,int &iX,int &iY);
    void TracerCavaliereFilaire(CDC* pDC);
    void TracerCavaliereEclairement(CDC* pDC);
    void TracerCavaliereTheme(CDC* pDC,int iNbSelection,int* itabSelection,int
iModeSuperposition);
```

```
  public:
     CPerspective(CCadre* pCadre,CImage* pImage);
     void GetParametreVisu(int &iLargeurVisu,int &iHauteurVisu);
     void SetRapport(double dFacteur) {m_dRapportVisu=dFacteur/m_dFpix;};
     void SetSoleilIntensite(int iIntensite);
     void SetMarge(int iMargeXBas,int iMargeYBas,int iMargeXHaut,int iMargeYHaut)
{m_iMargeXBas=iMargeXBas;m_iMargeYBas=iMargeYBas;m_iMargeXHaut=iMargeXHaut;m_iMargeYHaut=iMarg
eYHaut;};
     unsigned char GetEclairement(int i,int j);
     BOOL Tracer(CDC* pDC,int iPas,int iXOrigine,int iYOrigine,int iLargeurVisu,int
iHauteurVisu,int iPerspective,int iMode,int iOLumiere,int iILumiere,int iOObserv,int
iIObserv);
     BOOL TracerBmp(int iLargeurVisu,int iHauteurVisu,int iPerspective,int iMode,int
iOLumiere,int iILumiere,int iOObserv,int iIObserv,int iNbSelection,int* itabSelection,int
iMoseSuperposition,char* chNomFichierBmp);
     BOOL Animer(int iLargeurVisu,int iHauteurVisu,int iPerspective,int iMode,int
iOLumiere,int iILumiere,int iOObservDebut,int iOObservFin,int iIObservDebut,int
iIObservFin,int iNbImages,int iNbSelection,int* itabSelection,int iModeSuperposition,char*
chNomFichierBmp);
     void PreviewAnimation(CDC* pDC,int iXOrigine,int iYOrigine,int iLargeurVisu,int
iHauteurVisu,int iPerspective,int iMode,int iOLumiere,int iILumiere,int iOObservDebut,int
iOObservFin,int iIObservDebut,int iIObservFin,int iNbImages);
  };
```

A.2.5. Exemples d'opérations dans un cadre

Cette section présente des exemples d'implantation de procédures d'exécution de commande, correspondant à des opérations sur les données d'une base SAVANE. Elles permettent notamment de montrer l'utilisation de la classe `CLecture`, et les méthodes de création de nouveaux attributs ou relations.

A.2.5.1. XQuestUnion()

La procédure XQuestUnion() effectue une union entre deux relations. Comme pour toutes les procédures d'exécution de commande, la procédure commence par la lecture des paramètres de la commande, en provenance du dialogue utilisateur ou d'une macro-commande. Elle crée ensuite dans un tableau contenant l'adresse des tuples émetteurs en fonction de la valeur de la clé de jointure de la relation émettrice, puis, pour chaque *tuple* de la relation réceptrice, l'union avec un *tuple* émetteur est recherchée. Dès qu'une union est détectée, les valeurs de attributs émetteurs sont ajoutés au *tuple* récepteur, qui est réécrit dans le nouveau fichier descriptif de la relation. Enfin, la procédure met à jour le schéma de la relation pour refléter l'ajout de nouveaux attributs temporaires.

```
void XQuestUnion(CCadre* pCadre)
  {
  HCURSOR hcurSave = ::SetCursor(LoadCursor(NULL, IDC_WAIT));

  //--- union
  CSchema* pSchema=pCadre->GetSchema();
  CWind* pWind=pCadre->GetWind();

  int i,j,k,iNothEmetteur,iNothRecepteur;
  int iNoattCleEmetteur,iNoattCleRecepteur;
  int iNbAttributsEmetteur,itabNoattEmetteur[NB_MAX_ATTRIBUTS];
  int iJointure;

  //--- lecture des paramètres
  FILE* f=fopen(base.m_chNomFtmp,"rb");
  fscanf(f,"%d%*c",&iNothEmetteur);
  fscanf(f,"%d%*c",&iNoattCleEmetteur);
  fscanf(f,"%d%*c",&iNbAttributsEmetteur);
```

```
   for(i=0;i < iNbAttributsEmetteur;i++)
      {
      fscanf(f,"%d%*c",&itabNoattEmetteur[i]);
      }
   fscanf(f,"%d%*c",&iNothRecepteur);
   fscanf(f,"%d%*c",&iNoattCleRecepteur);
   fscanf(f,"%d%*c",&iJointure);
   fclose(f);

   //--- traitement
   int iNoFichier=pSchema->FchoixDescriptif();
   if(iNoFichier == -1)
      {
      ::SetCursor(hcurSave);
      return;
      }

   CString strTexte;
   int iNbObjetsRecepteur=pSchema->NombreObjets(iNothRecepteur);

   int tabpRecepteur[NB_MAX_ATTRIBUTS];
   int iNbaRecepteur=pSchema->m_pRelations[iNothRecepteur]->m_iNba;
   for(i=1;i <= iNbaRecepteur;i++)tabpRecepteur[i]=pSchema->m_pRelations[iNothRecepteur]-
>m_pAttributs[i]->m_iNumero;
   int iNovarCleRecepteur=tabpRecepteur[iNoattCleRecepteur];

   int iNbObjetsEmetteur=pSchema->NombreObjets(iNothEmetteur);
   int tabpEmetteur[NB_MAX_ATTRIBUTS];
   int iNbaEmetteur=pSchema->m_pRelations[iNothEmetteur]->m_iNba;
   for(i=1;i <= iNbaEmetteur;i++)tabpEmetteur[i]=pSchema->m_pRelations[iNothEmetteur]-
>m_pAttributs[i]->m_iNumero;
   int iNovarCleEmetteur=tabpEmetteur[iNoattCleEmetteur];

   float vEmetteur[NB_MAX_ATTRIBUTS];
   float vRecepteur[NB_MAX_ATTRIBUTS];
   float fValCleEmetteur,fValCleRecepteur;
   float vprim[NB_MAX_ATTRIBUTS];

   //--- tableaux pour la clé émettrice et les pointeurs dans le fichier émetteur
   float* ftabCle=NULL;
   long* itabPtr=NULL;

   ftabCle=(float *) malloc((iNbObjetsEmetteur + 1)*sizeof(float));
   itabPtr=(long *) malloc((iNbObjetsEmetteur + 1)*sizeof(long));

   //--- lecture de l'émetteur pour remplir les tableaux
   CLecture lectureEmetteur;
   if(lectureEmetteur.Open(pCadre,iNothEmetteur) == FALSE)
      {
      free(ftabCle);
      free(itabPtr);
      ::SetCursor(hcurSave);
      return;
      }
   int iObjet=0;
   j=lectureEmetteur.Lire(vEmetteur);
   while(j != -1)
      {
      if(j == 1)
         {
         iObjet++;
         ftabCle[iObjet]=vEmetteur[iNovarCleEmetteur];
         itabPtr[iObjet]=lectureEmetteur.GetPointeurTuple();
         }
      j=lectureEmetteur.Lire(vEmetteur);
      }
   lectureEmetteur.Close();

   //--- codification des valeurs dans le cas d'une clé de jointure nominale
   int iTypeCleEmetteur=pSchema->m_pRelations[iNothEmetteur]->m_pAttributs[iNoattCleEmetteur]-
>m_iType;
   int iTypeCleRecepteur=pSchema->m_pRelations[iNothRecepteur]-
>m_pAttributs[iNoattCleRecepteur]->m_iType;
   if(iTypeCleEmetteur == 1 || iTypeCleEmetteur == 5)
      {
      int iNbvalbRecepteur=pSchema->m_pRelations[iNothRecepteur]-
>m_pAttributs[iNoattCleRecepteur]->m_iNbvalb;
      int iPtrvalRecepteur=pSchema->m_pRelations[iNothRecepteur]-
>m_pAttributs[iNoattCleRecepteur]->m_iPtrval;
```

```
     int iNbcarAttrRecepteur=pSchema->m_pRelations[iNothRecepteur]-
>m_pAttributs[iNoattCleRecepteur]->m_iNbcar;

     int iNbvalbEmetteur=pSchema->m_pRelations[iNothEmetteur]-
>m_pAttributs[iNoattCleEmetteur]->m_iNbvalb;
     int iPtrvalEmetteur=pSchema->m_pRelations[iNothEmetteur]-
>m_pAttributs[iNoattCleEmetteur]->m_iPtrval;
     int iNbcarAttrEmetteur=pSchema->m_pRelations[iNothEmetteur]-
>m_pAttributs[iNoattCleEmetteur]->m_iNbcar;

     if(iPtrvalEmetteur != iPtrvalRecepteur)
       {
       //--- indexation des modalites de l'attribut de référence
       CIndexAttribut index(pSchema,iNothRecepteur,iNoattCleRecepteur);
       if(index.Init())
         {
         FILE* FileValeurs=NULL;
         if(iTypeCleEmetteur == 1)FileValeurs=fopen(base.m_chNomFichierValeurs,"rb");
         else FileValeurs=fopen(pSchema->m_chNomFtval,"rb");

         if(FileValeurs != NULL)
           {
           //--- recodification
           int iRes,iVal,iPtr;
           char chNomValeur[33];
           for(i=1;i <= iNbObjetsEmetteur;i++)
             {
             iVal=(int) (ftabCle[i] + 0.5);
             if(iVal > 0)
               {
               iPtr=(iPtrvalEmetteur+iVal-2)*iNbcarAttrEmetteur;
               fseek(FileValeurs,iPtr,SEEK_SET);
               fread(chNomValeur,iNbcarAttrEmetteur,1,FileValeurs);
               chNomValeur[iNbcarAttrEmetteur]=0;
               strstrip(chNomValeur);
               iRes=index.Recherche(chNomValeur);
               ftabCle[i]=(float) iRes;
               }
             }

           fclose(FileValeurs);
           }
         index.Fermer();
         }
       }
     }

   //--- union
   int iType;
   float vInconnu[NB_MAX_ATTRIBUTS];
   for(i=0;i < iNbAttributsEmetteur;i++)
     {
     iType=pSchema->m_pRelations[iNothEmetteur]->m_pAttributs[itabNoattEmetteur[i]]->m_iType;
     if(iType == 1 || iType == 5)vInconnu[i]=0.;
     else vInconnu[i]=FINCONNU;
     }

   int iObjetRecepteur=1;
   CLecture lectureRecepteur;
   if(lectureRecepteur.Open(pCadre,iNothRecepteur,iNoFichier,iNbAttributsEmetteur) == FALSE)
     {
     free(ftabCle);
     free(itabPtr);
     ::SetCursor(hcurSave);
     return;
     }
   CLecture lecture2Emetteur;
   if(lecture2Emetteur.Open(pCadre,iNothEmetteur) == FALSE)
     {
     free(ftabCle);
     free(itabPtr);
     ::SetCursor(hcurSave);
     return;
     }
   i=lectureRecepteur.Lire(vRecepteur);
   while(i != -1)
     {
     if(i == 1)
       {
       //--- initialisation du tupe
```

```
for(j=1;j <= iNbaRecepteur;j++)vprim[j]=vRecepteur[tabpRecepteur[j]];
for(j=0;j < iNbAttributsEmetteur;j++)vprim[iNbaRecepteur+1+j]=vInconnu[j];
fValCleRecepteur=vRecepteur[iNovarCleRecepteur];

strTexte.Format("Objet %d sur %d",iObjetRecepteur,iNbObjetsRecepteur);
pMainFrame->SetStatusTexte(strTexte);

if(iTypeCleRecepteur == 1 || iTypeCleRecepteur == 5)
  {
  //--- cas clés de jointure nominales
  if(fValCleRecepteur > 0.)
    {
    j=0;
    iObjet=1;
    while(iObjet <= iNbObjetsEmetteur && j == 0)
      {
      fValCleEmetteur=ftabCle[iObjet];
      if(fValCleEmetteur > 0.)
        {
        switch(iJointure)
          {
          default:
            break;
          case 0: //--- equijointure
            if(fValCleEmetteur == fValCleRecepteur)j=1;
            break;
          case 1: //--- different
            if(fValCleEmetteur != fValCleRecepteur)j=1;
            break;
          }
        if(j == 1)
          {
          //--- lire le tuple emetteur
          lecture2Emetteur.LireTuple(itabPtr[iObjet],vEmetteur);
          for(k=0;k < iNbAttributsEmetteur;k++)
            {
            vprim[iNbaRecepteur+1+k]=vEmetteur[tabpEmetteur[itabNoattEmetteur[k]]];
            }
          }
        }
      iObjet++;
      }
    }
  }
else
  {
  //--- cas clés de jointure numériques
  if(fValCleRecepteur > FINCONNU)
    {
    j=0;
    iObjet=1;
    while(iObjet <= iNbObjetsEmetteur && j == 0)
      {
      fValCleEmetteur=ftabCle[iObjet];
      if(fValCleEmetteur > FINCONNU)
        {
        switch(iJointure)
          {
          default:
            break;
          case 0: //--- equijointure
            if(fValCleEmetteur == fValCleRecepteur)j=1;
            break;
          case 1: //--- different
            if(fValCleEmetteur != fValCleRecepteur)j=1;
            break;
          case 2:
            if(fValCleEmetteur <= fValCleRecepteur)j=1;
            break;
          case 3: //---
            if(fValCleEmetteur >= fValCleRecepteur)j=1;
            break;
          case 4: //---
            if(fValCleEmetteur < fValCleRecepteur)j=1;
            break;
          case 5: //---
            if(fValCleEmetteur > fValCleRecepteur)j=1;
            break;
          }
        if(j == 1)
```

```
                    {
                    //--- lire le tuple emetteur
                    lecture2Emetteur.LireTuple(itabPtr[iObjet],vEmetteur);
                    for(k=0;k < iNbAttributsEmetteur;k++)
                       {
                       vprim[iNbaRecepteur+1+k]=vEmetteur[tabpEmetteur[itabNoattEmetteur[k]]];
                       }
                    }
                 }
              iObjet++;
              }
           }
        }

      //--- écriture des valeurs dans le récepteur
      lectureRecepteur.Ecrire(vprim);
      }
    i=lectureRecepteur.Lire(vRecepteur);
    iObjetRecepteur++;
    }
  lectureRecepteur.Close();
  lecture2Emetteur.Close();

  free(ftabCle);
  free(itabPtr);

  //--- schema
  if(pSchema->m_pRelations[iNothRecepteur]->m_iType < 10)
    {
    pSchema->m_pRelations[iNothRecepteur]->m_iType+=10;
    pSchema->m_pRelations[iNothRecepteur]->m_iNoFichDescriptif=iNoFichier;
    pSchema->m_iFdisDescriptif[iNoFichier]=1;
    for(i=1;i <= pSchema->m_pRelations[iNothRecepteur]->m_iNba;i++)pSchema-
>m_pRelations[iNothRecepteur]->m_pAttributs[i]->m_iNumero=i;
    }
  else
    {
    int iNo=pSchema->m_pRelations[iNothRecepteur]->m_iNoFichDescriptif;
    unlink(pSchema->m_chFprovDescriptif[iNo]);
    pSchema->m_pRelations[iNothRecepteur]->m_iNoFichDescriptif=iNoFichier;
    pSchema->m_iFdisDescriptif[iNoFichier]=1;
    pSchema->m_iFdisDescriptif[iNo]=0;
    }

  //--- creation des nouveaux attributs dans le schema
  int iNbcar,iNoatt,iNbvalb,iPtrval;
  char chNom[20];
  char chDescription[128];
  for(k=0;k < iNbAttributsEmetteur;k++)
    {
    iNoatt=itabNoattEmetteur[k];

    strcpy(chNom,pSchema->m_pRelations[iNothEmetteur]->m_pAttributs[iNoatt]->m_chNom);
    iType=pSchema->m_pRelations[iNothEmetteur]->m_pAttributs[iNoatt]->m_iType;
    iNbcar=pSchema->m_pRelations[iNothEmetteur]->m_pAttributs[iNoatt]->m_iNbcar;
    iNbvalb=pSchema->m_pRelations[iNothEmetteur]->m_pAttributs[iNoatt]->m_iNbvalb;
    iPtrval=pSchema->m_pRelations[iNothEmetteur]->m_pAttributs[iNoatt]->m_iPtrval;
    strcpy(chDescription,"");
    pSchema->NouvelAttribut(iNothRecepteur,iType,iNbcar,iNbvalb,iPtrval,chNom,chDescription);
    }

  ::SetCursor(hcurSave);
  }
```

A.2.5.2. XQuestJoinGeo()

La procédure `XQuestJoinGeo()` crée une nouvelle relation par équi-jointure géométrique, en utilisant les représentations matricielles des deux relations. Elle crée une image matricielle résultant de l'intersection des deux images matricielles des deux relations initiales, crée ensuite les *tuples* correspondant à cette image, à partir des valeurs des *tuples* des deux relations initiales. Enfin, l'image résultante est vectorisée pour créer les arcs correspondants aux zones de la nouvelle relation.

```
void XQuestJoindreGeoEqui(CCadre* pCadre)
```

```
{
HCURSOR hcurSave = ::SetCursor(LoadCursor(NULL, IDC_WAIT));

//--- equijointure géométrique
CSchema* pSchema=pCadre->GetSchema();
CWind* pWind=pCadre->GetWind();

int iNoth1,iNoth2,iIntersection;
char chNomRelation[_MAX_PATH];

//--- lecture des paramètres
FILE* FileMenu=fopen(base.m_chNomFtmp,"rb");
fscanf(FileMenu,"%d%*c",&iNoth1);
fscanf(FileMenu,"%d%*c",&iNoth2);
fscanf(FileMenu,"%d%*c",&iIntersection);
fgets(chNomRelation,_MAX_PATH,FileMenu);
fclose(FileMenu);

chNomRelation[strlen(chNomRelation)-1]='\0';

//--- rasterisation si nécessaire
int iTypeRelation=pSchema->m_pRelations[iNoth1]->m_iType;
if(iTypeRelation == 4 || iTypeRelation == 14)
   {
   CYX cyx(pCadre);
   cyx.Rasteriser(iNoth1);
   }
iTypeRelation=pSchema->m_pRelations[iNoth2]->m_iType;
if(iTypeRelation == 4 || iTypeRelation == 14)
   {
   CYX cyx(pCadre);
   cyx.Rasteriser(iNoth2);
   }

int i,iNo,iNbarc,iPtrarc,nz,iNzNew,iNbpt;
int iNba1,iNba2;
int iNbObjets1,iNbObjets2;
int iCode,iCode1,iCode2;
int iX,iY;
long adresse;
float xc,yc,xb,yb,xh,yh,v1[NB_MAX_ATTRIBUTS],v2[NB_MAX_ATTRIBUTS];
float xcNew,ycNew,xbNew,ybNew,xhNew,yhNew;
float vInconnu1[NB_MAX_ATTRIBUTS],vInconnu2[NB_MAX_ATTRIBUTS];
FILE *FileDescriptif1,*FileDescriptif2,*FileDescriptif3;
FILE *FileImage1,*FileImage2,*FileImage3;
long *ltabObjets1,*ltabObjets2;

//--- indexation des objets de la relation 1
iNbObjets1=pSchema->m_pRelations[iNoth1]->m_iNbObjets;
if(iNbObjets1 == -1)iNbObjets1=pSchema->NombreObjets(iNoth1);
ltabObjets1=(long *) malloc((iNbObjets1+1)*sizeof(long));
if(ltabObjets1 == NULL)
   {
   ::SetCursor(hcurSave);
   ::ErreurDeMemoire();
   return;
   }

iNba1=pSchema->m_pRelations[iNoth1]->m_iNba;
iCode=0;
Clecture lecture1;
if(lecture1.Open(iNoth1))
   {
   int i=lecture1.Lire(v);
   while(i != -1)
      {
      if(i == 1)
         {
         iCode++;
         ltabObjets1[iCode]=lecture1.GetPointeurTuple();
         }
      i=lecture1.Lire(v);
      }
   lecture1.Close();
   }

//--- indexation des objets de la relation 2
iNbObjets2=pSchema->m_pRelations[iNoth2]->m_iNbObjets;
if(iNbObjets2 == -1)iNbObjets2=pSchema->NombreObjets(iNoth2);
ltabObjets2=(long *) malloc((iNbObjets2+1)*sizeof(long));
```

```
if(ltabObjets2 == NULL)
   {
   ::SetCursor(hcurSave);
   ::ErreurDeMemoire();
   free(ltabObjets1);
   return;
   }

iNba2=pSchema->m_pRelations[iNoth2]->m_iNba;

iCode=0;
Clecture lecture2;
if(lecture2.Open(iNoth2))
   {
   int i=lecture2.Lire(v);
   while(i != -1)
      {
      if(i == 1)
         {
         iCode++;
         ltabObjets2[iCode]=lecture2.GetPointeurTuple();
         }
      i=lecture2.Lire(v);
      }
   lecture2.Close();
   }

//--- remplissage des inconnus
int iTypeAttr;
for(i=1; i <= iNba1;i++)
   {
   iTypeAttr=pSchema->m_pRelations[iNoth1]->m_pAttributs[i]->m_iType;
   switch(iTypeAttr)
      {
      case 1:
      case 5:
      case 6:
         vInconnu1[i]=(float) CINCONNU;
         break;
      case 2:
      case 3:
      case 4:
      default:
         vInconnu1[i]=FINCONNU;
         break;
      }
   }

for(i=1; i <= iNba2;i++)
   {
   iTypeAttr=pSchema->m_pRelations[iNoth2]->m_pAttributs[i]->m_iType;
   switch(iTypeAttr)
      {
      case 1:
      case 5:
      case 6:
         vInconnu2[i]=(float) CINCONNU;
         break;
      case 2:
      case 3:
      case 4:
      default:
         vInconnu2[i]=FINCONNU;
         break;
      }
   }

//--- pour la nouvelle relation
int iNoFichierDescriptif=pSchema->FchoixDescriptif();
if(iNoFichierDescriptif == -1)
   {
   free(ltabObjets1);
   free(ltabObjets2);
   return;
   }
int iNoFichierImage=pSchema->FchoixImage();
if(iNoFichierImage == -1)
   {
   free(ltabObjets1);
   free(ltabObjets2);
```

```
    return;
    }

  int *itabImage1=(int *) malloc(config.m_iDefX*(sizeof(int)));
  int *itabImage2=(int *) malloc(config.m_iDefX*(sizeof(int)));
  if(itabImage1 == NULL || itabImage2 == NULL)
    {
    ::ErreurDeMemoire();
    free(ltabObjets1);
    free(ltabObjets2);
    if(itabImage1 != NULL)free(itabImage1);
    if(itabImage2 != NULL)free(itabImage2);
    return;
    }

  int iNo1=pSchema->m_pRelations[iNoth1]->m_iNoFichImage;
  int iNo2=pSchema->m_pRelations[iNoth2]->m_iNoFichImage;
  FileImage1=fopen(pSchema->m_chFprovImage[iNo1],"rb");
  FileImage2=fopen(pSchema->m_chFprovImage[iNo2],"rb");
  FileImage3=fopen(pSchema->m_chFprovImage[iNoFichierImage],"wb");
  if(FileImage1 == NULL || FileImage2 == NULL || FileImage3 == NULL)
    {
    free(itabImage1);
    free(itabImage2);
    free(ltabObjets1);
    free(ltabObjets2);
    if(FileImage1 != NULL)fclose(FileImage1);
    if(FileImage2 != NULL)fclose(FileImage2);
    if(FileImage3 != NULL)fclose(FileImage3);
    return;
    }

  CTableauCode* pTableauCode=new CTableauCode();

  iY=0;
  while(iY < config.m_iDefY)
    {
    iX=0;
    while(iX < config.m_iDefX)
      {
      fscanf(FileImage1,"%d %d%*c",&iCode1,&iNbpt);
      for(i=0;i < iNbpt;i++){itabImage1[iX]=iCode1;iX++;}
      }
    iX=0;
    while(iX < config.m_iDefX)
      {
      fscanf(FileImage2,"%d %d%*c",&iCode2,&iNbpt);
      for(i=0;i < iNbpt;i++){itabImage2[iX]=iCode2;iX++;}
      }
    //--- croisement des images
    iCode1=itabImage1[0];
    iCode2=itabImage2[0];
    iNbpt=1;
    iX=1;
    while(iX < config.m_iDefX)
      {
      while(iX < config.m_iDefX)
        {
        if(itabImage1[iX] != iCode1 || itabImage2[iX] != iCode2)break;
        iX++;
        iNbpt++;
        }
      //--- traiter le segment
      if((iIntersection == TRUE && (iCode1 == 0 || iCode2 == 0)) || (iIntersection == FALSE
&& iCode1 == 0 && iCode2 == 0))iCode=0;
      else iCode=pTableauCode->AjouterCellule(iCode1,iCode2);

      fprintf(FileImage3,"%d %d\n",iCode,iNbpt);

      //--- segment suivant
      iNbpt=0;
      if(iX < config.m_iDefX)
        {
        iCode1=itabImage1[iX];
        iCode2=itabImage2[iX];
        iX++;
        iNbpt=1;
        }
      }
    if(iNbpt > 0)fprintf(FileImage3,"%d %d\n",iCode,iNbpt);
```

```
  iY++;
  }
fclose(FileImage1);
fclose(FileImage2);
fclose(FileImage3);

//--- fichier descriptif
iNo=pSchema->m_pRelations[iNoth1]->m_iNoFichDescriptif;
FileDescriptif1=fopen(pSchema->m_chFprovDescriptif[iNo],"rb");
iNo=pSchema->m_pRelations[iNoth2]->m_iNoFichDescriptif;
FileDescriptif2=fopen(pSchema->m_chFprovDescriptif[iNo],"rb");
FileDescriptif3=fopen(pSchema->m_chFprovDescriptif[iNoFichierDescriptif],"wb");

iNbarc=1;
iPtrarc=1;
fwrite(&iNbarc,SIZEINT,1,FileDescriptif3);
fwrite(&iPtrarc,SIZEINT,1,FileDescriptif3);

iNzNew=20;
xcNew=(pWind->m_fFx1+pWind->m_fFx2)/2.F;
ycNew=(pWind->m_fFy1+pWind->m_fFy2)/2.F;
xbNew=pWind->m_fFx1;
ybNew=pWind->m_fFy1;
xhNew=pWind->m_fFx2;
yhNew=pWind->m_fFy2;
CCelluleCode* ptrCellule=pTableauCode->GetTete();
while(ptrCellule != NULL)
  {
  ptrCellule=pTableauCode->LireCellule(ptrCellule,iCode1,iCode2);

  if(iCode1 > 0)
    {
    fseek(FileDescriptif1,ltabObjets1[iCode1],SEEK_SET);
    for(i=1;i <= iNba1;i++)fread(&v1[i],SIZEVAL,1,FileDescriptif1);
    }
  else
    {
    for(i=1;i <= iNba1;i++)v1[i]=vInconnu1[i];
    }

  if(iCode2 > 0)
    {
    fseek(FileDescriptif2,ltabObjets2[iCode2],SEEK_SET);
    for(i=1;i <= iNba2;i++)fread(&v2[i],SIZEVAL,1,FileDescriptif2);
    }
  else
    {
    for(i=1;i <= iNba2;i++)v2[i]=vInconnu2[i];
    }

  fwrite(&iNzNew,SIZEINT,1,FileDescriptif3);
  fwrite(&xcNew,SIZECOOR,1,FileDescriptif3);
  fwrite(&ycNew,SIZECOOR,1,FileDescriptif3);
  fwrite(&xbNew,SIZECOOR,1,FileDescriptif3);
  fwrite(&ybNew,SIZECOOR,1,FileDescriptif3);
  fwrite(&xhNew,SIZECOOR,1,FileDescriptif3);
  fwrite(&yhNew,SIZECOOR,1,FileDescriptif3);
  for(i=1;i <= iNba1;i++)fwrite(&v1[i],SIZEVAL,1,FileDescriptif3);
  for(i=1;i <= iNba2;i++)fwrite(&v2[i],SIZEVAL,1,FileDescriptif3);

  iNzNew++;
  }
iNzNew=0;
fwrite(&iNzNew,SIZEINT,1,FileDescriptif3);
fwrite(&xcNew,SIZECOOR,1,FileDescriptif3);
fwrite(&ycNew,SIZECOOR,1,FileDescriptif3);
fwrite(&xbNew,SIZECOOR,1,FileDescriptif3);
fwrite(&ybNew,SIZECOOR,1,FileDescriptif3);
fwrite(&xhNew,SIZECOOR,1,FileDescriptif3);
fwrite(&yhNew,SIZECOOR,1,FileDescriptif3);
for(i=1;i <= iNba1;i++)fwrite(&vInconnu1[i],SIZEVAL,1,FileDescriptif3);
for(i=1;i <= iNba2;i++)fwrite(&vInconnu2[i],SIZEVAL,1,FileDescriptif3);

iNbarc=0;
iPtrarc=0;
fwrite(&iNbarc,SIZEINT,1,FileDescriptif3);
fwrite(&iPtrarc,SIZEINT,1,FileDescriptif3);

fclose(FileDescriptif1);
```

```
  fclose(FileDescriptif2);
  fclose(FileDescriptif3);

  delete pTableauCode;
  free(itabImage1);
  free(itabImage2);
  free(ltabObjets1);
  free(ltabObjets2);

  //--- création du schéma
  int iNoFichierFeuille=-1;
  int iNoFichierArc=-1;
  int iNoth3=pSchema-
>NouvelleRelation(4,chNomRelation,iNoFichierFeuille,iNoFichierDescriptif,iNoFichierArc);
  if(iNoth3 > 0)
    {
    pSchema->m_pRelations[iNoth3]->m_iNoFichImage=iNoFichierImage;
    if(iNoFichierImage >= 0)pSchema->m_iFdisImage[iNoFichierImage]=1;
    pSchema->m_pRelations[iNoth3]->m_iType=24;
    //--- attributs
    int iType,iNbcar,iNbvalb,iPtrval;
    char chNomAttribut[20];
    char chDescription[128];
    strcpy(chDescription,"");
    for(i=1;i <= iNba1; i++)
      {
      strcpy(chNomAttribut,pSchema->m_pRelations[iNoth1]->m_pAttributs[i]->m_chNom);
      iNbcar=pSchema->m_pRelations[iNoth1]->m_pAttributs[i]->m_iNbcar;
      iNbvalb=pSchema->m_pRelations[iNoth1]->m_pAttributs[i]->m_iNbvalb;
      iPtrval=pSchema->m_pRelations[iNoth1]->m_pAttributs[i]->m_iPtrval;
      iType=pSchema->m_pRelations[iNoth1]->m_pAttributs[i]->m_iType;
      pSchema-
>NouvelAttribut(iNoth3,iType,iNbcar,iNbvalb,iPtrval,chNomAttribut,chDescription);
      }
    for(i=1;i <= iNba2; i++)
      {
      strcpy(chNomAttribut,pSchema->m_pRelations[iNoth2]->m_pAttributs[i]->m_chNom);
      iNbcar=pSchema->m_pRelations[iNoth2]->m_pAttributs[i]->m_iNbcar;
      iNbvalb=pSchema->m_pRelations[iNoth2]->m_pAttributs[i]->m_iNbvalb;
      iPtrval=pSchema->m_pRelations[iNoth2]->m_pAttributs[i]->m_iPtrval;
      iType=pSchema->m_pRelations[iNoth2]->m_pAttributs[i]->m_iType;
      pSchema-
>NouvelAttribut(iNoth3,iType,iNbcar,iNbvalb,iPtrval,chNomAttribut,chDescription);
      }

    //--- vectorisation et recupération du nombre d'arcs
    CVectoriser vectoriser(pCadre,TRUE);
    vectoriser.Relation(iNoth3);
    iNbarc=vectoriser.GetNbarc();

    FileDescriptif3=fopen(pSchema->m_chFprovDescriptif[iNoFichierDescriptif],"rb+");
    if(FileDescriptif3 != NULL)
      {
      fwrite(&iNbarc,SIZEINT,1,FileDescriptif3);
      fclose(FileDescriptif3);
      }
    //--- calcul des centroïdes et fenêtres de zones
    pSchema->m_pRelations[iNoth3]->CalculDesCentroides();
    }

  ::SetCursor(hcurSave);
  }
```

A.2.5.3. XCrisCalcul()

Le code présenté ici est une partie de l'implémentation de la procédure générale de création d'un attribut par calcul. Il s'agit d'une opération interne à une relation, comme toutes les opérations du menu CRIS. Le code montre les étapes importantes de cette procédures : création du tableau d'indirection schéma interne-schéma externe, lecture des objets (avec la classe `CLecture`) et utilisation de la classe `CCalculateur` pour déterminer la valeur du nouvel attribut à partir des attributs de l'objet, mise à jour du schéma de la relation avec création d'un nouvel attribut.

```
//--- tableau d'indirection schéma interne-schéma externe
     int tabp[NB_MAX_ATTRIBUTS];
     int iNba=pSchema->m_pRelations[iNoth]->m_iNba;
     for(i=1;i <= iNba;i++)tabp[i]=pSchema->m_pRelations[iNoth]->m_pAttributs[i]->m_iNumero;

     int iNoFichier=pSchema->FchoixDescriptif();
     if(iNoFichier == -1)return;

     CLecture lecture;
     if(lecture.Open(pCadre,iNoth,iNoFichier,1))
       {
       //--- lecture des objets
       double v[NB_MAX_ATTRIBUTS],fResultat;
       double vprim[NB_MAX_ATTRIBUTS];

       i=lecture.Lire(v);
       while(i != -1)
         {
         if(i == 1)
           {
           dResultat=calculateur.Calcul(chFormule,v);
           iNbObjets++;

           if(pSchema->m_pRelations[iNoth]->m_iType < 10)
             {
             for(i=1;i <= iNba;i++)vprim[i]=v[tabp[i]];
             vprim[iNba+1]=fResultat;
             lecture.Ecrire(vprim);
             }
           else
             {
             v[iNba+1]=fResultat;
             lecture.Ecrire(v);
             }
           }
         i=lecture.Lire(v);
         }
       lecture.Close();

       //--- mise a jour de la relation
       if(pSchema->m_pRelations[iNoth]->m_iType < 10)
         {
         pSchema->m_pRelations[iNoth]->m_iType+=10;
         pSchema->m_pRelations[iNoth]->m_iNoFichDescriptif=iNoFichier;
         pSchema->m_iFdisDescriptif[iNoFichier]=1;
         for(i=1;i <= pSchema->m_pRelations[iNoth]->m_iNba;i++)pSchema->m_pRelations[iNoth]-
>m_pAttributs[i]->m_iNumero=i;
         }
       else
         {
         int iNo=pSchema->m_pRelations[iNoth]->m_iNoFichDescriptif;
         unlink(pSchema->m_chFprovDescriptif[iNo]);
         pSchema->m_pRelations[iNoth]->m_iNoFichDescriptif=iNoFichier;
         pSchema->m_iFdisDescriptif[iNoFichier]=1;
         pSchema->m_iFdisDescriptif[iNo]=0;
         }

       //--- creation du nouvel attribut dans le schema
       pSchema->NouvelAttribut(iNoth,4,0,0,0,chNomAttribut,chDescription);

       iOperationOk=1;
       }
```

A.2.5.4. XTri()

La procédure Xtri() est utilisée à plusieurs reprises dans SAVANE pour trier les valeurs d'un attribut d'une relation (par exemple, dans la procédure `XCrisTri()`, dans la procédure de détermination de quantiles, dans le calcul de la médiane, etc.). Elle utilise la classe `CLecture` pour lire la valeur des objets de la relation, et la procédure `qsort()` de la librairie standard C++ pour trier les valeurs.

```
void XTri(CCadre* pCadre,int iNoth, int iNoatt,float *fTableau)
  {
  //--- tri des valeurs d'un attribut, relations non image
```

```
  CSchema* pSchema=pCadre->GetSchema();
  if(pSchema->m_pRelations[iNoth]->m_iType != 5 && pSchema->m_pRelations[iNoth]->m_iType != 15
&& pSchema->m_pRelations[iNoth]->m_iType != 25)
    {
    //--- initialisation de la lecture
    float v[NB_MAX_ATTRIBUTS];
    int i;
    int iNovar=pSchema->m_pRelations[iNoth]->m_pAttributs[iNoatt]->m_iNumero;

    CLecture lecture;
    if(lecture.Open(pCadre,iNoth))
      {
      int iNbObjets=0;

      i=lecture.Lire(v);
      while(i != -1)
        {
        if(i == 1)
          {
          if(v[iNovar] > FINCONNU)
            {
            fTableau[iNbObjets]=v[iNovar];
            iNbObjets++;
            }
          }
        i=lecture.Lire(v);
        }
      lecture.Close();

      qsort(fTableau,iNbObjets,sizeof(float),comparnum);
      }
    }
  }

int comparnum(const void *arg1,const void *arg2)
  {
  if(*((float *) arg1) < *((float *) arg2))return -1;
  if(*((float *) arg1) > *((float *) arg2))return 1;
  return 0;
  }
```

A.3. Savamer

A.3.1. La classe CAmers

La classe `CAmers` de SAVAMER gère les couples de points saisis entre image à redresser et espace géographique. Elle est utilisée dans de nombreuses autres classes du programme. On peut saisir jusqu'à 10 000 amers pour une image.

```
class CAmers
  {
  friend class CTranslation;
  friend class CSimilitude;
  friend class CSysteme1;
  friend class CSysteme3;
  friend class CTessel;

  private:
    double m_dtabXDiff[NB_MAX_AMER];
    double m_dtabYDiff[NB_MAX_AMER];
    //--- coordonnées dans la vue de la base
    int m_itabXVue[NB_MAX_AMER];
    int m_itabYVue[NB_MAX_AMER];

    //--- coordonnées dans la base savane
    double m_dtabXSav[NB_MAX_AMER];
    double m_dtabYSav[NB_MAX_AMER];
    //--- coordonnées image
    int m_itabXImage[NB_MAX_AMER];
    int m_itabYImage[NB_MAX_AMER];

    int m_iNbpt;
```

```
    int m_iSelection;
    BOOL m_bASauver;
    char m_chNomFichier[_MAX_PATH];

  public:
    CAmers();

    void Init();
    BOOL Lire(char *chNomFichierAmers);
    void Ecrire();
    BOOL Fermer();

    void SavToVue();
    void Tracer(int iTarget,CDC* pDc);
    void TracerLesLiens(int iTarget,CDC* pDc);
    void TracerUnPoint(int iTarget,CDC* pDc,int iNo,COLORREF couleur);
    void Supprimer(int iNo);
    void AjouterSavane(double dXSav,double dYSav,int iXImage,int iYImage);
    void AjouterProjection(double dXProj,double dYProj,int iXImage,int iYImage);
    int Selectionner(int iTarget,CPoint point);
    int SelectionnerParNumero(int iNo);
    int GetSelection() {return m_iSelection;};
    void SetPointVue(int iNo,int iXVue,int iYVue,int iXImage,int iYImage);
    void SetPointProjection(int iNo,double dXProj,double dYProj,int iXImage,int iYImage);
    void GetPointVue(int iNo,int &iXVue,int &iYVue,int &iXImage,int &iYImage);
    void GetPointSavane(int iNo,double &dXSav,double &dYSav,int &iXImage,int &iYImage);
    void GetPointProjection(int iNo,double &dXProj,double &dYproj,int &iXImage,int &iYImage);
    int GetNbpt() {return m_iNbpt;};
    void ASauver() {m_bASauver=TRUE;};
    BOOL IsASauver() {return m_bASauver;};
    void Echanger(int,int);

    BOOL Fenetre(double &dXMin,double &dYMin,double &dXMax,double &dYMax);
  };
```

A.3.2. La classe CSysteme1

La classe Csystème1 permet d'effectuer un redressement polynomial avec trois points d'amers.

```
class CSysteme1
  {
  private:
    double m_dResolution;
    CAmers* m_pAmers;
    CImage* m_pImage;

    double m_aPToI;
    double m_bPToI;
    double m_cPToI;
    double m_aprimPToI;
    double m_bprimPToI;
    double m_cprimPToI;

    double m_aIToP;
    double m_bIToP;
    double m_cIToP;
    double m_aprimIToP;
    double m_bprimIToP;
    double m_cprimIToP;

    //--- fenetre d'arrivee, en projection
    double m_dFx1;
    double m_dFy1;
    double m_dFx2;
    double m_dFy2;
    int m_iNbligDestination;
    int m_iNbcolDestination;


  public:
    CSysteme1(CAmers* pAmers,CImage* pImage,double dResolution);

    BOOL Equations();
    void ImageToProjection(int iX,int iY,double &dX,double &dY);
```

```
    void ProjectionToImage(double dX,double dY, int &iX,int &iY);
    void ProjectionToImage(double dX,double dY,double &dXImage,double &dYImage);
    void FenetreDestination();
    void Recalage(int iModeRecalage);
    void RecalageAmers();

    double GetResolution() {return m_dResolution;};
    void GetFenetreDestination(double &dXb,double &dYb,double &dXh,double &dYh,int
&iNbcol,int &iNblig);
  };
```

La méthode `Equations()` de la classe calcule, à partir des trois premiers points d'amers, les coefficients des formules de calculs. Les méthodes `ProjectionToImage()` et `ImageToProjection()` effectuent les calculs de changement de repère :

```
BOOL CSysteme1::Equations()
  {
  double dX,dY,dX0,dY0;

  double dXi0=(double) m_pAmers->m_itabXImage[0];
  double dYi0=(double) m_pAmers->m_itabYImage[0];
  double dXi1=(double) m_pAmers->m_itabXImage[1];
  double dYi1=(double) m_pAmers->m_itabYImage[1];
  double dXi2=(double) m_pAmers->m_itabXImage[2];
  double dYi2=(double) m_pAmers->m_itabYImage[2];

  ::SavToProj(m_pAmers->m_dtabXSav[0],m_pAmers->m_dtabYSav[0],dX0,dY0);

  ::SavToProj(m_pAmers->m_dtabXSav[0],m_pAmers->m_dtabYSav[0],dX,dY);
  double dXp0=dX-dX0;
  double dYp0=dY-dY0;

  ::SavToProj(m_pAmers->m_dtabXSav[1],m_pAmers->m_dtabYSav[1],dX,dY);
  double dXp1=dX-dX0;
  double dYp1=dY-dY0;

  ::SavToProj(m_pAmers->m_dtabXSav[2],m_pAmers->m_dtabYSav[2],dX,dY);
  double dXp2=dX-dX0;
  double dYp2=dY-dY0;

  double dDelta=(dXi1-dXi0)*(dYi2-dYi0)-(dXi2-dXi0)*(dYi1-dYi0);
  if(dDelta == 0)return FALSE;

  m_aIToP=((dXp1-dXp0)*(dYi2-dYi0)-(dXp2-dXp0)*(dYi1-dYi0))/dDelta;
  m_bIToP=((dXp2-dXp0)*(dXi1-dXi0)-(dXp1-dXp0)*(dXi2-dXi0))/dDelta;
  m_cIToP=dXp0-m_aIToP*dXi0-m_bIToP*dYi0;

  m_aprimIToP=((dYp1-dYp0)*(dYi2-dYi0)-(dYp2-dYp0)*(dYi1-dYi0))/dDelta;
  m_bprimIToP=((dYp2-dYp0)*(dXi1-dXi0)-(dYp1-dYp0)*(dXi2-dXi0))/dDelta;
  m_cprimIToP=dYp0-m_aprimIToP*dXi0-m_bprimIToP*dYi0;

  dDelta=(dXp1-dXp0)*(dYp2-dYp0)-(dXp2-dXp0)*(dYp1-dYp0);
  if(dDelta == 0)return FALSE;

  m_aPToI=((dXi1-dXi0)*(dYp2-dYp0)-(dXi2-dXi0)*(dYp1-dYp0))/dDelta;
  m_bPToI=((dXi2-dXi0)*(dXp1-dXp0)-(dXi1-dXi0)*(dXp2-dXp0))/dDelta;
  m_cPToI=dXi0-m_aPToI*dXp0-m_bPToI*dYp0;

  m_aprimPToI=((dYi1-dYi0)*(dYp2-dYp0)-(dYi2-dYi0)*(dYp1-dYp0))/dDelta;
  m_bprimPToI=((dYi2-dYi0)*(dXp1-dXp0)-(dYi1-dYi0)*(dXp2-dXp0))/dDelta;
  m_cprimPToI=dYi0-m_aprimPToI*dXp0-m_bprimPToI*dYp0;

  return TRUE;
  }

void CSysteme1::ImageToProjection(int iX,int iY,double &dXProj,double &dYProj)
  {
  dXProj=m_aIToP*((double) iX) + m_bIToP*((double) iY) + m_cIToP;
  dYProj=m_aprimIToP*((double) iX) + m_bprimIToP*((double) iY) + m_cprimIToP;
  }

void CSysteme1::ProjectionToImage(double dXProj,double dYProj,int &iX,int &iY)
  {
  iX=(int) (m_aPToI*dXProj + m_bPToI*dYProj + m_cPToI);
```

```
  iY=(int) (m_aprimPToI*dXProj + m_bprimPToI*dYProj + m_cprimPToI);
  }
```

Le calcul de la fenêtre de redressement est effectué par la fonction `FenetreDestination()` :

```
void CSysteme1::FenetreDestination()
  {
  double dX,dY;
  ImageToProjection(0,0,m_dFx1,m_dFy1);
  m_dFx2=m_dFx1;
  m_dFy2=m_dFy1;
  ImageToProjection(m_pImage->m_iNbcol,0,dX,dY);
  if(dX < m_dFx1)m_dFx1=dX;
  if(dY < m_dFy1)m_dFy1=dY;
  if(dX > m_dFx2)m_dFx2=dX;
  if(dY > m_dFy2)m_dFy2=dY;
  ImageToProjection(m_pImage->m_iNbcol,m_pImage->m_iNblig,dX,dY);
  if(dX < m_dFx1)m_dFx1=dX;
  if(dY < m_dFy1)m_dFy1=dY;
  if(dX > m_dFx2)m_dFx2=dX;
  if(dY > m_dFy2)m_dFy2=dY;
  ImageToProjection(0,m_pImage->m_iNblig,dX,dY);
  if(dX < m_dFx1)m_dFx1=dX;
  if(dY < m_dFy1)m_dFy1=dY;
  if(dX > m_dFx2)m_dFx2=dX;
  if(dY > m_dFy2)m_dFy2=dY;

  double dXProj1,dYProj1,dXProj2,dYProj2;

  m_pAmers->Fenetre(dXProj1,dYProj1,dXProj2,dYProj2);
  if(dXProj1 < m_dFx1)m_dFx1=dXProj1;
  if(dYProj1 < m_dFy1)m_dFy1=dYProj1;
  if(dXProj2 > m_dFx2)m_dFx2=dXProj2;
  if(dYProj2 > m_dFy2)m_dFy2=dYProj2;
  //--- aligner la fenêtre de recalage sur la résolution des pixels
  double dXProj0,dYProj0;
  ::SavToProj(m_pAmers->m_dtabXSav[0],m_pAmers->m_dtabYSav[0],dXProj0,dYProj0);

  double dXb=m_dFx1+dXProj0;
  double dYb=m_dFy1+dYProj0;
  double dXh=m_dFx2+dXProj0;
  double dYh=m_dFy2+dYProj0;

  dXb=((long) (dXb/m_dResolution))*m_dResolution;
  dYb=((long) (dYb/m_dResolution))*m_dResolution;
  dXh=((long) (dXh/m_dResolution)+1)*m_dResolution;
  dYh=((long) (dYh/m_dResolution)+1)*m_dResolution;

  m_dFx1=dXb-dXProj0;
  m_dFy1=dYb-dYProj0;
  m_dFx2=dXh-dXProj0;
  m_dFy2=dYh-dYProj0;

  m_iNbligDestination=(int) ((m_dFy2-m_dFy1)/m_dResolution);
  m_iNbcolDestination=(int) ((m_dFx2-m_dFx1)/m_dResolution);
  int iReste=4-(m_iNbcolDestination % 4);
  if(iReste != 4)m_iNbcolDestination+=iReste;

  m_dFx2=m_dFx1 + m_iNbcolDestination*m_dResolution;
  }
```

La méthode `Recalage()` effectue à la fois le redressement et le ré-échantillonage. Par exemple, pour un ré-échantillonnage par plus proche voisin, le calcul est le suivant :

```
        //--- recalage plus proche voisin
        ilig=0;
        dY=m_dFy1+(m_dResolution/2.);
        while(ilig < m_iNbligDestination)
           {
           icol=0;
           dX=m_dFx1+(m_dResolution/2.);
           while(icol < m_iNbcolDestination)
              {
```

```
            ProjectionToImage(dX,dY,iX,iY);
            if(iX >= 0 && iX < m_pImage->m_iNbcol && iY >= 0 && iY < m_pImage->m_iNblig)
               buffer[icol]=m_pImage->m_bPixels[(iY*m_pImage->m_iNbcol)+iX];
            else buffer[icol]=0;

            dX+=m_dResolution;
            icol++;
            }
          fwrite(buffer,1,m_iNbcolDestination,FileBmp);

          dY+=m_dResolution;
          ilig++;
          }
```

```
ilig=0;
dY=m_dFy1+(m_dResolution/2.);
while(ilig < m_iNbligDestination)
   {
   icol=0;
   dX=m_dFx1+(m_dResolution/2.);
   while(icol < m_iNbcolDestination)
      {
      ProjectionToImage(dX,dY,dXImage,dYImage);
      alfa0=dXImage-floor(dXImage);
      alfa1=1.-alfa0;
      beta0=dYImage-floor(dYImage);
      beta1=1.-beta0;
      iX=(int) dXImage;
      iY=(int) dYImage;

      if(iX >= 0 && iX < m_pImage->m_iNbcol-1 && iY >= 0 && iY < m_pImage->m_iNblig-1)
         {
         dVal00=(double) m_pImage->m_bPixels[(iY*m_pImage->m_iNbcol)+iX];
         dVal01=(double) m_pImage->m_bPixels[(iY*m_pImage->m_iNbcol)+iX+1];
         dVal10=(double) m_pImage->m_bPixels[((iY+1)*m_pImage->m_iNbcol)+iX];
         dVal11=(double) m_pImage->m_bPixels[((iY+1)*m_pImage->m_iNbcol)+iX+1];

         buffer[icol]=(int)
(beta0*alfa0*dVal00+beta0*alfa1*dVal01+beta1*alfa0*dVal10+beta1*alfa1*dVal11);
         }
      else buffer[icol]=0;
      dX+=m_dResolution;
      icol++;
      }
   fwrite(buffer,1,m_iNbcolDestination,FileBmp);

   dY+=m_dResolution;
   ilig++;
   }
```

Le calcul bicubique calcule la valeur en fonction de la position du point sur une fenêtre de 4*4 voisins :

```
ilig=0;
dY=m_dFy1+(m_dResolution/2.);
while(ilig < m_iNbligDestination)
   {
   icol=0;
   dX=m_dFx1+(m_dResolution/2.);
   while(icol < m_iNbcolDestination)
      {
      ProjectionToImage(dX,dY,dXImage,dYImage);
      iX=(int) dXImage;
      iY=(int) dYImage;

      if(iX >= 1 && iX < m_pImage->m_iNbcol-2 && iY >= 1 && iY < m_pImage->m_iNblig-2)
         {
         alfa1=dXImage-floor(dXImage);
         alfa0=1.+alfa1;
         alfa2=1.-alfa1;
         alfa3=2.-alfa1;
         beta1=dYImage-floor(dYImage);
         beta0=1.+beta1;
         beta2=1.-beta1;
```

```
      beta3=2.-beta1;

      CX0=alfa0*(-alfa0*alfa0 + 5.*alfa0 - 8.) + 4.;
      CX1=alfa1*alfa1*(alfa1 - 2.) + 1.;
      CX2=alfa2*alfa2*(alfa2 - 2.) + 1.;
      CX3=alfa3*(-alfa3*alfa3 + 5*alfa3 - 8.) + 4.;
      CY0=beta0*(-beta0*beta0 + 5*beta0 - 8.) + 4.;
      CY1=beta1*beta1*(beta1 - 2.) + 1.;
      CY2=beta2*beta2*(beta2 - 2.) + 1.;
      CY3=beta3*(-beta3*beta3 + 5.*beta3 - 8.) + 4.;

      iIndice=(iY-1)*m_pImage->m_iNbcol+iX;
      dVal00=(double) m_pImage->m_bPixels[iIndice-1];
      dVal01=(double) m_pImage->m_bPixels[iIndice];
      dVal02=(double) m_pImage->m_bPixels[iIndice+1];
      dVal03=(double) m_pImage->m_bPixels[iIndice+2];

      iIndice=iY*m_pImage->m_iNbcol+iX;
      dVal10=(double) m_pImage->m_bPixels[iIndice-1];
      dVal11=(double) m_pImage->m_bPixels[iIndice];
      dVal12=(double) m_pImage->m_bPixels[iIndice+1];
      dVal13=(double) m_pImage->m_bPixels[iIndice+2];

      iIndice=(iY+1)*m_pImage->m_iNbcol+iX;
      dVal20=(double) m_pImage->m_bPixels[iIndice-1];
      dVal21=(double) m_pImage->m_bPixels[iIndice];
      dVal22=(double) m_pImage->m_bPixels[iIndice+1];
      dVal23=(double) m_pImage->m_bPixels[iIndice+2];

      iIndice=(iY+2)*m_pImage->m_iNbcol+iX;
      dVal30=(double) m_pImage->m_bPixels[iIndice-1];
      dVal31=(double) m_pImage->m_bPixels[iIndice];
      dVal32=(double) m_pImage->m_bPixels[iIndice+1];
      dVal33=(double) m_pImage->m_bPixels[iIndice+2];

      dVal=CY0*(CX0*dVal00 + CX1*dVal01 + CX2*dVal02 + CX3*dVal03);
      dVal+=CY1*(CX0*dVal10 + CX1*dVal11 + CX2*dVal12 + CX3*dVal13);
      dVal+=CY2*(CX0*dVal20 + CX1*dVal21 + CX2*dVal22 + CX3*dVal23);
      dVal+=CY3*(CX0*dVal30 + CX1*dVal31 + CX2*dVal32 + CX3*dVal33);

      buffer[icol]=(int) dVal;
      }
    else buffer[icol]=0;

    dX+=m_dResolution;
    icol++;
    }
  fwrite(buffer,1,m_iNbcolDestination,FileBmp);

  dY+=m_dResolution;
  ilig++;
  }
```

A.3.3. La classe CTessel

La classe `CTessel` permet d'effectuer le calcul de la triangulation de Delaunay à partir des points d'amers, et le redressement polynomail de degré 1 dans chacun des triangles. La triangulation de Delaunay est calculée par insertion successive des points d'amer. Le redressement utilise une image rasterisée de la triangulation pour déterminer le triangle auquel appartient le point à redresser, et déterminer ainsi les coefficients du redressement de degré 1.

```
class CTessel
  {
  friend class CYX;

  private:
    double m_dResolution;
    CAmers* m_pAmers;
    CImage* m_pImage;
```

```
    CSimilitude *m_pSimilitude;
    CSysteme1 *m_pSysteme1;
    int m_iRecalageInitial;

    //--- fenetre d'arrivee, en projection
    double m_dFx1;
    double m_dFy1;
    double m_dFx2;
    double m_dFy2;
    double m_dXProj0;
    double m_dYProj0;
    int m_iNbligDestination;
    int m_iNbcolDestination;

    //--- structures
    NOEUD *m_TableNoeuds;
    int m_iNombredeNoeuds;
    int m_iNumeroTriangle;

    //--- fonctions locales de tesselation
    void GenerationTriangle(FILE *FileArc,FILE *FileDat,int ,int ,int );
    void InitTriangulation(int *triangle,double flxb,double flyb,double flxh,double flyh);
    int InsertionPoint(POINTR2 point,TRIANGLE *triangle );

    void supprimerliste(LISTE *tete) ;
    LISTE *init_liste();
    void suppression(LISTE *tete, int indice);
    int prede_succ(LISTE *tete, int indice, int *predecesseur, int *successeur) ;
    int succ(LISTE *tete, int indice) ;
    LISTE *insertion(LISTE *tete, LISTE *noeud, int indice);
    void init_pile();
    void supprimepile();
    void empiler(int a, int b, int c, int d)     ;
    void depiler(int *a, int *b, int *c, int *d);
    int pile_vide() ;
    void insertion_liste_ordonnee(int a, int b, int c);
    int chercher_le_triangle(int *a, int *b, int *c, POINTR2 point);
    int cherche_point_oppose(int a, int b, int c);
    int adjacent(int a, int b, int c);
    int est_un_triangle(int a, int b, int c);
    void creation_triangle(int a, int b, int c);
    void del_triangle(int a, int b, int c);
    int split_triangle(int a, int b, int c, POINTR2 point, TRIANGLE *triangle);
    void split3_triangle(int a, int b, int c, int d, TRIANGLE *triangle);
    void split4_triangle(int a, int b, int c, int d, int e, TRIANGLE *triangle);
    int swap(int a, int b, int c, int d);
    void split_quad(int a, int b, int c, int d, TRIANGLE *triangle);
    int position_point_segment(POINTR2 p1, POINTR2 p2, POINTR2 p0s, POINTR2 p1s);
    double pseudo_angle(POINTR2 p1, POINTR2 p2);
    double angle_point(POINTR2 a, POINTR2 b, POINTR2 c);
    int point_non_alignes(POINTR2 p1, POINTR2 p2, POINTR2 p3);
    int quadrilatere_strictement_convexe(POINTR2 pa, POINTR2 pb, POINTR2 pc, POINTR2 pd);
    int compare_point(POINTR2 p1, POINTR2 p2);
    int modulo(int i, int j, int k);
    void copier_point(POINTR2 p1, POINTR2 *p2);

  public:
    CTessel(CAmers* pAmers);
    CTessel(CAmers* pAmers,CImage* pImage,CSimilitude* pSimilitude);
    CTessel(CAmers* pAmers,CImage* pImage,CSysteme1* pSysteme1);
    ~CTessel();
    int Tesselation();
    int TesselationSansRecalage();
    void Raster();
    void Recalage(int iNbTriangles,int iModeRecalage);
  };
```

Voici une partie du code de la procédure de redressement par triangulation :

```
void CTessel::Recalage(int iNbTriangle,int iModeRecalage)
  {
  char chNomFichierDat[_MAX_PATH];
  strcpy(chNomFichierDat,schema.m_chDirUser);
  strcat(chNomFichierDat,"\\ftdat");
  char chNomFichierTriangle[_MAX_PATH];
  strcpy(chNomFichierTriangle,schema.m_chDirUser);
  strcat(chNomFichierTriangle,"\\ftImageTriangle");

  char chNomFichier[_MAX_PATH];
```

```
  char chNomFichierDepl[_MAX_PATH];
  FILE *FileBmp,*FileDepl;

  strcpy(chNomFichier,m_pImage->m_chNomFichier);
  strcat(chNomFichier,"R.bmp");
  strcpy(chNomFichierDepl,m_pImage->m_chNomFichier);
  strcat(chNomFichierDepl,"RDiff.bmp");

  if((FileBmp=fopen(chNomFichier,"wb")) != NULL  && (FileDepl=fopen(chNomFichierDepl,"wb")) !=
NULL)
    {
    //--- fichier déplacement
    BITMAPFILEHEADER BitmapFileHeaderDepl;
    BITMAPINFOHEADER BitmapInfoHeaderDepl;
    RGBQUAD PaletteEntriesDepl[256];

    //--- BITMAPFILEHEADER
    char bm[3];
    strcpy(bm,"BM");
    memcpy(&BitmapFileHeaderDepl.bfType,bm,2);

    int iSize=sizeof(BITMAPFILEHEADER)+sizeof(BITMAPINFOHEADER)+sizeof(PaletteEntriesDepl);
    iSize+=(m_iNbligDestination*m_iNbcolDestination);
    BitmapFileHeaderDepl.bfSize=iSize;
    BitmapFileHeaderDepl.bfReserved1=0;
    BitmapFileHeaderDepl.bfReserved2=0;
    BitmapFileHeaderDepl.bfOffBits=sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER) +
sizeof(PaletteEntriesDepl);

    //--- BITMAPINFOHEADER
    BitmapInfoHeaderDepl.biSize=(DWORD) sizeof(BITMAPINFOHEADER);
    BitmapInfoHeaderDepl.biWidth=m_iNbcolDestination;
    BitmapInfoHeaderDepl.biHeight=m_iNbligDestination;
    BitmapInfoHeaderDepl.biPlanes=(WORD) 1;
    BitmapInfoHeaderDepl.biBitCount=(WORD) 8;
    BitmapInfoHeaderDepl.biCompression=BI_RGB;
    BitmapInfoHeaderDepl.biSizeImage=(DWORD) (m_iNbligDestination*m_iNbcolDestination);
    BitmapInfoHeaderDepl.biXPelsPerMeter=(LONG) m_dResolution;
    BitmapInfoHeaderDepl.biYPelsPerMeter=(LONG) m_dResolution;
    BitmapInfoHeaderDepl.biClrUsed=(DWORD) 0;
    BitmapInfoHeaderDepl.biClrImportant=(DWORD) 0;

    for(int i=0;i < 128;i++)
      {
      PaletteEntriesDepl[i].rgbBlue= (BYTE) i*2;
      PaletteEntriesDepl[i].rgbGreen= (BYTE) i*2;
      PaletteEntriesDepl[i].rgbRed= (BYTE) i*2;
      PaletteEntriesDepl[i].rgbReserved= (BYTE) 0;
      }
    for(i=128;i < 256;i++)
      {
      PaletteEntriesDepl[i].rgbBlue= (BYTE) 255;
      PaletteEntriesDepl[i].rgbGreen= (BYTE) 255;
      PaletteEntriesDepl[i].rgbRed= (BYTE) 255;
      PaletteEntriesDepl[i].rgbReserved= (BYTE) 0;
      }

    fwrite(&BitmapFileHeaderDepl,sizeof(BITMAPFILEHEADER),1,FileDepl);
    fwrite(&BitmapInfoHeaderDepl,sizeof(BITMAPINFOHEADER),1,FileDepl);
    fwrite(&PaletteEntriesDepl,sizeof(PaletteEntriesDepl),1,FileDepl);

    //--- creation du tableau des coefficients
    int iCode;
    double *alpha;
    double *beta;
    double *gamma;
    double *alphaprim;
    double *betaprim;
    double *gammaprim;
    unsigned char *bufferDiff;
    int *itabImage;

    alpha=(double *) malloc(iNbTriangle*sizeof(double));
    beta=(double *) malloc(iNbTriangle*sizeof(double));
    gamma=(double *) malloc(iNbTriangle*sizeof(double));

    if(alpha == NULL || beta == NULL || gamma == NULL)
      {
      ::ErreurDeMemoire();
      if(alpha != NULL)free(alpha);
```

```
   if(beta != NULL)free(beta);
   if(gamma != NULL)free(gamma);

   fclose(FileBmp);
   fclose(FileDepl);
   unlink(chNomFichier);
   unlink(chNomFichierDepl);
   unlink(chNomFichierTriangle);
   unlink(chNomFichierDat);
   return;
   }

alphaprim=(double *) malloc(iNbTriangle*sizeof(double));
betaprim=(double *) malloc(iNbTriangle*sizeof(double));
gammaprim=(double *) malloc(iNbTriangle*sizeof(double));

if(alphaprim == NULL || betaprim == NULL || gammaprim == NULL)
   {
   ::ErreurDeMemoire();
   if(alphaprim != NULL)free(alphaprim);
   if(betaprim != NULL)free(betaprim);
   if(gammaprim != NULL)free(gammaprim);

   free(alpha);
   free(beta);
   free(gamma);
   fclose(FileBmp);
   fclose(FileDepl);
   unlink(chNomFichier);
   unlink(chNomFichierDepl);
   unlink(chNomFichierTriangle);
   unlink(chNomFichierDat);
   return;
   }

bufferDiff=(unsigned char *) malloc(m_iNbcolDestination*sizeof(unsigned char));
itabImage=(int *) malloc(m_iNbcolDestination*sizeof(int));
if(bufferDiff == NULL || itabImage == NULL)
   {
   ::ErreurDeMemoire();
   if(bufferDiff != NULL)free(bufferDiff);
   if(itabImage != NULL)free(itabImage);

   free(alpha);
   free(beta);
   free(gamma);
   free(alphaprim);
   free(betaprim);
   free(gammaprim);
   fclose(FileBmp);
   fclose(FileDepl);
   unlink(chNomFichier);
   unlink(chNomFichierDepl);
   unlink(chNomFichierTriangle);
   unlink(chNomFichierDat);
   return;
   }

FILE* FileDat=fopen(chNomFichierDat,"rb");
if(FileDat == NULL)
   {
   free(bufferDiff);
   free(itabImage);
   free(alpha);
   free(beta);
   free(gamma);
   free(alphaprim);
   free(betaprim);
   free(gammaprim);
   fclose(FileBmp);
   fclose(FileDepl);
   unlink(chNomFichier);
   unlink(chNomFichierDepl);
   unlink(chNomFichierTriangle);
   }

fread(&iCode,sizeof(int),1,FileDat);
while(!feof(FileDat))
   {
   fread(&alpha[iCode],sizeof(double),1,FileDat);
```

```
        fread(&beta[iCode],sizeof(double),1,FileDat);
        fread(&gamma[iCode],sizeof(double),1,FileDat);
        fread(&alphaprim[iCode],sizeof(double),1,FileDat);
        fread(&betaprim[iCode],sizeof(double),1,FileDat);
        fread(&gammaprim[iCode],sizeof(double),1,FileDat);

        fread(&iCode,sizeof(int),1,FileDat);
        }

     fclose(FileDat);
     unlink(chNomFichierDat);

     FILE* FileImage=fopen(chNomFichierTriangle,"rb");
     if(FileImage == NULL)
        {
        ::ErreurDeFichier(chNomFichierTriangle);
        free(bufferDiff);
        free(itabImage);
        free(alpha);
        free(beta);
        free(gamma);
        free(alphaprim);
        free(betaprim);
        free(gammaprim);
        fclose(FileBmp);
        fclose(FileDepl);
        unlink(chNomFichier);
        unlink(chNomFichierDepl);
        unlink(chNomFichierTriangle);
        return;
        }

     if(m_pImage->m_iType == 6)
        {
        //--- 8 bits
        BITMAPFILEHEADER BitmapFileHeader;
        BITMAPINFOHEADER BitmapInfoHeader;

        //--- BITMAPFILEHEADER
        char bm[3];
        strcpy(bm,"BM");
        memcpy(&BitmapFileHeader.bfType,bm,2);

        int iSize=sizeof(BITMAPFILEHEADER)+sizeof(BITMAPINFOHEADER)+sizeof(m_pImage-
>m_PaletteEntries);
        iSize+=(m_iNbligDestination*m_iNbcolDestination);
        BitmapFileHeader.bfSize=iSize;
        BitmapFileHeader.bfReserved1=0;
        BitmapFileHeader.bfReserved2=0;
        BitmapFileHeader.bfOffBits=sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER) +
sizeof(m_pImage->m_PaletteEntries);

        //--- BITMAPINFOHEADER
        BitmapInfoHeader.biSize=(DWORD) sizeof(BITMAPINFOHEADER);
        BitmapInfoHeader.biWidth=m_iNbcolDestination;
        BitmapInfoHeader.biHeight=m_iNbligDestination;
        BitmapInfoHeader.biPlanes=(WORD) 1;
        BitmapInfoHeader.biBitCount=(WORD) 8;
        BitmapInfoHeader.biCompression=BI_RGB;
        BitmapInfoHeader.biSizeImage=(DWORD) (m_iNbligDestination*m_iNbcolDestination);
        BitmapInfoHeader.biXPelsPerMeter=(LONG) m_dResolution;
        BitmapInfoHeader.biYPelsPerMeter=(LONG) m_dResolution;
        BitmapInfoHeader.biClrUsed=(DWORD) 0;
        BitmapInfoHeader.biClrImportant=(DWORD) 0;

        fwrite(&BitmapFileHeader,sizeof(BITMAPFILEHEADER),1,FileBmp);
        fwrite(&BitmapInfoHeader,sizeof(BITMAPINFOHEADER),1,FileBmp);
        fwrite(&m_pImage->m_PaletteEntries,sizeof(m_pImage->m_PaletteEntries),1,FileBmp);


        //--- calcul et écriture des pixels
        unsigned char *buffer;

        buffer=(unsigned char *) malloc(m_iNbcolDestination*sizeof(unsigned char));
        if(buffer == NULL)
           {
           ::ErreurDeMemoire();
           free(bufferDiff);
           free(itabImage);
           free(alpha);
```

```
   free(beta);
   free(gamma);
   free(alphaprim);
   free(betaprim);
   free(gammaprim);
   fclose(FileBmp);
   fclose(FileDepl);
   fclose(FileImage);
   unlink(chNomFichier);
   unlink(chNomFichierDepl);
   unlink(chNomFichierTriangle);
   return;
   }

double dX,dY,dXProj,dYProj,dDistance;
int j,iX,iY,ilig,icol,iDiff;
int iNbpt;

//--- init du progressCtrl
char chTitreProgress[100];
switch(config.m_iLangage)
   {
   case FRANCAIS:
   default:
      strcpy(chTitreProgress," Recalage en cours :");
      break;
   case ESPAGNOL:
      strcpy(chTitreProgress," Calculo en curso :");
      break;
   case ANGLAIS:
      strcpy(chTitreProgress," Working :");
      break;
   }
pMainFrame->InitProgress(chTitreProgress);
int iStep,iCount;

iStep=m_iNbligDestination/10;
if(iStep == 0)iStep=1;
iCount=0;

if(iModeRecalage == 0)
   {
   //--- recalage deux points
   ilig=0;
   dY=m_dFy1+(m_dResolution/2.);
   while(ilig < m_iNbligDestination)
      {
      //--- lire la ligne triangle
      for(icol=0;icol < m_iNbcolDestination;icol++)itabImage[icol]=0;
      icol=0;
      while(icol < m_iNbcolDestination)
         {
         fscanf(FileImage,"%d %d%*c",&iCode,&iNbpt);
         j=0;
         while(j < iNbpt)
            {
            itabImage[icol]=iCode;
            icol++;
            j++;
            }
         }

      icol=0;
      dX=m_dFx1+(m_dResolution/2.);
      while(icol < m_iNbcolDestination)
         {
         //--- recalage Voronoi
         iCode=itabImage[icol];
         if(iCode > 0)
            {
            dXProj=alpha[iCode]*dX + beta[iCode]*dY + gamma[iCode];
            dYProj=alphaprim[iCode]*dX + betaprim[iCode]*dY + gammaprim[iCode];

            //--- deplacement du au voronoi
            dDistance=sqrt((dX-dXProj)*(dX-dXProj)+(dY-dYProj)*(dY-dYProj));
            iDiff=(int) (dDistance/m_dResolution);
            if(iDiff > 255)iDiff=255;
            bufferDiff[icol]=iDiff;
```

```
               if(m_iRecalageInitial == SIMILITUDE)m_pSimilitude-
>ProjectionToImage(dXProj,dYProj,iX,iY);
               else m_pSysteme1->ProjectionToImage(dXProj,dYProj,iX,iY);
               if(iX >= 0 && iX < m_pImage->m_iNbcol && iY >= 0 && iY < m_pImage->m_iNblig)
                 buffer[icol]=m_pImage->m_bPixels[(iY*m_pImage->m_iNbcol)+iX];
               else buffer[icol]=0;
               }
             else
               {
               buffer[icol]=0;
               bufferDiff[icol]=255;
               }

             dX+=m_dResolution;
             icol++;
             }
           fwrite(buffer,1,m_iNbcolDestination,FileBmp);
           if(config.m_bImageDifference)fwrite(bufferDiff,1,m_iNbcolDestination,FileDepl);

           dY+=m_dResolution;
           ilig++;

           iCount++;
           if(iCount > iStep)
             {
             iCount=0;
             pMainFrame->StepProgress();
             }
           }
         }

       else if(iModeRecalage == 1)
         {
         //--- recalage bilineaire
         double dXImage,dYImage;
         double alfa0,alfa1,beta0,beta1;
         double dVal00,dVal01,dVal10,dVal11;

         ilig=0;
         dY=m_dFy1+(m_dResolution/2.);
         while(ilig < m_iNbligDestination)
           {
           //--- lire la ligne triangle
           for(icol=0;icol < m_iNbcolDestination;icol++)itabImage[icol]=0;
           icol=0;
           while(icol < m_iNbcolDestination)
             {
             fscanf(FileImage,"%d %d%*c",&iCode,&iNbpt);
             j=0;
             while(j < iNbpt)
               {
               itabImage[icol]=iCode;
               icol++;
               j++;
               }
             }

           icol=0;
           dX=m_dFx1+(m_dResolution/2.);
           while(icol < m_iNbcolDestination)
             {
             //--- recalage Voronoi
             iCode=itabImage[icol];
             if(iCode > 0)
               {
               dXProj=alpha[iCode]*dX + beta[iCode]*dY + gamma[iCode];
               dYProj=alphaprim[iCode]*dX + betaprim[iCode]*dY + gammaprim[iCode];

               //--- deplacement du au voronoi
               dDistance=sqrt((dX-dXProj)*(dX-dXProj)+(dY-dYProj)*(dY-dYProj));
               iDiff=(int) (dDistance/m_dResolution);
               if(iDiff > 255)iDiff=255;
               bufferDiff[icol]=iDiff;

               if(m_iRecalageInitial == SIMILITUDE)m_pSimilitude-
>ProjectionToImage(dXProj,dYProj,dXImage,dYImage);
               else m_pSysteme1->ProjectionToImage(dXProj,dYProj,dXImage,dYImage);
               alfa0=dXImage-floor(dXImage);
               alfa1=1.-alfa0;
               beta0=dYImage-floor(dYImage);
```

```
                beta1=1.-beta0;
                iX=(int) dXImage;
                iY=(int) dYImage;

                if(iX >= 0 && iX < m_pImage->m_iNbcol-1 && iY >= 0 && iY < m_pImage-
>m_iNblig-1)
                  {
                  dVal00=(double) m_pImage->m_bPixels[(iY*m_pImage->m_iNbcol)+iX];
                  dVal01=(double) m_pImage->m_bPixels[(iY*m_pImage->m_iNbcol)+iX+1];
                  dVal10=(double) m_pImage->m_bPixels[((iY+1)*m_pImage->m_iNbcol)+iX];
                  dVal11=(double) m_pImage->m_bPixels[((iY+1)*m_pImage->m_iNbcol)+iX+1];

                  buffer[icol]=(int)
(beta0*alfa0*dVal00+beta0*alfa1*dVal01+beta1*alfa0*dVal10+beta1*alfa1*dVal11);
                  }
                else buffer[icol]=0;
                }
              else
                {
                buffer[icol]=0;
                bufferDiff[icol]=255;
                }

              dX+=m_dResolution;
              icol++;
              }
            fwrite(buffer,1,m_iNbcolDestination,FileBmp);
            if(config.m_bImageDifference)fwrite(bufferDiff,1,m_iNbcolDestination,FileDepl);

            dY+=m_dResolution;
            ilig++;

            iCount++;
            if(iCount > iStep)
              {
              iCount=0;
              pMainFrame->StepProgress();
              }
            }
          }

        else if(iModeRecalage == 2)
          {
          //--- recalage bicubique
          int iIndice;
          double dXImage,dYImage;
          double alfa0,alfa1,alfa2,alfa3,beta0,beta1,beta2,beta3;
          double CX0,CX1,CX2,CX3,CY0,CY1,CY2,CY3;
          double dVal00,dVal01,dVal02,dVal03;
          double dVal10,dVal11,dVal12,dVal13;
          double dVal20,dVal21,dVal22,dVal23;
          double dVal30,dVal31,dVal32,dVal33;
          double dVal;

          ilig=0;
          dY=m_dFy1+(m_dResolution/2.);
          while(ilig < m_iNbligDestination)
            {
            //--- lire la ligne triangle
            for(icol=0;icol < m_iNbcolDestination;icol++)itabImage[icol]=0;
            icol=0;
            while(icol < m_iNbcolDestination)
              {
              fscanf(FileImage,"%d %d%*c",&iCode,&iNbpt);
              j=0;
              while(j < iNbpt)
                {
                itabImage[icol]=iCode;
                icol++;
                j++;
                }
              }

            icol=0;
            dX=m_dFx1+(m_dResolution/2.);
            while(icol < m_iNbcolDestination)
              {
              //--- recalage Voronoi
              iCode=itabImage[icol];
              if(iCode > 0)
```

```
                {
                dXProj=alpha[iCode]*dX + beta[iCode]*dY + gamma[iCode];
                dYProj=alphaprim[iCode]*dX + betaprim[iCode]*dY + gammaprim[iCode];

                //--- deplacement du au voronoi
                dDistance=sqrt((dX-dXProj)*(dX-dXProj)+(dY-dYProj)*(dY-dYProj));
                iDiff=(int) (dDistance/m_dResolution);
                if(iDiff > 255)iDiff=255;
                bufferDiff[icol]=iDiff;

                if(m_iRecalageInitial == SIMILITUDE)m_pSimilitude-
>ProjectionToImage(dXProj,dYProj,dXImage,dYImage);
                else m_pSysteme1->ProjectionToImage(dXProj,dYProj,dXImage,dYImage);
                iX=(int) dXImage;
                iY=(int) dYImage;

                if(iX >= 1 && iX < m_pImage->m_iNbcol-2 && iY >= 1 && iY < m_pImage-
>m_iNblig-2)
                  {
                  alfa1=dXImage-floor(dXImage);
                  alfa0=1.+alfa1;
                  alfa2=1.-alfa1;
                  alfa3=2.-alfa1;
                  beta1=dYImage-floor(dYImage);
                  beta0=1.+beta1;
                  beta2=1.-beta1;
                  beta3=2.-beta1;

                  CX0=alfa0*(-alfa0*alfa0 + 5.*alfa0 - 8.) + 4.;
                  CX1=alfa1*alfa1*(alfa1 - 2.) + 1.;
                  CX2=alfa2*alfa2*(alfa2 - 2.) + 1.;
                  CX3=alfa3*(-alfa3*alfa3 + 5*alfa3 - 8.) + 4.;
                  CY0=beta0*(-beta0*beta0 + 5*beta0 - 8.) + 4.;
                  CY1=beta1*beta1*(beta1 - 2.) + 1.;
                  CY2=beta2*beta2*(beta2 - 2.) + 1.;
                  CY3=beta3*(-beta3*beta3 + 5.*beta3 - 8.) + 4.;

                  iIndice=(iY-1)*m_pImage->m_iNbcol+iX;
                  dVal00=(double) m_pImage->m_bPixels[iIndice-1];
                  dVal01=(double) m_pImage->m_bPixels[iIndice];
                  dVal02=(double) m_pImage->m_bPixels[iIndice+1];
                  dVal03=(double) m_pImage->m_bPixels[iIndice+2];

                  iIndice=iY*m_pImage->m_iNbcol+iX;
                  dVal10=(double) m_pImage->m_bPixels[iIndice-1];
                  dVal11=(double) m_pImage->m_bPixels[iIndice];
                  dVal12=(double) m_pImage->m_bPixels[iIndice+1];
                  dVal13=(double) m_pImage->m_bPixels[iIndice+2];

                  iIndice=(iY+1)*m_pImage->m_iNbcol+iX;
                  dVal20=(double) m_pImage->m_bPixels[iIndice-1];
                  dVal21=(double) m_pImage->m_bPixels[iIndice];
                  dVal22=(double) m_pImage->m_bPixels[iIndice+1];
                  dVal23=(double) m_pImage->m_bPixels[iIndice+2];

                  iIndice=(iY+2)*m_pImage->m_iNbcol+iX;
                  dVal30=(double) m_pImage->m_bPixels[iIndice-1];
                  dVal31=(double) m_pImage->m_bPixels[iIndice];
                  dVal32=(double) m_pImage->m_bPixels[iIndice+1];
                  dVal33=(double) m_pImage->m_bPixels[iIndice+2];

                  dVal=CY0*(CX0*dVal00 + CX1*dVal01 + CX2*dVal02 + CX3*dVal03);
                  dVal+=CY1*(CX0*dVal10 + CX1*dVal11 + CX2*dVal12 + CX3*dVal13);
                  dVal+=CY2*(CX0*dVal20 + CX1*dVal21 + CX2*dVal22 + CX3*dVal23);
                  dVal+=CY3*(CX0*dVal30 + CX1*dVal31 + CX2*dVal32 + CX3*dVal33);

                  buffer[icol]=(int) dVal;
                  }
                else buffer[icol]=0;
                }
              else
                {
                buffer[icol]=0;
                bufferDiff[icol]=255;
                }

              dX+=m_dResolution;
              icol++;
              }
            fwrite(buffer,1,m_iNbcolDestination,FileBmp);
```

```
            if(config.m_bImageDifference)fwrite(bufferDiff,1,m_iNbcolDestination,FileDepl);

            dY+=m_dResolution;
            ilig++;

            iCount++;
            if(iCount > iStep)
               {
               iCount=0;
               pMainFrame->StepProgress();
               }
            }
          }

        pMainFrame->CloseProgress();
        free(buffer);
        }

     else if(m_pImage->m_iType == 8)
        {
        //--- 24 bits, 3 octets par pixel, processus identique
        ….
        }

     free(bufferDiff);
     free(itabImage);
     free(alpha);
     free(beta);
     free(gamma);
     free(alphaprim);
     free(betaprim);
     free(gammaprim);

     fclose(FileImage);
     fclose(FileBmp);
     fclose(FileDepl);
     if(config.m_bImageDifference == FALSE)unlink(chNomFichierDepl);
     unlink(chNomFichierTriangle);
     }

   //--- fichier .car
   double dProj[20];
   projection.GetProjection(dProj);
   double dXProj0,dYProj0;
   ::SavToProj(m_pAmers->m_dtabXSav[0],m_pAmers->m_dtabYSav[0],dXProj0,dYProj0);

   char chNomImage[_MAX_PATH];
   strcpy(chNomImage,m_pImage->m_chNomFichier);
   strcat(chNomImage,"R");
   double dXb=m_dFx1+dXProj0;
   double dYb=m_dFy1+dYProj0;
   ::CreationFichierCar(chNomImage,m_dResolution,1,dXb,dYb,m_iNbcolDestination,m_iNbligDestinat
ion,m_pImage->GetType(),dProj);

   if(config.m_bImageDifference)
      {
      strcpy(chNomImage,m_pImage->m_chNomFichier);
      strcat(chNomImage,"RDiff");
      dXb=m_dFx1+dXProj0;
      dYb=m_dFy1+dYProj0;

   ::CreationFichierCar(chNomImage,m_dResolution,1,dXb,dYb,m_iNbcolDestination,m_iNbligDestinat
ion,m_pImage->GetType(),dProj);
      }
   }
```

A.4. Savedit

A.4.1. La classe CArc

La classe `CArc` de SAVEDIT est plus complète que la classe `CArc` de SAVANE. Elle comprend en particulier des variables membres conservant l'état de l'arc

(simplicité, extra-simplicité) et permettant de dessiner ces arcs avec une couleur dépendant de leur état de validité. L'allocation mémoire pour les points (tableau `m_ArrayPoint`) est dynamique. Elle utilise les fonctionnalités de la classe `CArray` de la MFC.

```
class CArc
  {
  private:
    int m_iNbpt; //--- nombre de points de l'arc
    int m_iNz1; //--- numéro de zone adjacente dans le cas d'un document de type zone
    int m_iNz2; //--- numéro de zone adjacente dans le cas d'un document de type zone
    int m_iSens; //--- sens de l'arc
    int m_iSelection; //--- point sélectionné dans l'arc
    double m_dXminSav,m_dYminSav,m_dXmaxSav,m_dYmaxSav; //--- fenêtre de l'arc
  public:
    CArray<CPoint,CPoint> m_ArrayPoint; / :--- tableau des points
    int m_iSimplicite; //--- indicateur de simplificté
    int m_iExtraSimplicite; //--- indicateur d'extra-simplificté
    int m_iTestAngleFerme; //--- indicateur d'angle entre deux points

  public:
    CArc();
    CArc(int iNbpt);
    ~CArc();
    int GetNbpt() {return m_iNbpt;}
    int GetSens() {return m_iSens;}
    int GetNoZone1() {return m_iNz1;};
    int GetNoZone2() {return m_iNz2;};
    void SetNbpt(int iNbpt) {m_iNbpt=iNbpt;};
    void SetSens(int iSens) {m_iSens=iSens;};
    void SetNoZone1(int iNo) {m_iNz1=iNo;};
    void SetNoZone2(int iNo) {m_iNz2=iNo;};

    void GetPoint(int i,double &dX,double &dY)
     {
     dX=m_ArrayPoint[i].m_dX;
     dY=m_ArrayPoint[i].m_dY;
     };
    void SetPoint(int i,double &dX,double &dY)
     {
     m_ArrayPoint[i].m_dX=dX;
     m_ArrayPoint[i].m_dY=dY;
     };
    void AddPoint(CPoint point) {m_ArrayPoint.Add(point);};
    void AddPoint(double dX,double dY)
     {
     CPoint point(dX,dY);
     m_ArrayPoint.Add(point);
     };
    void SetPointGrow(int i,CPoint point) {m_ArrayPoint.SetAtGrow(i,point);}
    void SetPointGrow(int i,double dX,double dY)
     {
     CPointMygale point(dX,dY);
     m_ArrayPoint.SetAtGrow(i,point);
     }
    int GetSelection() {return m_iSelection;}
    void SetSelection(int iNo) {if(iNo >= 0 && iNo < m_iNbpt)m_iSelection=iNo; else
m_iSelection=-1;}
    int SelectionnerUnPoint(double dXGeo,double dYGeo);
    void SupprimerUnPoint(int iNoPoint);
    void GetFenetreSav(double &dXmin,double &dYmin,double &dXmax,double &dYmax)
     {
     dXmin=m_dXminSav;
     dYmin=m_dYminSav;
     dXmax=m_dXmaxSav;
     dYmax=m_dYmaxSav;
     }

    BOOL Intersect(double dX1,double dY1,double dX2,double dY2);
    BOOL TestSimplicite(double& dXRes,double& dYRes);
    BOOL TestExtraSimplicite(CArc* pArc,double& dXRes,double& dYRes,int& iNoPointArc1,int&
iNoPointArc2);
    BOOL TestAngleEntreSegments(double dAngleMinimum,double& dXRes,double& dYRes);
    BOOL TestIntersection(double dLong1,double dColat1,double dLong2,double dColat2,double&
dXRes,double& dYRes);
    void CalculFenetreSav();
    void CalculCentroide(double &dX,double &dY);
```

```
    void GetCentroide(double &dXGeo,double &dYGeo);
    double DistanceAUnPoint(double dXGeo,double dYGeo);
    int DistanceAUnPoint(int iXVue,int iYVue);
    int InsererUnPoint(double dXGeo,double dYGeo);
    BOOL InsererUnPoint(double dXGeo,double dYGeo,int iNoPoint);
    int PointLePlusProche(double dXGeo,double dYGeo,double dDistanceRecherche,double
dDistanceJointure,double& dXGeoRes,double& dYGeoRes,double &dDistance,int& iNoPoint);
    void Filtrer(double dSurfaceMinimum);
    void FiltrerMetre(double dSurfaceMinimum);
    void Nettoyer();
    void Arrondir(int iPrecision);
    void Molodensky(CMolodensky* pMolodensky);
    void Translation(double dXProj,double dYProj);
    void Tracer(CDC* pDc);
    void TracerAvecTranslation(CDC* pDc,int iTranslationX,int iTranslationY);
    void Tracer(CDC* pDc,COLORREF Couleur);
    void TracerAvecLesPoints(CDC* pDC);
    void TracerAvecLesPoints(CDC* pDC,COLORREF Couleur);
    void EffacerLesPoints(CDC* pDC);
    void TracerUnPoint(CDC* pDC,int iNoPoint,COLORREF Couleur);
    void Recopier(CArcMygale* pArc);
    void ChangerLeSens();
    void Chainer(BOOL bSens,int iTypeCoordonnees,int& iIndex,int& iNbptPolygon,double
*dtabPoints);
  };
```

La méthode `TestSimplicite()` permet de tester la simplicité d'un arc. Elle utilise la procédure `SegmentIntersection()` qui contrôle l'intersection de deux segments de droite :

```
BOOL CArc::TestSimplicite(double& dXRes,double& dYRes)
  {
  dXRes=0.;
  dYRes=0.;
  m_iSimplicite=0;
  if(m_iNbpt < 3)return TRUE;

  double dX1,dY1,dX2,dY2,dX3,dY3,dX4,dY4;
  int j,k;
  for(j=1;j < m_iNbpt;j++)
     {
     GetPoint(j-1,dX1,dY1);
     GetPoint(j,dX2,dY2);
    for(k=j+1;k < m_iNbpt;k++)
       {
       GetPoint(k-1,dX3,dY3);
       GetPoint(k,dX4,dY4);
       if(SegmentIntersection(dX1,dY1,dX2,dY2,dX3,dY3,dX4,dY4,dXRes,dYRes))
          {
          m_iSimplicite=1;
          return FALSE;
          }
       }
     }
  return TRUE;
  }
```

La méthode `TestExtraSimplicite()` permet de tester l'intersection d'un arc avec un autre. Elle utilise également la procédure `SegmentIntersection()` qui contrôle l'intersection de deux segments de droite, après avoir testé l'intersection possible des arcs en utilisant les *BoundingBox* des deux arcs (méthode `Intersect()` de `CArc`).

```
BOOL CArc::TestExtraSimplicite(CArc* pArc,double& dXRes,double& dYRes,int& iNoPointArc1,int&
iNoPointArc2)
  {
  dXRes=0.;
  dYRes=0.;
  iNoPointArc1=0;
  iNoPointArc2=0;
  if(pArc == NULL)return TRUE;
  if(pArc == this)return TRUE;
```

```
  //--- tester l'intersection des fenêtre des arcs
  double dXMin,dYMin,dXMax,dYMax;
  pArc->GetFenetreSav(dXMin,dYMin,dXMax,dYMax);
  if(Intersect(dXMin,dYMin,dXMax,dYMax) == FALSE)return TRUE;

  double dX1,dY1,dX2,dY2,dX3,dY3,dX4,dY4;
  int j,k;
  int iNbpt=pArc->GetNbpt();
  for(j=1;j < m_iNbpt;j++)
     {
     GetPoint(j-1,dX1,dY1);
     GetPoint(j,dX2,dY2);
    for(k=1;k < iNbpt;k++)
        {
        pArc->GetPoint(k-1,dX3,dY3);
        pArc->GetPoint(k,dX4,dY4);
      if(SegmentIntersection(dX1,dY1,dX2,dY2,dX3,dY3,dX4,dY4,dXRes,dYRes))
          {
          iNoPointArc1=j-1;
          iNoPointArc2=k-1;
          return FALSE;
          }
        }
     }
  return TRUE;
  }
```

La méthode `PointLePlusProche()` permet de déterminer les coordonnées du point de l'arc le plus proche d'un point donné :

```
int CArc::PointLePlusProche(double dXGeo,double dYGeo,double dDistanceRecherche,double
dDistanceJointure,double& dXGeoRes,double& dYGeoRes,double &dDistance,int& iNoPoint)
  {
  //--- dDistanceRecherche en minutes
  //--- retourne le type, la distance avec l'arc, le point résultat, le numéro du point dans
l'arc
  //--- type : -1 echec, 0 nouveau point, 1 premier noeud, 2 dernier noeud, 3 point de l'arc
  int i;
  iNoPoint=-1;
  dXGeoRes=0.;
  dYGeoRes=0.;
  dDistance=-DINCONNU;

  if(m_iNbpt < 2)return -1;

  double dX1,dY1,dX2,dY2,dXSav,dYSav;
  dX1=m_dXminSav - dDistanceRecherche;
  dX2=m_dXmaxSav + dDistanceRecherche;
  dY1=m_dYminSav - dDistanceRecherche;
  dY2=m_dYmaxSav + dDistanceRecherche;
  wind.GeoToSav(dXGeo,dYGeo,dXSav,dYSav);

  if(dXSav < dX1 || dXSav > dX2 || dYSav < dY1 || dYSav > dY2)return -1;

  //--- premier passage : test sur les points de l'arc
  double dP1P2;

  int iType=-1;
  iNoPoint=-1;
  double dDistanceAuCarre=dDistanceRecherche*dDistanceRecherche;
  for(i=0;i < m_iNbpt;i++)
     {
     GetPoint(i,dX1,dY1);
     dP1P2=((dXGeo-dX1)*(dXGeo-dX1))+((dYGeo-dY1)*(dYGeo-dY1));
     if(dP1P2 < dDistanceAuCarre)
        {
        iNoPoint=i;
        dXGeoRes=dX1;
        dYGeoRes=dY1;
        dDistanceAuCarre=dP1P2;
        }
     }

  if(iNoPoint == 0)iType=1; //--- premier point
  else if(iNoPoint == m_iNbpt-1)iType=2; //--- dernier point
  else if(iNoPoint > 0)iType=3; //--- un point de l'arc
```

```
  dDistance=sqrt(dDistanceAuCarre);
  if(dDistance < dDistanceJointure)
    {
    //--- le point de l'arc répond à la question, on sort
    return iType;
    }

  //--- deuxième passage : test sur les segments de l'arc
  double d,dX,dY,cosTeta,sinTeta,dXPrim,dYPrim;

  GetPoint(0,dX1,dY1);
  for(i=1;i < m_iNbpt;i++)
    {
    GetPoint(i,dX2,dY2);
    dP1P2=((dX2-dX1)*(dX2-dX1))+((dY2-dY1)*(dY2-dY1));
    if(dP1P2 > 0.)
      {
      dP1P2=sqrt(dP1P2);
      //--- changement de repere pour les calculs
      cosTeta=(dX2-dX1)/dP1P2;
      sinTeta=(dY2-dY1)/dP1P2;
      dX=dXGeo-dX1;
      dY=dYGeo-dY1;
      dXPrim=dX*cosTeta+dY*sinTeta;
      dYPrim=dY*cosTeta-dX*sinTeta;
      if(dXPrim >= 0. && dXPrim <= dP1P2)
        {
        d=fabs(dYPrim);
        if(d < dDistance)
          {
          dDistance=d;
          iNoPoint=i-1;
          iType=0; //--- nouveau point à créer
          //--- rechangement de repère pour avoir le point d'intersection sur le segment
(dYPrim = 0)
          dXGeoRes=dXPrim*cosTeta + dX1;
          dYGeoRes=dXPrim*sinTeta + dY1;
          }
        }
      }
    dX1=dX2;
    dY1=dY2;
    }

  if(iNoPoint >= 0)
    {
    return iType;
    }
  return -1;
  }
```

A.4.2. La classe CSaveditDocument

Toutes les variables et procédures liée à document de saisie sont regroupées dans une classe nommée `CSaveditDocument`. On notera en particulier les variables qui conservent les coordonnées des éléments graphiques et la valeur ou la clé des objets (`CArc* m_ptabArc; double* m_dtabPoint; double* m_dtabValeur; char* m_ctabCle`). L'allocation mémoire de ces variables est effectuée dans les méthodes `Nouveau()` ou `Ouvrir()` de l'objet `CSaveditDocument`, en fonction du type des objets.

```
class CSaveditDocument
  {
  private:
    BOOL m_bOuvert;
    BOOL m_bASauver;
    BOOL m_bARasteriser;
    char m_chNomFichier[_MAX_PATH];
    //--- paramètres du .car
    int m_iVersion;
```

```
    int m_iTypeObjet;
    int m_iTypeSaisie;
    char m_chNomDocument[100];
    int m_iEchelle;
    char m_chOperateur[100];
    char m_chDatum[100];
    char m_chDate[100];
    char m_chCreation[100];
    double m_dX1,m_dY1,m_dX2,m_dY2;
    int m_iNbcharCle;

    //--- objets graphiques, mémoire allouée en fonction du type
    int m_iNoZoneLibre;
    int m_iNbObjet;
    int m_iNbArc;
    CArc* m_ptabArc[NB_MAX_ARC];
    int m_itabNoZone[NB_MAX_OBJET];
    double* m_dtabPoint;
    double* m_dtabValeur;
    char* m_ctabCle;

    //--- édition et sélection
    int m_iNbObjetSelection;
    int m_iObjetSelection;
    int m_itabObjetSelection[NB_MAX_SELECTION];

    int m_iNbArcSelection;
    int m_iArcSelection;
    int m_itabArcSelection[NB_MAX_SELECTION];

  //--- pour le undo
  public:
    CArcMygale* m_ptabArcModifie[NB_MAX_UNDO];
    CArcMygale* m_ptabArcSauvegarde[NB_MAX_UNDO];
    int m_itabTypeUndo[NB_MAX_UNDO + 1];

  //--- tracer
  public:
    BOOL m_bDegrade;
    double m_dIntervalleDegrade;

  //--- pour l'initialisation des clés nominales
  public:
    int m_iCleValeurInitiale;
    int m_iCleIncrement;
    int m_iCleNbChiffres;
    CString m_strClePrefixe;
    CString m_strCleSuffixe;

  private:
    BOOL Sauver();

  public:
    CMygaleDocument();
    ~CMygaleDocument();

    //--- procédures de gestion des fichiers
    void Init();
    BOOL Nouveau(const char* chNom,int iEchelle,const char* chOperateur,int iType,int
iSaisie,int iNbcharCle);
    BOOL Ouvrir(const char* chNomFichier);
    BOOL Ajouter(const char* chNomFichier);
    BOOL ImporterShapefile(const char* chNomFichier,int iNoattCle,const char* strPrefixe,BOOL
bCleToujoursNominal,CProjection* pProjection,int iHemisphere,BOOL bTypeShape,int iTypeMygale);
    BOOL ImporterMygale(const char* chNomFichier,CProjection* pProjection);
    BOOL ImporterPoint(const char* chNomFichier,CDecodage* pDecodage,const char* chPrefixe,
      const char* chNom,const char* chOperateur,int iEchelle,int iCleMygale,int iNbcharCle);
    BOOL ImporterDxf(const char* chNomFichier,CProjection* pProjection,int iHemisphere,
      int iTousLesLayers,bool* btabLayer,
      const char* chNom,const char* chOperateur,int iEchelle,int iTypeObjet,int
iTypeSaisie,int iNbcharCle);
    BOOL ImporterLignes(const char* chNomFichier,CProjection* pProjection,int iHemisphere,
      const char* chPrefixe,const char* chNom,const char* chOperateur,int iEchelle,int
iTypeSaisie,int iNbcharCle);
    BOOL Enregistrer();
    BOOL EnregistrerCar();
    BOOL EnregistrerSous();
    BOOL ExporterEnShapefile(const char* chNomFichier,int iTypeShapefile,int
iTypeCoordonnees);
    BOOL Fermer();
```

```
     void SupprimerToutesLesZones();
     int SupprimerLesZonesSansArcs();
     int SupprimerLesArcsLibres();
     int SupprimerLesArcsEnDouble();
     int NettoyerTousLesArcs();
     void Precision(int iPrecision);

     //--- changement de datum par Molodensky
     void Molodensky(CMolodensky* pMolodensky);

     BOOL IsTypeZone() {if(m_iTypeObjet == TYPE_ZONE_MYGALE)return TRUE; else return FALSE;};
     BOOL IsTypeLigne() {if(m_iTypeObjet == TYPE_LIGNE_MYGALE)return TRUE; else return
FALSE;};
     BOOL IsTypePoint() {if(m_iTypeObjet == TYPE_POINT_MYGALE)return TRUE; else return
FALSE;};

     int GetTypeObjet() {return m_iTypeObjet;};
     int GetTypeSaisie() {return m_iTypeSaisie;};
     void GetNomFichier(char* chNom) {strcpy(chNom,m_chNomFichier);};
     void GetNomDocument(char* chNom) {strcpy(chNom,m_chNomDocument);};
     void SetNomDocument(const char* chNom) {strcpy(m_chNomDocument,chNom);};
     void GetOperateur(char* chNom) {strcpy(chNom,m_chOperateur);};
     void SetOperateur(const char* chNom) {strcpy(m_chOperateur,chNom);};
     void GetDatum(char* chNom) {strcpy(chNom,m_chDatum);};
     void SetDatum(const char* chNom) {strcpy(m_chDatum,chNom);m_bASauver=TRUE;};
     int GetEchelle() {return m_iEchelle;};
     void SetEchelle(int iEchelle) {m_iEchelle=iEchelle;};
     void GetDate(char* chNom) {strcpy(chNom,m_chDate);};
     void GetCreation(char* chNom) {strcpy(chNom,m_chCreation);};
     BOOL IsOuvert() {return m_bOuvert;};
     BOOL IsASauver() {return m_bASauver;};
     void ASauver() {m_bASauver=TRUE;m_bARasteriser=TRUE;};
     void ARasteriser() {m_bARasteriser=TRUE;};

     BOOL FenetreSav(double &dFx1,double &dFy1,double &dFx2,double &dFy2);

     //--- procédures sur les objets
     void GetValue(int iNoObjet,char* chCle,double &dValeur);
     void GetValuePourDBIII(int iNoObjet,char *chCle,double &dValeur);
     CArc* GetPtrArc(int iNoArc) {if(iNoArc >= 0 && iNoArc < m_iNbArc)return
m_ptabArc[iNoArc]; else return NULL;};
     int GetNoZone(int iObjet) {if(iObjet >= 0 && iObjet < m_iNbObjet)return
m_itabNoZone[iObjet]; else return -1;};

     int GetNbObjet() {return m_iNbObjet;};
     int GetNbArc() {return m_iNbArc;};
     int GetNbPoints();
     void Translation(double dXProj,double dYProj);

     //--- procédures de sélection
     int SelectionnerUnObjet(double dX,double dY,double& dDistance);
     int SelectionnerUnArc(double dX,double dY,double& dDistance);

     int GetObjetSelection() {return m_iObjetSelection;};
     int GetObjetSelection(int i) {if(i >= 0 && i < m_iNbObjetSelection)return
m_itabObjetSelection[i]; else return -1;};
     int GetNbObjetSelection() {return m_iNbObjetSelection;};
     void SetObjetSelection(int iNo,int iModeSelection);
     void InitObjetSelection() {m_iObjetSelection=-1;m_iNbObjetSelection=0;};

     int GetArcSelection() {return m_iArcSelection;};
     int GetArcSelection(int i) {if(i >= 0 && i < m_iNbArcSelection)return
m_itabArcSelection[i]; else return -1;};
     int GetNbArcSelection() {return m_iNbArcSelection;};
     void SetArcSelection(int iNo,int iModeSelection);
     void InitArcSelection();
     BOOL IsSelectionFermee();
     BOOL IsSelectionFermee(CDC* pDC);
     void SetArcSelectionZone(int iNoObjet,int iMode);
     void SelectionnerTousLesArcs();
     void SelectionnerTousLesObjets();
     void SelectionSurLaValeur(const char* chCle,double dValeurSelection);

     //--- procédures sur le undo
     void InitUndo() {m_itabTypeUndo[0]=UNDO_RIEN;m_ptabArcModifie[0]=NULL;}
     void RestaurerUnArc(int i);

     int AjouterUnObjet(double dXGeo,double dYGeo,const char* chCle,double dValeur);
     void ModifierUnObjet(int iNoObjet,const char* chCle,double dValeur);
```

```
    void SupprimerUnObjet(int iNoObjet,BOOL bSupprimerLesArcs);

    void GetPosVue(int iNoObjet,int &iXVue,int &iYVue);
    void SetPosVue(int iNoObjet,int iXVue,int iYVue);
    void GetCentreZoneVue(int iNoZone,int &iXVue,int &iYVue);
    void GetPosGeo(int iNoObjet,double &dXGeo,double &dYGeo);
    void SetPosGeo(int iNoObjet,double dXGeo,double dYGeo);

    //--- pour le type ligne ou zone, fonctions sur les arcs
    int AjouterUnArc(int iNz1,int iNz2);
    void SupprimerUnArc(int iNoArc);
    BOOL FermerUnArc(int iNoArc);
    BOOL JoindreDeuxArcs(int iNoArcReference,int iNoArcAModifier);
    BOOL ReunirDeuxArcs(int iNoArc1,int iNoArc2);
    BOOL FusionnerDeuxArcs(int iNoArcReference,int iNoArcAModifier);
    int DiviserUnArc(int iNoArc,double dXGeo,double dYGeo);
    int CouperUnArc(int iNoArc,double dXGeo,double dYGeo,int iNoPoint);
    int CouperUnArc(int iNoArc,int iNoPoint);
    BOOL UnirUnArc(int iNoArc);
    void DeplacerLaSelection(int iTranslationX,int iTranslationY);
    void DupliquerLaSelection(int iTranslationX,int iTranslationY);
    void ExclureTousLesArcs();
    void FiltrerTousLesArcs(double dSurfaceMinimum);
    int JoindreTousLesArcsValeur(double dDistanceEnMetre,int iNbIterations);
    int JoindreTousLesArcs(double dDistanceEnMetre,int iNbIterations);
    int ReunirTousLesArcs();
    int DiviserEtJoindreParIntersection();
    int DiviserEtJoindreParExtremite(double dDistanceEnMetre);
    void CalculFenetreDesArcs();
    void ModifierParIntersection(BOOL bValeurInitiale,double dValeurMinimum,double
dIntervalle,double dLong1,double dColat1,double dLong2,double dColat2);
    void BougerLesNoeuds(double dXGeoOld,double dYGeoOld,double dXGeo,double dYGeo);

    //--- procédures de test
    void TestValeur(double dIntervalleMaximum,double dIntervalleMinimum);
    void TestLigneValeur(CDC* pDC,int iNoArc,double dIntervalleMaximum,double
dIntervalleMinimum);
    void TestSimplicite();
    void TestExtraSimplicite();
    void Rasteriser(const char* chNomFichierImage);
    BOOL IsZoneFermee(int iNoZone);

    //--- procédures de tracé des documents
    void Tracer(CDC* pDC);
    void TracerUnObjet(CDC* pDC,int iNoObjet,COLORREF Couleur);
    void TracerUnArc(CDC* pDC,int iNoArc);
    void TracerUnArc(CDC* pDC,int iNoArc,COLORREF Couleur);
    void TracerLaSelection(CDC* pDC,int iTranslationX,int iTranslationY);
    void TracerFermeture(CDC* pDC);
  };
```

De nombreuses méthodes de cette classe sont liées à la vérification de contraintes d'intégrité spatiale ou au netoyage de documents importés à partir d'autres systèmes. Par exemple, la méthode `SupprimerLesArcsEnDouble()` permet d'éliminer les arcs qui apparaissent deux fois dans le document, comme c'est la cas de tous les arcs directement importés d'un document au format ShapeFile.

```
int CSaveditDocument::SupprimerLesArcsEnDouble()
  {
  if(IsOuvert() == FALSE)return -1;
  if(IsTypeZone() == FALSE)return -1;

  int iNbArcSupprimes=0;

  CArcMygale *pArc1,*pArc2;
  bool bOui;
  int iNoZone1,iNoZone2,iNoZone3,iNoZone4;
  int iArc1,iArc2,iNbpt1,iNbpt2,iPointDebut1,iPointDebut2,iPointFin1,iPointFin2,iNbCommuns;
  int i,j,iSens;
  double dX,dY,dX1,dY1,dX2,dY2;
  double dXMinArc1,dYMinArc1,dXMaxArc1,dYMaxArc1;
  double dXMinArc2,dYMinArc2,dXMaxArc2,dYMaxArc2;
  CPointMygale point;

  iArc1=0;
```

```
  while(iArc1 < m_iNbArc)
     {
     pArc1=m_ptabArc[iArc1];
     if(pArc1 != NULL)
        {
        iNoZone1=pArc1->GetNoZone1();
        iNoZone2=pArc1->GetNoZone2();
        if(iNoZone1 < 20 || iNoZone2 < 20)
           {
           //--- l'arc 1 est candidat
           pArc1->GetFenetreSav(dXMinArc1,dYMinArc1,dXMaxArc1,dYMaxArc1);
           iArc2=iArc1+1;
           while(iArc2 < m_iNbArc)
              {
              pArc2=m_ptabArc[iArc2];
              if(pArc2 != NULL)
                 {
                 iNoZone3=pArc2->GetNoZone1();
                 iNoZone4=pArc2->GetNoZone2();
                 if(iNoZone3 < 20 || iNoZone4 < 20)
                    {
                    //--- intersection des fenêtres ?
                    pArc2->GetFenetreSav(dXMinArc2,dYMinArc2,dXMaxArc2,dYMaxArc2);
                    if(dXMaxArc2 >= dXMinArc1 && dXMinArc2 <= dXMaxArc1 && dYMaxArc2 >= dYMinArc1
&& dYMinArc2 <= dYMaxArc1)
                       {
                       //--- recherche d'une séquence commune entre arc 1 et arc 2
                       iNbpt1=pArc1->GetNbpt();
                       iNbpt2=pArc2->GetNbpt();
                       iPointDebut1=0;
                       while(iPointDebut1 < iNbpt1)
                          {
                          pArc1->GetPoint(iPointDebut1,dX1,dY1);
                          iPointDebut2=0;
                          while(iPointDebut2 < iNbpt2)
                             {
                             pArc2->GetPoint(iPointDebut2,dX2,dY2);
                             if(dX1 == dX2 && dY1 == dY2)
                                {
                                //--- un point identique, on cherche les suivants dans un sens
                                iNbCommuns=1;
                                iPointFin1=iPointDebut1;
                                iPointFin2=iPointDebut2;
                                bOui=TRUE;
                                while(iPointFin1 < iNbpt1-1 && iPointFin2 < iNbpt2-1 && bOui)
                                   {
                                   pArc1->GetPoint(iPointFin1+1,dX1,dY1);
                                   pArc2->GetPoint(iPointFin2+1,dX2,dY2);
                                   if(dX1 == dX2 && dY1 == dY2)
                                      {
                                      iNbCommuns++;
                                      iPointFin1++;
                                      iPointFin2++;
                                      }
                                   else bOui=FALSE;
                                   }
                                if(iNbCommuns == 1)
                                   {
                                   //--- on essaye dans l'autre sens
                                   iPointFin1=iPointDebut1;
                                   iPointFin2=iPointDebut2;
                                   bOui=TRUE;
                                   while(iPointFin1 < iNbpt1-1 && iPointFin2 > 0 && bOui)
                                      {
                                      pArc1->GetPoint(iPointFin1+1,dX1,dY1);
                                      pArc2->GetPoint(iPointFin2-1,dX2,dY2);
                                      if(dX1 == dX2 && dY1 == dY2)
                                         {
                                         iNbCommuns++;
                                         iPointFin1++;
                                         iPointFin2--;
                                         }
                                      else bOui=FALSE;
                                      }
                                   }
                                if(iNbCommuns > 1)
                                   {
                                   //--- des points en commun, arc 1 à découper
                                   iSens=pArc1->GetSens();
                                   if(iPointFin1 < iNbpt1-1)
```

```
   {
   //--- bout final
   if(m_iNbArc >= NB_MAX_ARC)return iNbArcSupprimes;
   m_ptabArc[m_iNbArc]=new CArcMygale(iNbpt1-iPointFin1+1);
   if(m_ptabArc[m_iNbArc] == NULL)return iNbArcSupprimes;
   j=0;
   for(i=iPointFin1;i < iNbpt1;i++)
      {
      pArc1->GetPoint(i,dX,dY);
      point.Set(dX,dY);
      m_ptabArc[m_iNbArc]->SetPointGrow(j,point);
      j++;
      }
   m_ptabArc[m_iNbArc]->SetSens(iSens);
   m_ptabArc[m_iNbArc]->SetNbpt(j);
   m_ptabArc[m_iNbArc]->SetNoZone1(iNoZone1);
   m_ptabArc[m_iNbArc]->SetNoZone2(iNoZone2);
   m_ptabArc[m_iNbArc]->CalculFenetreSav();
   m_iNbArc++;
   }

if(iPointDebut1 > 0)
   {
   //--- bout initial
   if(m_iNbArc >= NB_MAX_ARC)return iNbArcSupprimes;
   m_ptabArc[m_iNbArc]=new CArcMygale(iPointDebut1+1);
   if(m_ptabArc[m_iNbArc] == NULL)return iNbArcSupprimes;
   j=0;
   for(i=0;i <= iPointDebut1;i++)
      {
      pArc1->GetPoint(i,dX,dY);
      point.Set(dX,dY);
      m_ptabArc[m_iNbArc]->SetPointGrow(j,point);
      j++;
      }
   m_ptabArc[m_iNbArc]->SetSens(iSens);
   m_ptabArc[m_iNbArc]->SetNbpt(j);
   m_ptabArc[m_iNbArc]->SetNoZone1(iNoZone1);
   m_ptabArc[m_iNbArc]->SetNoZone2(iNoZone2);
   m_ptabArc[m_iNbArc]->CalculFenetreSav();
   m_iNbArc++;
   }

if(iNoZone1 == iNoZone3 && iNoZone1 >= 20)
   {
   //--- suppression de l'arc qui reste, car même zone adjacente
   SupprimerUnArc(iArc1);
   iNbArcSupprimes++;
   }
else
   {
   //--- arranger l'arc iArc1 avec les points communs
   j=0;
   for(i=iPointDebut1;i <= iPointFin1;i++)
      {
      pArc1->GetPoint(i,dX,dY);
      pArc1->SetPoint(j,dX,dY);
      j++;
      }
   pArc1->SetNbpt(j);
   if(iNoZone1 < 20)
      {
      if(iNoZone3 >= 20)
            {
            pArc1->SetNoZone1(iNoZone3);
            pArc1->SetNoZone2(2);
            }
      else
            {
            pArc1->SetNoZone1(0);
            pArc1->SetNoZone2(0);
            }
      }
   else
      {
      if(iNoZone1 < iNoZone3)
            {
            pArc1->SetNoZone1(iNoZone1);
            pArc1->SetNoZone2(iNoZone3);
            }
```

```
                        else
                              {
                              if(iNoZone3 >= 20)
                                    {
                                    pArc1->SetNoZone1(iNoZone3);
                                    pArc1->SetNoZone2(iNoZone1);
                                    }
                              else
                                    {
                                    pArc1->SetNoZone1(iNoZone1);
                                    pArc1->SetNoZone2(2);
                                    }
                              }
                        }
                     }
                  //--- arc 2
                  Ordonner(iPointDebut2,iPointFin2);
                  iSens=pArc2->GetSens();
                  if(iPointFin2 < iNbpt2-1)
                     {
                     //--- bout final
                     if(m_iNbArc >= NB_MAX_ARC)return iNbArcSupprimes;
                     m_ptabArc[m_iNbArc]=new CArcMygale(iNbpt2-iPointFin2+1);
                     if(m_ptabArc[m_iNbArc] == NULL)return iNbArcSupprimes;
                     j=0;
                     for(i=iPointFin2;i < iNbpt2;i++)
                        {
                        pArc2->GetPoint(i,dX,dY);
                        point.Set(dX,dY);
                        m_ptabArc[m_iNbArc]->SetPointGrow(j,point);
                        j++;
                        }
                     m_ptabArc[m_iNbArc]->SetSens(iSens);
                     m_ptabArc[m_iNbArc]->SetNbpt(j);
                     m_ptabArc[m_iNbArc]->SetNoZone1(iNoZone3);
                     m_ptabArc[m_iNbArc]->SetNoZone2(iNoZone4);
                     m_ptabArc[m_iNbArc]->CalculFenetreSav();
                     m_iNbArc++;
                     }
                  if(iPointDebut2 > 0)
                     {
                     //--- bout initial
                     if(m_iNbArc >= NB_MAX_ARC)return iNbArcSupprimes;
                     m_ptabArc[m_iNbArc]=new CArcMygale(iPointDebut2+1);
                     if(m_ptabArc[m_iNbArc] == NULL)return iNbArcSupprimes;
                     j=0;
                     for(i=0;i <= iPointDebut2;i++)
                        {
                        pArc2->GetPoint(i,dX,dY);
                        point.Set(dX,dY);
                        m_ptabArc[m_iNbArc]->SetPointGrow(j,point);
                        j++;
                        }
                     m_ptabArc[m_iNbArc]->SetSens(iSens);
                     m_ptabArc[m_iNbArc]->SetNbpt(j);
                     m_ptabArc[m_iNbArc]->SetNoZone1(iNoZone3);
                     m_ptabArc[m_iNbArc]->SetNoZone2(iNoZone4);
                     m_ptabArc[m_iNbArc]->CalculFenetreSav();
                     m_iNbArc++;
                     }
                  SupprimerUnArc(iArc2);
                  iNbArcSupprimes++;
                  pArc1=NULL;
                  pArc2=NULL;

                  //--- on recommence
                  InitUndo();
                  ASauver();

                  goto SUITE;
                  }
               }
            iPointDebut2++;
            }
         iPointDebut1++;
         }
      }
   }
}
iArc2++;
```

```
            }
          }
        }
     iArc1++;
SUITE: i=0;
     }

  return iNbArcSupprimes;
  }
```

La méthode `IsZoneFermee()` teste la fermeture d'une zone, en calculant l'occurrence des extremités des arcs de la zone :

```
BOOL CSaveditDocument::IsZoneFermee(int iNoZone)
  {
  int i,j,iNoZone1,iNoZone2,iNbpt,iNbBout,itabNb[NB_MAX_SELECTION];
  BOOL bTrouve;
  double dXGeo,dYGeo,dtabX[NB_MAX_SELECTION],dtabY[NB_MAX_SELECTION];

  if(iNoZone < 20 || iNoZone > m_iNoZoneLibre)return FALSE;

  iNbBout=0;
  for(i=0;i < m_iNbArc;i++)
    {
    iNoZone1=m_ptabArc[i]->GetNoZone1();
    iNoZone2=m_ptabArc[i]->GetNoZone2();
    if(iNoZone1 == iNoZone || iNoZone2 == iNoZone)
      {
      iNbpt=m_ptabArc[i]->GetNbpt();
      m_ptabArc[i]->GetPoint(0,dXGeo,dYGeo);
      bTrouve=FALSE;
      for(j=0;j < iNbBout;j++)
        {
        if(dtabX[j] == dXGeo && dtabY[j] == dYGeo)
          {
          itabNb[j]=1-itabNb[j];
          bTrouve=TRUE;
          break;
          }
        }
      if(bTrouve == FALSE)
        {
        dtabX[iNbBout]=dXGeo;
        dtabY[iNbBout]=dYGeo;
        itabNb[iNbBout]=1;
        iNbBout++;
        }
      m_ptabArc[i]->GetPoint(iNbpt-1,dXGeo,dYGeo);
      bTrouve=FALSE;
      for(j=0;j < iNbBout;j++)
        {
        if(dtabX[j] == dXGeo && dtabY[j] == dYGeo)
          {
          itabNb[j]=1-itabNb[j];
          bTrouve=TRUE;
          break;
          }
        }
      if(bTrouve == FALSE)
        {
        dtabX[iNbBout]=dXGeo;
        dtabY[iNbBout]=dYGeo;
        itabNb[iNbBout]=1;
        iNbBout++;
        }
      }
    }

  for(j=0;j < iNbBout;j++)
    {
    if(itabNb[j] != 0)return FALSE;
    }

  return TRUE;
  }
```

La méthode `TestValeur()` permet de mettre en évidence les erreurs de valeur numérique pour des courbes de niveaux (document de type Ligne, saisie par valeur numérique). On calcule pour cela les intersections de toutes les lignes saisies avec des lignes horizontales (y=constante), en testant ensuite les différences de valeur entre deux points d'intersection consécutifs.

```
void CSaveditDocument::TestValeur(double dIntervalleMaximum,double dIntervalleMinimum)
  {
  if(IsOuvert() == FALSE)return;
  if(m_iTypeObjet != TYPE_LIGNE_MYGALE)return;
  if(m_iTypeSaisie != SAISIE_VALEUR && m_iTypeSaisie != SAISIE_CLEVALEUR)return;
  if(m_iNbObjet == 0)return;

  int i,j,iNbpt,iNoObjet1,iNoObjet2;
  double dValeur1,dValeur2,dDifference;
  double dFx1,dFy1,dFx2,dFy2;
  double dXMin,dYMin,dXMax,dYMax;
  double dX0,dY0,dX1,dY1,dXGeo,dYGeo;
  double dX,dY,dXa,dYa,dXb,dYb,dIntervalleLigne;

  for(i=0;i < NB_MAX_OBJET;i++)m_itabNoZone[i]=0;
  //-- calcul de la fenêtre du document, en coordonnées savane
  int iNbLignesRaster=500 ;
  FenetreSav(dFx1,dFy1,dFx2,dFy2);
  dIntervalleLigne=(dFy2-dFy1)/iNbLignesRaster;

  double dMarge=(dFy2-dFy1)/20.;
  dY=dFy1 + dMarge;
  while(dY <= dFy2 - dMarge)
    {
    //--- Pour chaque ligne raster, calcul des intersections
    CChaine chaine;
    for(i=0;i < NB_MAX_ARC;i++)
      {
      //--- intersections de l'arc avec la ligne
      if(m_ptabArc[i] != NULL)
        {
        m_ptabArc[i]->GetFenetreSav(dXMin,dYMin,dXMax,dYMax);

        if(dY >= dYMin && dY <= dYMax)
          {
          iNbpt=m_ptabArc[i]->GetNbpt();
          m_ptabArc[i]->GetPoint(0,dXGeo,dYGeo);
          wind.GeoToSav(dXGeo,dYGeo,dX0,dY0);

          for(j=1;j < iNbpt;j++)
            {
            m_ptabArc[i]->GetPoint(j,dXGeo,dYGeo);
            wind.GeoToSav(dXGeo,dYGeo,dX1,dY1);

            //--- ordre sur Y
            if(dY1 > dY0)
              {
              dXa=dX0;
              dYa=dY0;
              dXb=dX1;
              dYb=dY1;
              }
            else
              {
              dXa=dX1;
              dYa=dY1;
              dXb=dX0;
              dYb=dY0;
              }
            //--- intersection
            if(dY >= dYa && dY <= dYb)
              {
              if(dYa == dYb)dX=dXa;
              else dX=((dXb-dXa)*(dY-dYa)/(dYb-dYa)) + dXa;
              chaine.InsererCellule(dX,i);
              }

            dX0=dX1;
            dY0=dY1;
            }
```

```
            }
          }
        }
      //--- test des différences
      CCellule* pCellule=chaine.m_ptrDebut;
      while(pCellule != NULL)
        {
        iNoObjet1=pCellule->m_iNoObjet;
        dValeur1=m_dtabValeur[iNoObjet1];

        pCellule=pCellule->m_ptrSuivant;
        if(pCellule != NULL)
          {
          iNoObjet2=pCellule->m_iNoObjet;
          dValeur2=m_dtabValeur[iNoObjet2];
          //--- tests
          if(dIntervalleMaximum > 0.)
            {
            dDifference=fabs(dValeur2-dValeur1);
            if(dDifference > dIntervalleMaximum)
              {
              m_itabNoZone[iNoObjet1]=1;
              m_itabNoZone[iNoObjet2]=1;
              }
            }
          if(dIntervalleMinimum > 0.)
            {
            dDifference=fabs(dValeur2-dValeur1);
            if(dDifference < dIntervalleMinimum)
              {
              m_itabNoZone[iNoObjet1]=1;
              m_itabNoZone[iNoObjet2]=1;
              }
            }
          }
        }

      dY+=dIntervalleLigne;
      }
    }
```

Nous présentons enfin des extraits des méthodes de lecture des documents SAVEDIT pour chaque type de document (point, ligne, zone), ce qui nous permet de montrer le format de stockage de ces documents.

Format des documents de type point :

```
m_iNbObjet=0;
while(fread(buffer,iTailleBuffer,1,FileObjets) == 1)
  {
  memcpy(&dLong,&buffer[0],8);
  memcpy(&dColat,&buffer[8],8);

  m_dtabPoint[iIndice]=dLong;
  iIndice++;
  m_dtabPoint[iIndice]=dColat;
  iIndice++;

    switch(m_iTypeSaisie)
     {
     case SAISIE_VALEUR:
       memcpy(&m_dtabValeur[m_iNbObjet],&buffer[16],8);
       break;
     case SAISIE_CLE:
       memcpy(&m_ctabCle[m_iNbObjet*m_iNbcharCle],&buffer[16],m_iNbcharCle);
       break;
     case SAISIE_CLEVALEUR:
       memcpy(&m_dtabValeur[m_iNbObjet],&buffer[16],8);
       memcpy(&m_ctabCle[m_iNbObjet*m_iNbcharCle],&buffer[24],m_iNbcharCle);
       break;

     default:
       break;
     }
    m_iNbObjet++;
```

```
    }
```

Pour les lignes, la procédure de lecture est la suivante :

```
CPointMygale point;

strcpy(chNomFichierObjets,m_chNomFichier);
strcat(chNomFichierObjets,".lg");
FILE* FileObjets=fopen(chNomFichierObjets,"rb");
if(FileObjets == NULL)
   {
   ::ErreurDeFichier(chNomFichierObjets);
   if(m_ctabCle != NULL)free(m_ctabCle);
   if(m_dtabValeur != NULL)free(m_dtabValeur);
   m_ctabCle=NULL;
   m_dtabValeur=NULL;
   return FALSE;
   }

int iTailleBuffer=8; //--- pour nbpt et sens
if(m_iTypeSaisie == SAISIE_VALEUR || m_iTypeSaisie == SAISIE_CLEVALEUR)iTailleBuffer+=8;
if(m_iTypeSaisie == SAISIE_CLE || m_iTypeSaisie ==
SAISIE_CLEVALEUR)iTailleBuffer+=m_iNbcharCle;

int iIndice=0;
int i,iNbpt,iSens;while(fread(buffer,iTailleBuffer,1,FileObjets) == 1)
   {
   memcpy(&iNbpt,&buffer[0],4);
   memcpy(&iSens,&buffer[4],4);

   switch(m_iTypeSaisie)
     {
     case SAISIE_VALEUR:
      memcpy(&m_dtabValeur[m_iNbObjet],&buffer[8],8);
      break;
     case SAISIE_CLE:
      memcpy(&m_ctabCle[m_iNbObjet*m_iNbcharCle],&buffer[8],m_iNbcharCle);
      break;
     case SAISIE_CLEVALEUR:
      memcpy(&m_dtabValeur[m_iNbObjet],&buffer[8],8);
      memcpy(&m_ctabCle[m_iNbObjet*m_iNbcharCle],&buffer[16],m_iNbcharCle);
      break;

     default:
      break;
     }
   //--- lecture de l'arc
   m_ptabArc[m_iNbArc]=new CArcMygale(iNbpt);
   if(m_ptabArc[m_iNbArc] == NULL)
     {
     ::ErreurDeMemoire();
     fclose(FileObjets);

     if(m_ctabCle != NULL)free(m_ctabCle);
     if(m_dtabValeur != NULL)free(m_dtabValeur);
     m_ctabCle=NULL;
     m_dtabValeur=NULL;
     for(i=0;i < NB_MAX_ARC;i++)
      {
      if(m_ptabArc[i] != NULL)delete m_ptabArc[i];
      m_ptabArc[i]=NULL;
      }
     return FALSE;
     }

   for(i=0;i < iNbpt;i++)
     {
     if(fread(buffer,16,1,FileObjets) != 1)
      {
      ::ErreurDeFichier(chNomFichierObjets);
      fclose(FileObjets);

      if(m_ctabCle != NULL)free(m_ctabCle);
      if(m_dtabValeur != NULL)free(m_dtabValeur);
      m_ctabCle=NULL;
      m_dtabValeur=NULL;
      for(int i=0;i < NB_MAX_ARC;i++)
         {
```

```
       if(m_ptabArc[i] != NULL)delete m_ptabArc[i];
       m_ptabArc[i]=NULL;
       }
     return FALSE;
     }
    memcpy(&point.m_dX,&buffer[0],8);
    memcpy(&point.m_dY,&buffer[8],8);
    m_ptabArc[m_iNbArc]->SetPointGrow(i,point);
    }
  m_ptabArc[m_iNbArc]->SetNbpt(iNbpt);
  m_ptabArc[m_iNbArc]->SetSens(iSens);

  //--- calcul de la fenêtre de l'arc
  m_ptabArc[m_iNbArc]->CalculFenetreSav();

  m_iNbArc++;
  m_iNbObjet++;
  }
fclose(FileObjets);
```

Enfin, pour les zones, la procédure de lecture est la suivante :

```
FILE* FileObjets=fopen(chNomFichierObjets,"rb");
if(FileObjets == NULL)
   {
   ::ErreurDeFichier(chNomFichierObjets);

   free(m_dtabPoint);
   if(m_ctabCle != NULL)free(m_ctabCle);
   if(m_dtabValeur != NULL)free(m_dtabValeur);
   m_dtabPoint=NULL;
   m_ctabCle=NULL;
   m_dtabValeur=NULL;
   return FALSE;
   }

int iTailleBuffer=20;
if(m_iTypeSaisie == SAISIE_VALEUR || m_iTypeSaisie == SAISIE_CLEVALEUR)iTailleBuffer+=8;
if(m_iTypeSaisie == SAISIE_CLE || m_iTypeSaisie ==
SAISIE_CLEVALEUR)iTailleBuffer+=m_iNbcharCle;

int iNoZone,iIndice;
double dLong,dColat;

iIndice=0;
while(fread(buffer,iTailleBuffer,1,FileObjets) == 1)
   {
   memcpy(&iNoZone,&buffer[0],4);
   memcpy(&dLong,&buffer[4],8);
   memcpy(&dColat,&buffer[12],8);

   m_itabNoZone[m_iNbObjet]=iNoZone;
   if(iNoZone >= m_iNoZoneLibre)m_iNoZoneLibre=iNoZone+1;

   m_dtabPoint[iIndice]=dLong;
   iIndice++;
   m_dtabPoint[iIndice]=dColat;
   iIndice++;

   switch(m_iTypeSaisie)
     {
     case SAISIE_VALEUR:
      memcpy(&m_dtabValeur[m_iNbObjet],&buffer[20],8);
      break;
     case SAISIE_CLE:
      memcpy(&m_ctabCle[m_iNbObjet*m_iNbcharCle],&buffer[20],m_iNbcharCle);
      break;
     case SAISIE_CLEVALEUR:
      memcpy(&m_dtabValeur[m_iNbObjet],&buffer[20],8);
      memcpy(&m_ctabCle[m_iNbObjet*m_iNbcharCle],&buffer[28],m_iNbcharCle);
      break;

     default:
      break;
     }
   m_iNbObjet++;
   }
fclose(FileObjets);
```

```
//--- lecture des arcs
FILE* FileArcs=fopen(chNomFichierArcs,"rb");
if(FileArcs == NULL)
   {
   ::ErreurDeFichier(chNomFichierArcs);

   free(m_dtabPoint);
   if(m_ctabCle != NULL)free(m_ctabCle);
   if(m_dtabValeur != NULL)free(m_dtabValeur);
   m_dtabPoint=NULL;
   m_ctabCle=NULL;
   m_dtabValeur=NULL;
   return FALSE;
   }

iTailleBuffer=16; //--- pour nbpt,sens,iNz1,iNz2
int i,iNbpt,iSens,iNz1,iNz2;
CPointMygale point;
while(fread(buffer,iTailleBuffer,1,FileArcs) == 1)
   {
   memcpy(&iNbpt,&buffer[0],4);
   memcpy(&iSens,&buffer[4],4);
   memcpy(&iNz1,&buffer[8],4);
   memcpy(&iNz2,&buffer[12],4);

   //--- lecture de l'arc
   m_ptabArc[m_iNbArc]=new CArcMygale(iNbpt);
   if(m_ptabArc[m_iNbArc] == NULL)
     {
     ::ErreurDeMemoire();
     fclose(FileArcs);

     free(m_dtabPoint);
     if(m_ctabCle != NULL)free(m_ctabCle);
     if(m_dtabValeur != NULL)free(m_dtabValeur);
     m_dtabPoint=NULL;
     m_ctabCle=NULL;
     m_dtabValeur=NULL;
     for(i=0;i < NB_MAX_ARC;i++)
      {
      if(m_ptabArc[i] != NULL)delete m_ptabArc[i];
      m_ptabArc[i]=NULL;
      }
     return FALSE;
     }

   for(i=0;i < iNbpt;i++)
     {
     if(fread(buffer,16,1,FileArcs) != 1)
      {
      ::ErreurDeFichier(chNomFichierArcs);
      fclose(FileArcs);

      free(m_dtabPoint);
      if(m_ctabCle != NULL)free(m_ctabCle);
      if(m_dtabValeur != NULL)free(m_dtabValeur);
      m_dtabPoint=NULL;
      m_ctabCle=NULL;
      m_dtabValeur=NULL;
      for(int i=0;i < NB_MAX_ARC;i++)
         {
         if(m_ptabArc[i] != NULL)delete m_ptabArc[i];
         m_ptabArc[i]=NULL;
         }
      return FALSE;
      }
     memcpy(&point.m_dX,&buffer[0],8);
     memcpy(&point.m_dY,&buffer[8],8);
     m_ptabArc[m_iNbArc]->SetPointGrow(i,point.m_dX,point.m_dY);
     }
   m_ptabArc[m_iNbArc]->SetNbpt(iNbpt);
   m_ptabArc[m_iNbArc]->SetSens(iSens);
   m_ptabArc[m_iNbArc]->SetNoZone1(iNz1);
   m_ptabArc[m_iNbArc]->SetNoZone2(iNz2);

   //--- calcul de la fenêtre de l'arc
   m_ptabArc[m_iNbArc]->CalculFenetreSav();

   m_iNbArc++;
```

```
  }
fclose(FileArcs);
```

RESUME

Cet ouvrage présente un travail de recherche et de développement informatique visant à apporter une réponse concrète à la question suivante : « comment construire un système d'information géographique complet et opérationnel, en suivant les principes théoriques de la gestion de données et en les adaptant aux données géographiques ? ».

Le sujet principal de cet ouvrage est de montrer sur un exemple concret l'implication des principes théoriques de la gestion de données dans la réalisation pratique d'un logiciel de type SIG. Elle présente l'architecture et la réalisation d'un système d'information géographique opérationnel – le système SAVANE - à partir des nombreux concepts, techniques et algorithmes nécessaires à cette réalisation. Ce travail de recherche a été effectué dans le cadre de l'IRD (Institut de recherche pour le développement).

L'ouvrage expose l'ensemble des principes nécessaires à la réalisation et à la bonne utilisation d'un SIG, en décrivant chaque aspect nécessaire à la construction du système et en expliquant les choix effectués, selon la démarche suivante :

- exposer avec précision tous les aspects concernant la définition et l'utilisation de l'information géographique ;
- exposer et développer les principes de la gestion de données (modèle relationnel et objet) pour les étendre aux données localisées ;
- développer ou mettre en œuvre les algorithmes nécessaires à l'implantation de ces principes dans un système d'information ;
- construire un système opérationnel, mettant en œuvre les principes théoriques et répondant à un cahier des charges fonctionnel couvrant l'ensemble des besoins nécessaires à l'utilisation de ce système dans le cadre de projets appliqués, notamment en géographie et dans le cadre de la recherche pour le développement..

L'exposé des méthodes est souvent suivi de la présentation d'algorithmes et de leur réalisation concrète. Des références renvoient fréquemment le lecteur à une annexe contenant l'implantation effective des structures et des algorithmes.