Ministère de l'Education Nationale

UNIVERSITE MONTPELLIER II

UFR

IUP

GENIE MATHEMATIQUE ET INFORMATIQUE

RAPPORT DE STAGE

effectué à

« l'Institut de Recherche pour le Développement »

dans les locaux de la « Maison de la Télédétection »

du 01/09/2006 au 22/12/2006

par Fouré Olivier

Directeurs de Stage :

M. SOURIS Marc

Tuteur pédagogique :

Mme AHRONOVITZ Yolande

Quito 3D

Développement d'application interactive 3D

de visite virtuelle en milieu urbain.

Remerciements :

Je tiens à remercier ceux qui de près ou de loin ont contribué à la réalisation de ce travail.

Je remercie plus particulièrement :

M. Duminil Julien, sans qui rien n'aurait été aussi loin.

M. Souris Marc, pour ce sujet de stage, son aide et ses conseils judicieux.

Mme Martiny Monique, pour son accueil chaleureux au sein de l'unité espace.

M. Demoraes Florent, pour toute l'aide et le temps qu'il nous a consacré.

Sans oublier, Chloé, Mme Ahronovitz Yolande, Mme Paulet Régine, M. Huynh Frédéric, M. Auberval Nicolas, Mlle Pecriaux Magalie et les stagiaires de la MTD.

Sommaire

I. Résumé du stage

A l'occasion d'une exposition sur la cartographie de la ville de Quito en Equateur, la mairie de Quito à demandé à l'Institut de Recherche pour le Développement de réaliser une application interactive performante de visite virtuelle sur la ville pour le grand public.

L'Institut de Recherche pour le Développement (IRD), anciennement connu sous le nom d'ORSTOM est un établissement public français à caractère scientifique et technologique. Ses travaux de recherches recouvrent trois grands domaines : Les Milieux Et Environnements, Les Ressources Vivantes et Les Sociétés Et Santé. Ses missions sont la recherche, l'expertise et valorisation, le soutien et la formation, et l'information scientifique.

Ce stage s'est effectué au sein de la Maison de la Télédétection (MTD), complexe constituant un pole de recherche appliquée en télédétection et information géographique. Ses activités sont structurées autour de produits issus du traitement de données et d'informations spatialisées. Le cadre général est celui de l'aide à la gestion de l'environnement, des ressources et des territoires.

Bien qu'il existe plusieurs applications permettant de modéliser une zone géographique, aucune n'est adaptée à l'utilisation pour le grand public. C'est pourquoi il nous a été demandé de développer une solution.

Sur le modèle d'un simulateur de vol, le logiciel permet d'explorer une zone de 880 Km² autour de Quito et ce de plusieurs façons : comme un avion, comme un piéton, de façon libre, mais aussi à l'aide de circuits prédéfinis.

Afin de rester fidèle au relief de la région de Quito, d'assurer la fluidité nécessaire à la simulation, le logiciel emploie les techniques les plus performantes en visualisation de terrain virtuel : gestion dynamique du niveau de détail, chargement de données orienté multiprocesseur, algorithme de simplification du relief, utilisation optimale de la carte graphique.

Pour répondre efficacement à ces besoins, l'application est développée en C++ à l'aide de Visual Studio 2005 et utilise les bibliothèques de DirectX 9.

Etant destiné au grand public, la simplicité d'utilisation et la fiabilité du logiciel ont été des éléments très importants de la réalisation.

II. Etat de l'art

Dans ce chapitre sont décrits les grandes tendances actuelles dans le secteur des techniques de visualisation.

1. Vue d'ensemble

Tout simulateur s'appuie sur des données de relief, fournies par des satellites la plupart du temps sous la forme d'une grille d'altitude. Une telle grille est un tableau de valeurs à deux dimensions. Il est caractérisé par sa largeur, sa longueur et sa précision. Par exemple une grille d'altitude de 20000 x 20000 avec 5m de précision recouvre une surface totale de (20000 * 5) x (20000 * 5) m².

Le problème qui se pose concerne l'affichage de ce relief. En effet une scène virtuelle est composée d'une multitude de triangles, car c'est ce que les cartes graphiques gèrent le mieux, or si l'on essaie naïvement de créer tous les triangles entre les valeurs de la grille d'altitude on se retrouve dans le cas suivant :

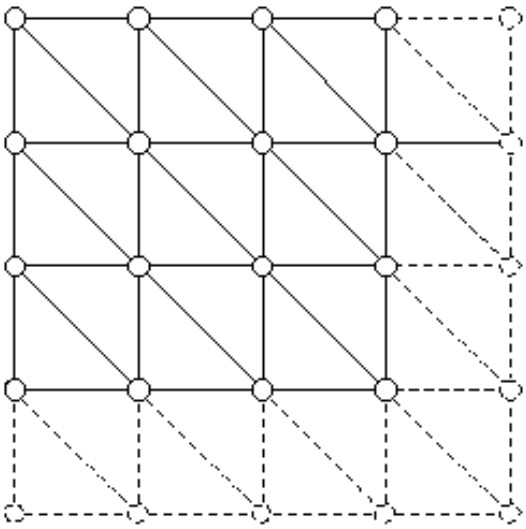


Figure 1 : Maillage complet d'une grille d'altitude

Nombre de triangle maximal :

(20000 – 1) * (20000 – 1) * 2

= 800 millions de triangles

Ce qu'il faut préciser c'est que ces 800 millions de triangles doivent être envoyés à la carte graphique à chaque image… ce qui est possible si l'on peux se contenter d'une image toute les 10 secondes (sur du matériel récent bien entendu)

Pour contourner ce problème, d'importantes recherches sont menées, principalement au niveau algorithmique.

Les résultats de ces recherches permettent aux développeurs d'utiliser des techniques de visualisation de plus en plus performantes, en terme de vitesse d'affichage et de précision pour des applications en temps réel.

Les principaux utilisateurs d'applications utilisant ces techniques sont des chercheurs, des collectivités territoriales, des professionnels du graphisme ou de simples possesseurs d'ordinateur qui utilisent des logiciels tels que Google Earth, Nasa World Wind ou tout simplement les derniers jeux à la mode.

De ces recherches ressortent deux techniques de visualisations, celles-ci permettent d'avoir un regard synthétique sur ce qui se fait.

2. Algorithme ROAM (Real-time Optimally Adapting Meshes) par M. Duchaineau

Cet algorithme permet de simplifier la géométrie du terrain en temps réel.
Le principe est le suivant :

- Initialisation :
  - Un ensemble de triangles est pré-calculé et est gardé en mémoire sous forme d'arbre binaire.

- Boucle d'affichage :
  - On calcule l'ensemble de triangles de la manière suivante : deux files de priorités basées sur le taux d'erreur du relief sont utilisées pour diviser et fusionner les triangles de l'image précédente. Seul un sous ensemble de triangles est modifié et on ne divise ni fusionne les triangles tout juste créés.
  - On affiche cet ensemble de triangles.

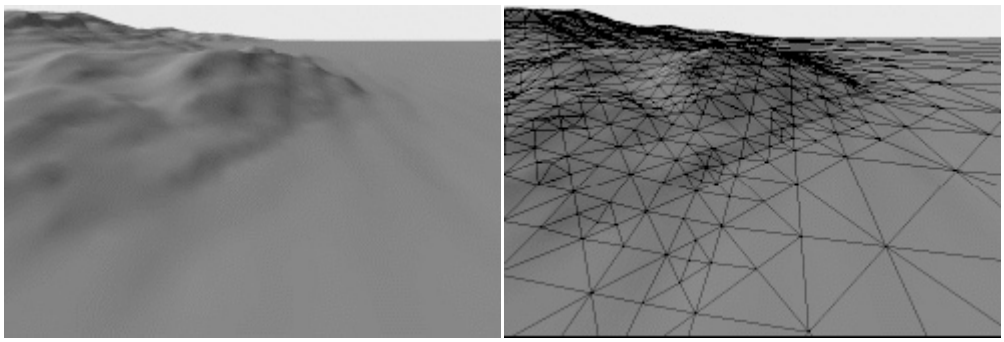


Figure 2 : Illustration de l'algorithme ROAM

Les principaux avantages de cette techniques sont :

- Cohésion temporelle :

D'une image à l'autre on ne modifie pas plus d'une fois les triangles. Ce qui permet d'avoir des transitions douces.

- La modification du terrain se fait en priorité là où le taux d'erreurs est le plus important.

Maintenant les inconvénients :

- Cette méthode nécessite la mémorisation des informations de chaque triangle, on doit en plus des informations de géométrie avoir une référence vers chaque triangle voisin, les besoins en mémoire sont importants.

3. Continuous LOD (Level Of Detail) par P. Lindstrom

Chaque bloc peut être décomposé en une grille de sous-blocs de même dimension.

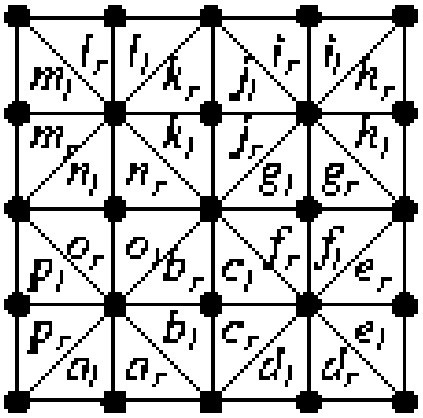


Figure 3 : Continuous LOD

Pour ne pas avoir à considérer tous les points de la surface pour la simplification, l'algorithme procède par bloc.

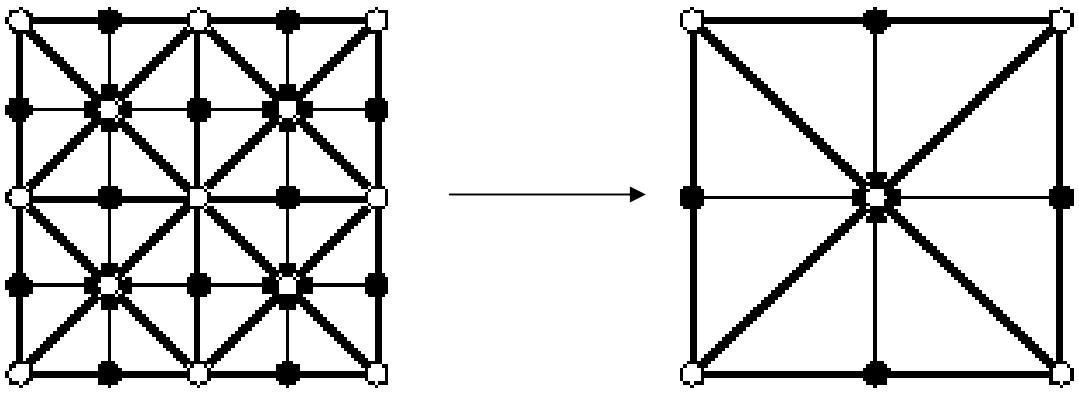


Figure 4 : Simplification avec Continuous LOD

Les principaux avantages de cette techniques sont :

- Simplification par bloc, simplifie la charge du processeur.
- Pas besoin de stocker les informations sur chaque triangle, seul une matrice permet de savoir si l'on a divisé ou pas.

Maintenant les inconvénients :

- On recalcule tout le relief à chaque image, on ne fait que diviser.

III. Solution élaborée

1. Principe de l'algorithme

Tout d'abord il faut préciser que l'on utilise des grilles d'altitude carrées de taille égale à 2^n + 1 (avec n >= 1). Grâce à ces dimensions il est facile de partitionner la carte à l'aide d'un réseaux régulier de triangles isocèles, en effet si l'on divise un triangle le nouveau point créé fait partie de la grille originale.

A l'initialisation la carte est composée de deux triangles. Ensuite en fonction du relief local on divise les triangles recouvrant une zone pas assez détaillée.

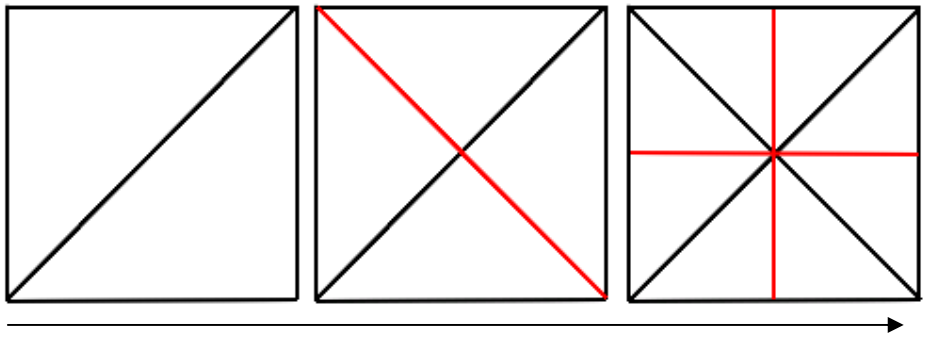


Figure 5 : Division

Cette technique s'appelle « Niveaux de détail dynamique ». En effet en fonction de plusieurs critères on va augmenter ou diminuer le nombre de triangles en une certaine zone de la carte.

Ces critères sont les suivants :

- **Le taux d'erreur du relief local**, qui est calculé à l'initialisation de façon récursive. Il représente la différence entre les altitudes réelles et le relief affiché par le triangle

recouvrant. Voici l'algorithme permettant de calculer le taux d'erreur pour chaque triangle :

- E = [0.0, 0.0, 0.0, …] (arbre binaire)
  iA = indice de A (dans la grille d'altitudes)
  h(iA) = altitude du point A

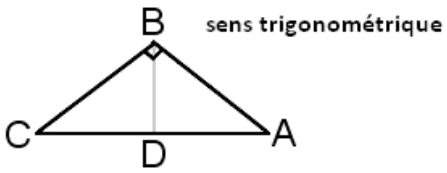


Figure 6 : Notation

- Etape(iA, iB, iC)
  iD = (iA + iC) / 2
  si iD existe
    - Err1 = Etape(iC, iD, iB)
    - Err2 = Etape(iB, iD, iA)
    - E[iD] = max(E[iD], |(h(iC) + h(iA)) / 2 – h(iD)| + max(Err1, Err2))
    - retourner E[iD]

  sinon (on se situe à la plus petite division possible, la grille originale)
    - retourner 0.0

On effectue ce calcul à l'initialisation de la carte en appelant deux fois la fonction « Etape(…) » avec les deux triangles de départ, ceux qui couvrent la carte entièrement.

Cet algorithme signifie que le taux d'erreur d'un triangle correspond au maximum entre le taux d'erreur maximum de ses deux triangles fils et la différence entre les altitudes réelles et le relief affiché par ce triangle recouvrant.

Ce critère permet de simplifier de grandes étendues et de préciser les zones où le relief est important.

- **La distance**. On l'utilise la distance au carré pour accentuer l'effet de détail proche et réduire plus encore les détails lointains.

- **L'angle de vue**, en effet si la caméra est pointée vers le sol l'œil ne distingue plus le relief, on peux donc diminuer le nombre de triangles.

- **L'altitude**. Plus on est proche du sol plus les détails sont importants.

- **Le champs de vision.** On peux déterminer si un triangle se situe dans la partie visible de la carte, on calcule les six plans de la pyramide de visibilité et si pour un des plans les trois points du triangle sont du coté non visible alors le triangle n'est pas visible. On considère qu'un triangle partiellement visible est visible.

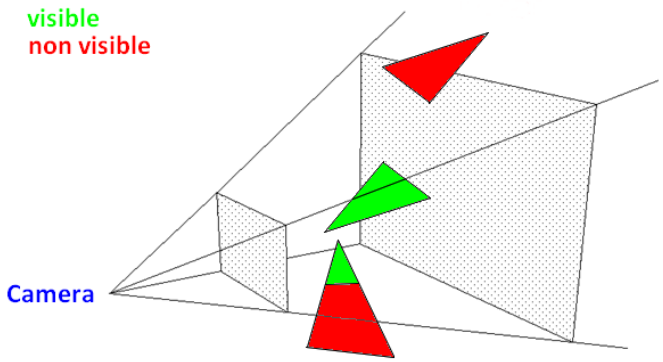


Figure 7 : Pyramide de visibilité

En croisant ces critères on obtient un filtre qui va permettre, par exemple, d'afficher une montagne lointaine tout en simplifiant une zone plane proche.

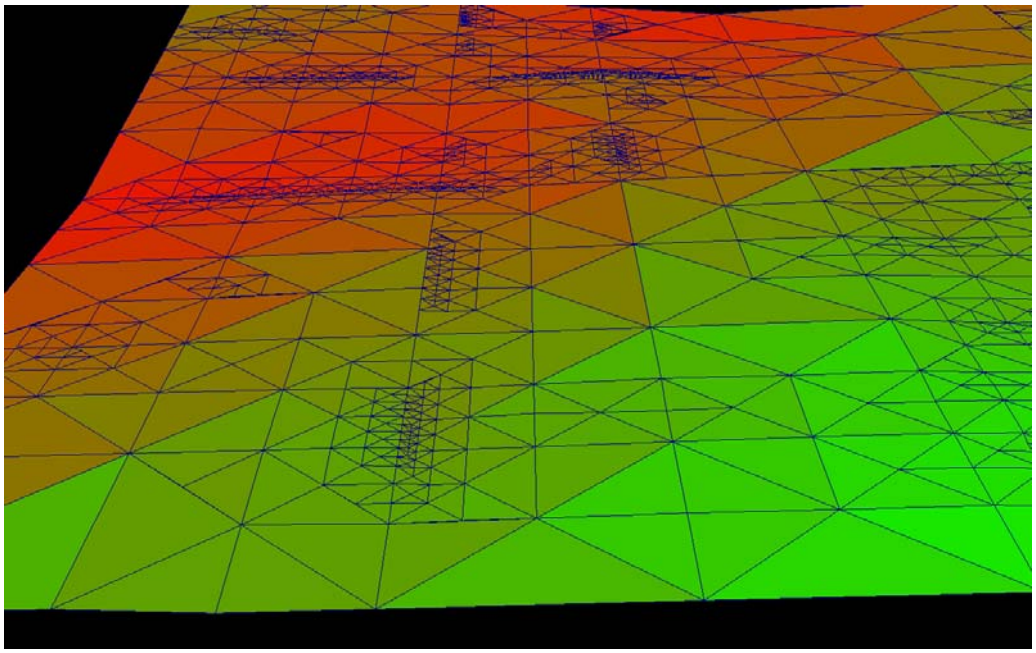


Figure 8 : Gestion du détail dynamique

Un problème important survient lorsqu'on travaille avec plusieurs niveaux de détails. En effet, si on divise un triangle sans tenir compte des triangles voisins on produit du « cracking ». Visuellement cela se traduit par une faille dans le terrain.

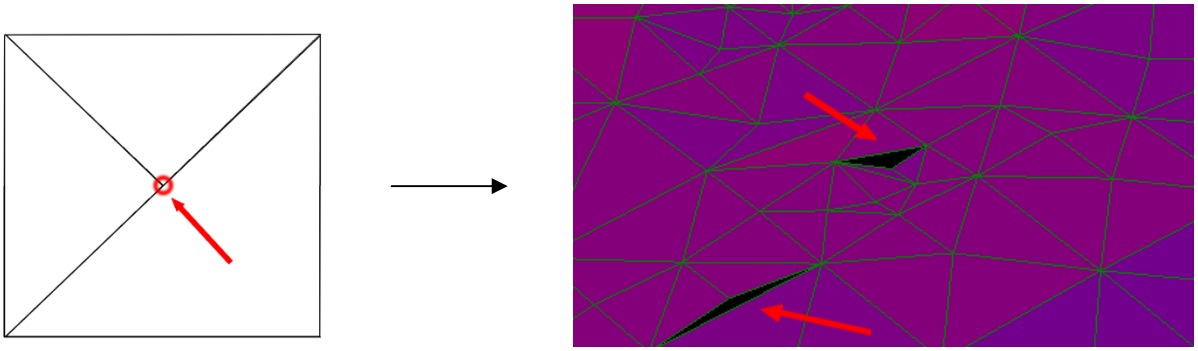


Figure 9 : Cracking ou discontinuité du maillage

La solution à ce problème est assez simple. A chaque fois que l'on divise un triangle il faut vérifier s'il possède un triangle voisin du coté de son hypoténuse, si c'est le cas alors il faut diviser ce triangle. Pour faire cette division il faut vérifier si le triangle est au même niveau que son voisin, si c'est le cas on divise les deux triangles. Dans le cas contraire il faut diviser le triangle parent au voisin et propager la division de voisin en voisin jusqu'à arriver à un voisin du même niveau ou si il n'y a pas de voisin (on est au bord de la carte).

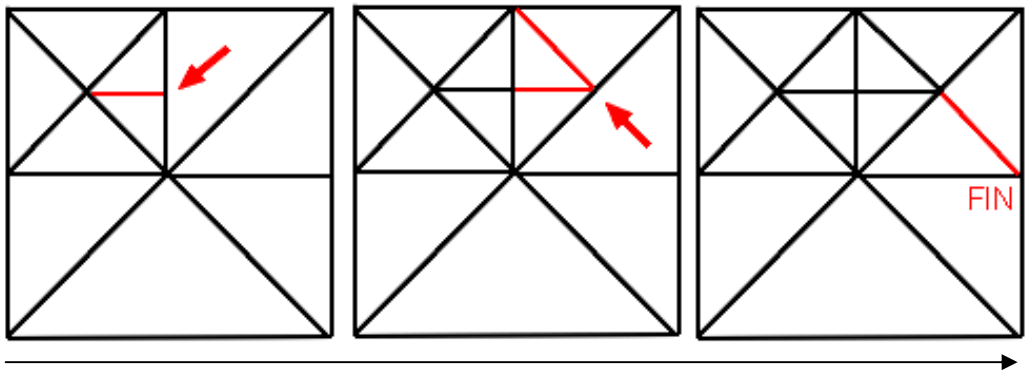


Figure 10 : Solution au cracking (voir annexe pour un schéma plus détaillé)

Le niveau d'un triangle est le nombre de divisions nécessaires à l'obtention de ce triangle depuis le triangle de base.

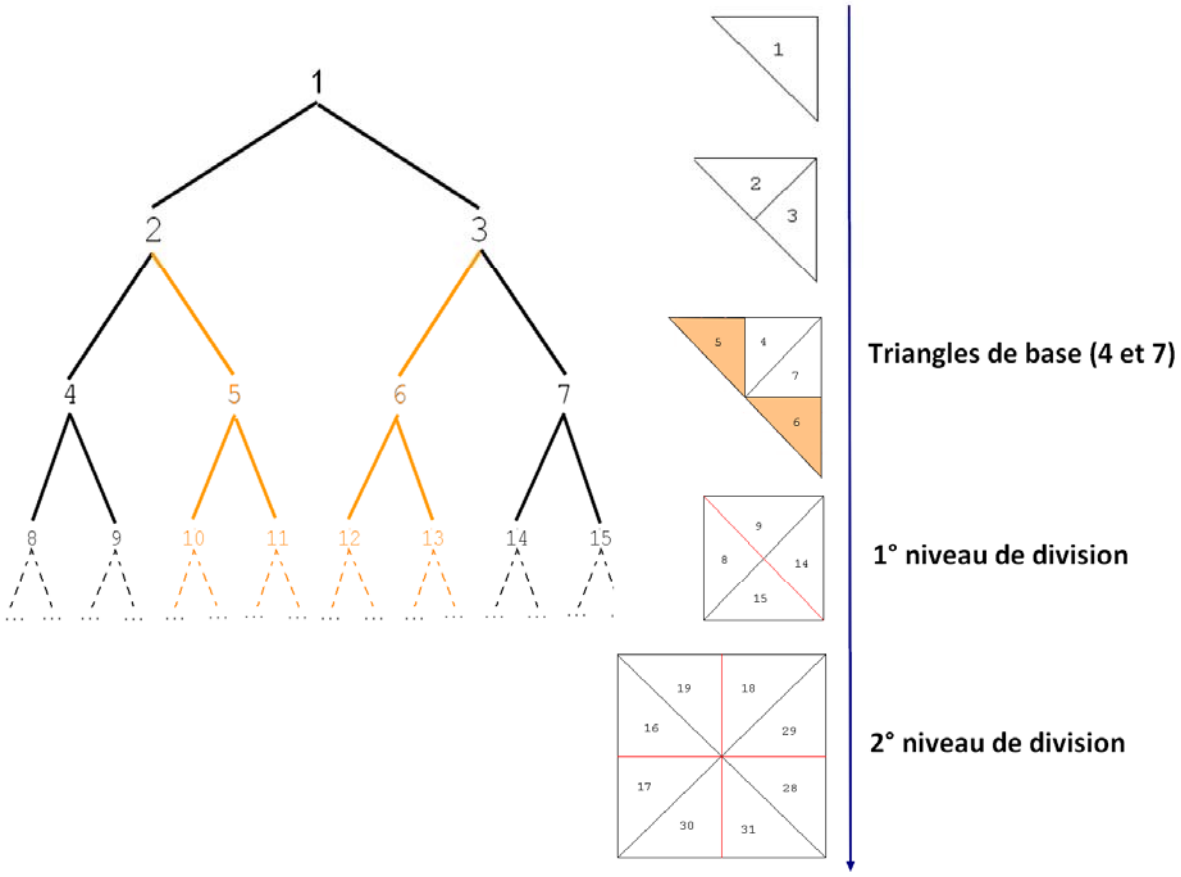


Figure 11 : Relation arbre binaire <=> découpage

Nous avons choisit les triangles 4 et 7 comme triangles de base, car parmi les méthodes que nous avons utilisé, nous calculons le numéro du triangle voisin. Et ce calcul impose de partir de deux triangles voisins au niveau de l'hypoténuse ayant le même ancêtre, configuration qui se retrouve avec le triangles 4 et 7.

2. Méthodes utilisées

La première méthode élaborée à recréer à chaque image la totalité de la géométrie du terrain, en divisant récursivement les deux triangles de base jusqu'à atteindre le taux d'erreur voulu.

L'algorithme correspondant est le suivant :

Soit la fonction / Affiche(triangle tri)

```
si OnPeutDiviser(tri) et OnDoitDiviser(tri)
        Affiche(FilsGauche(tri))
        Affiche(FilsDroit(tri))
Sinon
        Dessiner(tri) // qui affiche réellement le triangle
```

Boucle d'affichage :

```
Affiche(triangle de base 1)
Affiche(triangle de base 2)
```

La fonction OnPeutDiviser(triangle) permet de vérifier si l'on a atteint le maillage original (auquel cas on ne peux plus diviser).

OnDoitDiviser(triangle) décide en fonction des critères de détails si le triangle doit être divisé.

La simplicité de cette méthode a fait qu'elle fut la première implémentée.

Mais elle possède un inconvénient, en effet la complexité n'est pas adaptée à l'affichage d'un grand nombre de triangles.

Soit une grille d'altitude de 1025 * 1025, on sait que 2^10 + 1 = 1025. Si on affiche le maillage maximal alors on obtient 2^10 * 2^10 triangles ce qui fait environ 1 million.

Sachant que le transfert de données vers la carte graphique prend un peu de temps (de l'ordre de quelques milli secondes), que la complexité dépend du nombre de triangle et qu'à chaque image tout les triangles sont finalement transférés vers la carte graphique, il est clair que cet algorithme n'est pas adapté à l'affichage d'un grand nombre de triangles.

C'est pourquoi il nous a fallu amener diverses améliorations, la principale étant de travailler sur un ensemble de triangles qui est gardé en mémoire dans la carte graphique.

Au lieu de partir de deux triangles de base et de reconstruire toute la géométrie, on se contente de modifier celle de l'image précédente et ce directement dans la mémoire de carte graphique. Avec cette méthode on implémente la fusion de triangles, sinon le nombre de triangles ne ferait que croître.

La principe de la fusion est simple, si on ne doit pas diviser le triangle que l'on traite ni son « frère » alors on supprime les deux triangles pour les remplacer par le triangle « parent ».

Tout comme la division si on ne tient pas compte du triangle « voisin » on va produire du « cracking ». A cette étape si le triangle « parent » du triangle que l'on traite possède un triangle « voisin » d'hypoténuse alors ce « voisin » doit avoir deux « fils » car auparavant on a fait attention à éviter le cracking. Dans ce cas il faut aussi fusionner ces deux triangles « cousins ».

Dans la figure qui suit si A1 est le triangle courant alors A2 est son frère, A leur père, B le voisin du père, B1 et B2 les fils de B mais surtout cousins de A1 et A2.

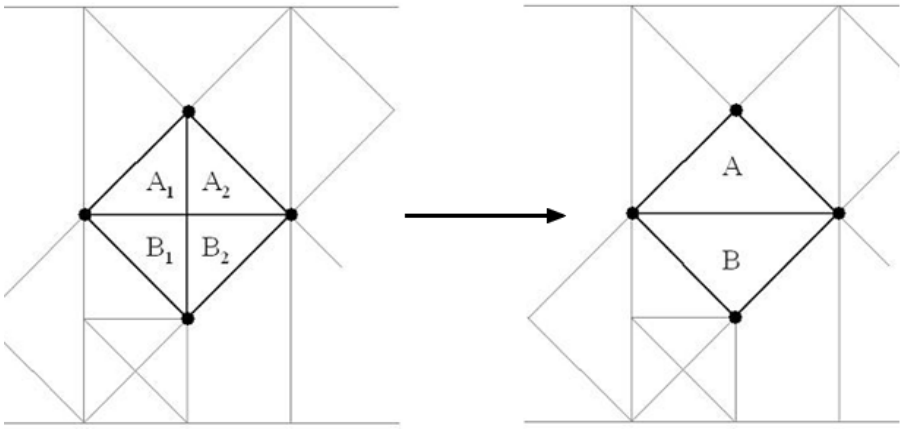


Figure 12 : Fusion

Le seul cas où l'on ne doit pas fusionner les « cousins » c'est quand on divise un triangle dont l'hypoténuse fait partie du bord de la carte.

Cette méthode permet de minimiser les échanges entre la carte graphique et le processeur, passage obligé qui reste un goulot d'étranglement pour les performances. De plus à chaque fois que l'on calcule la géométrie on ne modifie qu'une portion de l'ensemble des triangles ce qui permet de limiter le temps passé à calculer la géométrie.

Garder les informations de géométrie dans la carte graphique permet plusieurs choses :

- On peux réutiliser les coordonnées des points qui composent un triangle pour un autre triangle, du coup on économise de la mémoire (ce qui est important dans une carte graphique).
- On ne fait plus qu'un appel système pour indiquer les triangles que l'on souhaite afficher, ce qui fait gagner énormément de temps par rapport à la méthode précédente.

En réalité seules les positions des points sont stockés dans la carte graphique, toutes les informations sur les voisins des triangles affichés sont stockés dans la mémoire vive. De cette façon on n'accède à la mémoire de la carte graphique qu'en mode écriture ce qui accélère encore les échanges matériels. Bien que cela prenne relativement beaucoup de mémoire les ordinateurs actuels sont équipés pour supporter pareille charge.

Cette dernière méthode permet de limiter le nombre de triangles par image, le temps passé à calculer la géométrie et donc au final le nombre d'images par seconde.

Voici son l'algorithme :

Initialisation :

Triangles = [triangle de base 1, triangle de base 2]

A chaque image :

Pour 10% des triangles de l'ensemble Triangles

Si OnPeutDiviser(triangle courant) et OnDoitDiviser(triangle courant)

Diviser(triangle courant)

Sinon si OnPeutFusionner(triangle courant) et OnDoitFusionner(tri courant)

Fusionner(triangle courant)

Fin si

Fin pour

Dessiner(Triangles)

Cette méthode permet d'obtenir un nombre d'image par seconde bien plus important que la méthode précédente, surtout lorsque l'on affiche beaucoup de triangles. De plus on évite un trop brusque changement de niveau de détails, phénomène appelé « popping »

3. Gestion d'une mosaïque de cartes

Pour parvenir à faire tourner l'application sur une grande variété d'ordinateurs actuels, on découpe la totalité de la surface à simuler en une mosaïque de cartes carrées de deux kilomètres de large. En effet, les zones éloignées n'ayant pas besoin d'être détaillées on utilise des données moins précises en fonction de la distance à la caméra, ce qui permet d'alléger la mémoire de l'ordinateur tout en ayant une qualité d'image maximale.

Chaque élément de la mosaïque est une carte qui possède deux types de données :

- Les données de relief, une grille d'altitude aussi appelée « MNT » (Modèle Numérique de Terrain).
- Une image satellite, appelée « texture ».

A partir des données originales on produit plusieurs versions de moins en moins précises. On obtient des MNT et des textures avec une précisions de plus en plus basses. Cet ensemble de fichier permet d'avoir plusieurs niveaux de détails pour une même carte.

Ces données ont été produite en dehors de l'exécution du logiciel et ne nécessitent aucun traitement supplémentaire à l'exécution.

En l'occurrence on modélise une zone de 20000 m * 44000 m avec une image satellite de 1 m de précision. Une telle image fait environ 3 Go en utilisant un format non compressé sans perte de qualité. Sachant que les cartes graphiques les plus récentes n'embarquent pas plus de 512 Mo de mémoire vive, on comprend tout l'intérêt de ne pas charger toute la zone en qualité maximale.

Afin d'avoir une bonne qualité visuelle malgré le peu de mémoire de la carte graphique, il suffit de charger en qualité maximale les images satellites des cartes proches et de laisser les autres images des cartes en qualité inférieure.

On retrouve ce problème avec les données de relief, en effet le MNT original fait environ 800 Mo. Pour les mêmes raisons on utilise les données de précision maximale que pour les cartes proches.

Le chargement des toutes ces données s'effectue dans un processus en parallèle avec une priorité inférieure. Cela qui permet de ne pas bloquer le logiciel pendant le chargement des ressources. Ainsi au lancement du logiciel on peux voir l'ensemble de la mosaïque de cartes se charger au fur et à mesure, d'abord le relief et ensuite l'image satellite.

Une fois ce premier chargement terminé, le gestionnaire de l'ensemble de cartes décide, pendant l'étape du calcul de l'image, quelles données doivent être chargées en fonction de la distance.

La présence de plusieurs cartes avec plusieurs niveaux de détails différents entraine un autre problème, il s'agit du « cracking » mais au niveau au dessus. En effet deux cartes adjacentes n'ont pas forcément le même relief à cet endroit car cela dépend du taux d'erreur, de la distance, etc.

Pour résoudre ce problème on utilise la technique de la « minijupe », technique qui consiste à habiller chaque carte d'un contour vertical le long des bords. Cela ne requiert que quelques triangles en plus et presque aucun calcul. Sa simplicité et son efficacité visuelle a rendu incontournable son utilisation.

IV. Mise en œuvre

1. Technologies employées

Le programme est développé en C++, langage qui garantit des performances inégalées pour un langage orienté objet.

La bibliothèque de fonctions DirectX 9 permet d'accéder simplement à l'affichage de scènes numériques, à l'utilisation de fichiers sonores, au chargement de divers formats d'images, à des éléments d'interface personnalisable, etc.

L'utilisation de cette bibliothèque implique que le programme ne pourra fonctionner que sous le système d'exploitation Microsoft Windows.

Dans le logiciel il existe plusieurs manières de se déplacer, l'une d'elle est un déplacement automatique. Il permet de déplacer la caméra le long d'une suite de couple « position, direction » avec une simple interpolation linéaire.

Ces parcours sont définis dans un script externe écrit en Lua. Ce qui fait la force de Lua c'est que les scripts ne sont pas compilés, c'est à dire que si on souhaite ajouter un parcours, il ne faut pas recompiler le logiciel, il suffit de modifie le fichier Lua en respectant une syntaxe très simple, en effet le script sera chargé au lancement de l'application.

2. Structures de l'application

En plus de l'algorithme lui-même, l'implémentation à une grande influence sur les performances. Je vais expliquer plus en détails les structures utilisées pour l'implémentation de notre deuxième méthode.

Tout d'abord il faut savoir comment sont stockées les informations de relief dans la mémoire de la carte graphique que l'on nommeras GPU (Graphic Process Unit).

Chaque donnée de relief est composée de cinq valeurs :

- Trois pour les coordonnées spatiales.
- Deux pour les coordonnées de texture.

Ces données sont stockées sous la forme d'un simple tableau à une dimension dans la mémoire GPU, nommé « Vertex Buffer ».

Pour afficher des triangles, il suffit de préciser GPU quels sommets sont utilisés, avec un tableau d'indices, appelé « Index Buffer ». Ce tableau est aussi stocké dans la mémoire GPU.

Par exemple le schéma suivant montre que nous avons quatre sommets dans le « Vertex Buffer » et que nous avons deux triangles dans l' « Index Buffer », le premier utilise les sommets 0, 1 et 2 et le deuxième les sommets 3, 0 et 2.

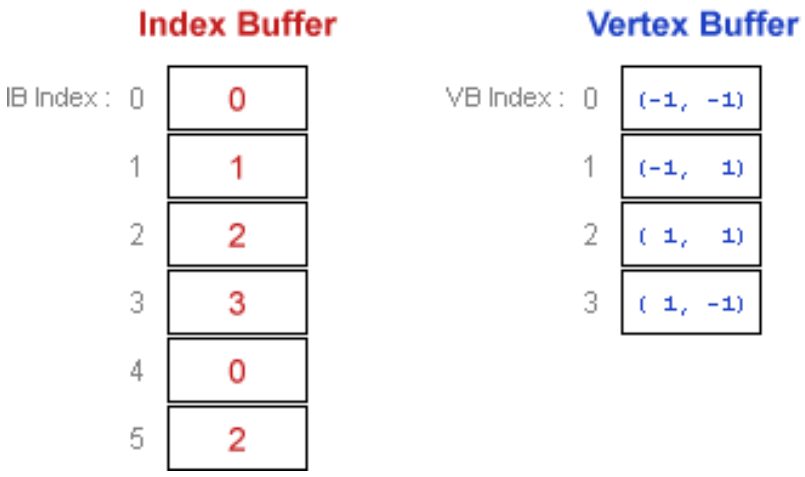


Figure 13 : Buffer de vertices indexé

Notre algorithme requiert la modification de cet espace mémoire fréquemment, on doit soit ajouter un sommet et trois indices (quand on divise un triangle) soit en retirer un et trois indices aussi (suite à la fusion de deux triangles).

Pour qu'il n'y ai pas de fuite mémoire ni d'erreurs, nos tableaux « Index Buffer » et « Vertex Buffer » ne doivent pas contenir de trous. Le problème se pose lorsque l'on fusionne deux triangles car il faut retirer des élément des tableaux en mémoire. Sans entrer dans les détails, on remplace dans chaque tableau l'élément que l'on veut retire par le dernier élément.

Mais cette façon de procéder implique d'autres modifications, dans le GPU sont stockées les données de relief et on garde dans la mémoire vive une copie de ces données afin de connaître les valeurs sans passer par la carte graphique (ce qui nous ralentirai).

Voici la liste des structures toutes accompagnées d'une explication.

- ```
  struct GPUVertice
  {
        D3DXVECTOR3 position;
        float u, v;
  };
  ```

  Cette structure correspond à un point dans l'espace, il est stocké dans la carte graphique.

- ```
  struct GPUTriangle
  {
        int a, b, c;
  };
  ```

  Cela correspond à un triangle, en effet les valeurs a, b et c sont des indices du tableau de GPUVertice. Pour chaque triangle on a trois coordonnées.

- ```
  struct CPUPoint
  {
        int vertice;
        GPUVertice gpuvertice;
        float ecart;
  };
  ```

  C'est un point du maillage originel, si on l'utilise alors il est associé à un GPUVertice. Si ce point est le au milieu de l'hypoténuse d'un triangle alors l'ecart est le taux d'erreur de ce triangle.

- ```
  struct CPUVertice
  {
        int point;
        int triangle;
  };
  ```

  Cette structure permet de faire le lien entre un vertice et l'un des triangles qui l'utilise. Grâce à cette structure lorsque le vertice est déplacé on peux mettre à jour tous les triangles qui font référence à ce vertice.

- ```
  struct CPUTriangle
  {
        int id;
        char niveau;
        int vg, vd, vb;
        GPUTriangle gputriangle;
        bool dejaFait;
  };
  ```

  Pour chaque triangle on connaît son id, afin de savoir si c'est un fils gauche ou droit, ce qui est nécessaire pour les opérations sur la géométrie.
  On connaît son niveau, c'est à dire le nombre de division nécessaires pour obtenir ce triangle depuis l'un des deux triangles de base.
  On connaît aussi les indices des triangles voisins, ce qui évite de les recalculer à chaque fois.
  ```

Et pour finir on sait si il faut ignorer le triangle lors du calcul de la géométrie de l'image, par exemple si ce triangle vient juste d'être créé.

Chaque carte possède toutes ces structures, elles permettent de gérer à leur niveau toute la géométrie du terrain.

Bien que cela requiert plus de mémoire vive, le gain de performance engendré est indéniablement préférable.

3. Interface utilisateur

Sachant que le logiciel sera utilisé lors d'une exposition par un nombre important de visiteurs nous avons créé une interface simplifiée traduite en Espagnol. On peux voir dans la partie haute de l'image qui suit cette interface.

Dans le coin supérieur gauche se trouve un indicateur d'altitude. Dans le coin supérieur droit on retrouve trois éléments d'interface :

- La première liste déroulante permet de sélectionner le mode de déplacement.
- La seconde permet de choisir un circuit prédéfini.
- Le bouton proche du bord permet d'afficher dans une petite fenêtre contenant des informations sur le logiciel.

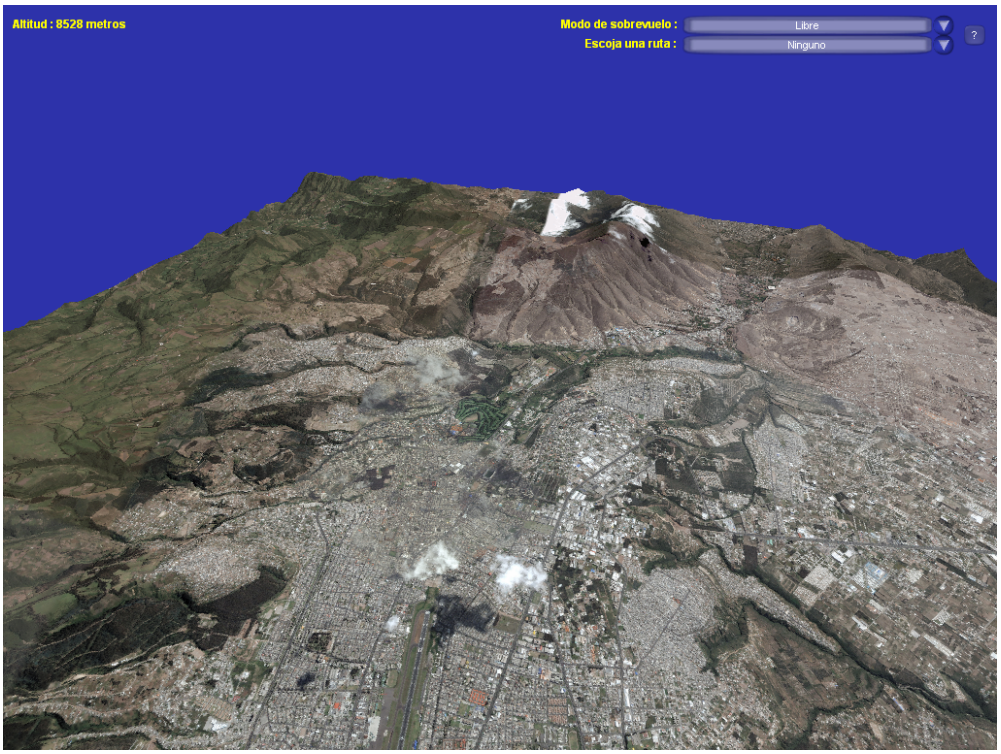


Figure 14 : Interface

Pour que la prise en main soit facile nous avons réduit le nombre de touches nécessaires aux déplacements, ainsi seul la souris et les touches fléchées permettent de se déplacer, toutes les touches sont décrites dans les annexes.

La souris permet de changer la direction de la caméra lorsque l'on maintient le bouton gauche alors que maintenir le bouton droit permet de décaler la position courante.

Il y a plusieurs types de déplacement, créés afin de permettre à un maximum de visiteurs de trouver ce qui leur convient le mieux.

Le mode libre permet de se déplacer comme on se le souhaite.

Dans le mode avion le déplacement vers l'avant est constant, mais ajustable dans une certaine mesure.

Dans le mode sol, le l'utilisateur se déplace en regardant vers la bas tout en étant proche du sol (sans pour autant regarder à la verticale).

Pour finir, le logiciel comporte un mode de déplacement automatique. Il suffit de choisir un chemin prédéfini et la caméra se déplace indépendamment de l'utilisateur.

V. Conclusion

1. Résultats obtenus

Les objectifs du stage sont tous accomplis, l'application est fiable, simple d'utilisation, performante.

L'application est fiable car aucune fuite mémoire n'est à signaler et elle tourne plusieurs heures sans erreurs.

L'application est simple d'utilisation car la prise en main est immédiate, les contrôles sont intuitifs et l'interface est réduite à l'essentiel pour ne pas surcharger l'utilisateur d'options inutiles.

L'application est performante car elle permet de visualiser avec précision une surface de 880 km² avec un nombre d'image par seconde élevé.

Malgré tout, des améliorations sont encore possibles. L'affichage de bâtiments, la gestion dynamique des options, l'utilisation d'un joystick, l'intégration d'une boussole et d'autres choses encore sont des aspects du logiciel que nous n'avons pas eu le temps de développer.

2. Difficultés Rencontrées

La principale difficulté a été de comprendre les algorithmes, les principes des méthodes existantes. Bien que nos sources soient nombreuses, la plupart sont en anglais et restent toujours imprécises quand à la mise en place de ces algorithmes.

Par exemple on trouvera dans toutes les descriptions d'algorithme qu'il faut diviser le voisin d'un triangle pour éviter le cracking mais aucune ne dit comment trouver le voisin. Il a donc fallu tester plusieurs méthodes et au final on parvient à comprendre ce qu'ils ont voulu dire. Le réel problème c'est que les algorithmes ne sont pas assez expliqués, ils semblent destinés à ceux qui comprennent déjà la majorité des principes.

Un autre problème s'est posé, celui de la communication. En effet, d'un coté notre maitre de stage était en Thaïlande la plupart du temps tandis que le logiciel devait servir à une exposition à Quito en Equateur. Il y un décalage de + 6 heures dans un sens et de – 6 heures dans l'autre. De plus en Equateur ils parlent Espagnol mais pas exactement l'Espagnol que l'on apprend en cours.

Il a fallu transmettre l'ensemble des données à notre correspondant à Quito, ce qui s'est fait petit à petit au rythme de téléchargements nocturnes (heure de Quito) car la bande passante est rare là-bas et le courrier contenant le dvd de données semble faire le tour du monde avant d'y arriver.

Malgré l'humeur festive régnant à Quito lors de l'inauguration de l'exposition, le logiciel n'a pas fonctionné. On ne sait toujours pas à ce jour la raison de ce problème mais on peut penser à plusieurs réponse :

- Mauvaise installation due à une incompréhension lors des échanges email en Espagnol
- La puissance de l'ordinateur exposé n'est pas adaptée ou alors il n'y avait tout simplement pas assez de mémoire vive.
- Raison mystique, en effet le musée pourrait être hanté par les premier habitants de Quito.

Nous avons donc envoyé une version « configuration minimale » qui depuis tourne sans le moindre problème, du moins nous n'avons eu aucun retour négatif, seulement des demandes de fonctionnalités supplémentaires…

3. Apports

Ce stage m'a apporté beaucoup, j'ai compris qu'une bonne organisation, ainsi qu'un travail de conception réfléchi sans trop de détails (pour laisser la place à l'imprévu) est très important pour produire un logiciel ambitieux.

De plus j'ai apprit à me servir d'outils de développement comme Visual Studio 2005, logiciel souvent utilisé en entreprise, ce qui élargi mon champs de compétences.

Je me rend compte que le développement en équipe, bien qu'il ne soit pas toujours facile, apporte autant de richesse au programme final qu'à l'expérience des développeurs.

Pour finir, je voulais souligner le fait que ce stage à été très enrichissant, qu'il m'a permis de passer ces quatre derniers mois à faire ce que j'aime, qu'il m'a permis de rencontrer des gens très intéressants et si c'était à refaire je n'hésiterais pas.

VI. Glossaire

Popping : Phénomène visuel résultant d'un brusque changement de détail.

Cracking : Discontinuité du maillage qui donne visuellement l'impression d'une faille.

MTD : Maison de la Télédétection. http://www.teledetection.fr/ Ce complexe constitue un pôle de recherche appliquée en télédétection et information géographique.

IRD : Institut de recherche pour le Développement. http://www.ird.fr/ L'Institut de recherche pour le développement a pour mission de développer des projets scientifiques centrés sur les relations entre l'homme et son environnement dans la zone tropicale.

Lua : Lua est un langage de programmation léger, flexible et extensible pouvant être intégré dans un programme ou utilisé à part. Il est intensivement utilisé dans les jeux commerciaux.

Vertice ou vertex : Sommet dans l'espace.

Visual Studio 2005 : Environnement de développement intégré, créé par Microsoft. Souvent utilisé en entreprise pour sa puissance et sa simplicité.

GPU : Graphic Process Unit, fait référence au processeur de traitement graphique. Désigne de façon plus commune la carte graphique.

DirectX : DirectX est une suite de bibliothèques multimédia intégrée au système d'exploitation Windows.

VII. Bibliographie

Virtual Terrain Project : Liste de documents traitants du niveau de détail des terrains.
http://www.vterrain.org/LOD/Papers/

Real-Time, Continuous Level of Detail Rendering of Height Fields par Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust.
http://www.cc.gatech.edu/gvu/people/peter.lindstrom/papers/siggraph96/

Visualization of Large Terrains Made Easy par Peter Lindstrom, Valerio Pascucci.
http://www.gvu.gatech.edu/people/peter.lindstrom/papers/visualization2001a/

ROAMing Terrain: Real-time Optimally Adapting Meshes par Mark Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, Mark B. Mineev-Weinstein.
http://www.cognigraph.com/ROAM_homepage/index.html

DirectX 9.0 Programmer's Reference par Microsoft, fourni avec le kit de développement de DirectX.

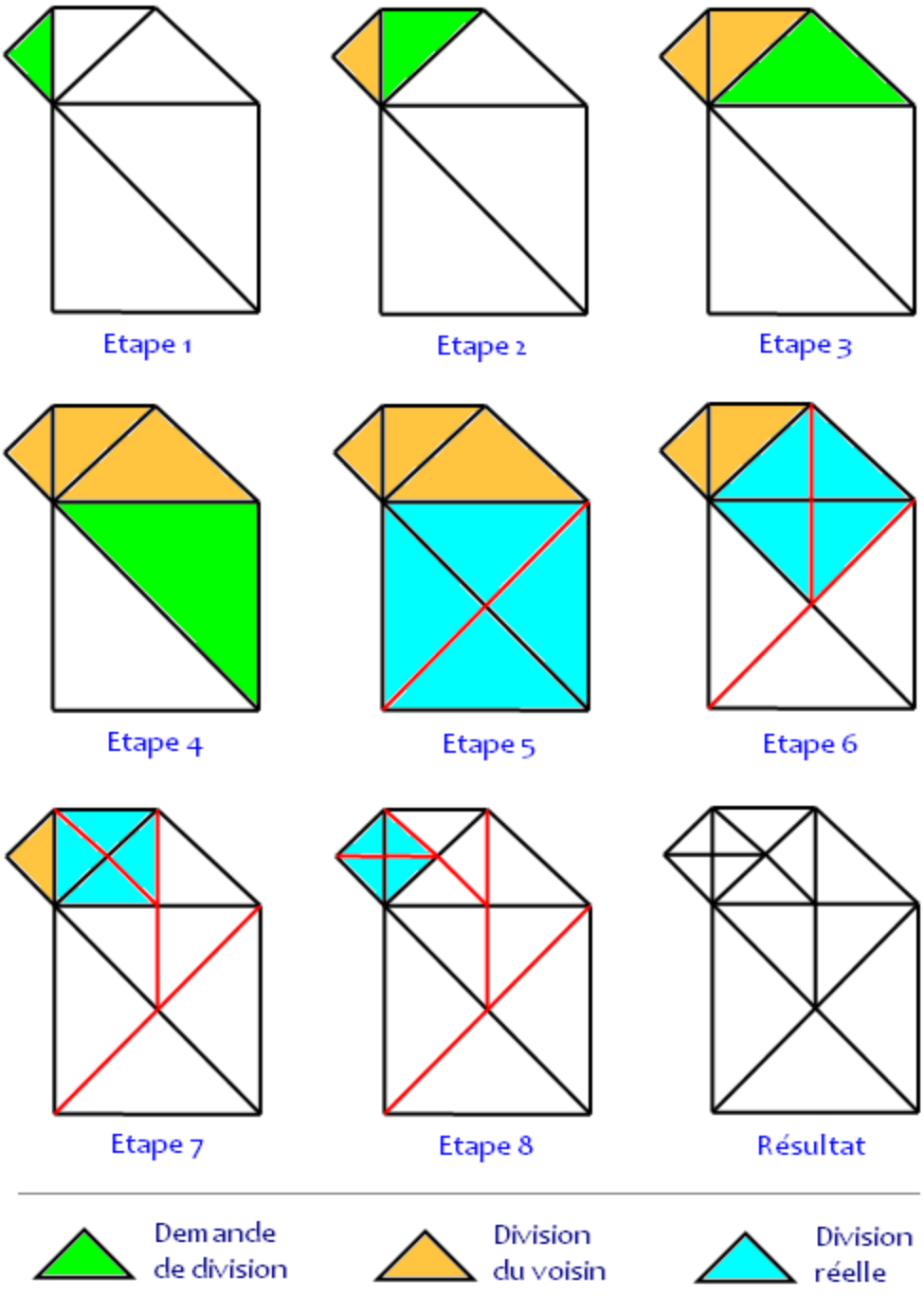
DirectX 9, Programmation de jeux 3D par Laurent Testud, édité par CampusPress.

Maison de la télédétection (MTD) en Languedoc-Roussillon.
http://www.teledetection.fr/

Institut de Recherche pour le Développement (IRD).
http://www.mpl.ird.fr/

VIII. Annexes

1. Division récursive d'un triangle sans *cracking*

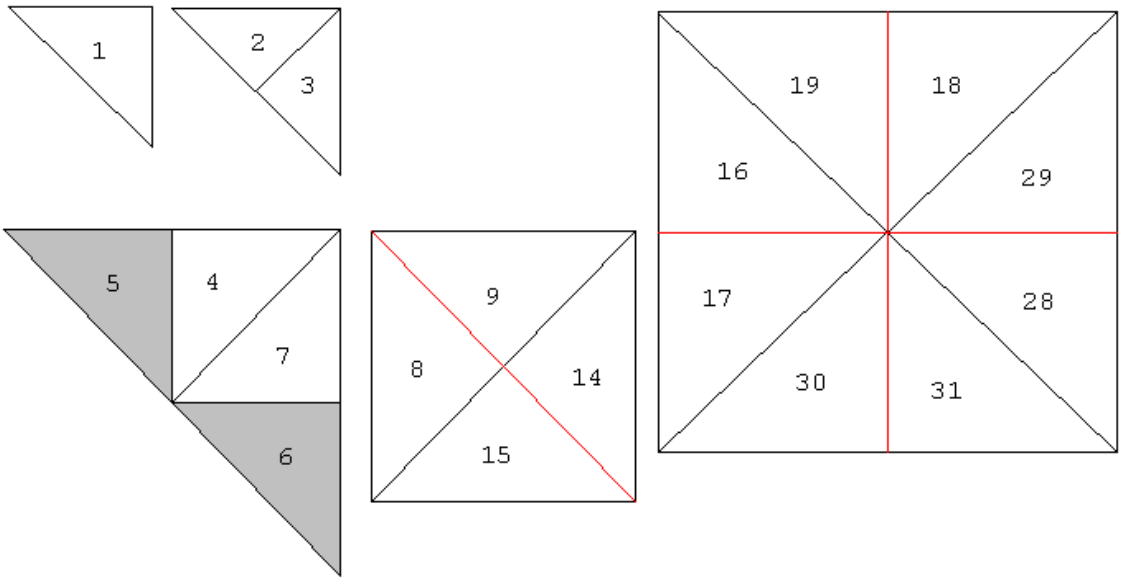


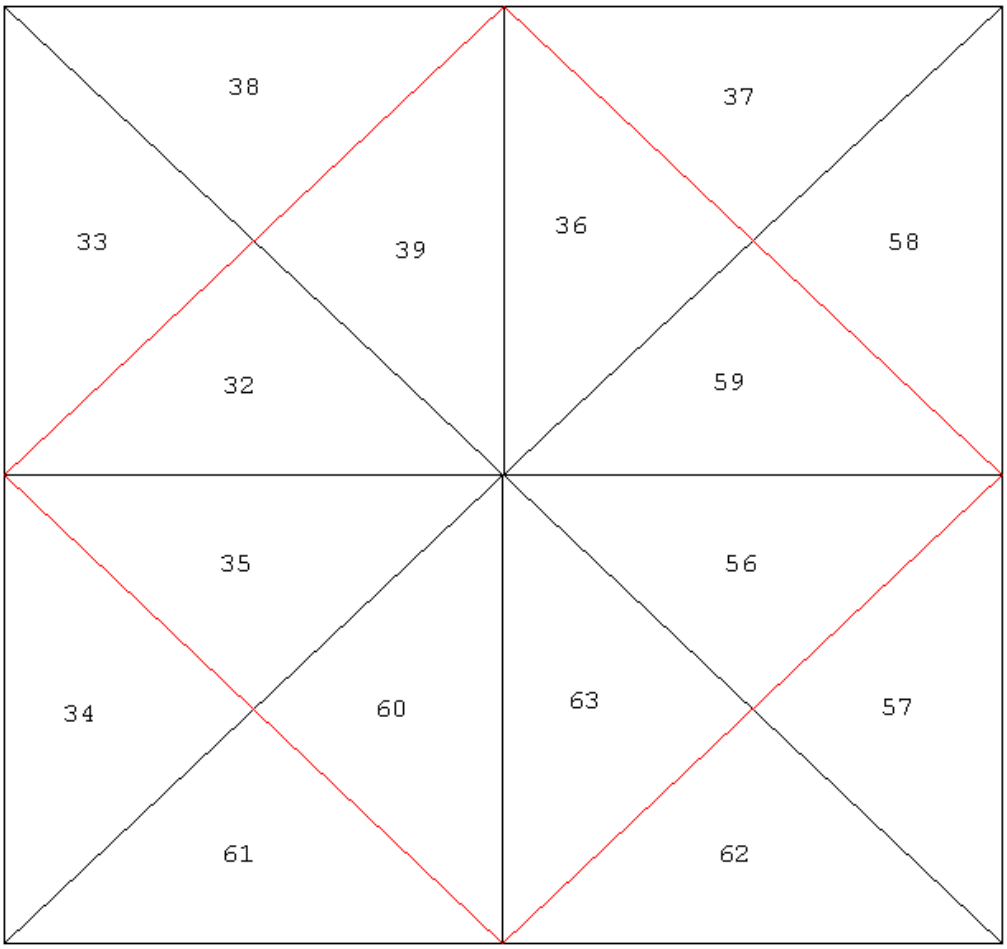
2. Quadrillage pour faire des traces de l'algorithme sur une grille 9x9

On peut facilement y reproduire un niveau de détail particulier, y noter les identifiants de triangles et des points, ainsi que marquer les points.

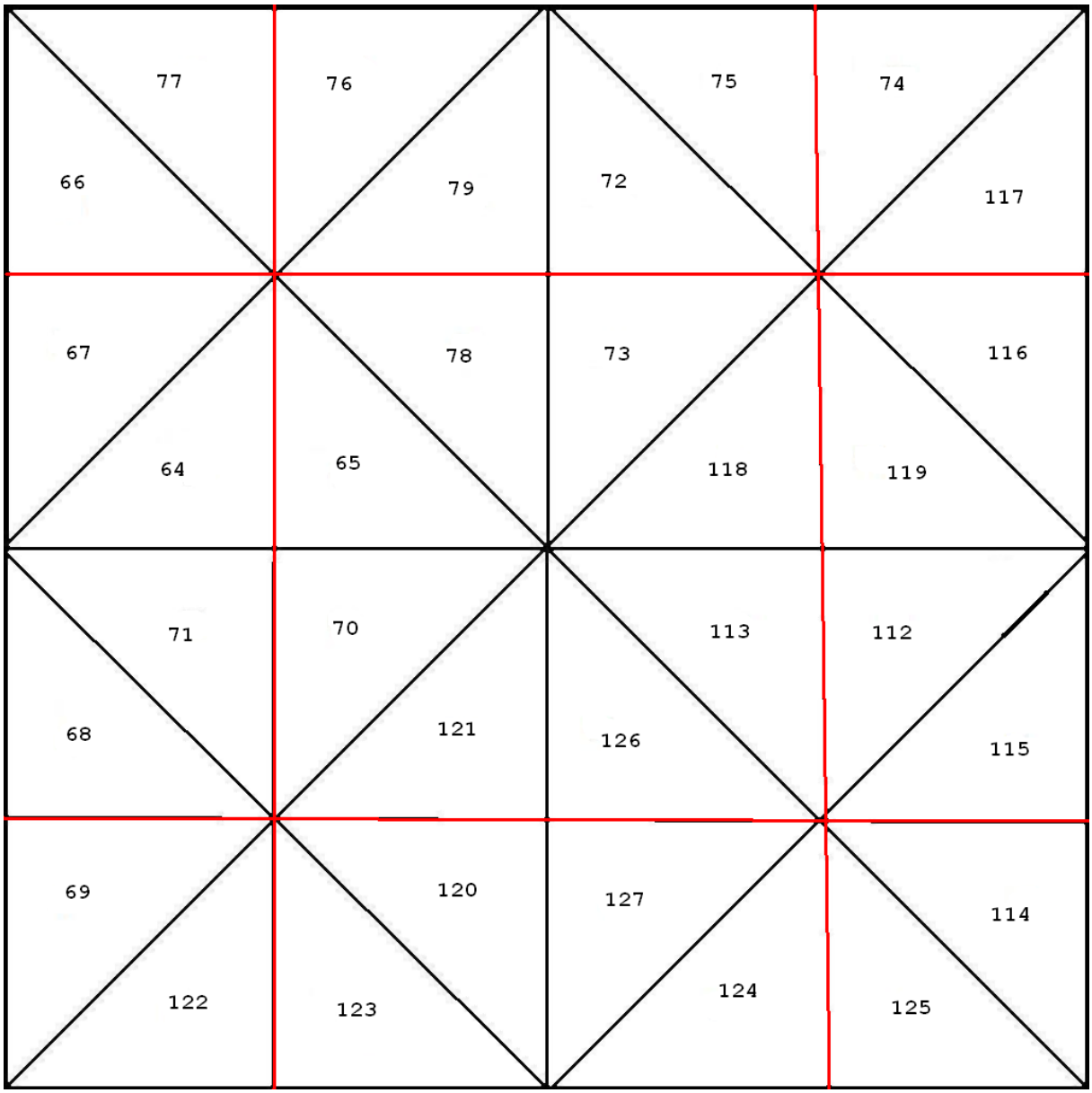
3. Numérotation des triangles jusqu'au niveau 4

Niveaux 0 à 3 :





Niveau 4 :


77
76
75
74
66
79
72
117
67
78
73
116
64
65
118
119
71
70
113
112
68
121
126
115
69
120
127
114
122
123
124
125

4. Description des commandes de navigation en fonction du mode de survol choisi

- **Mode libre :**

Avancer / Reculer 

Tourner à gauche / à droite  ou 

Incliner en avant / en arrière 

Décaler horizontalement vers la gauche / la droite 

Décaler verticalement vers le haut / le bas 

Monter / descendre 

- **Modo avion :**

Tourner à gauche / à droite  ou 

Incliner en avant / en arrière 

Augmenter / réduire la vitesse 

- **Mode à ras du sol :**

Avancer / Reculer 

Tourner à gauche / à droite  ou 

Décaler horizontalement vers la gauche / la droite 

Monter / descendre 

IX. Rapport technique

1. Configuration.

La configuration du logiciel s'effectue grâce au fichier « config.h ». On peux y changer les caractéristiques des données à charger :

- Le thème de la surface, dans le cadre de ce stage « Quito ».
- Les dimensions de la mosaïque de cartes.
- L'extension des données, mas ne change pas la façon dont elles sont lues.
- Les coordonnées du point de référence de la zone modélisé, en coordonnées projetées.
- La précision et la résolution maximale des morceaux de relief et d'images satellites.
- Le facteur d'accentuation du relief.
- La résolution du mode plein écran.
- Les touches de déplacement et les textes qui sont affichés.

Toute modification de ce fichier nécessite une recompilation.

2. Format des données.

Les données doivent remplir certaine conditions, chaque fichier doit être de la forme :

- Les morceaux de relief sont nommés « Data\Modeles\%1_%2_%3_%4.bin » avec :
  - %1 = Le thème de la surface modélisé
  - %2 = résolution de cette partie du relief égale à 2^n + 1 avec n >= 1
  - %3 = coordonnées de ligne dans la mosaïque de cartes, nombre positif
  - %4 = coordonnées de colonne dans cette même mosaïque, nombre positif
  - Le fichier bin contient %2 * %2 valeurs de type **float (4 octets)** au format binaire

- Les images satellites sont nommées « Data\Textures\%1_%2_%3_%4.bmp » avec :
  - %1 = idem que le précédent
  - %2 = résolution de cette partie de l'image égale à 2^n + 1 avec n >= 1
  - %3 = idem que le précédent
  - %4 = idem que le précédent

Le logiciel supporte tous les types d'images que DirectX prend en charge : BMP, JPG, TGA, PNG, DDS, PPM, DIB, HDR et PFM.

3. Ajouter un nom de lieu ou un chemin prédéfini

Lieu :

Dans l'application, placez-vous au dessus du lieu concerné, et appuyez sur la touche L.

La ligne de script Lua est directement copiée dans le presse-papier.

Ouvrez ensuite le fichier *Data\Scripts\Locations.lua* et collez le texte à la suite des autres lieux.

Mettez le nom de votre choix, et sauvegardez le fichier.

Il faut relancer l'application pour qu'elle prenne en compte le nouveau lieu.

Chemin :

Ouvrez le fichier *Data\Scripts\Paths.lua*.

Ajouter un nouveau nom de chemin selon le même modèle que les autres.

Dans l'application, placez-vous à une position clé, et appuyez sur la touche P.

La ligne de script Lua est directement copiée dans le presse-papier.

Collez le texte à l'endroit voulu, et donnez-lui un numéro et le temps voulu.

Recommencez ainsi pour chaque position clé, puis sauvegardez le fichier.

Il faut relancer l'application pour qu'elle prenne en compte le nouveau chemin.

4. Description rapide des fichiers et des classes

Chaque classe possède son propre fichier, la classe Application est donc définie dans les fichiers « Application.h » et « Application.cpp ».

La classe *Application* est la classe principale.

La classe *MultiCarte* permet de gérer la mosaïque de cartes. Plus précisément c'est elle qui décide des niveaux de détails des données.

La classe *Carte* contient principalement l'algorithme de simplification du relief.

La classe *Chargeur* permet de charger dans un processus différent n'importe quel type de données, ici elle permet de charger les données de relief et les images satellites en arrière plan.

La classe *ChampDeVision* permet l'utilisation de la pyramide de vue.

La classe *Camera* gère la caméra dans l'environnement virtuel.

La classe *Lua* constitue l'interface de communication entre le logiciel et les scripts écrits en Lua.

5. Intégrer le moteur dans une autre application

L'intégration du moteur d'affiche est simple. Il suffit de créer une instance de la classe *Application* et d'appeler les fonctions publiques de cette classe au moment voulu.

Le constructeur de cette classe nécessite un pointeur vers un ***CDXUTDialog***, ce qui permet d'afficher des éléments d'interface.

On demande le chargement des données avec la fonction Charger(…) avec comme paramètre un pointeur vers un ***IDirect3DDevice9.***

Le déchargement des données s'effectue avec la fonction Decharger()

La fonction de pré-calcul de l'affichage se fait grâce à la fonction PreCalculer(…) appelée avec en paramètre le temps écoulé depuis la dernier appel à cette fonction.

L'affichage se fait avec la fonction Afficher(…., …..) avec comme premier paramètre un pointeur de ***IDirect3DDevice9*** et comme deuxième paramètre le temps écoulé depuis la dernière image.

Pour finir les divers événements sont gérés par les fonctions :

- GestionClavier(UINT iLettre, bool bPresse, bool bPresseAlt);
- GestionSouris(bool bGauche, bool bDroit, bool bMilieu, int iMoletteDelta, int xPos, int yPos);
- GestionInterface(CDXUTControl* pControl, UINT iEvenement, int iControle);

6. Changer les modes de déplacement

La gestion des modes de déplacement se trouve dans la méthode *FrameMove* de la classe *Camera*.