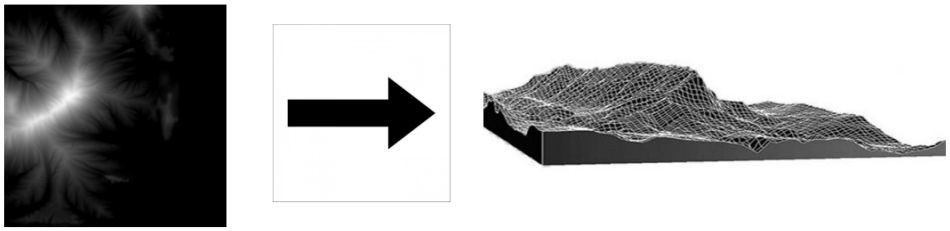
IMAC
INGÉNIEUR
ESIPE

Synthèse d'image I
Programmation-Algorithmique +
Projet
VisuTerrImac

Le projet de ce semestre consiste en la réalisation d'une application permettant de visualiser un terrain, à l'aide d'une image en entrée appelée *height map* . Cela vous permettra ainsi de rendre un MNT (Modèle Numérique de Terrain) que vous agrémenterez d'objets et de végétation



Pour ce faire, vous allez également devoir exploiter une structure de données de type arbre : le QuadTree. Cette structure de données vous permettra d'optimiser le rendu notamment. Vous allez donc devoir réaliser un programme en C/C++ qui va permettre à un utilisateur de survoler ce terrain. Le but étant :

- De vous familiariser avec la structures d'arbres dite quadtree.
- De vous faire utiliser les fonctions de dessin 3D OpenGL.
- De vous faire coder certains algorithmes fondamentaux sur les arbres.
- De vous faire manipuler des images.

Modalités de rendu

- **Nombre de personnes par projet :** 2 ou 3
- **Livrable :**
  — Les sources C/C++ ainsi qu'un système de compilation permettant de compiler sur les machines de l'université (Linux avec les packages SDL, SDL2, OpenGL, SFML, GLU, GLUT, SDL_image, SDL_ttf disponibles ainsi que make, gcc, g++ et cmake)
  — Un rapport au format PDF décrivant le projet (en particulier les aspects qui diffèrent du jeu original, inutile de répéter le sujet), votre méthode de travail, des détails techniques si vous souhaitez détailler une fonctionnalité, les difficultés rencontrées et enfin les améliorations possibles.
- **Soutenance :** Si les conditions le permettent, vous pourrez défendre votre projet durant une soutenance en faisant une démonstration de votre projet et en reprenant les différents points du rapport. Votre présentation durera 10 minutes et sera suivi d'une série de questions du jury pour compléter votre soutenance.
- **Date de rendu (non définitive) :** 01 juin 2020
- **Date de soutenance :** 2 juin 2020 13h30-18h

1 L'application

Votre groupe est chargé, au sein d'une entreprise d'éditeur de logiciel, de créer un prototype opérationnel permettant de visualiser un terrain et de l'agrémenter avec quelques décors. Le nom prévu pour ce progiciel est VisuTerrImac, mais la division marketing vous suivra sur tout nouveau nom de votre choix.

Le terrain sera représenté sur le plan (O,x,y) et ses élévations sur l'axe z. Le terrain sera centré sur l'origine.

1.1 Les données d'entrée

1.1.1 La *height map*

Le terrain, ou plus exactement son élévation, sera en fait représenté par une image que l'on nomme traditionnellement *height map* . Chaque pixel représente une position sur le terrain, et sa valeur indique l'élévation de ce point. Ainsi chaque pixel (i,j) représente un point (x,y) du terrain, et la valeur du pixel indique donc l'élévation en z de ce point. A priori cette valeur sera comprise entre 0 et 255 (ou entre 0 et 1 suivant le format d'image utilisé) et nous verrons au prochain paragraphe comment le convertir en élévation "réelle", ainsi que la conversion entre (i,j) et (x,y). Un défaut des *height map* est qu'elle ne peuvent représenter des grottes ou des surplombs puisque chaque point du terrain n'a qu'une seule élévation.

Cette image peut être au format PPM (ou plus exactement PGM) mais nous vous laissons toute latitude d'exploiter d'autres formats de fichiers.

Entre chaque groupe carré de 2x2 pixels, on trouve donc 2 triangles. Autrement dit si votre *height map* est de résolution w, h alors le nombre de triangle constituant votre terrain sera de $2 * (w - 1) * (h - 1)$

1.1.2 Les paramètres et le fichier d'entrée

La *height map* ne suffit pas à définir correctement la taille "réelle" du terrain. Votre terrain sera donc chargé via un fichier texte au format `.timac` qui contiendra :

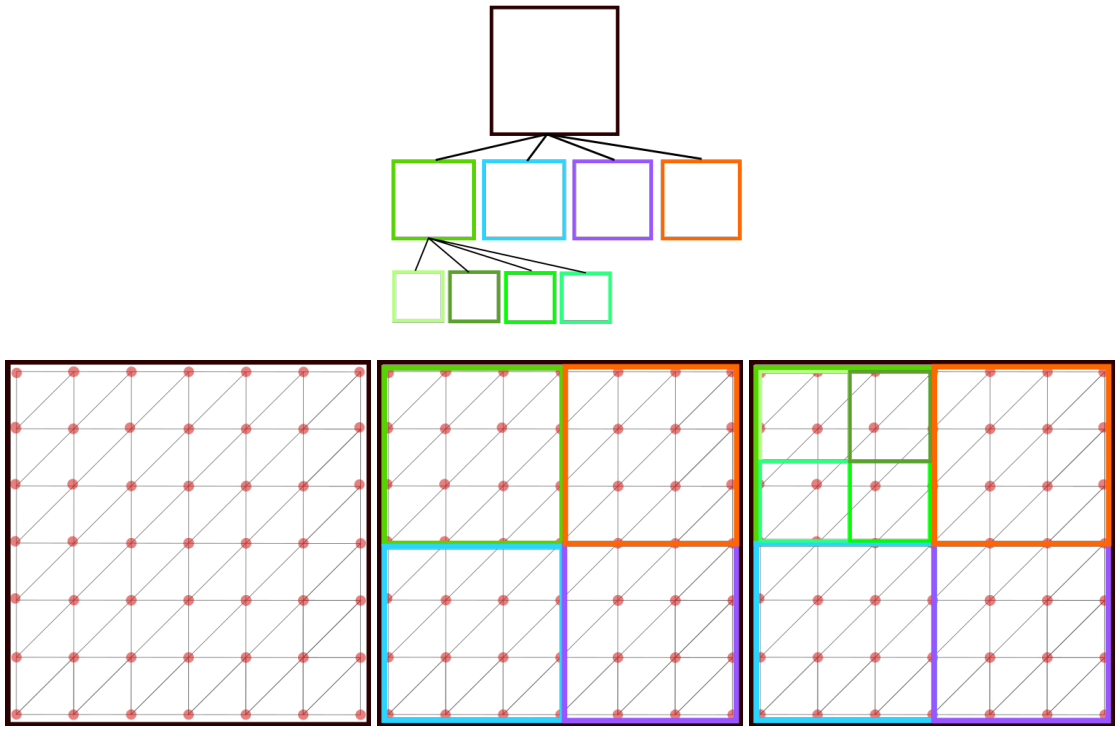
- Un paramètre donnant le nom de fichier de la *height map* à charger
- Un paramètre x_size indiquant la longueur totale du terrain sur l'axe X
- Un paramètre y_size indiquant la longueur totale du terrain sur l'axe Y
- Deux paramètres z_min et z_max indiquant les hauteurs minimales et maximales du terrain.
- Deux paramètres z_near et z_far donnant les plan near et far de la caméra
- Un paramètre fov qui donne l'angle visible par la caméra.

Les paramètres z_min et z_max permettent de définir n'importe quelle altitude d'un pixel / point. Ainsi, si val est la valeur (entre 0 et 1) du pixel/point, son élévation sera $z_{min} + val(z_{max} - z_{min})$

De même, la taille et la pixel réel d'un pixel de ligne i et de colonne j peuvent s'obtenir grâce aux paramètres x_size et y_size sachant que le le terrain est centré sur l'origine (en X,Y).

1.2 Le Quad Tree

Le *Quad Tree* ou Arbre Quaternaire est un arbre dont chaque noeud peut avoir jusqu'à 4 enfants. Semblable à l'arbre binaire, cette structure permet d'organiser des données pour mieux les retrouver, il est souvent utilisé pour partitionner des données dans un espace bidimensionnel, notamment pour ranger des points. Chaque noeud de l'arbre représente un carré partitionnant l'espace et chaque enfant est une partition de son parent, la racine de l'arbre est un carré représentant la totalité de l'espace.



Voici les propriétés à respecter
— Chaque noeud qui n'est pas une feuille aura obligatoirement 4 enfants qui le divisent en quatre espaces (supérieur droit, supérieur gauche, inférieur droit, inférieur gauche)
— Chaque noeud représente un espace et possède 4 coordonnées qui délimite celui-ci
— Chaque feuille contient les 4 points (4 pixels de la heightmap) qui se trouve à l'intérieur du carré qu'elle représente, elle contient par conséquent 2 triangles du MNT

Note : Il existe l'équivalent 3D, l'octree qui range des points selon l'axe X, Y et Z.

1.3 La caméra

1.3.1 Gestion de la caméra

La caméra pourra se déplacer dans le monde sans difficulté c'est à dire sans collision. Le mode de déplacement s'appelle 'freefly' c'est à dire que la caméra se déplace comme dans un FPS. Pour ce faire, vous pouvez utiliser l'instruction GLU appelée `gluLookAt` qui prend globalement 3 paramètres : la position de la caméra, le point visé et le vecteur 'up'.

Pour le point visé, vous devez prendre la position + une direction de regard. Cette direction de regard peut se définir grâce aux coordonnées sphériques :

$$\vec{F} = (\cos(\theta)\sin(\phi), \sin(\theta)\sin(\phi), \cos(\phi)) \quad (1)$$

Le vecteur 'up' $\vec{U}$ doit être orthogonal à la direction de regard. Pour le calculer, vous pouvez passer par un vecteur 'left' (direction gauche de la caméra)

$$\vec{L} = (\cos(\theta + \pi/2), \sin(\theta + \pi/2), 0) \tag{2}$$

Votre application permettra donc, à l'aide de touche au clavier, de déplacer la caméra (position) et de changer la direction du regard.

Trinôme uniquement : L'application proposera également un mode de visualisation ou la caméra reste à une distance fixe (1 mètre par ex.) du sol. Essayez de proposer un mouvement le plus fluide possible.

1.3.2 Optimisation liée à la caméra

Grâce au quadtree et à la position de la caméra, nous allons pouvoir procéder à des optimisations diminuant le temps de rendu. En effet, le quadtree permet de répartir l'ensemble des triangles constituant le terrain (et les objets) dans des zones carrées.

La première optimisation s'appelle *frustum culling* et permet d'éliminer toutes les zones non visibles par la caméra. Pour ce faire vous déterminerez dans un premier temps la zone visible depuis la caméra grâce à sa position bien sûr, mais aussi grâce aux paramètres $fovy$, z_{near} et z_{far}. Ensuite, en prenant en compte cette zone, vous parcourerez le quadtree. Toutes les noeuds ou feuilles qui sont totalement à l'exterieur de la zone peuvent être omis (non rendu). Si il y a intersection, alors il faut soit voir les fils, soit dessiner.

La seconde optimisation à réaliser s'appelle de *Level Of Detail* ou LOD. Cette optimisation consiste à déterminer la distance d'un noeud du quadtree vis à vis de la caméra. Les binômes prendront le point central du noeud, alors que les trinomes calculeront le point le plus proche du noeud vis à vis de la caméra. Une fois cette distance acquise, vous **choisirez** une méthode permettant de déterminer si on dessine le noeud (les 2 triangles formés des 4 points extrêmes du noeud) ou si on descend sur les fils (si le noeud n'est pas une feuille bien sûr).

1.4 Les décorations et l'environnement

1.4.1 La décoration du terrain

Sur le terrain, vous placerez aléatoirement des arbres. Ces arbres devront être placé à la bonne hauteur sur le terrain. En terme de modélisation ces arbres seront représentés par des *billboards*. Ces *billboards*sont en réalité des quads (rectangles) sur lequel est placé une texture en RGBA (avec transparence donc) d'arbre. Vous en trouverez sur ce site. La caractéristique principal des *billboards*c'est surtout qu'ils **sont toujours orientés vers la caméra**.

1.4.2 L'environnement

Le ciel sera constitué par une skybox. C'est un cube (pas forcément très grand) bleu ou texturé qui est dessiné à la position de la caméra. Lorsque vous dessinez ce cube, il suffit de désactiver le rendu sur le buffer de profondeur grâce à l'instruction `glDepthMask(GL_FALSE)`. Le réactiver se fait avec `GL_TRUE`.

Par ailleurs, votre application proposera un éclairage directionnel (de type soleil) associé à un modèle de Lambert. Votre application devra d'ailleurs permettre de faire évoluer cette direction d'éclairage à l'appui sur une touche comme si le soleil se levait puis se couchait. Il n'est pas demandé qu'il y ait une cohérence entre la skybox et cet éclairage.

Enfin votre terrain devra être coloré en fonction de la hauteur. Les binômes pourront utiliser des couleurs normales mais les trinômes devront exploiter des textures sur ces terrains.

1.5 Pour aller plus loin

Vous pouvez ajouter des améliorations à l'application comme par exemple :
— ajouter ou retirer un arbre
— modifier la hauteur d'un point du terrain
— sauvegarde / chargement de MNT
— afficher des décors animés
— affichage d'eau
— une skybox et un éclairage cohérent
— ...

2 Développement

Pour faire l'application vous devez respecter les contraintes décrites dans cette partie, tout écart devra être au préalable validé par vos évaluateurs et justifié dans le rapport et la soutenance.

2.1 Synthèse d'Image - GUI

Pour la partie Synthèse d'images, vous aurez à réaliser les éléments suivants, qui reprennent les spécifications vues précédemment.

2.1.1 Chargement et affichage du terrain

L'application devra donc, en ligne de commande, charger un fichier de configuration décrit en 1.1.2.

2.1.2 Affichage de la végétation et de l'environnement

En reprenant les spécifications précédentes, vous afficherez également, en tant que *billboards*, les arbres de la scène.

2.1.3 Déplacement dans le terrain

A l'appui sur des touches que vous définirez, vous déplacerez la caméra sur le terrain.

2.1.4 Modèle d'éclairage

Votre application proposera une lumière directionnelle (soleil) qui pourra varier suivant la 'journée'. Ainsi, l'appui d'une touche fera "tourner" le soleil comme si le temps passait. Le terrain sera illuminé en utilisant un modèle de Lambert dont le coefficient de réflexion diffuse sera soit une couleur dépendant de l'altitude (pour les binômes), soit d'une texture (pour les trinomes).

2.1.5 Visualisation du Quadtree

A l'appui sur une touche, votre application basculera en visualisation du quadtree. Dans ce mode, qui remplace le modèle d'illumination, le terrain pourra être vu en mode 'fil de fer' ou en mode plein. Et les triangles du terrain seront dessinés avec une couleur différente suivant leur appartenance à un noeud ou à une feuille. L'idée est que vous proposiez un mode de visualisation le plus clair possible de la structure sous jacente du quadtree

2.1.6 (trinome uniquement) Réparation des cracks

En passant d'un LOD à un autre, le maillage peut présenter des trous. Faites en sorte de détecter et de réparer cela.

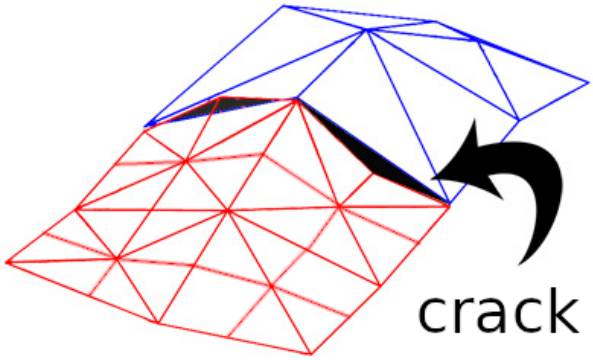


Figure 1 – une cassure dans le maillage de terrain dûe à deux niveaux de LOD différents

2.2 Algorithmique - Parcours d'arbre

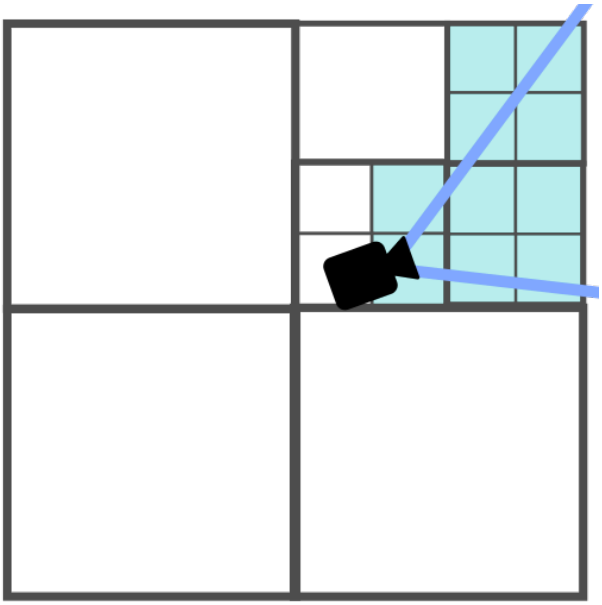
2.2.1 Création du QuadTree

Une fois tous vos points instanciés, il faudra les ranger dans le quadtree. En partant de la racine vous allez diviser en quatre chaque noeud jusqu'à ce que chaque feuille ne puisse contenir que 4 points.

2.2.2 Utilisation du QuadTree pour le frustum culling

Votre caméra possède un champs de vision qu'on représentera ici comme un triangle (défini par le centre optique de la caméra, les plans droites, gauche, et z_{far}), tout ce qui est dans ce trapèze sera affiché à l'écran. Par opposition on peut donc s'abstraire de ce qui n'est pas dans ce triangle.

Pour filtrer les points à dessiner vous allez calculer les intersections de ce trapèze avec chaque carré représenté par les noeuds de votre arbre. Si un noeud croise le trapèze alors vous pouvez continuer le test dans les noeuds enfant, autrement vous ignorez le noeud (et donc tous ses enfants). Si vous finissez par tomber sur une feuille lors de vos tests alors les triangles de cette feuille doivent être dessinés.



Référence : https ://jeux.developpez.com/tutoriels/theorie-des-collisions/formes-complexes/

2.2.3 Utilisation du QuadTree pour le LOD

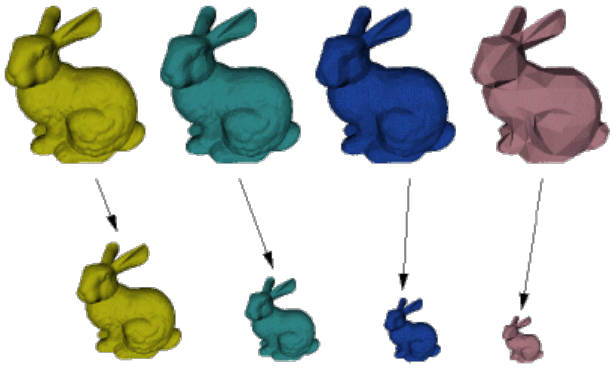


Figure 2 – Simplication and LOD Hierarchy

Pour réduire le nombre de triangles à dessiner vous allez simplifier les triangles les plus éloignés, pour ce faire, plutôt que de dessiner les triangles "normaux", il faudra dessiner les noeuds parent de ces points sous forme de deux triangles qui auront pour coordonnées celles qui délimitent les noeuds en question. Plus vous vous éloignez de la caméra moins le noeud que vous dessinerez sera profond. Nous vous laissons libre de choisir la méthode la plus appropriée pour le choix de la profondeur de dessin (dans l'arbre).

Attention il faudra donc prendre en compte l'axe Z dans les délimitations des noeuds de votre arbre.

3 Notation

Partie Commune (6)
Rapport (1)
Soutenance (1)
Compilable par l'évaluateur (1)
Architecture logiciel et propreté du code (1)
Fonctionnalités du jeu (2)

Partie Synthèse d'Images (10)
Chargement et affichage du terrain
Affichage de la végétation
Déplacement caméra
Modèle d'illumination et skybox
Visualisation du quadtree
Réparation des cracks

Partie Algorithmique (10)
Chargement, création du quadtree
Utilisation du quadtree pour le dessin
Utilisation du quadtree pour le *frustum culling*
Utilisation du quadtree pour le LOD

Bonus (4) (Soutenance, Originalité, Fonctionnalités supplémentaires...)

4 Remarques

— Il est très important que vous réfléchissiez avant de commencer à coder aux principaux modules, algorithmes et aux principales structures de données que vous utiliserez pour votre application . Il faut également que vous vous répartissiez le travail et que vous déterminiez les tâches à réaliser en priorité.
— Ne rédigez pas le rapport à la dernière minute sinon il sera bâclé et cela se sent toujours.
— Le projet est à faire par binôme. Il est impératif que chacun d'entre vous travaille sur une partie et non pas tous "en même temps" (plusieurs qui regarde un travailler). sinon vous n'aurez pas le temps de tout faire. C'est encore plus vrai pour les trinômes.
— Utilisez des bibliothèques ! Notamment pour les types abstraits.
— N'oubliez pas de tester votre application à chaque spécification implémentée. Il est impensable de tout coder puis de tout vérifier après. Pour les tests, confectionnez vous tout d'abord de petites cartes (taille 5 par 5 par exemple) avec un chemin extrêmement simple. Si cela marche vous pouvez passer à plus gros ou plus complexe.
— Vos chargés de TD et CM sont là pour vous aider. Si vous ne comprenez pas un algorithme ou avez des difficultés sur un point, n'attendez pas la soutenance pour nous

en parler.