Rapport de soutenance

Team UMAD


AutoMap

Table des matières

Chapitre 1

Introduction

1.1 Présentation du projet

Le projet de cette deuxième année consiste à créer un programme qui permet de représenter une carte topologique en trois dimensions. Une carte topologique est une carte représentant les différentes altitudes grâce à des différentes zones de couleurs. Au final le programme permettra à l'utilisateur d'appliquer divers traitements d'image, de modifier plusieurs options ou encore de se déplacer dans le modèle 3D grâce à une caméra.

1.2 Présentation de l'équipe

1.2.1 Weng Rémi - Ixartz

Avec le projet de l'année dernière, j'ai beaucoup appris à travailler en groupe ce qui n'était pas le cas avant d'intégrer EPITA. Avec ce nouveau projet et avec un groupe venant de 4 groupes différents, nous avons tous dû s'adapter. J'ai choisi de travailler sur le traitement d'image afin d'approfondir ce domaine puisque les travaux pratiques donnés par les ACDC de l'année m'ont beaucoup intéressé. Cela me permettra d'utiliser les algorithmes de traitement d'image dans un projet conséquent dans un contexte plus concret.

1.2.2 Mirzaian Vincent - Gratteux

Ce projet m'a très emballé dès le premier coup d'œil au cahier des charges. Cette deuxième année commence très fort. En effet durant ma première année, j'ai pu toucher à la manipulation d'images durant nos TP ou encore durant le projet d'INFOSUP. Par ailleurs, le fait de le faire en Ocaml rajoute une part de motivation, car cela me permettra d'utiliser des librairies très reconnues telles que SDL ou encore OpenGL. De plus, le fait que l'équipe se connaisse depuis maintenant plus d'un an, cela nous permettra de rester soudés et ainsi facilitera le développement du projet.

1.2.3 Anselme Jeremy - Levak

Je pratique depuis un certain nombre d'années le monde de la 3D et j'ai eu à plusieurs reprises la possibilité de coder tout ou partie de moteurs 3D. Cela a commencé par un apprentissage - sans prétention - de Blender en 3eme. Cela s'est poursuivit en 1ere par un TPE dont le thème principal était les modèles de simulation 3D, pour donner vie à un moteur 3D codé de A à Z sur calculatrice (suite à une remarque de notre professeur qui nous indiquait qu'on ne faisait qu'utiliser les logiciels de 3D sans savoir ce que cela impliquait). En Terminale, j'ai poursuivi le projet en le rendant utile pour un certain nombre d'étudiants. Coder sur calculatrice impliquait de minimiser les calculs inutiles et d'optimiser chaque partie au maximum. En rentrant en SUP j'ai continué cette merveilleuse aventure lors d'un TP sur le RayTracing. Rien qu'après avoir commencé à implémenter les fonctions, j'ai trouvé évident la lenteur de Blender sur les rendus. Cette année le projet de SPE va me permettre de toucher à un nouvel aspect de la programmation de moteurs 3D : celui d'utiliser la carte graphique via des bibliothèques comme OpenGL. Je trouve donc ce parcours tout à fait noble, celui de commencer à la base, puis de monter les échelons avec en vue le résultat final. Avec l'accord des autres membres de projet, je vais développer le moteur 3D tout en donnant un grand nombre d'informations concernant mes recherches comme le ferait n'importe quel groupe soudé.

1.2.4 Waser Rémi - Radion

Passionné par l'informatique depuis mon plus jeune âge, j'ai toujours voulu réaliser un projet de grande ampleur avec des personnes que je ne connaissais pas. Le projet de l'an dernier m'a permis de découvrir ce concept et ainsi me conditionner aux futurs travaux de groupe qui m'attends durant mes années à EPITA. Ce deuxième projet m'a donc permis de continuer cette expérience et me renforcer en tant que membre d'un groupe de travail. De plus, ce projet étant commun à tout les groupes, on peut par conséquence voir l'avancement et la motivation des autres groupes et nous permettre de nous situer par rapport à de nombreuses personnes et groupes de travail.

1.3 Organisation du travail

Le projet comporte plusieurs grandes phases qui permettent assez aisément de se répartir les travaux entre les différents membres de l'équipe.

	Contours	Gestion du bruit	Échantil-lonnage	Interface graphique	Moteur 3D	Site Web
Vincent	X	X		X		
Rémi Wa.			X	X		
Jeremy			X		X	
Rémi We.	X	X				X

Chapitre 2

Pré-traitement

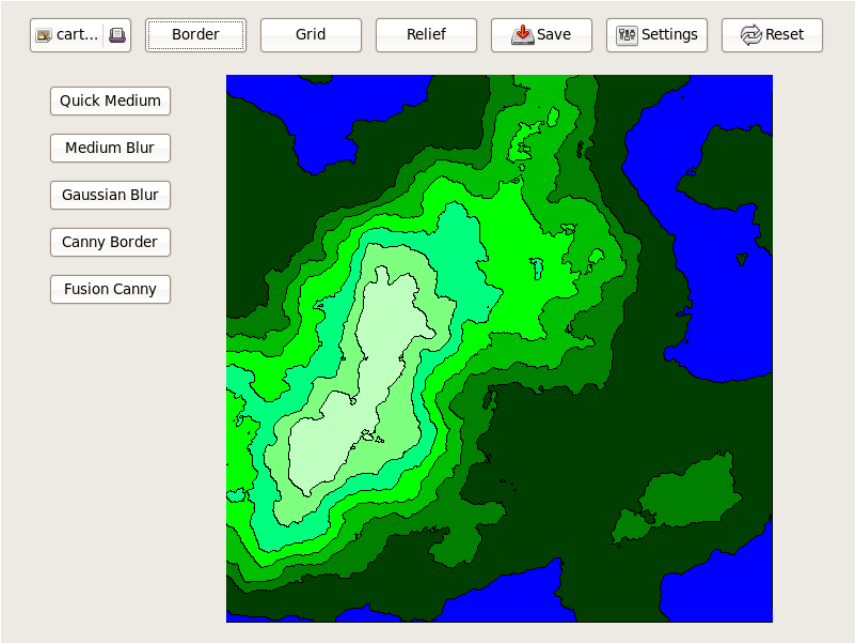
2.1 Les bibliothèques

Il existe de nombreuses bibliothèques permettant de manipuler des images via Ocaml. En faisant nos propres recherches nous avons découvert deux principales bibliothèques permettant d'appliquer divers traitements sur des images : OcamlSdl et CamlImages. Mais très vite nous nous sommes portés vers OcamlSdl.
En effet, cette bibliothèque a de nombreux atouts dont le premier est la simplicité d'utilisation. OcamlSdl représente une image par l'intermédiaire d'une matrice de pixels. Les fonctions get_pixel et put_pixel déjà intégrées, nous pouvions mettre en place divers traitements très rapidement. De plus, OcamlSdl étant déjà installé sur les racks, nous avions la chance de pouvoir directement commencer à coder le projet sans délai.

2.2 Détection des contours

La détection des contours a surement été la partie du projet le plus facile à réaliser. L'objectif était de faire apparaître des bordures noires autour des différentes zones de la carte topologique. La manière la plus simple était donc de parcourir l'image, et en comparant successivement le

pixel courant avec son pixel voisin, il suffisait donc de comparer leur couleur, et de mettre le pixel courant en noir si les deux couleurs différaient. Comme un carte topologique est composés de différentes zones avec des formes diverses et plutôt arrondies, il fallait faire une comparaison horizontale et verticale des pixels.
Comme certaines images comportent des zones de bruits, c'est-à-dire des zones de l'image dans lesquelles se trouvent des pixels parasites, nous avons appliqué un seuil à notre fonction de traçage de contours pour que ce dernier ne prenne pas en compte ce bruit lors du parcours de l'image.

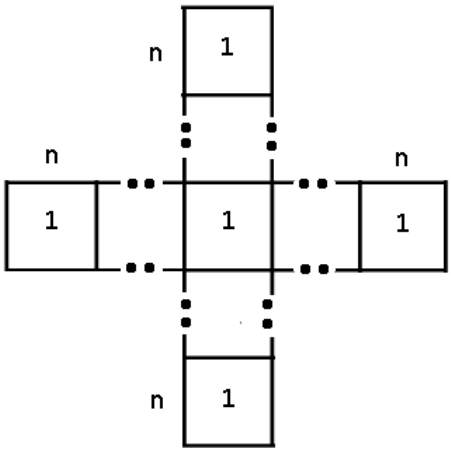


2.3 Pré-traitement de l'image

La détection des contours avec l'algorithme de base c'est à dire parcourir tous les pixels de l'image et lorsqu'il y a un changement, mettre le pixel courant en noir a une certaine limite. En effet, cet algorithme ne marche plus avec les images avec beaucoup de bruits puisqu'il met des contours autour des ces derniers ce qui pose un gros problème pour la suite. Après de multiples recherches sur Internet pour trouver des algorithmes plus complexes sur la détection des contours, nous avons choisi l'algorithme de Canny qui se déroule en plusieurs étapes.

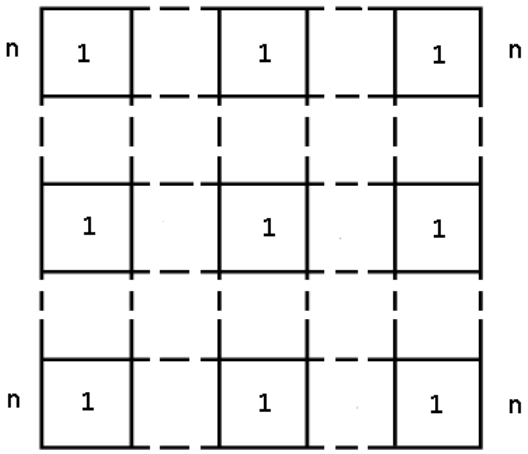
2.3.1 Filtre moyen (rapide)

Pour chaque pixel de l'image, nous faisons la moyenne du composante rouge des pixels situés à n (pourra être choisi par l'utilisateur) postion verticalement ou horizontalement (mais pas les deux en même temps) du pixel courant. Puis, nous faisons la même chose pour les composants vert et bleu.



2.3.2 Filtre moyen

Nous avons mis en place un autre filtre moyen basé sur le même principe que le filtre précédent mais en faisant la moyenne des pixels situés à n position autour du pixel courant (verticalement, horizontalement ou les deux en même temps).



2.3.3 Filtre Gaussien

Le filtre Gaussien utilise le même principe que le filtre moyen mais de façon pondéré. Plus un pixel est proche du pixel courant plus son poids est important alors qu'un pixel éloigné a un poids beaucoup plus faible. Cela explique pourquoi l'utilisateur ne pourra pas choisir le pas.

$$\frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

2.3.4 Filtre de Sobel

Le filtre de Sobel permet de calculer approximativement le gradient de l'intensité de chaque pixel. Ceci permet d'indiquer la direction de la variation du clair au sombre (le gradient pointe dans la direction du changement d'intensité). Le gradient est nul lorsqu'on se trouve dans une zone d'intensité constante.

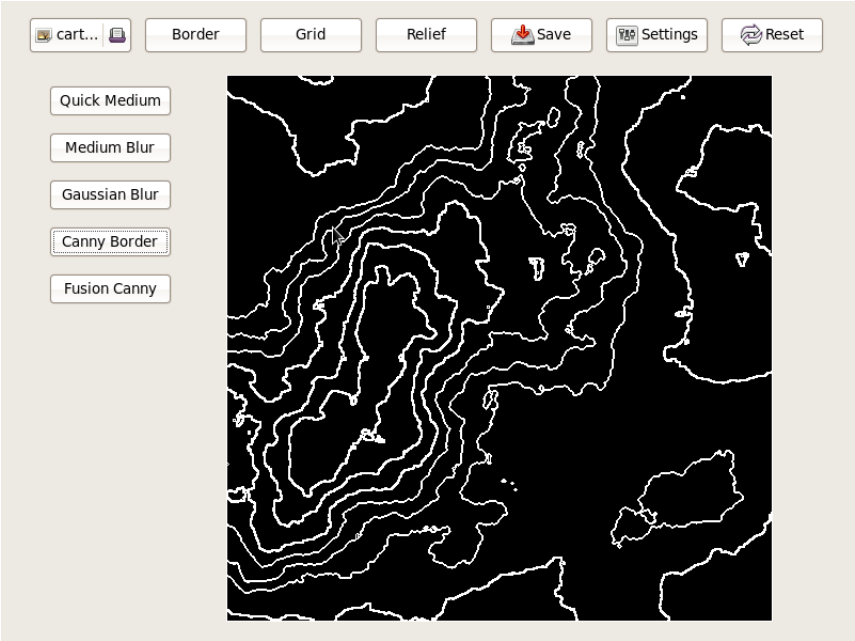
$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A \text{ et } G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

Légende : *A représente l'image source (tableau de pixel)*
G_x, le gradient horizontal
G_y, le gradient vertical
G, la norme du gradient

2.3.5 Algorithme de Canny

Le filtre de Canny se déroule en plusieurs étapes. La première est la réduction du bruit en utilisant le filtre Gaussien (mais rien n'empêche d'utiliser les filtres moyennes, à déterminer en fonction de l'image source et des contours qu'on veut obtenir). La deuxième est le filtre de Sobel qui va calculer le gradient de chaque pixel. Puis, la dernière étape est le seuillage des contours (plus précisément un seuillage par hystérésis) : si l'intensité du gradient est inférieur au seuil

bas, le pixel n'est pas un contour alors que si l'intensité est supérieur au seuil haut, le pixel est considéré comme un contour. Enfin, si l'intensité du gradient est entre le seuil bas et le seuil haut, le pixel est un coutour si celui-ci est connecté à un autre contour. On obtient ainsi une image binaire : le contour et les autres. L'image binaire pourra ensuite être fusionné à l'image source pour obtenir l'image avec des contours.



2.4 Gestion des couleurs et des altitudes

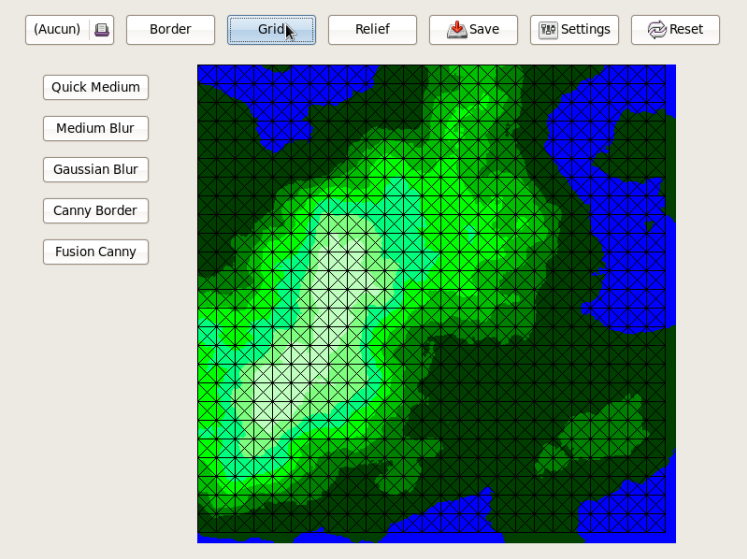
Notre objectif final est de reproduire en 3D, la carte topologique fournit au programme. Il était donc évident qu'il nous fallait pouvoir définir une altitude pour chaque couleur présente sur la carte. Nous avons d'abord commencé par mettre toutes les couleurs présentent dans une liste. Par la suite, nous avons codé une fonction qui permettait de prédéfinir dans une liste, les hauteurs pour chaque couleur. Pour cette première soutenance, nous avons décidé que la première couleur rencontrée sur la carte aura une hauteur nulle, la deuxième une hauteur de 10, la troisième une hauteur de 20 et ainsi de suite.
Ensuite il suffisait, de dire au générateur du fichier obj (nécessaire à l'ébauche du moteur 3D) de prendre ces altitudes pour les valeurs des composantes Y.

Chapitre 3

Échantillonnage de la carte

3.1 La triangulation

L'échantillonnage de la carte est surement la partie centrale de tout le projet, en effet ce traitement va permettre de récupérer les points de la carte nécessaire à la génération du fichier obj. Pour la représentation graphique du quadrillage, nous avons pris une technique différente de la technique habituelle. En effet, la méthode la plus simple voudrait que l'on trace simplement des lignes verticales, horizontales, et enfin des diagonales afin d'obtenir les triangles souhaités. Mais nous avons décidé de coder une fonction qui trace directement un carré avec ses diagonales, ainsi on peut directement rentrer dans notre fichier obj les points et les faces souhaitées. L'inconvénient est qu'il ya le présence de doublons de pixels lors de l'affichage du quadrillage, mais ceci est géré lors de la génération de l'obj, on a donc uniquement les faces souhaitées sans doublons.



3.2 Moteur 3D

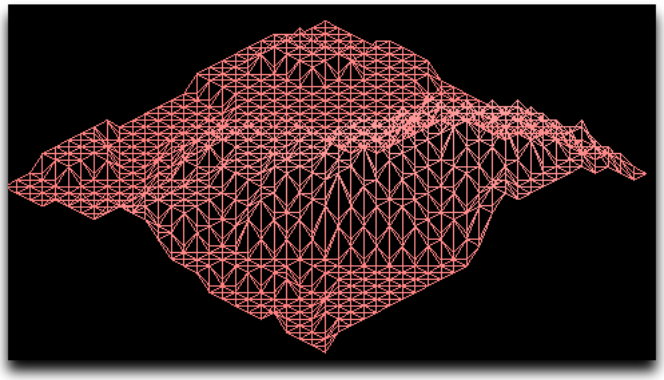
Le moteur 3D est la 2eme plus grande partie de AutoMap : elle se doit d'être intuitive et performante. Par défaut, nos racks sont équipés de la bibliothèque lablGL (un binding OpenGL v1 pour OCaml). C'est donc ce que nous utiliserons pour cette première soutenance. Cependant, il n'est pas impossible que l'on change pour glMLite qui bénéficie d'une version plus récente d'OpenGL (v2) où il est possible d'optimiser considérablement le moteur (plus de détails à la 2eme soutenance !).

Dans l'état actuel des choses, AutoMap affiche dans une fenêtre SDL séparée la visualisation 3D de la carte générée. Sa mise en place a été très rapide, il aura fallu plus de temps pour optimiser certaines méthodes barbares et utiliser des procédures plus ouvertes et donc moins restrictives à notre projet (AutoMap pourrait donc servir à visualiser d'autres modèles que les cartes générée).

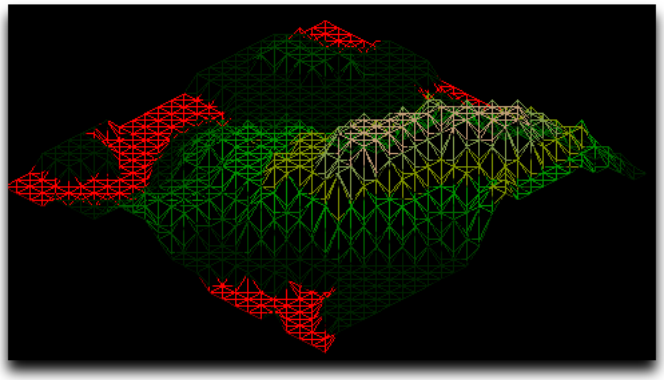
En effet, la carte générée est en réalité un fichier Wavefront (*.obj) qui contient une liste indexée de vertex et une liste de faces utilisant cette indexation pour éviter les répétitions de vertex (un vertex peut être attaché à plusieurs faces). Le moteur 3D lit donc une première fois le fichier à la recherche des faces (lignes commençant par "f"). Il construit une liste dynamique de ces dernières tout en essayant de trouver l'index maximal. En effet, ce nombre maximal détermine la longueur du tableau à taille fixe et à indexation rapide de vertex, qui sera parcouru dans un ordre non défini lors de l'affichage. Une fois la première passe terminée, on ferme le fichier et on créer un nouveau tableau (module Array en OCaml) de la taille précédemment déterminée. On rouvre le fichier pour capturer tous les vertex en les ajoutant à leur place dans le tableau. Cette méthode de double passe évite donc tout problème de génération, de fichier altéré, ou encore un ordre différent (définition des faces avant les vertex), tout en garantissant un accès rapide aux vertex lors de l'affichage.

Une fois la capture des données effectuée, il est grand tant de rentrer dans la boucle principale du programme qui gère aussi bien la souris (clic gauche pour bouger la vue, clic droit pour tourner autour du modèle, molette pour zoomer) que le clavier (ESC pour quitter, Z pour changer de mode de rendu). On peut en effet afficher le modèle dans plusieurs mode de rendu avec la touche Z :

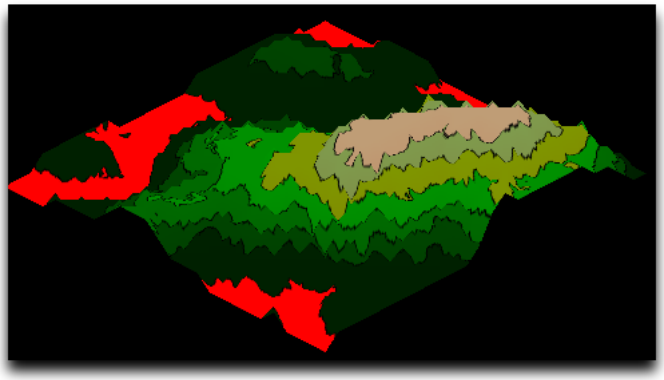
- fil de fer :



- fil de fer texturé (par l'image de l'utilisateur) :



- solide texturé :

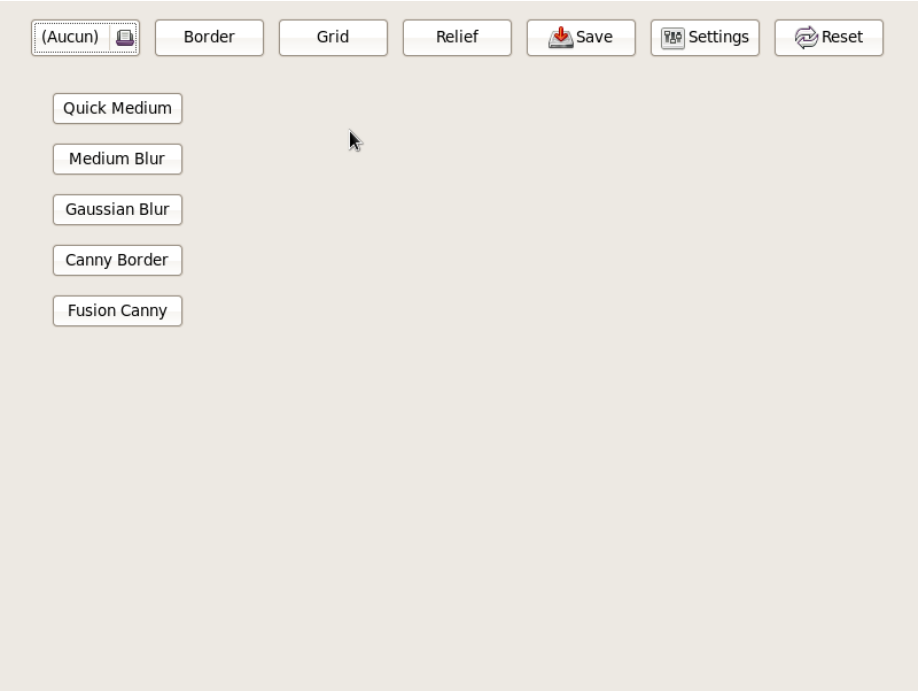


Nous avons commencé un algorithme de quad-tree qui devrait considérablement réduire le rapport temps de rendu / détails. La suite à la prochaine soutenance !

Chapitre 4

L'interface graphique

Pour l'interface graphique, il nous fallait encore utiliser une bibliothèque. Nous avons opté pour lablGTK, un binding entre GTK+ et Ocaml. L'avantage de cette bibliothèque est qu'elle est très complète et qu'elle permet de créer des interfaces graphiques avec de nombreux et divers éléments tels aue des boutons, des onglets, des barres de progression, des barres d'outils etc ...



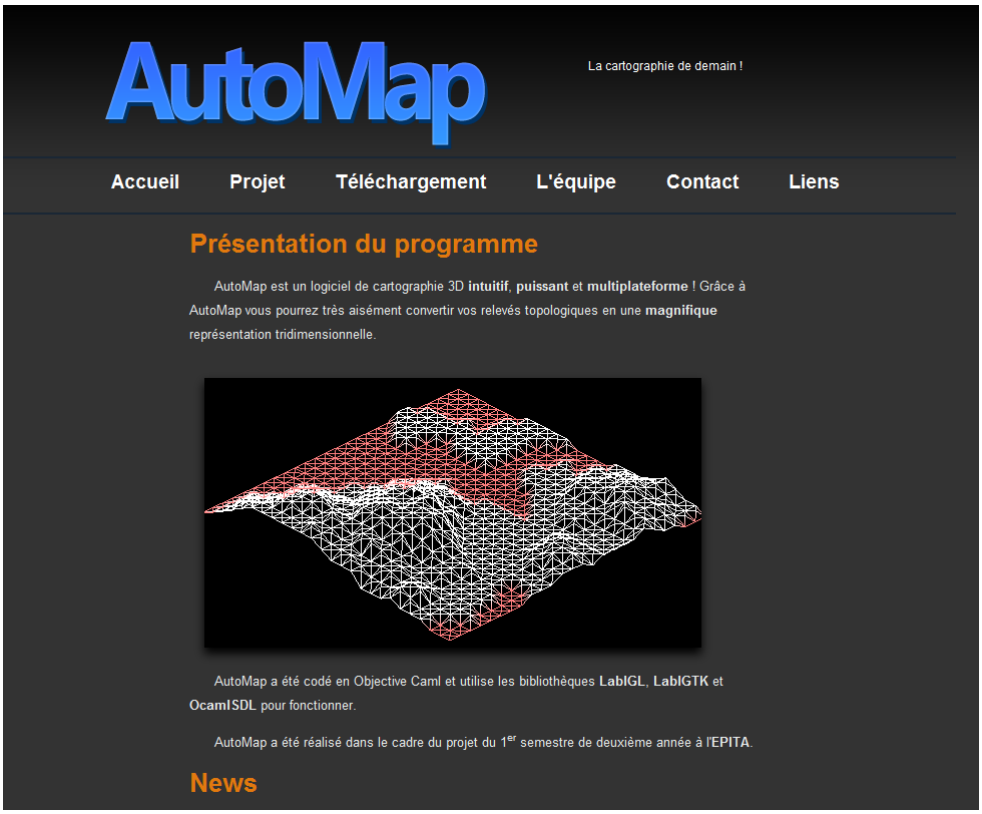
En revanche l'inconvénient de lablGTK, c'est le manque cruel d'exemples, et la difficulté de conception. Notre fenêtre est une GWindow, l'interface est composé de plusieurs boutons, permettant de loader une image, afficher les contours, afficher le quadrillage, lancer la visualisation 3D, sauvegarder le résultat, un bouton reset et un bouton option permettant à l'utilisateur de

définir le pas de l'échantillonnage (pas du quadrillage). Une des plus grandes difficultés a été de gérer le bouton qui permet d'ouvrir une image directement depuis son disque dur. En effet, la méthode "filename" que ce bouton possède renvoie une string 'option et non une string. Par la suite nous avons rajouté plusieurs boutons sur la gauche de l'interface permettant d'appliquer plusieurs traitements plus poussés comme un flou gaussien, un filtre médian ou encore l'application de l'agorithme de Canny. Évidemment, tous les traitement appliqués sur l'image s'affiche directement sur l'interface graphique et non pas dans une fenêtre SDL.

Chapitre 5

Site web

Un projet aussi conséquent a besoin d'être suivis par des internautes c'est pourquoi nous avons réalisé un site web. Il explicite l'avancée du projet et permet aux utilisateurs débutant de comprendre le principe de notre programme.
Le site est disponible à cette adresse : www.umad.fr.nf .



Chapitre 6

Objectifs pour la deuxième soutenance

Nous avons bien avancé pour cette première soutenance. Nous sommes en avance sur certains points, tel que la 3D, mais certains aspects du projet nécessitent d'être beaucoup plus approfondis. En effet, notre interface graphique reste encore assez basique, il faudra donc proposer à l'utilisateur beaucoup plus d'options. Il pourra donc choisir lui même l'altitude correspondante à chaque couleur de la carte, choisir entre plusieurs modes de triangulation pour la phase d'échantillonnage. Le moteur 3D devra également subir une révision complète pour permettre à l'utilisateur de se déplacer dans le modèle 3D grâce à une caméra à la première personne.

Chapitre 7

Conclusion

Globalement, à la fin de cette première soutenance, nous pouvons dire que nous sommes globalement en avance, surtout grâce à l'ébauche du moteur 3D déjà intégrée dans notre programme. Mais il nous reste encore beaucoup de choses à améliorer au point de vue des graphismes, des options ou encore de certaines optimisation à ajouter. La bonne ambiance du groupe nous permettra à coup sûr de mener ce projet à son terme.