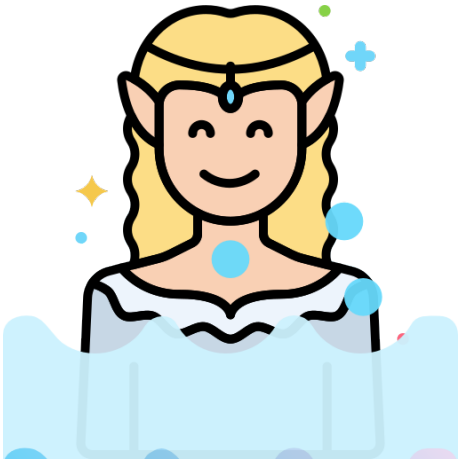
Rapport de soutenance 1

elfes & mer

Yann BOUDRY, Azéline AILLET, Pierre-Corentin AUGER

Vincent LIBESKIND, Scott TALLEC



Sommaire

1 Introduction

Ce rapport va vous présenter l'avancement du projet des elfes&mer. L'objectif final de notre projet est de pouvoir convertir une carte topographique en un modèle 3D explorable. Nous avons avancé sur trois parties différentes du projet en parallèle : le traitement de la carte et l'attribution du relief, la modélisation de la carte et la caméra libre, et la création de l'interface graphique et de notre site web.

La projet a avancé rapidement et nous sommes content de travailler ensemble sur ce projet complexe, impliquant énormément de fusions de codes et donc de discussions. Nous sommes curieux de savoir jusqu'où nous pourrons aller avec ce travail et nous allons nous donner au maximum pour voir nos objectifs se réaliser. Les cartes utilisées afin de tester notre programme sont issues du site *"https://www.geoportail.gouv.fr/donnees/carte-ign"*.

2 Présentation du groupe et du projet

2.1 Le groupe

Notre groupe s'est formé naturellement avec la classe S4E, la classe éphémère regroupant les élèves censés partir à l'étranger. Les membres du groupe elfes&mer sont: Yann Boudry notre chef de projet, ainsi que Azéline Aillet, Pierre-Corentin Auger, Vincent Libeskind et Scott Tallec. La classe étant seulement composée de 7 personnes, nous nous sommes très vite entendus, ce qui a mis une bonne ambiance dans le groupe. Malgré notre répartition dans différentes classes, nous essayons de faire plusieurs réunions afin de parler de l'avancement de chacun, et de mettre en communs les parties qui sont liées.

2.2 Notre projet

Nous avons décidé de développer un programme dont l'objectif principal est de traiter une carte topographique et d'effectuer une modélisation 3D. Le traitement de la carte topographique permettra d'extraire les lignes topographiques et de construire une carte en 3D. Différents paramètres tels que les routes, rivières pourraient être implémentées. Une caméra libre permettra de naviguer dans cette carte.

Le but de notre programme est de pouvoir convertir une carte topographique en un modèle 3D explorable. Cela pourrait permettre aux utilisateurs d'explorer une zone afin d'avoir un meilleur aperçu des distances ainsi que du dénivelé

de certains espaces géographiques. Ce logiciel pourrait être notamment utile aux aviateurs, randonneurs, ainsi qu'aux touristes.

2.3 Organisation dans le temps

Avancé du projet sur le semestre :

Tâches	Soutenance 1	Soutenance 2	Soutenance 3
Traitement de la carte	90 %	100 %	100 %
Attribution du relief	80 %	100 %	100 %
Modéliser la carte	40 %	80 %	100 %
Caméra libre	70 %	100 %	100 %
Amélioration du modèle	0 %	60 %	100 %
Site web	50 %	80 %	100 %
Interface	75 %	90 %	100 %

Répartition des tâches :

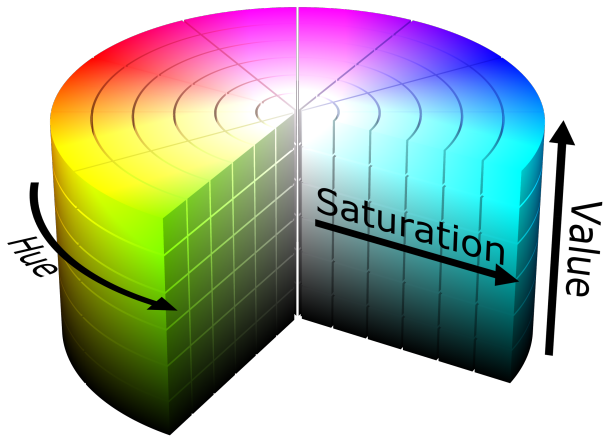
Tâches	Yann	Azeline	Pierre-Corentin	Vincent	Scott
Traitement de la carte		X		X	
Attribution du relief		X		X	
Modéliser la carte			X		X
Caméra libre			X		X
Amélioration du modèle	X	X			
Site web	X				

3 Traitement d'une carte topographique

3.1 Filtrage de la carte

3.1.1 Introduction

L'objectif de cette partie est l'extraction des lignes topographiques afin de pouvoir les exploiter pour l'attribution de l'altitude. Différentes méthodes existent afin d'extraire des lignes distinctes d'une image. Nous avons décidé d'effectuer un filtrage de couleurs afin de différencier les noms de villes, les rivières, les routes, le fond de la carte et les lignes topographiques. Nous allons utiliser les valeurs **RGB** (Rouge, Vert, Bleu en français) des pixels rencontrés afin de les convertir au format utilisé par le système **HSV** (Teinte, Saturation, Valeur en français). Le système **HSV** est une autre approche à la description d'une couleur.



HSV (Hue, Saturation, Lightness)

3.1.2 Conversion RGB en HSV

Formule de conversion des valeurs **RGB** des pixels en valeur **HSV** :

$$R' = R/255$$

$$G' = G/255$$

$$B' = B/255$$

Avec **R**, **G** et **B** les valeurs couleurs **RGB** du pixel.

$$Cmax = max(R', G', B')$$

$$Cmin = min(R', G', B')$$

$$\Delta = Cmax - Cmin$$

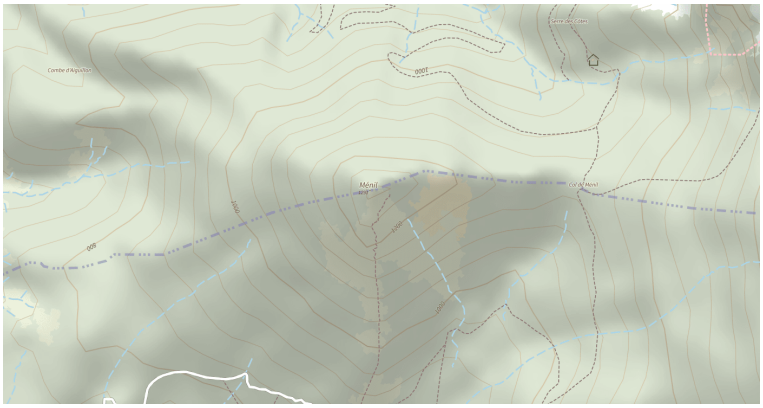
$$\text{H} = \begin{cases} 0°, & \Delta = 0 \\ 60° \times (\frac{G'-B'}{\Delta} mod 6), & Cmax = R' \\ 60° \times (\frac{B'-R'}{\Delta} + 2), & Cmax = G' \\ 60° \times (\frac{R'-G'}{\Delta} + 4), & Cmax = B' \end{cases}$$

$$\text{S} = \begin{cases} 0, & Cmax = 0 \\ \frac{\Delta}{Cmax}, & Cmax \neq 0 \end{cases}$$

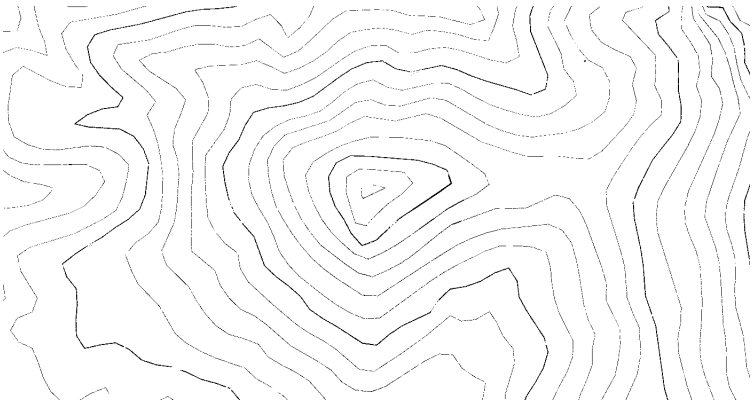
V = Cmax

La conversion des valeurs **RGB** de chaque pixel en **HSV** permet, par exemple, d'isoler les couleurs claires des couleurs sombres, ainsi que de récupérer certaines teintes de couleurs avec plus de précision. Les pixels appartenant aux lignes topographiques, rivières et routes sont isolées dans d'autres matrices afin de faciliter leur exploitation.

Nous utilisons pour notre projet la librairie **SDL2** afin de parcourir et traiter les cartes topographiques au format **BMP**.

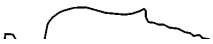


Carte IGN avant extraction



Lignes topographiques extraites

Rivières extraites



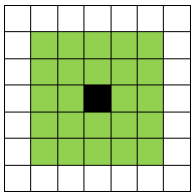
Extraction des routes

3.2 Reconstruction des lignes

Avec l'extraction des lignes topographiques de la carte IGN, l'apparition de trous dans les lignes est inévitable, nous devons donc les reconstruire. Pour ce faire, il faut dans un premier temps chercher les pixels correspondant à des fins de lignes, et dans un second temps relier ces pixels entre eux, de façon à reconstruire les lignes. Pour trouver les pixels de fin de ligne, nous devons tout d'abord amincir les lignes afin de n'avoir plus qu'un pixel d'épaisseur.

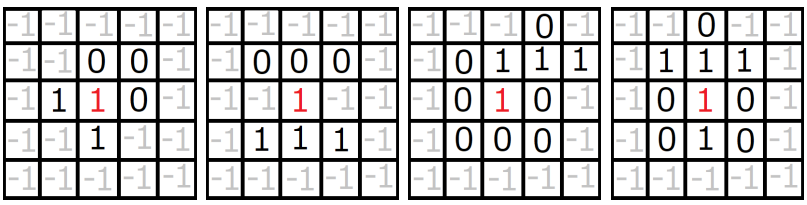
3.2.1 Amincissement des lignes

Pour l'amincissement des lignes, nous allons utiliser des filtres, qui correspondent à la position d'un pixel lorsqu'il peut être enlevé (mis en blanc). Pour savoir si un pixel peut être enlevé ou non, il faut qu'on regarde ses voisins. Les voisins d'un pixel sont les pixels adjacent à ce dernier (diagonales comprises). Dans notre cas, nous avons décidé de regarder les 24 voisins du pixel concerné, aussi appelé voisinage de Moore d'ordre 2.



Voisinage de Moore d'ordre 2 du pixel noir

Nos filtres ont donc 25 cases correspondant aux voisins et au pixel étudié situé au centre, avec des 1 pour le noir, des 0 pour le blanc, et des -1 pour les pixels dont la couleur peut être noir ou blanc. A partir d'un filtre correspondant à une situation dans laquelle le pixel peut être enlevé, nous créons d'autres filtres correspondant à la même situation en tournant les pixels. Voici quelques exemples de cas où le pixel peut être enlevé:



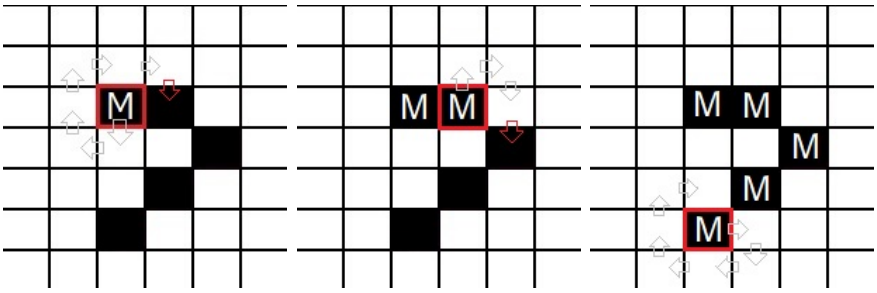
Exemples de pixels à enlever

3.2.2 Recherche des fins de lignes

Nous avons cherché comment nous pouvions trouver les pixels qui correspondent aux fins de lignes afin de les relier plus tard, et nous avons découvert la fonction Moore. Cette fonction prend le principe du voisinage de Moore, dont nous avons parlé dans la sous-partie précédente, et permet de retrouver le contour de lignes. Dans notre cas, nous allons l'utiliser pour retrouver les fins de lignes. Cet algorithme a besoin des voisins de Moore rangés dans l'ordre horaire en partant du voisin juste en dessous du pixel étudié. L'idée générale de l'algorithme est le suivant: à partir du voisin de dessous, passer sur chaque voisin du pixel dans l'ordre horaire, il y a alors 2 cas:

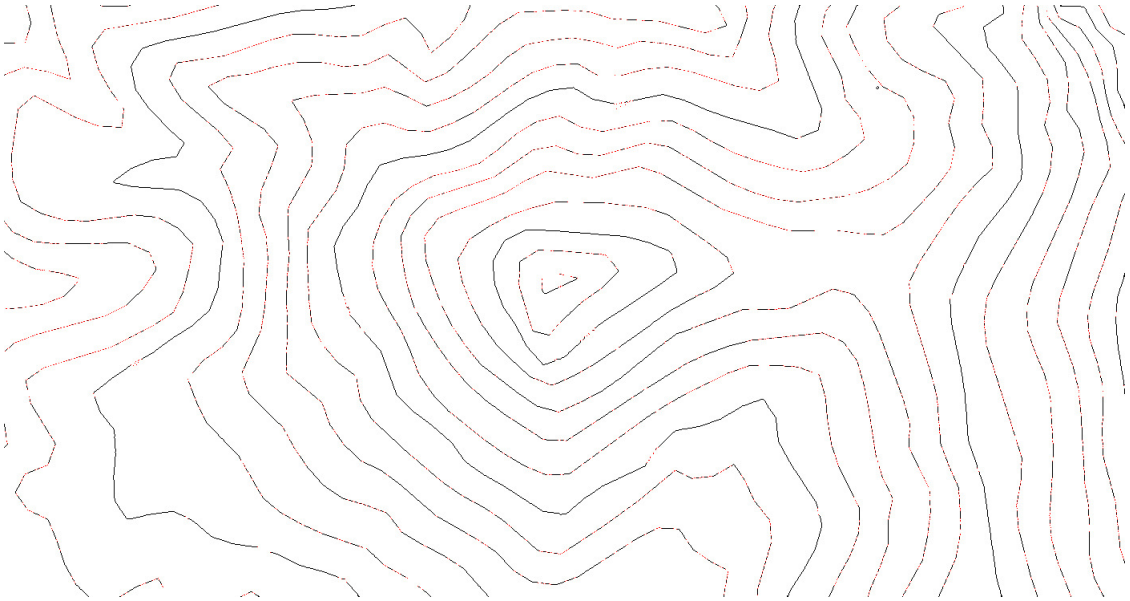
- Si le pixel est blanc, continuer avec les voisins du pixel
- Sinon (le pixel est noir), on revient en arrière sur le dernier pixel voisin blanc, et on relance alors l'algorithme sur ce nouveau pixel noir pour continuer la ligne

L'algorithme se finit lorsque le nouveau pixel noir correspond au pixel de départ. Dans notre cas, nous voulons marquer les pixels noir que nous rencontrons dans un tableau, et nous regardons si nous avons fait le tour du pixel, donc si nous n'avons pas trouvé de pixel noir autre que celui du début. Dans le cas où nous sommes sur le pixel de départ, il faut regarder les 8 voisins de Moore pour que ce soit une fin de ligne, mais si nous ne sommes pas sur le pixel du début, il ne faut en regarder que 7.



Simulation de l'algorithme modifié de Moore

Cet algorithme fonctionnait bien pour trouver les pixels de fin de lignes, cependant il était long et utilisait beaucoup de mémoire. Nous avons donc décidé de changer d'algorithme afin d'avoir un programme plus optimisé. Le nouvel algorithme est plus simple, il compte le nombre de pixels noir présents dans les 8 voisins du pixel étudié (voisinage de Moore d'ordre 1). S'il y a moins de 2 pixels noirs dans les voisins, ce pixel est une fin de ligne (nous les mettons en rouge pour plus de visibilité lors de la soutenance).



Après application de l'algorithme de recherche

3.2.3 Reconstruction des trous

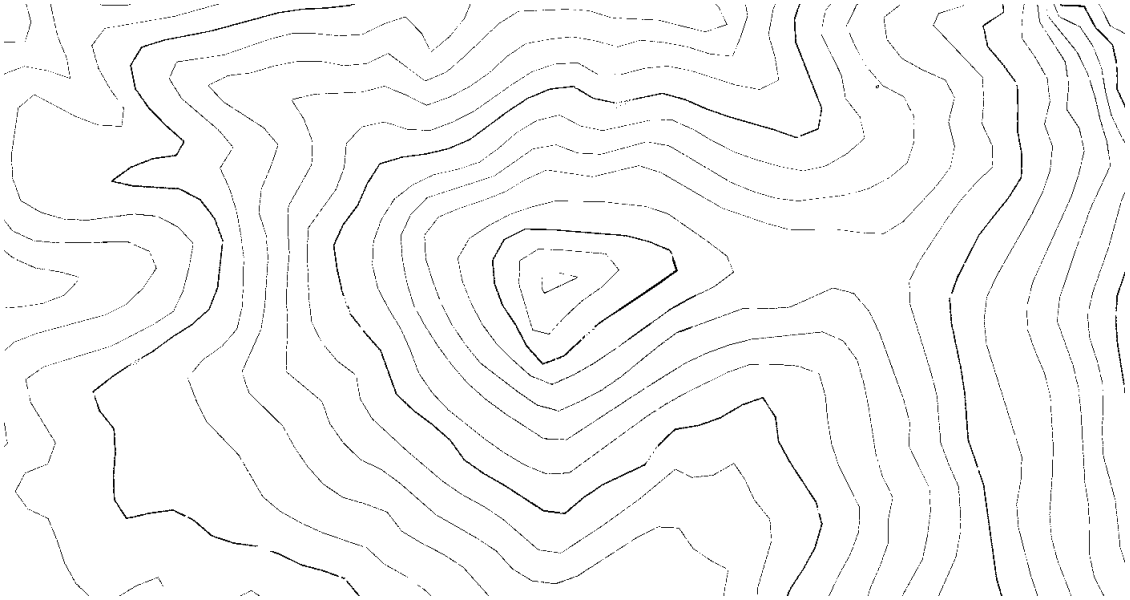
Pour reconstruire les lignes, il faut relier les pixels de fin de ligne entre eux, mais avec le bon. Pour chaque pixel de fin de ligne que nous avons marqué précédemment, et nous cherchons le pixel le plus proche qui est aussi une fin de ligne, et qui n'est pas déjà relié avec notre pixel de base. Pour savoir quel est le pixel le plus proche, nous calculons la distance euclidienne entre les 2 points. La distance euclidienne entre un point (x1,y2) et un point (x2,y2) est de:

$$D = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

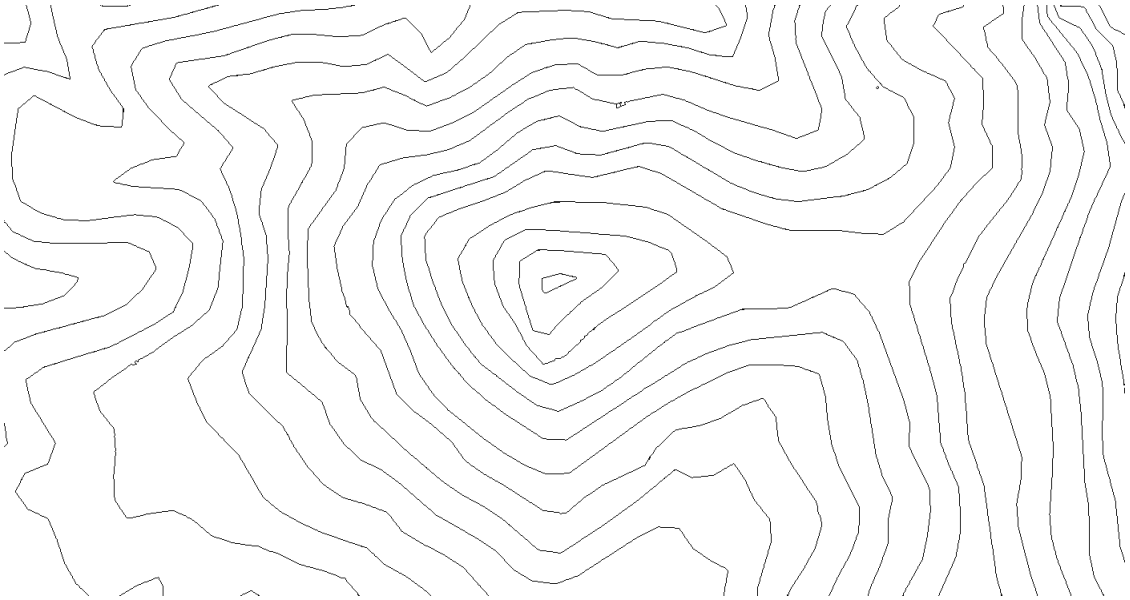
Les pixels de fin de ligne peuvent être reliés plusieurs fois dans certains cas si besoin. De plus, dans certains cas, le pixel de fin de ligne doit être relié avec le bord de l'image. Donc pour chaque pixel étudié, les 4 pixels collé au bord de l'image et de même abscisse ou ordonné que notre pixel sont rajoutés à la liste des pixels de fin de lignes, puis enlevé après l'étude du pixel. L'algorithme fonctionne comme suit pour chaque pixel de fin de ligne non relié: les 4 pixels du bord sont ajoutés à la liste, puis pour chaque pixel de la liste:

- La distance euclidienne est calculée
- Si on est sur un des pixel du bord, la distance est multipliée par 5 pour éviter que les point soient reliés trop souvent au bord
- On vérifie que la distance n'est pas trop grande
- On vérifie que les 2 pixels ne sont pas liés
- Si on est sur un des pixel du bord, on vérifie qu'une boucle n'est pas créée

Après avoir parcouru toute la liste, si on n'a pas trouvé de point correspondant à tous les critères, on supprime le pixel .Sinon, on trace une ligne entre le pixel étudié et le pixel correspondant à la plus courte distance.
Notre fonction principale de reconstruction des lignes rappelle plusieurs fois la fonction de recherche de fin de lignes et la fonction de reconstruction des trous, jusqu'à qu'il n'y ait plus de pixel de fin de lignes trouvé.



Lignes topographiques extraites

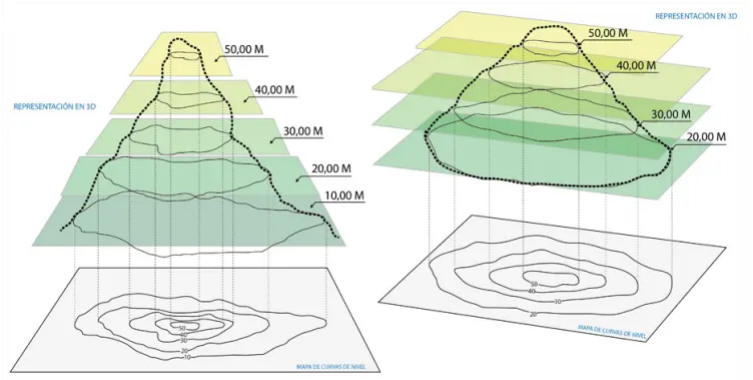


Après application de l'algorithme de reconstruction des lignes

4 Attribution du relief

4.1 Introduction

L'objectif de cet algorithme est d'attribuer pour chaque zone de la carte **IGN** une altitude afin de pouvoir réaliser la modélisation 3D de la carte. Le principal problème est qu'il y a peu d'informations disponibles sur les altitudes des lignes topographiques. On peut retrouver sur les lignes topographiques principales des indications d'altitude, mais elles sont difficilement exploitables. L'objectif pour cette première soutenance est de simuler l'altitude de chaque ligne topographique. Pour cela, on peut par exemple se baser sur les points les plus en altitudes qui sont les cercles non contenus dans d'autres cercles. En effet, ils représentent le sommet d'une montagne par exemple.



Exemple d'une modélisation 3D d'une carte topographique

Il faut ensuite leur attribuer l'altitude la plus élevée de la carte puis diminuer d'altitude pour chaque ligne topographique à proximité de ces points les plus hauts, et continuer ainsi jusqu'à ce que toutes les zones aient une attitude attribuée.

4.2 Labellisation des lignes topographiques et recherche des sommets

Il faut en premier lieu labelliser les zones (leur attribuer un numéro afin de les distinguer). Pour cela, l'image **BMP** contenant uniquement les lignes topographiques est parcourue. Dès qu'un pixel blanc non labellisé est trouvé, un parcours largeur est effectué à partir de celui-ci. L'objectif est d'attribuer le même label à tous les pixels d'une zone.

Les délimitations de cette zone sont les lignes topographiques ou les bordures l'entourant. Pendant le parcours largeur, toutes les lignes topographiques sont comptées. Un parcours profondeur appliqué dessus permet de les différencier. Ce système permet de détecter si l'on trouve une zone ne contenant pas d'autres zones.

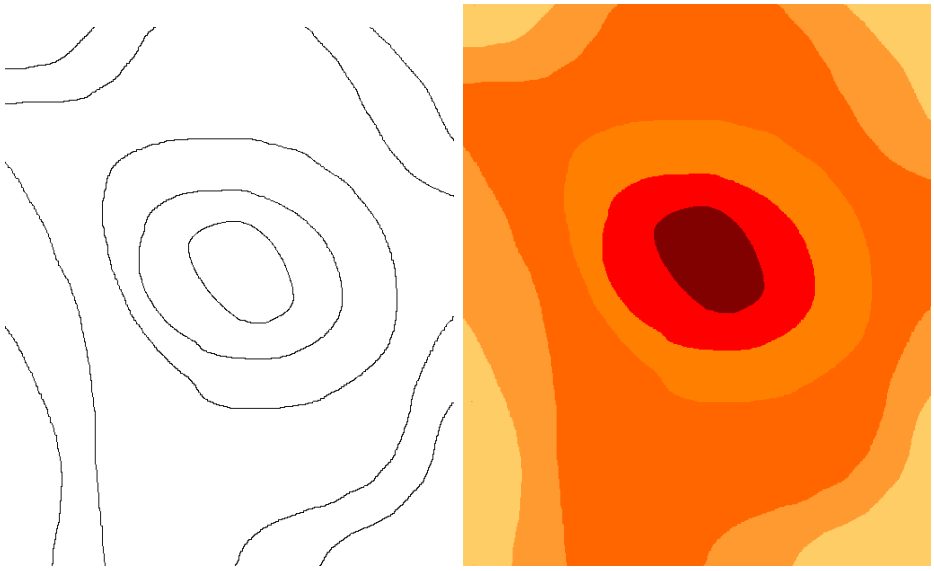
Dès qu'on tente un parcours sur un pixel en dehors de l'image, on peut alors en déduire qu'une bordure à été atteinte et que ce n'est pas une zone fermée, donc pas un **"cercle"**. Une fois le parcours largeur terminé, les labels correspondants aux zones fermées ne contenant pas d'autres zones sont récupérés.

4.3 Attribution de l'altitude

Une fois que toutes les zones fermées ne contenant pas d'autres zones sont récupérées, on effectue un parcours largeur dessus à l'aide d'une liste. Une altitude maximale est définie. Pour chaque zone dépilée, on réalise un parcours profondeur. L'objectif est de récupérer les potentiels labels dont l'altitude n'a pas encore été définie qui sont collées à cette même zone. On explore donc les zones voisines de la zone dépilée. Il faut également attribuer aux lignes topographiques (représentées par les pixels noirs) de la zone la bonne altitude. Une fois le parcours profondeur fini et le(s) label(s) récupérés, on les empile à leur tour dans la file.

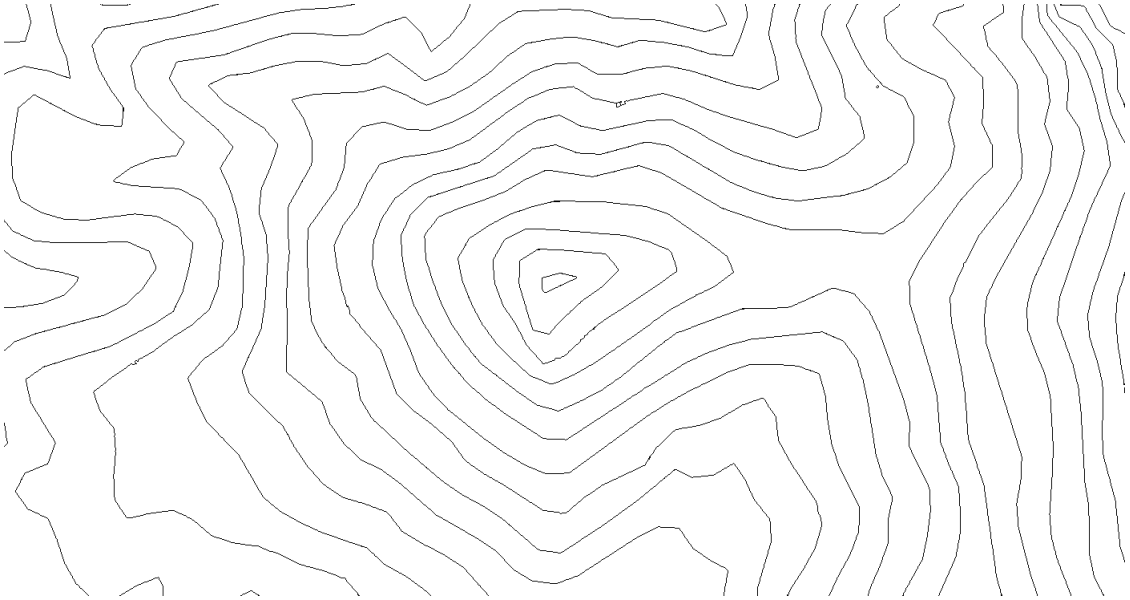
Ce processus est répété jusqu'à ce que la file soit vide. Cela permet donc d'explorer et d'attribuer la même altitude à chacun des voisins de la zone de départ. Il peut également y avoir plusieurs zones de départs. L'altitude est décrémentée afin de reconstituer l'altitude potentielle du terrain.

Toutes les altitudes de chacune des lignes topographiques sont stockées dans une matrice qui sert à la modélisation 3D de la carte IGN.

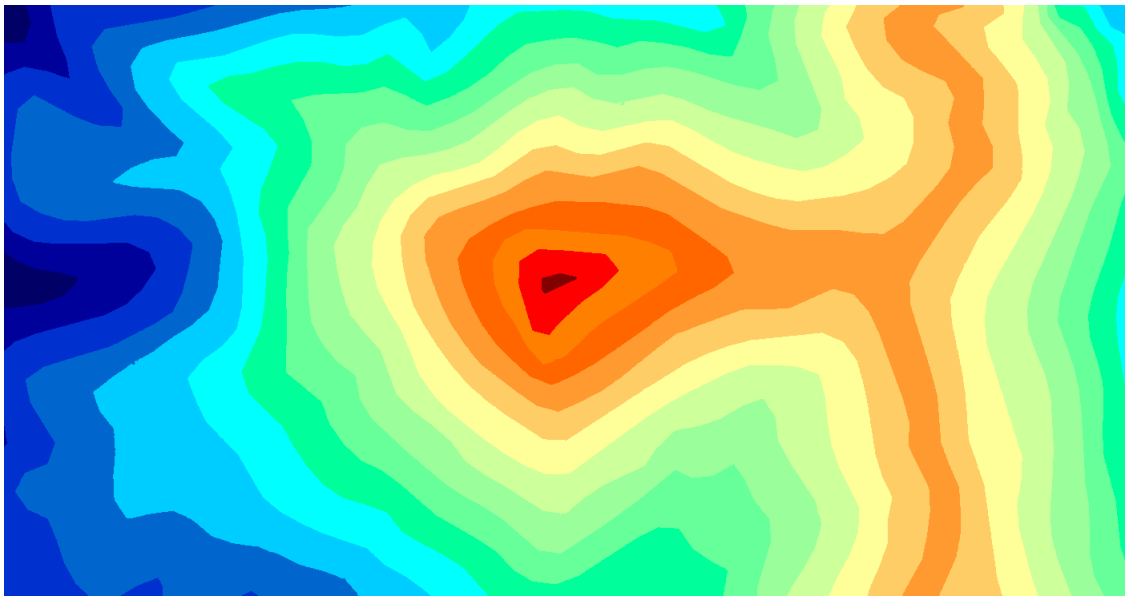


Avant **Après**

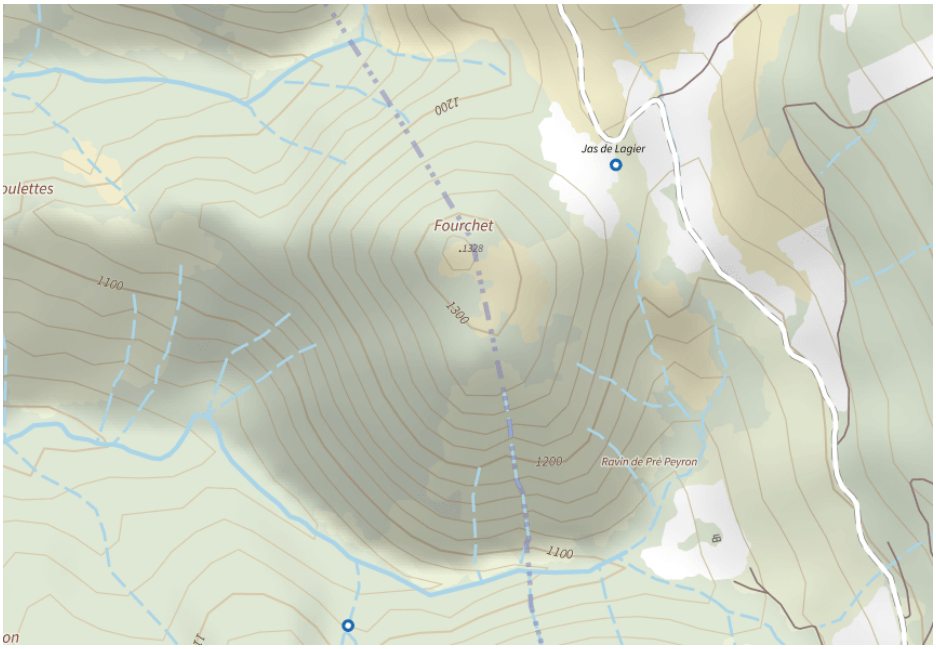
Note : plus les couleurs sont foncées, plus l'altitude est élevée.



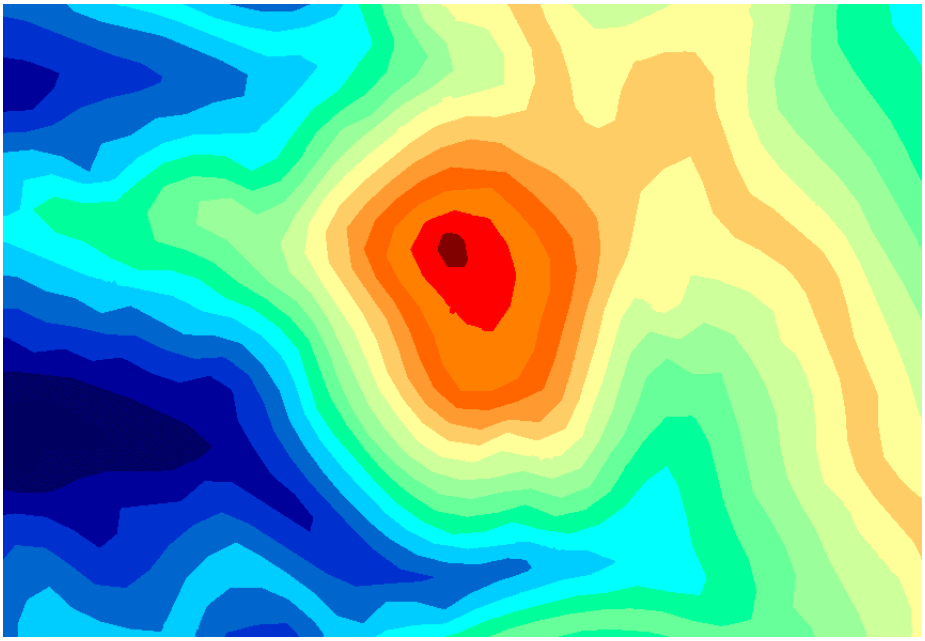
Lignes topographiques extraites et traitées



Carte résultante de l'algorithme d'attribution de l'altitude



Carte topographique IGN (Institut Géographique National)



Carte résultante de l'algorithme d'attribution de l'altitude

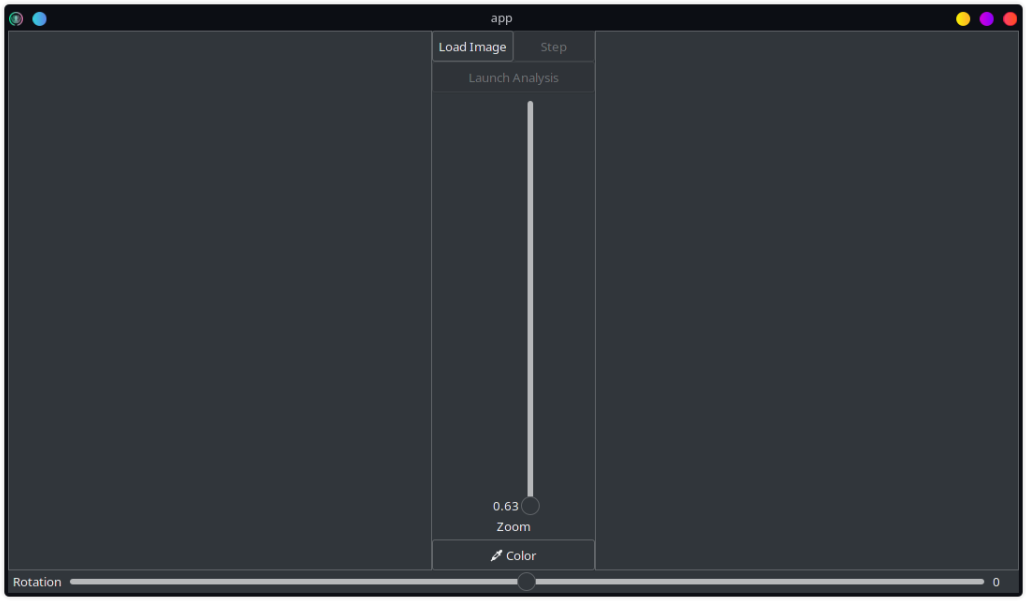
5 Application

5.1 Introduction

L'objectif est de mettre en commun toutes les parties du projet et avoir une interface graphique. Il faut pour cela créer une application. Pour cela nous avons décidé d'utiliser la librairie GTK3, associé à l'outil Glade qui permet de créer l'application graphiquement, puis de charger tous les composants dans notre programme.

5.2 La fenêtre principale

Voici la fenêtre qui apparaît au lancement de l'application. Elle est constituée de différentes parties:



Écran principal

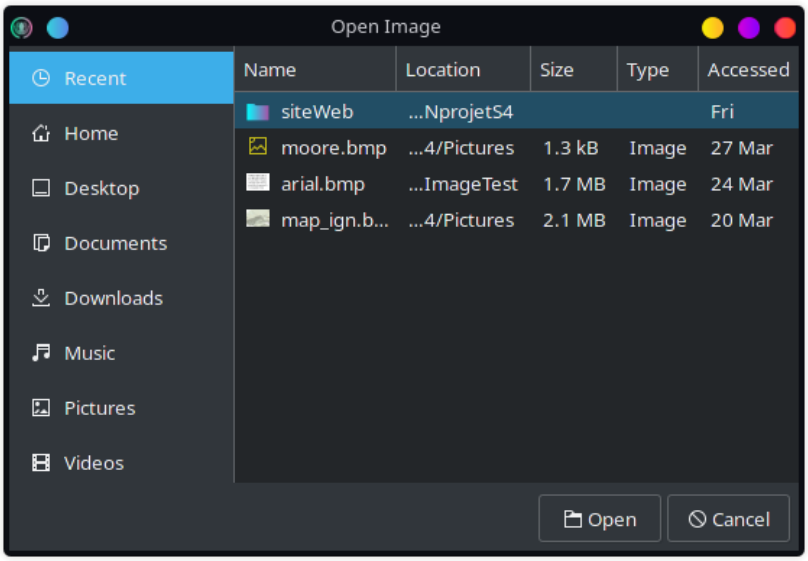
- Deux zones d'images, vide au lancement, à gauche et à droite, qui permettent de visualiser la carte IGN que l'on souhaite analyser. Au chargement de la carte, celle-ci est affichée identiquement sur les deux zones. C'est lors de l'analyse que la zone de droite sera mise à jour. Elles sont synchronisées en zoom, rotation, déplacement.
- Une zone qui regroupe les contrôles de l'application, en T inversé, et qui permet donc d'utiliser l'application à proprement parler.

5.2.1 Les commandes

Au lancement, les boutons "Step" et "Launch Analysis" ne sont pas cliquables car il faut en premier lieu charger une image. Le bouton "Load Image" permet cela. Il ouvre une fenêtre secondaire d'explorateur de fichier pour choisir une image BitMap à charger. Une fois l'image chargée, les deux premiers boutons deviennent sensibles pour permettre le lancement de l'analyse. "Step" effectue une analyse en pas à pas, en affichant chaque image après le nouveau traitement dans la zone de droite, alors que "Launch Analysis" fera tout sans étape intermédiaire et affichera le résultat final. Deux slider permettent de contrôler le zoom et la rotation de l'image, allant respectivement de 0.5 à 15 et de -180° à +180°. Finalement, le bouton "Color" ouvre une autre fenêtre de choix de couleur, qui est utilisé pour trouver les lignes topographiques sur la carte.

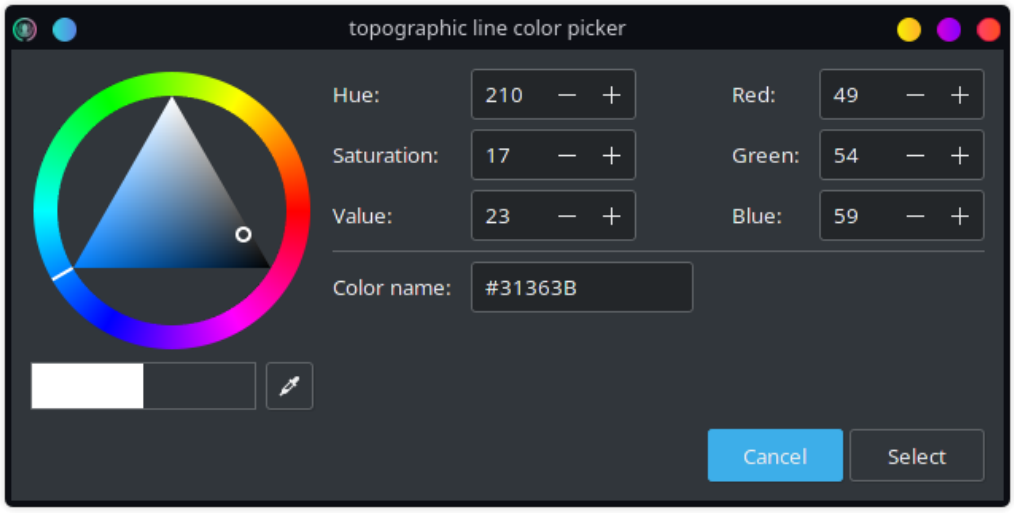
5.3 Les fenêtres secondaires

L'application propose 2 fenêtre secondaire pour son utilisation:



Explorateur de fichier

Celle-ci permet de choisir l'image à ouvrir pour l'analyse. Le type de fichier est restreint aux images BitMap car seul ce format est supporté.



Choix de couleur

Cette dernière est utilisée pour choisir la couleur des lignes topographiques à extraire. il est possible d'entrer les valeurs RGB directement mais le plus utile est la fonction pipette sur la partie gauche de la fenêtre qui prend la couleur présente sous la souris au moment du clic. A utiliser sur l'image après agrandissement grâce au zoom pour une meilleure précision.

6 Modélisation de la carte

La carte doit cependant être modélisée en 3D. Pour ce faire, nous avons choisi d'utiliser OpenGL, en particulier Glut (OpenGL Utility Toolkit). C'est une bibliothèque utilitaire offrant un jeu de routines pour la gestion des fenêtres OpenGL et les interactions avec le système d'exploitation, chose très utile pour notre rendement 3D.
L'avantage d'OpenGL est sa portabilité. En effet, il fonctionne sur la majorité des systèmes d'exploitation contrairement à DirectX qui marche exclusivement sur les plateformes Microsoft.

6.1 OpenGL

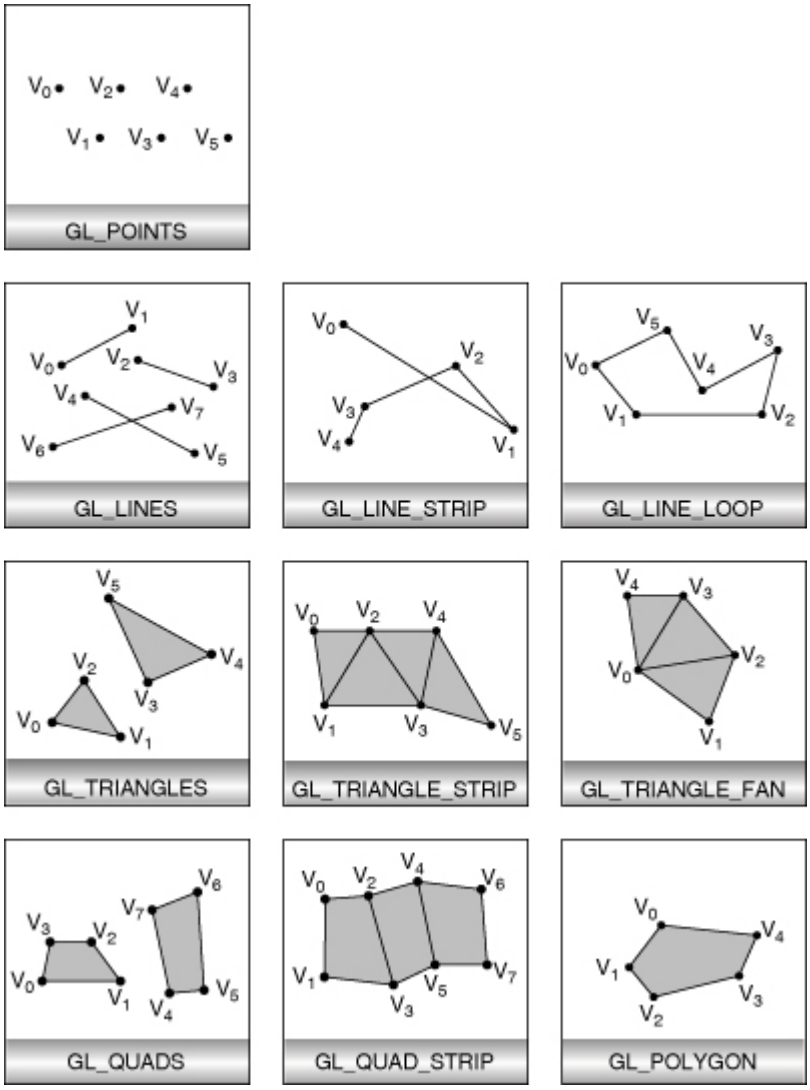
OpenGL a la particularité d'être une machine d'état. De ce fait, notre mise en œuvre repose sur la définition des états. L'API regroupe un nombre important de fonctionnalités pour éditer un état. Cela signifie que beaucoup de fonctions sont "contextuelles".

Dans notre fonction, nous initialisons donc évidemment OpenGL.

6.2 Traçage des Vertex

Après avoir défini l'état de notre machine à travers divers appels d'API, par exemple celles qui définissent la Transformation Viewport, la Transformation de Projection, la transformation Model-view (important plus tard pour la caméra), on peut commencer à fixer les figures complexes à tracer à partir de primitives géométriques qui sont les polygones les plus simples que l'API peut dessiner.

Pour tracer notre modèle, plusieurs possibilités se présentent. L'API nous permet donc de tracer les primitives géométriques suivantes:

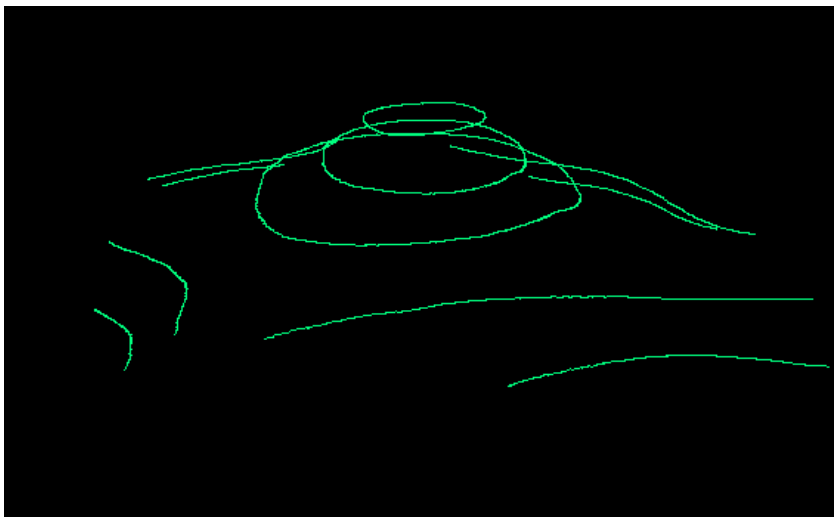


Représentation Graphiques des Primitive

Une possibilité éventuelle pour notre projet serait de tracer des triangles où un sommet appartient à une ligne de contour, et les autres à une ligne de contour adjacentes à celle-ci (GL_TRIANGLE_STRIP).

Pour notre première implémentation, nous avons décidé de tracer les points (GL_POINTS) à partir du tableau 2D décrivant les altitudes des lignes de contours.

Les points sont déterminés à partir du tableau des altitudes élaboré précédemment(lors de la colorisation de la carte). Pour notre première implémentation, nous allons nous contenter de parcourir ce tableau et de tracer les valeurs non-nulles. Avec la méthode mentionnée ci-dessus on obtient donc le résultat suivant:



Représentation Graphiques de la Topographie d'une Zone

7 Caméra libre

Au sein d'OpenGL, on trouve la fonction, gluLookAt, définie comme ceci :

```
void gluLookAt(GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ,
               GLDouble centerX, GLDouble centerY, GLDouble centerZ,
               GLDouble upX, GLDouble upY, GLDouble upZ)
```

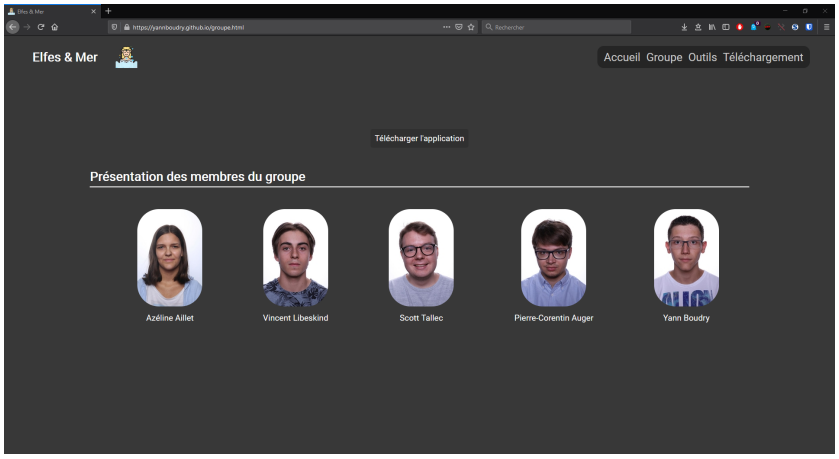
Cette fonction permet de simuler une caméra tel que :

- La position de la caméra est définie par eyeX, eyeY et eyeZ;
- La position du point que l'on regarde est définie par centerX, centerY et centerZ;
- L'orientation de la caméra est définie par upX, upY et upZ.

Ainsi, en initialisant des variables globales et en les assignant à gluLookAt, puis en les modifiant derrière, on peut faire bouger la caméra. Par exemple, pour faire une translation sur l'axe X, on incrémente ou décrémente eyeX et centerX par la même valeur.
Enfin, pour lier ces mouvements de caméra aux touches du clavier, on crée deux fonction qui seront appelées dans le main par glutKeyboardFunc ou glutSpecialFunc s'il s'agit d'une touche non-ASCII (comme les flèches ou bien F1, F2...).

8 Site Web

Le site est accessible en ligne gratuitement grâce à GitHub à l'addresse: https://yannboudry.github.io



8.1 Accueil

Cette page présente les évolutions du projet depuis sa création, les modifications les plus récentes apparaissant en premier.

8.2 Groupe

Il s'agit ici de la présentation des membres du groupe par une photo de chacun, ce qui vous permettra de nous reconnaître, une fois que nous pourrons de nouveau vivre sans masques.

8.3 Outils

Cette page liste tout les outils utilisés par notre groupe pour la réalisation du projet. (Git, GitHub, html, ...)

8.4 Liens et sources

Vous retrouverez finalement ici tout les documents réalisés pour le projet ainsi que l'application à télécharger

9 Conclusion

Nous sommes donc satisfaits de notre travail pour cette soutenance. Nous avons respecté nos objectifs en privilégiant le traitement et l'attribution de l'altitude de la carte topographique. Nous avons commencé l'implémentation de la modélisation 3D et fini la caméra libre. Pour la prochaine soutenance, nous souhaitons totalement finir la partie traitement de la carte, qui nécessite encore quelques optimisations et ajustements. Notre objectif pour la partie modélisation est d'avoir une base solide afin de terminer notre projet dans les temps pour la dernière soutenance.