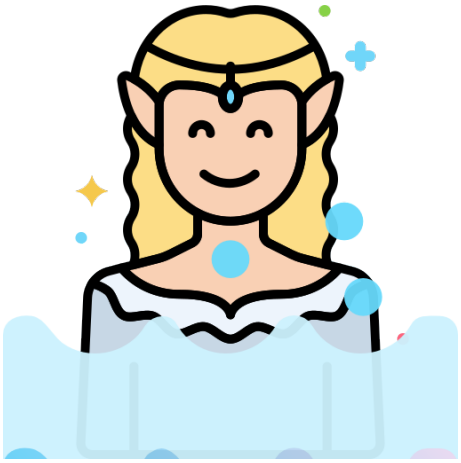
Rapport de soutenance 2

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 depuis la dernière soutenance. L'objectif final de notre projet est de pouvoir convertir une carte topographique en un modèle 3D explorable. Nous avons gardé la même répartition que pour la première soutenance, donc nous avons avancé sur trois parties différentes du projet en parallèle : l'amélioration et la fin du traitement de la carte et l'attribution du relief, la modélisation de la carte et la caméra libre, et l'amélioration 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 %	90 %	100 %
Amélioration du modèle	0 %	50 %	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

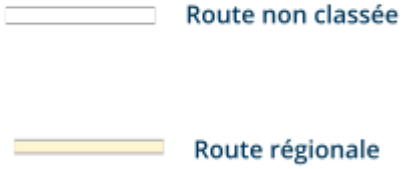
3.1 Filtrage de la carte

3.1.1 Introduction

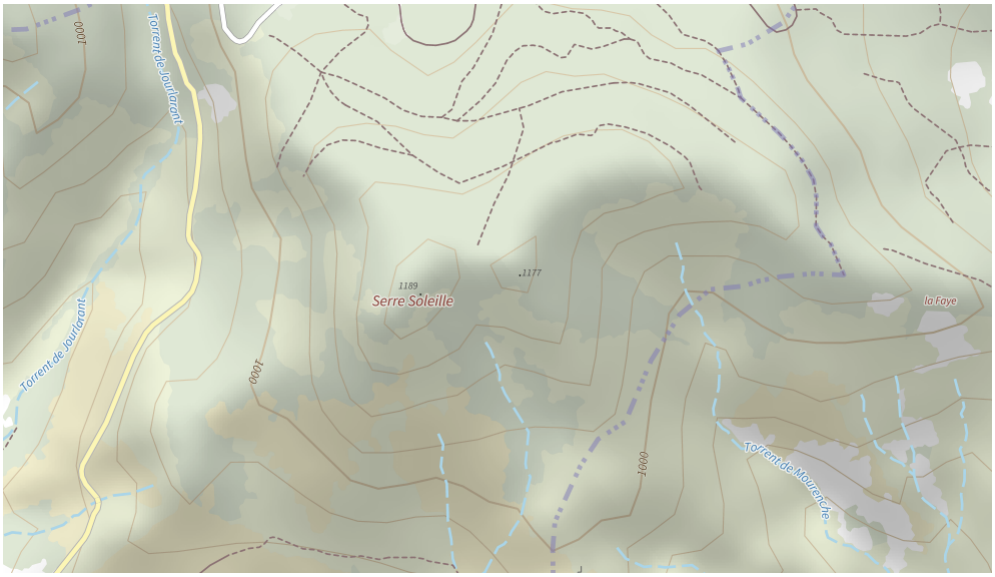
Lors de la première soutenance, le système d'extraction des lignes topographiques, routes et rivières a été mis en place. Ce système fonctionnait sur les cartes IGN issues du site *"https://www.geoportail.gouv.fr/"*. Pour cette seconde soutenance, nous avons décidé d'améliorer ce système d'extraction et de le compléter. Deux modes sont disponibles : un mode automatique (qui ne fonctionne que sur les cartes IGN issues du site *geoportail.gouv.fr*), et un mode manuel, qui peut s'adapter à des cartes personnalisées.

3.1.2 Nouveaux éléments extraits

Pour rappel, lors de la première soutenance, les lignes topographiques routes et rivières étaient extraites. Lors de l'étude des cartes **IGN**, il nous a semblé permettant d'extraire également les sentiers ainsi que de différencier deux types de routes : les routes locales ainsi que les routes non classées. En effet, ces deux types de routes sont facilement distinguables et présents sur la plupart de nos cartes.



Légende des routes

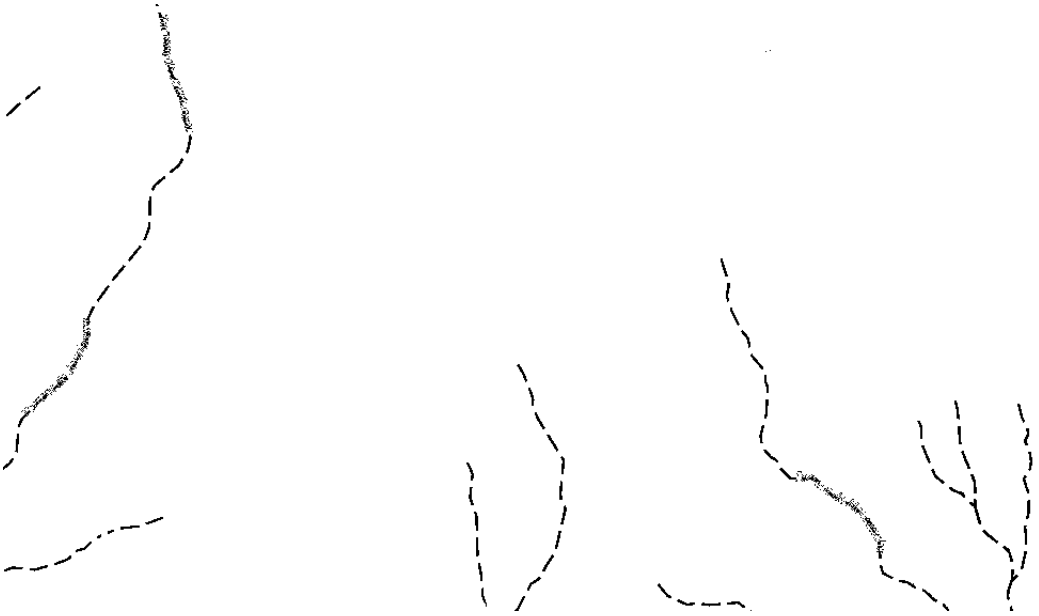


Carte IGN

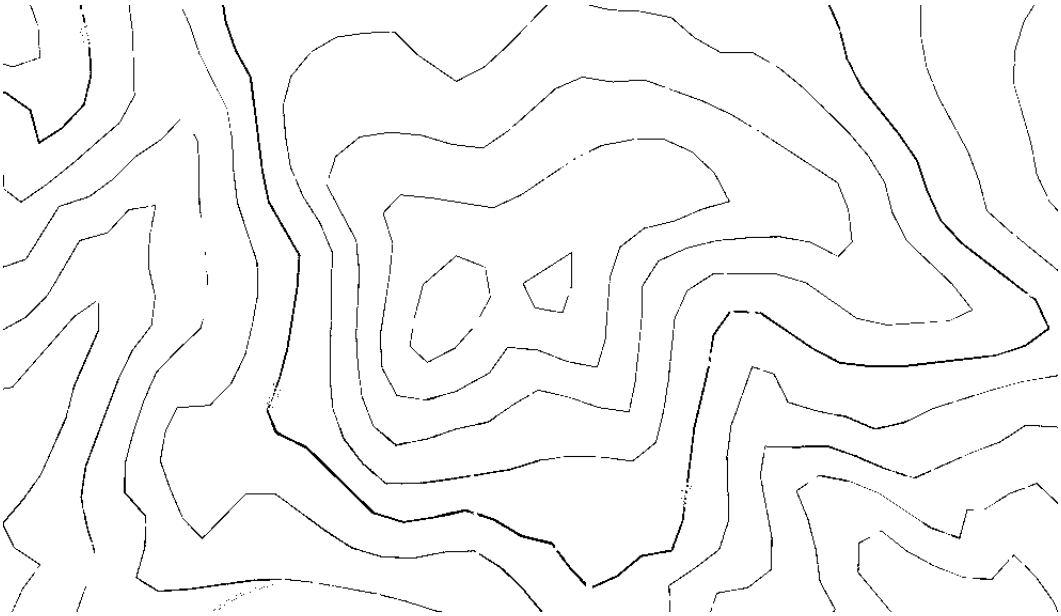
Route régionale

Route non classée

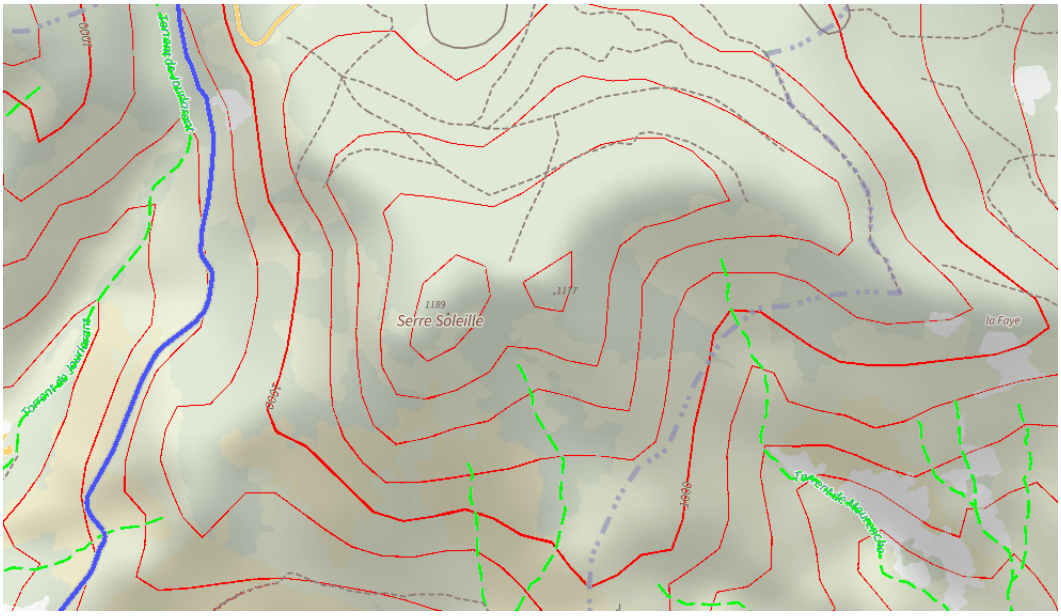
Sentier



Rivières



Lignes topographiques



Carte IGN avec tous les éléments extraits

Ci-dessus est présent un récapitulatif de tous les éléments extraits de la carte IGN lors de cette soutenance seconde soutenance.

3.1.3 Deux modes de fonctionnement

Comme énoncé en introduction, l'extraction possède deux modes de fonctionnement : un mode manuel et un mode automatique. Le mode manuel permet d'extraire à l'aide de 3 pipettes de couleurs les valeurs R, G, B des couleurs des lignes topographiques, routes et rivières. Un simple parcours sur l'image permet de récupérer les pixels correspondant à ces éléments à l'aide de 3 tableaux différents. Le mode automatique est identique à celui de la première soutenance. Il fonctionne uniquement sur les cartes IGN issues du site *"https://www.geoportail.gouv.fr/"*. Des tableaux permettent également de stocker les éléments extraits.

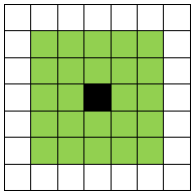
3.2 Reconstruction des lignes

3.2.1 Introduction

Lors de la première soutenance, l'algorithme permettant de reconstruire les lignes précédemment extraites a été créé. Pour rappel, celui-ci permet de remplir les trous apparu dans les lignes. Il commence par amincir les lignes, puis il recherche les pixels correspondant à des fins de lignes, et enfin il relie ces pixels entre eux, de façon à reconstruire les lignes. Cet algorithme fonctionnait bien avec la seule carte que nous avions lors de la première soutenance, mais lorsque nous l'avons lancé avec d'autres cartes, il ne fonctionnait pas bien. En effet, avec la nouvelle carte, certaines lignes étaient reliées entre elles, il restait des trous, et il y avait des traits en trop. Pour résoudre ces problèmes, il a fallu qu'on modifie les filtres d'amincissement des lignes, et qu'on améliore la fonction de reconstruction des lignes.

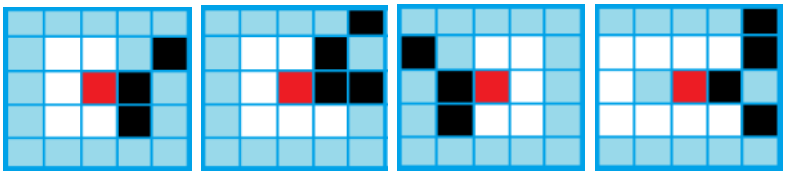
3.2.2 Filtrage de la carte

De base, les filtres que nous avions créés étaient seulement pour amincir les lignes, cependant, nous avons remarqué qu'il faillait aussi s'occuper des pixels seuls et qui ne correspondent pas aux lignes. En effet, lors de l'extraction des lignes, certains pixels possèdent la même couleur que celle des lignes, ainsi ils sont extrait. Pour s'occuper de ces cas, nous avons rajouté de nouveaux filtres. Nous allons aussi rajouter d'autres filtres pour prendre en compte des cas que nous n'avions pas vu avec la première carte. Pour rappel, les filtres que nous utilisons permettent de savoir quels pixels noir doivent être enlevés (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

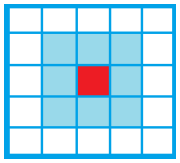
Nos filtres ont donc 25 cases correspondant aux voisins et au pixel étudié situé au centre, avec des 1 pour le noir, des 0 pour le blanc, et des -1 pour les pixels dont la couleur peut être noir ou blanc. Voici quelques exemples de cas que nous avons rajoutés:



Exemples de pixels à enlever

Note : rouge: pixel noir étudié, bleu: pixel noir ou blanc (-1)

Pour le cas des pixels seuls et ne correspondant pas aux lignes, on enlève déjà les pixels qui ont aucuns pixels noir dans son voisinage de Moore d'ordre 1. Ensuite nous créons un filtre qui regarde si le pixel et ses voisins d'ordre 1 sont entouré de blanc:



Filtre spécial

Si ce filtre passe, le pixel n'est pas tout de suite enlevé, les pixels voisins aux voisinages de Moore d'ordre 2 sont aussi étudiés. On vérifie qu'ils sont tous blancs, car si ce n'est pas le cas, les pixels appartiennent à une ligne. Ainsi, s'ils sont tous blanc, on enlève le pixel étudié.

3.2.3 Reconstruction des lignes

La reconstruction des lignes, se fera à peu près de la même manière que pour la première soutenance, mais avec quelques modifications. Pour rappel, notre algorithme de base recherchait pour chaque pixel de fin de ligne précédemment marqué, le pixel le plus proche qui est aussi une fin de ligne. Ce pixel le plus proche devait répondre seulement à deux critère: ne pas être plus loin qu'une certaine valeur, et ne pas déjà être relié avec le pixel étudié. Pour ce faire, on traçait une ligne imaginaire entre ces 2 points, et si on tombait sur un pixel noir, alors les points étaient déjà reliés. Cependant, cela ne suffit pas car les 2 points peuvent être relié par une courbe.

Donc pour cette deuxième soutenance, on a rajouté une fonction qui parcours les pixels noir voisins du pixel de base, et si on tombe sur le deuxième pixel, alors ils sont liés. Pour éviter de partir trop loin sur la ligne et d'empêcher de fermer une ligne de relief continue formant un cercle, on met une distance maximale à parcourir. Ainsi, les boucles involontaires qui auraient pu être formée si les points avaient été reliés, n'existent plus. Dans la nouvelle version de la fonction, les points ne peuvent plus être reliés plusieurs fois car il n'y a plus de pixels tout seul.

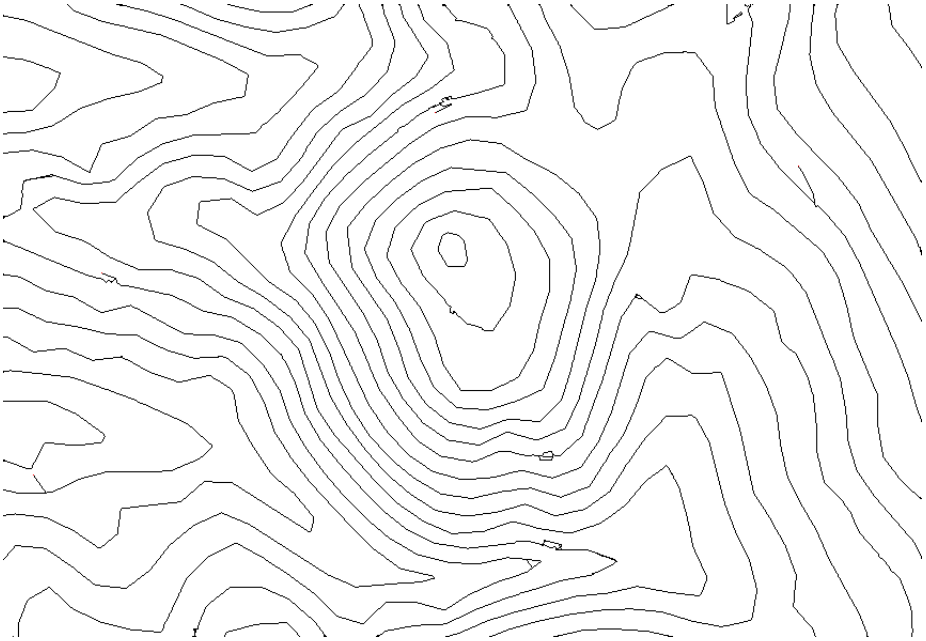
En plus de ces amélioration, la fonction va se transformer en fonction récursive. En effet, de base, on cherchait juste le pixel de fin de ligne le plus proche grâce à la distance euclidienne, et répondant aux critères, cependant maintenant, il va falloir vérifier que la plus petite distance est réciproque. C'est à dire que pour un point A étudié, s'il trouve que le point B est celui le plus proche, alors, il faut vérifier que pour le point B, le point A est aussi le plus proche. Pour ce faire, la fonction va devenir récursive, et dès qu'on trouve une distance minimale correspondant à un pixel (B) de fin de ligne, pour un pixel (A) étudié, on rappelle notre fonction pour ce nouveau pixel (B) avec la distance trouvée. Il y a alors deux cas possibles:

- Le point trouvé pour le nouveau point (B) correspond à l'ancien point (A), ainsi on prend la distance calculée comme distance minimale, puis on continue la liste, et si à la fin du parcours de la liste on a pas trouvé de distance plus petite, on trace un trait en les pixels (A) et (B)

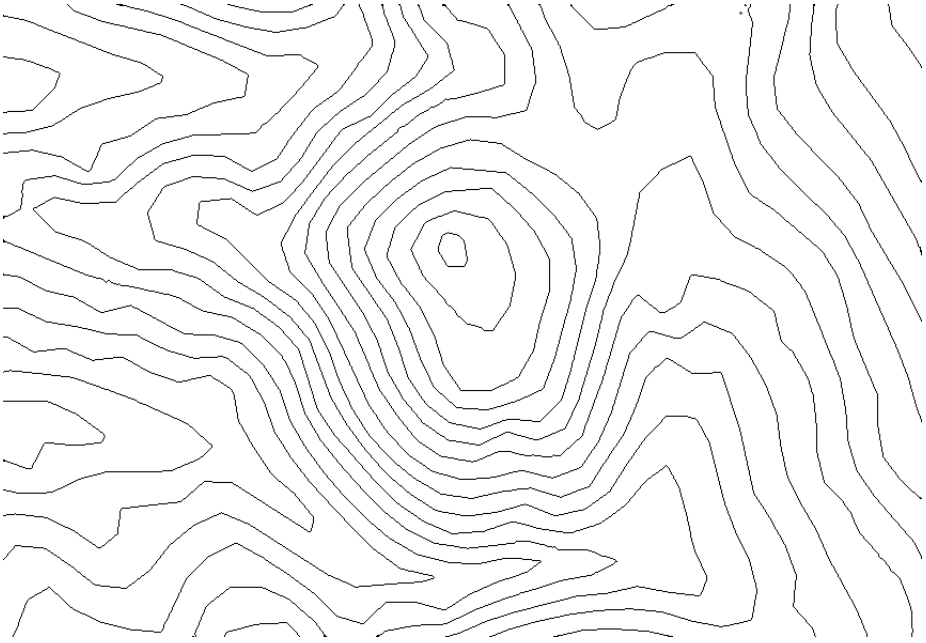
- Sinon, le nouveau point (B) a été relié à un autre point plus proche pour lui, et dans ce cas, on ne prend pas la distance calculée comme distance minimale et on continue la liste de fin de lignes

Dans certains cas, le pixel de fin de ligne doit être relié avec le bord de l'image. Pour ce faire, après avoir parcouru toute la liste de fin de ligne, on calcule la distance pour les 4 pixels collés au bord de l'image et de même abscisse ou ordonné que notre pixel étudié. Bien évidemment, dans ce cas, il n'y a pas besoin de rappeler notre fonction puisque les points collés au bord sont fictifs. De plus, dans ce cas, la distance calculée est multipliée par 5 pour éviter que les points soient reliés inutilement au bord. Ensuite, on prend la distance minimale entre celle déjà calculée avec la liste et celles calculées avec les nouveaux points, et on trace un trait entre le pixel étudié et le pixel correspondant à la distance minimale.

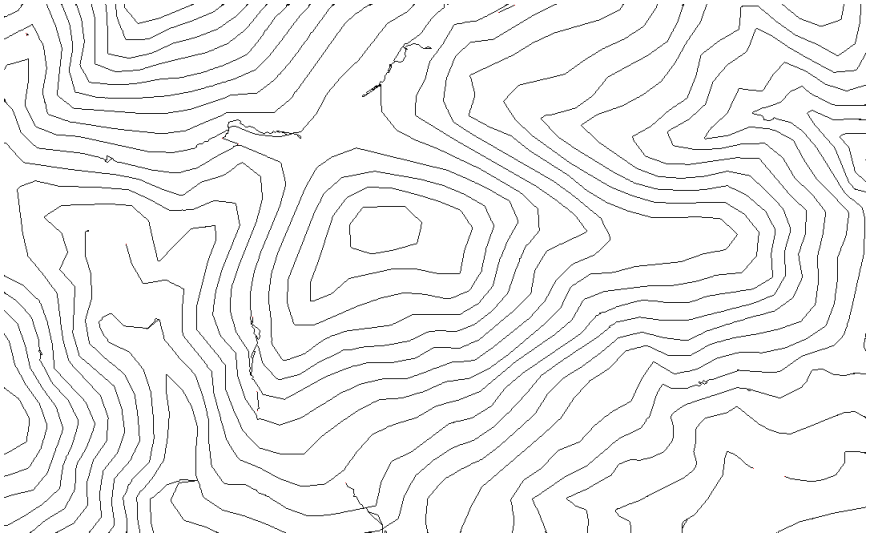
Après avoir parcouru toute la liste, si on n'a pas trouvé de point correspondant aux différents critères, on supprime le pixel. Dans certains cas, un passage de la fonction ne suffit pas, donc on la rappelle plusieurs fois, jusqu'à ce qu'il n'y ai plus de fins de lignes détectées. Cependant, on a mis un maximum de 5 fois pour éviter une récursion infinie, car 5 passages de la fonctions devrait suffire, et si ce n'est pas le cas, ça veut dire qu'il y a un problème quelque part. Pour finir, avant d'enregistrer l'image et d'envoyer les valeurs à la suite de notre programme, on refait le filtrage de l'image pour enlever des tas, potentiellement créé pas le tracé des lignes.



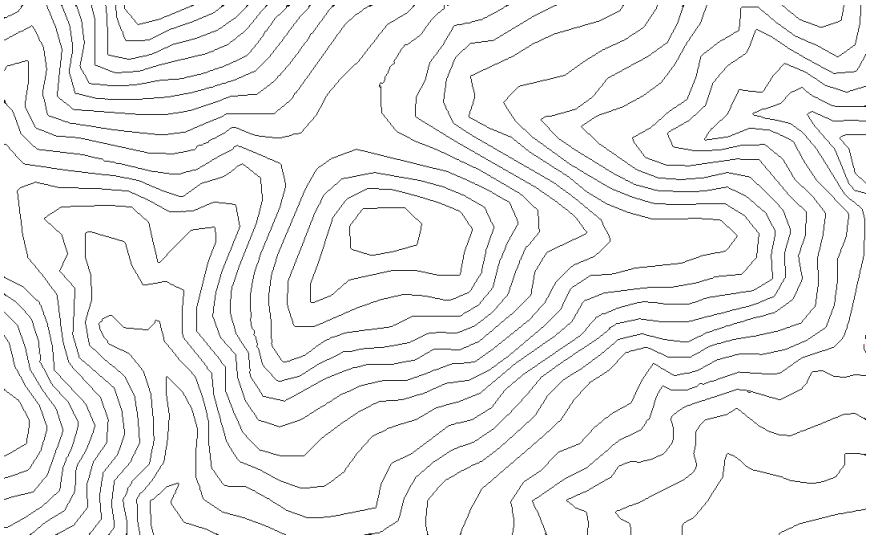
Le résultat de l'ancien algorithme de reconstruction des lignes



Le résultat du nouveau algorithme de reconstruction des lignes



Le résultat de l'ancien algorithme de reconstruction des lignes



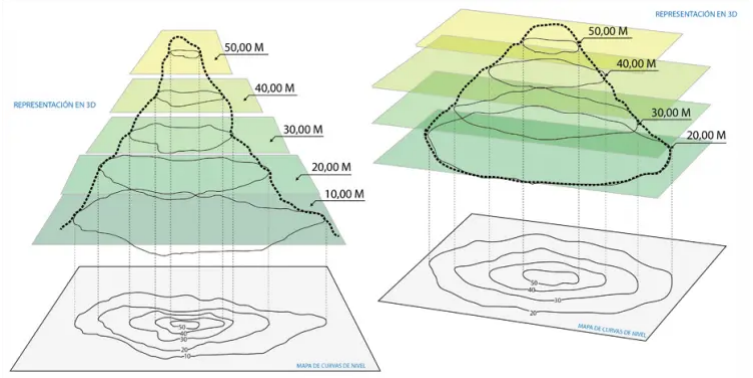
Le résultat du nouveau algorithme de reconstruction des lignes

4 Attribution du relief

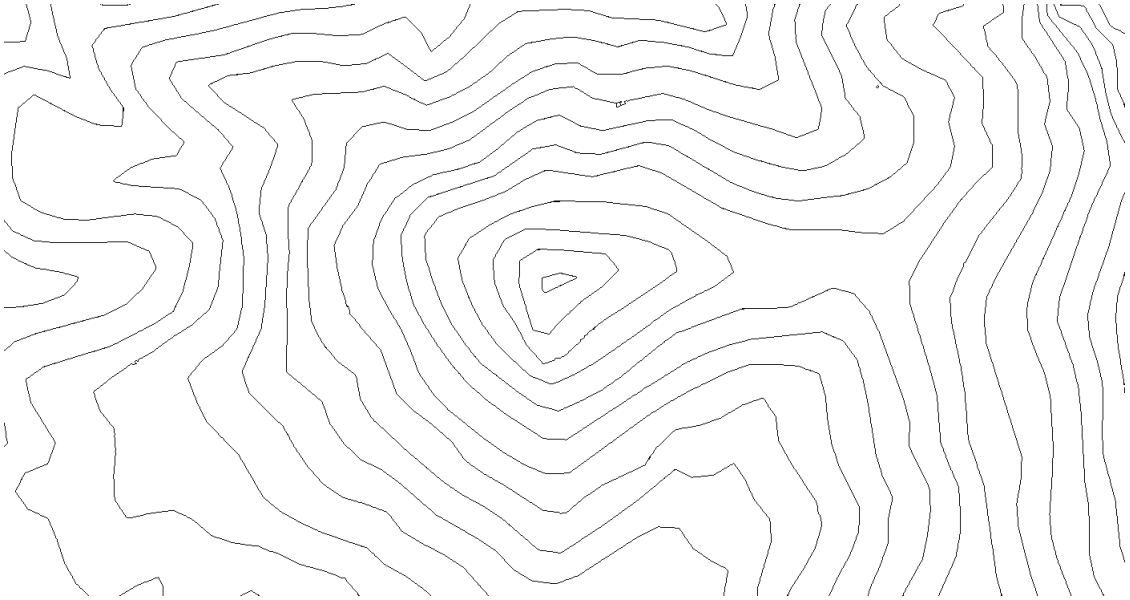
4.1 Introduction

Lors de la première soutenance, l'algorithme permettant d'attribuer de manière automatique l'altitude à partir d'une carte a été mis en place. Pour rappel, celui-ci attribue à chaque zone un label, ce qui permet de différencier les zones les unes des autres. Il récupère ensuite les zones fermées (sans bordures) qui ne contiennent aucune autre zone. A l'aide d'un parcours largeurs, l'altitude maximum est attribuée à ses zones, puis diminue au fur et à mesure du parcours des zones voisines. Le principal problème est que l'algorithme n'a aucune manière d'identifier une potentielle "remontée" du relief. C'est pourquoi lors de cette seconde soutenance, nous avons implémenté un nouvel algorithme et outil permettant d'effectuer une ré-attribution du relief de manière manuelle ou automatique.

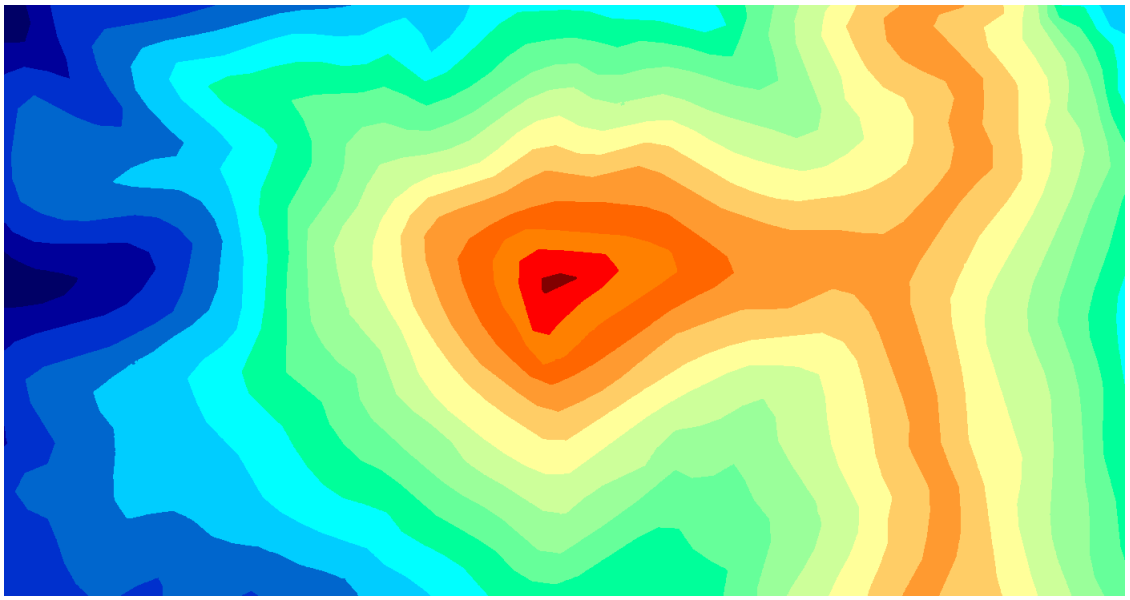
Voici pour rappel les objectifs de l'algorithme d'attribution du relief.



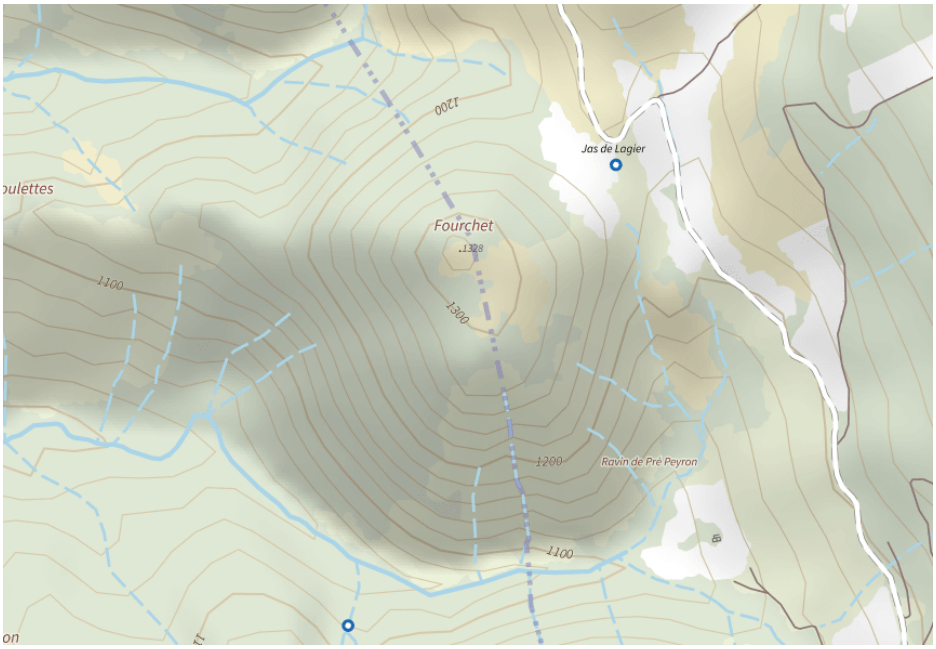
Exemple d'une modélisation 3D d'une carte topographique



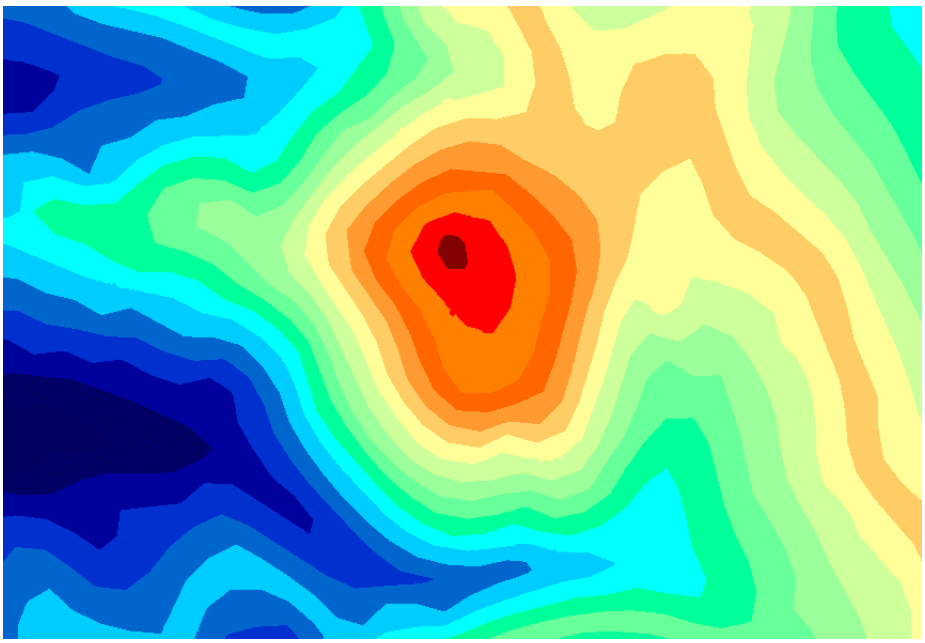
Lignes topographiques extraites et traitées



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

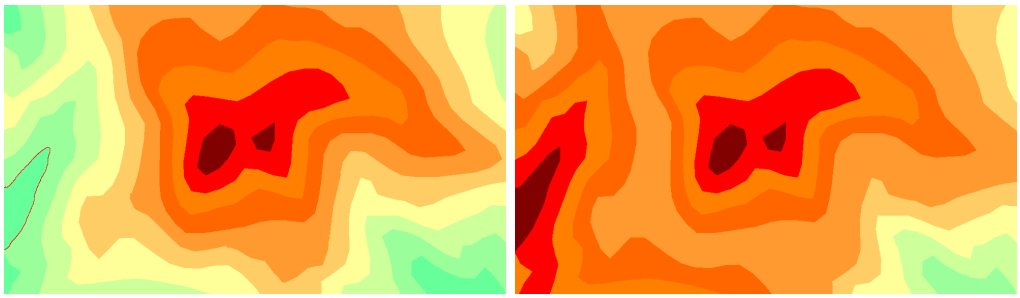
Note : plus les couleurs sont chaudes, plus l'altitude est élevée.

4.2 L'algorithme de ré-attribution du relief

4.2.1 Introduction

L'algorithme de ré-attribution du relief permet donc de modifier l'altitude d'une zone en récupérant les coordonnées x et y d'un point de celle-ci ainsi que la nouvelle altitude définie par l'utilisateur. Elle possède deux modes : un mode manuel qui modifie seulement l'altitude de la zone souhaitée et un mode automatique qui ajuste automatiquement les zones voisines de façon cohérente en plus de modifier l'altitude de la zone sélectionnée.

Voici un aperçu du rendu de cet algorithme (la zone entourée de rouges à gauche est la zone choisie par l'utilisateur et l'algorithme est en mode automatique).



Avant **Après**

Exemple : si on souhaite attribuer une altitude de 500 mètres à une zone dont l'altitude est de 1000 mètres, l'algorithme va parcourir les zones voisines et réajuster les altitudes (les diminuer dans ce cas précis).

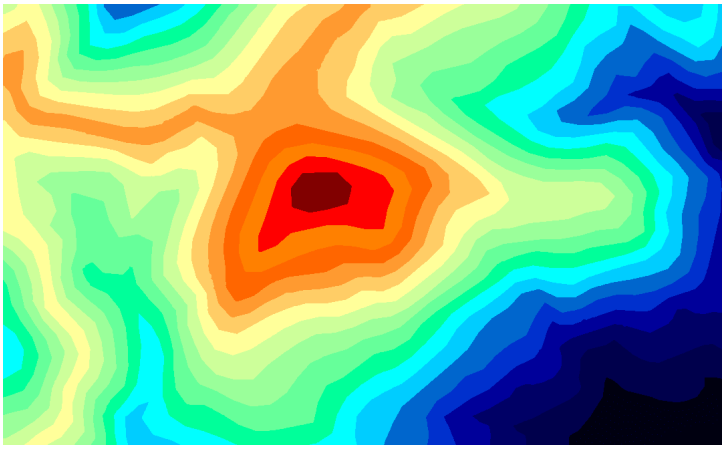
4.2.2 L'algorithme

Il faut premièrement récupérer le label de la zone grâce aux tableaux créés lors de l'algorithme d'attribution de l'altitude. Un tableau contenant les labels pour chaque pixel (x, y) nous permet de le faire facilement. On peut alors modifier le tableau d'élévation qui, pour chaque pixel (x, y), revoit son élévation. L'altitude de chaque pixel appartenant au label trouvé est donc mise à jour. Ce principe est appliqué aux deux modes de fonctionnement de l'algorithme.

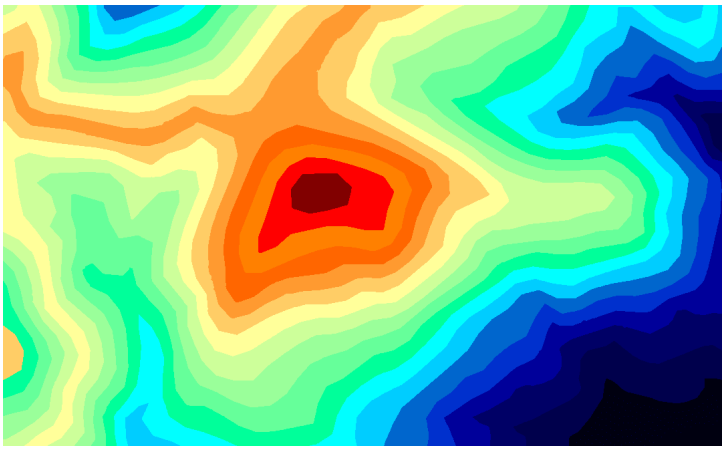
Si le mode automatique a été sélectionné, on réalise un parcours largeur à partir de la zone mise à jour. Une file est créée dans laquelle on enfile le label correspondant à la zone, puis on parcourt chacun des pixels de la zone à l'aide d'un second parcours largeur afin de récupérer tous les labels des zones voisines. En effet, dès qu'un pixel correspond à un label différent, on stocke son label dans une liste. Une fois le second parcours largeur terminé, on récupère alors les labels des zones voisines dans notre liste et on les enfile dans notre file. Le premier parcours largeur sur nos labels continus : pour chaque label dépilé, on met à jour l'altitude de celui-ci (en la diminuant ou en l'augmentant au fur et à mesure selon la situation de départ) et on effectue un parcours largeur afin de récupérer ces zones voisines. Ce processus est donc répété jusqu'à que notre file de labels soit vide ou que les altitudes des zones parcourues soient cohérentes. Un seuil établi à 200 mètres permet de définir ce seuil de cohérence. Si une zone possède une différence d'altitude supérieure à ce seuil, elle est alors traitée par l'algorithme, sinon ignorée.

Cet algorithme permet donc de réattribuer l'altitude de manière dynamique. L'utilisateur peut donc modifier les altitudes des zones de la carte topographique autant de fois qu'il le souhaite. Le tableau contenant les altitudes des lignes topographiques est alors mis a jour afin de modéliser correctement la carte résultante.

Exemple d'application de l'algorithme en mode manuel.

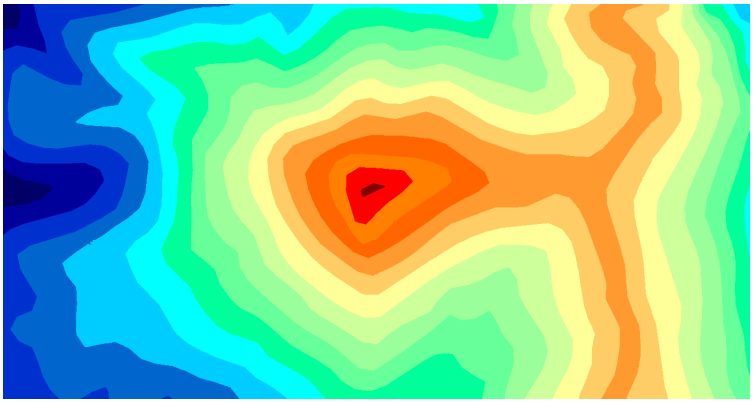


Avant (voir zone à gauche)

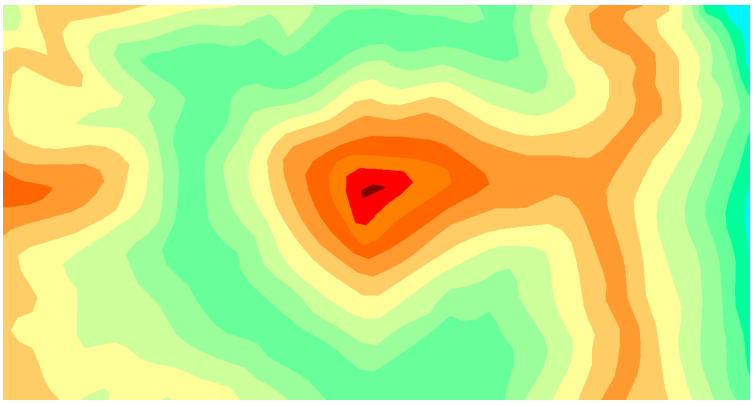


Après (voir zone à gauche)

Exemple d'application de l'algorithme en mode automatique.



Avant (voir zone à gauche)



Après (voir zone à gauche)

On peut observer une modification de l'altitude des zones voisines à la zone de départ sélectionnée par l'utilisateur. Dans ce cas précis, le problème d'une potentielle remonter du relief après la détection du point le plus haut est donc résolu. L'utilisateur peut signaler à l'application qu'une zone est à une certaine altitude et le mode automatique permet de modifier également les zones voisines afin de conserver une cohérence dans le relief.

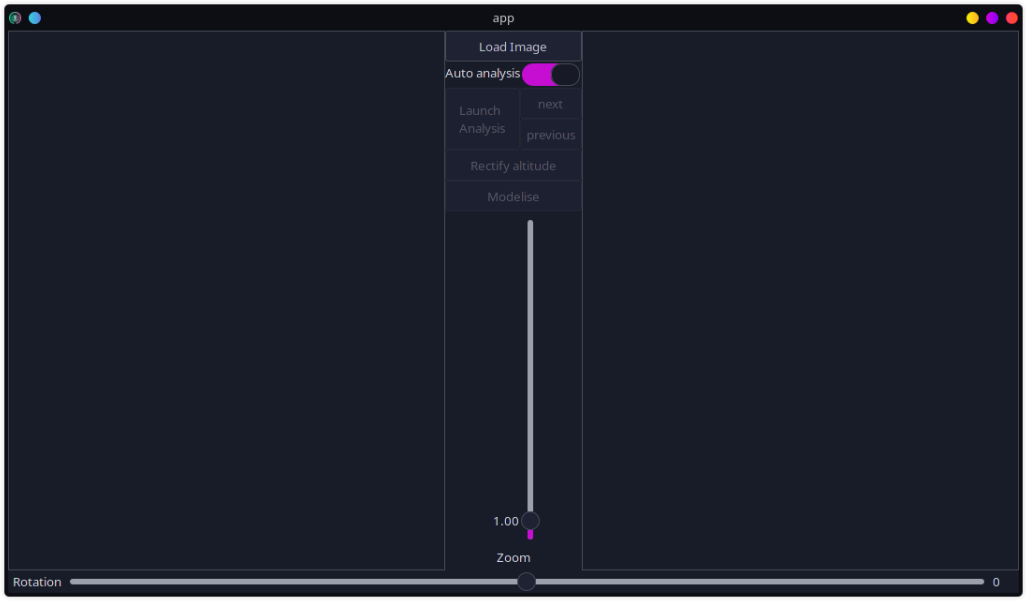
5 Application

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

Les contrôles de l'application se débloquent au fur et à mesure de l'avancement de l'exécution puisqu'il serait étrange de vouloir corriger l'altitude alors que l'analyse n'as pas encore été réalisée. La première étape consiste donc à charger une image en cliquant sur "Load Image". Cela ouvre une fenêtre secondaire d'explorateur de fichier pour choisir une image BitMap à charger. Une fois celle-ci chargée, les deux boutons suivants deviennent sensibles pour permettre le lancement de l'analyse ainsi que l'interrupteur "Auto Analysis".

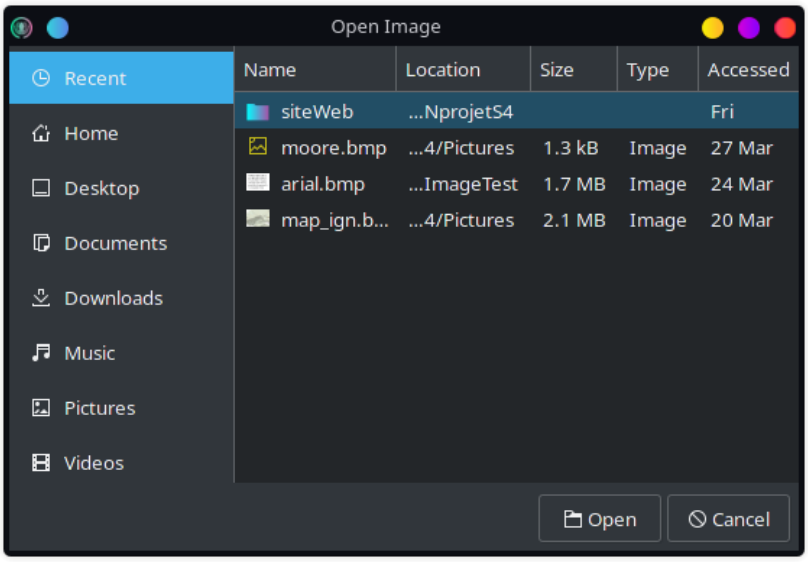
"Next" effectue une analyse en pas à pas, en affichant chaque image après le nouveau traitement dans la zone de droite, alors que "Launch Analysis" fera tout sans étape intermédiaire et affichera le résultat final. Les boutons "Next" et "Previous" permettent de se déplacer entre les différents images générées par le programme. Le changement d'état de "Auto Analysis" ouvre une fenêtre secondaire qui permet de choisir différentes couleurs utilisées pour l'analyse. L'état par défaut, comme son nom l'indique, va extraire les lignes automatiquement pour procéder à l'analyse. Une fois celle-ci effectuée, les deux derniers boutons sont débloqués:

"Rectify altitude" ouvre une autre fenêtre secondaire qui offre la possibilité de modifier l'altitude trouvée sur la carte de droite, au cas où l'analyse n'as pas respecté les altitudes écrites sur la carte.

"Modelise" affiche une deuxième fenêtre générée avec OpenGL et qui contient la modélisation de la carte. Finalement, 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°.

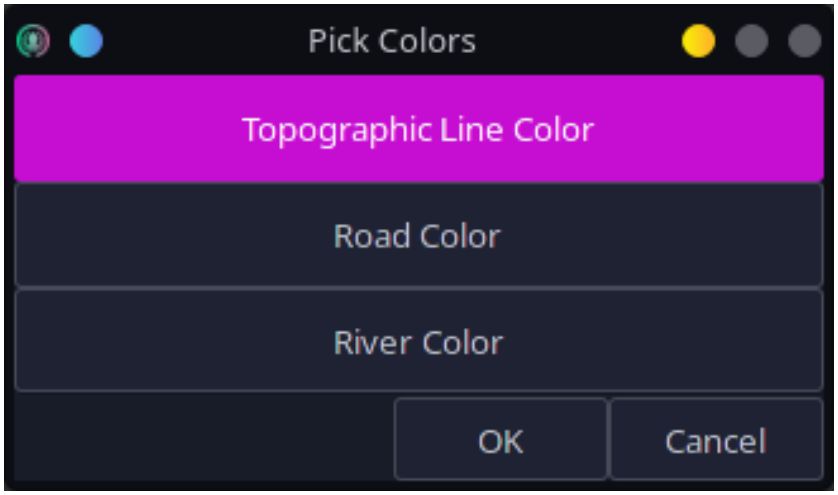
5.3 Les fenêtres secondaires

L'application propose plusieurs fenêtres secondaires pour son utilisation:



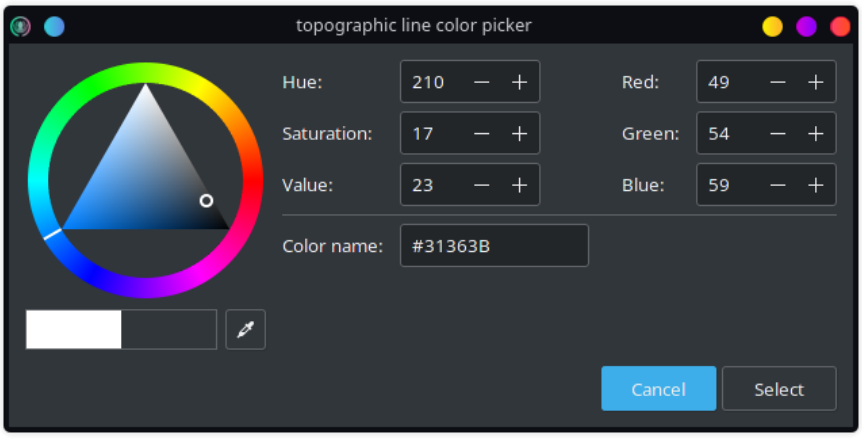
Explorateur de fichier

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.



Fenêtre de choix des couleurs

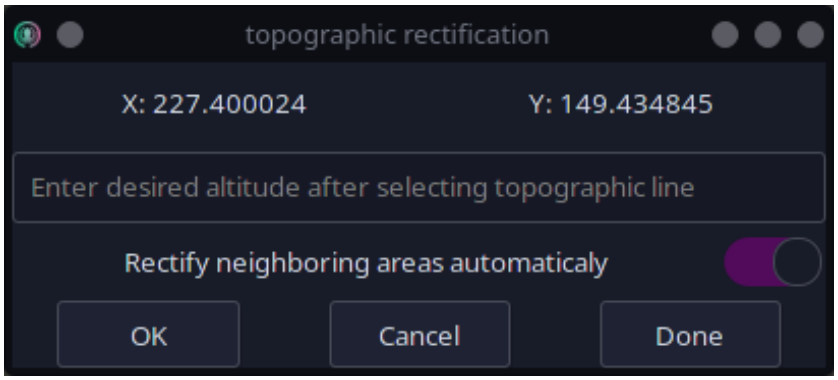
Le choix des couleurs se fait à travers cette fenêtre. Il est possible de changer les couleurs à extraire pour les lignes topographiques, les routes ainsi que les rivières. Il n'est pas possible de changer la taille de la fenêtre ni de la fermer directement car il ne s'agit que d'une petite fenêtre temporaire et la fermeture directe détruisait l'élément et rendait impossible sa réouverture. Une fois les couleurs choisies, il faut valider la sélection via le bouton "OK" et il est possible à tout moment d'annuler l'action et de refermer la fenêtre en cliquant sur "Cancel". Chacun des trois bouton de choix de couleur va ouvrir une autre fenêtre:



Choix des couleurs

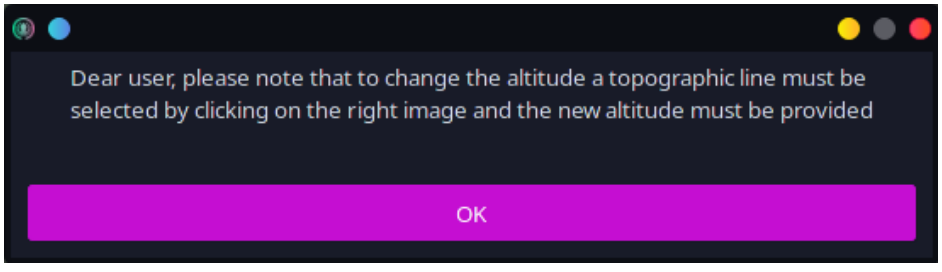
Il est ici possible d'entrer les valeurs RGB ou HSV directement dans les champs correspondant mais le plus utile est la fonction pipette sur la partie

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.



Correction des altitudes

Cette fenêtre est ouverte lors de la rectification d'altitude et va activer la sensibilité au clic de la souris sur l'image de droite. Il faut en effet choisir un point dont on veut changer l'altitude et ses coordonnées s'affichent sur la partie haute de la fenêtre. Dans la zone de texte située en dessous, il faut entrer l'altitude désirée en chiffres. La limite de caractères est de 5. Un interrupteur situé en dessous va permettre de choisir si l'altitude entrée ne va modifier que la zone cliquée ou si les zones adjacentes sont aussi recalculée pour garder une cohérence d'altitude. Finalement le bouton "OK" va mettre à jour l'altitude et la nouvelle carte s'affiche immédiatement. Le bouton "Cancel" va fermer la fenêtre sans changer d'altitude et le bouton "Done" sert à terminer les modifications et à fermer cette fenêtre. Si les condition de changement d'altitude ne sont pas satisfaites cette dernière boite de dialogue s'ouvre pour les rappeler:



Message d'information, correction d'altitude

6 Modélisation de la carte

En reprenant depuis la dernière soutenance, nous sommes toujours dans l'objectif de produire un modèle fidèle de la topographie indiqué dans sur une carte IGN. Nous avons réussi à extraire les altitudes des lignes de contour pour au final avoir une nuage de points décrivant la topographie de notre modèle.

Le but serait de tracer un polygone complexe avec ceci. Comme mentionné lors du rapport de la première soutenance, les primitives d'OpenGL nous permettrons de faire ceci à travers les triangles.

6.1 Triangulation 3D

6.1.1 Prise de décision

Initialement nous avons tenté d'écrire notre propre algorithme qui trie les points (sur la ligne de contour), de façon à ce que lorsqu'on parcours la liste, on parcourt la ligne de contour. Ceci se fait en regardant le point le plus proche, vérifiant si celui-ci est marqué sinon on l'ajoute à notre liste et on le marque. Ensuite sélectionner deux points "adjacents" et leurs points le plus proche. Et on ferait ce processus pour chacune des lignes de contours.

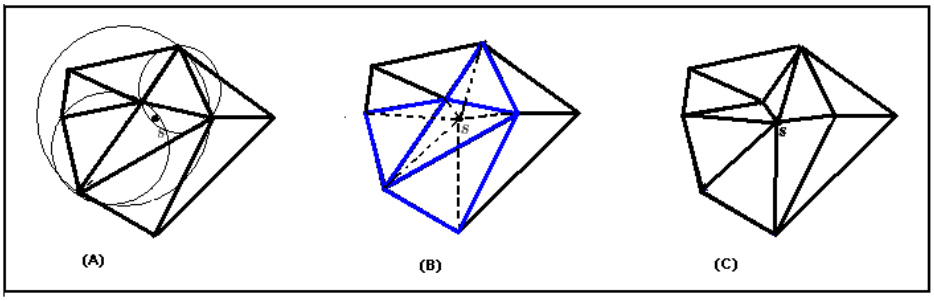
Or, cet algorithme posait des problèmes, par exemple le croisement de Triangles pouvait se produire menant à un incohérence dans l'image. Après plusieurs recherches nous avons trouvé une solution plus adaptée, la triangulation de Delaunay.

6.1.2 Triangulation de Delaunay

En mathématiques et plus particulièrement en géométrie algorithmique, la triangulation de Delaunay d'un ensemble P de points du plan est une triangulation DT(P) telle qu'aucun point de P n'est à l'intérieur du cercle circonscrit d'un des triangles de DT(P). En géométrie algorithmique, l'algorithme de Bowyer-Watson est une méthode pour calculer la triangulation de Delaunay d'un ensemble fini de points dans un espace euclidien de dimension quelconque. Un emploiera donc cet algorithme pour obtenir les triangles de Delaunay.

Pseudo-code:

```
def delaunay_bowyer_watson( points ):
    supertri = supertriangle(points) # Un triangle contenant tous les points à trianguler.
    # Qui devient le premier triangle de la triangulation.
    triangles = [ supertri ]
    fait = { supertri: False }
```



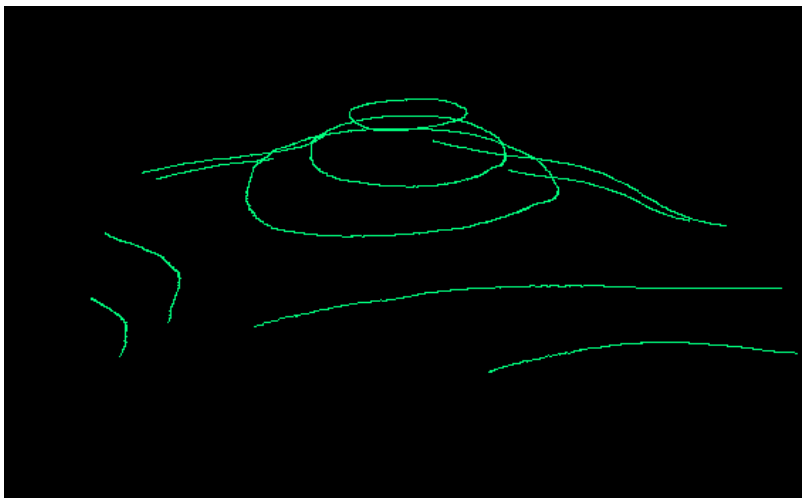

```
for sommet in points:
    # Tous les triangles dont le cercle circonscrit contient le sommet à ajouter sont identifiés,
    # l'ensemble de leurs arêtes forme un polygone englobant.

    # Démarrer avec un polygone englobant vide.
    englobant = []
    # Pour l'instant, il n'y a aucun triangle subdivisé y supprimer.
    à_supprimer = []
    for triangle in triangles:
        centre,rayon = cercle_circonscrit( triangle )
        # Si ce triangle respecte la condition de Delaunay.
        if x(centre) < x(sommet) and abs(x(sommet)-x(centre)) > radius:
            fait[triangle] = True # Le marquer comme traité.
        # Si le sommet courant est dans le cercle circonscrit du triangle courant,
        if dans_cercle( sommet, centre, rayon ):
            # ajouter les arêtes de ce triangle au polygone candidat,
            englobant += aretes( triangle )
            # préparer la suppression de ce triangle.
            à_supprimer += [ triangle ]
            fait.pop( triangle )
    # Effectue la suppression des triangles marqués précédemment.
    for triangle in à_supprimer:
         triangles.remove( triangle )
    # Créer de nouveaux triangles, en utilisant le sommet courant et le polygone englobant.
    for arête in englobant:
        point_A, point_B = arête
        # Ajoute un nouveau triangle à la triangulation.
        triangle = [ point_A, point_B, sommet ]
        triangles += [ triangle ]
        # Il faudra le considérer au prochain ajout de point.
        fait[triangle] = False
    # Supprime les triangles ayant au moins une arête en commun avec le super-triangle.
    triangulation = non_supertriangle( triangles )
return triangulation
```

En Bref, l'algorithme de DeLanay nous traduit des

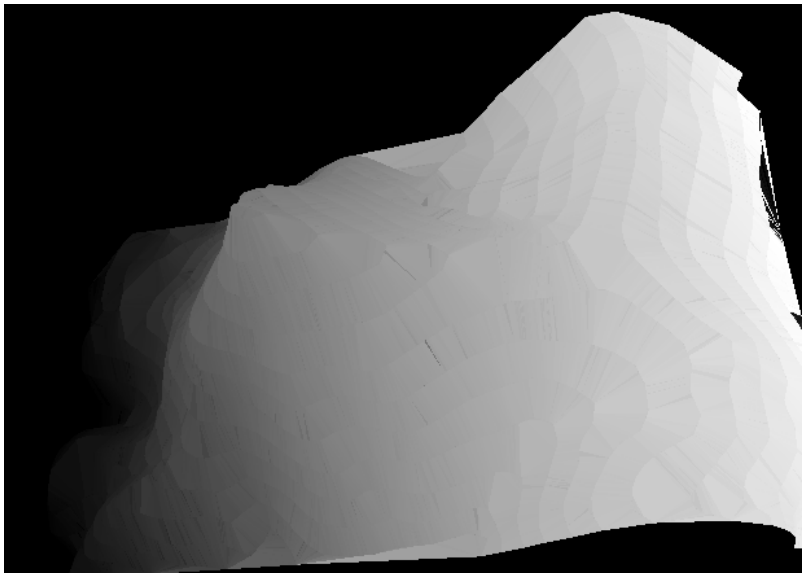
6.2 Tracer des Faces des Triangles

Premièrement, l'algorithme nécessite en entrée la liste du nuage de points (ou vertex) que nous obtenons grâce à l'extrapolation des altitudes depuis une image IGN qui est ensuite stockée dans une matrice 2D. Nous avons donc écrit une fonction permettant d'accéder à ceci. Après avoir obtenu nos Triangles de Delaunay, il nous suffit de les tracer. On parcourt de 3 à 3 la liste des triangles, les sommets sont fixés dans le sens antihoraire.

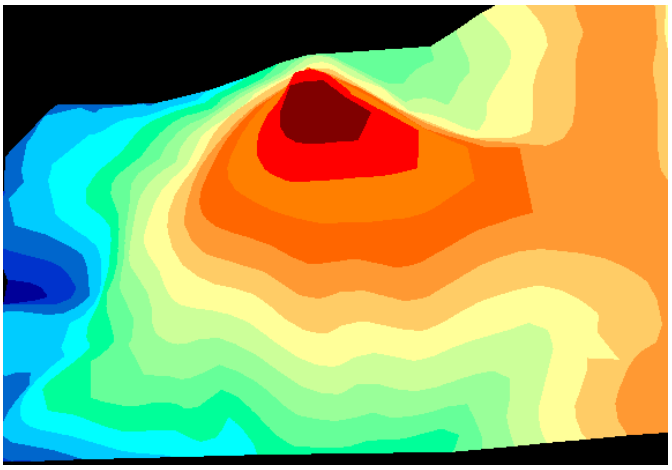


Avant : Nuage de Points

Ensuite, on a fixé le même dégradé de couleurs que pour l'attribution des couleurs sur la représentation 2D. Ce qui nous procure le résultat suivant.



Modèle initiale où chaque triangle a une teinte unique



Modèle colorié en fonction des altitudes

6.3 Les améliorations

Amélioration sur l'itération précédente du projet

- Passage de glut à freeglut(Plus moderne)
- Centrage automatique du modèle lors du lancement de la modélisation
- Optimisation sur le stockage des vertex pour la modélisation
- Proportionnalité entre x et y est maintenant correcte et automatisé.

- Triangulation 3D et coloration des triangles mentionnés ci-dessus.

6.4 Modern OpenGL

Depuis la première soutenance, une grande quantité de recherches ont été faites sur l'implémentation OpenGL Modern. Jusqu'à présent nous employons OpenGL Legacy mais ceci provoque quelques soucis sur le long terme. Notamment par rapport à l'implémentation de l'éclairage, les matériels et les textures. Il est donc prévu pour la prochaine soutenance de réadapter le code de OpenGL 2.X(Legacy) à OpenGL 3.X (Moderne).

7 Caméra libre

Lors de la première soutenance, nous avions réussi à implémenter une caméra virtuelle à l'aide de gluLookAt, et avions implémenté les translations et rotations.

Pour cette soutenance, nous avons tout d'abord ajouté l'option de faire les translations plus rapidement en apuuyant sur la touche Shift.

De plus, nous nous sommes rendus compte qu'utiliser les touches du clavier pour les rotations n'était pas forcément la chose la plus pratique. C'est pourquoi nous somme passé à l'utilisation de la souris pour contrôler l'orientation de la caméra.

En effet, au sein d'OpenGL, on trouve la fonction glutMouseFunc qui prend en paramètre une fonction qui va gérer les évènements. Par exemple, lorsqu'on presse ou on relâche une des touches de la souris.

```
void glutMouseFunc(void (*func)(int button, int state,
                                int x, int y));
```

Par la suite, nous allons nous servir de la fonction glutMotionFunc, qui prend en paramètre une fonction permettant de gérer les mouvements de la souris.

```
void glutMotionFunc(void (*func)(int x, int y));
```

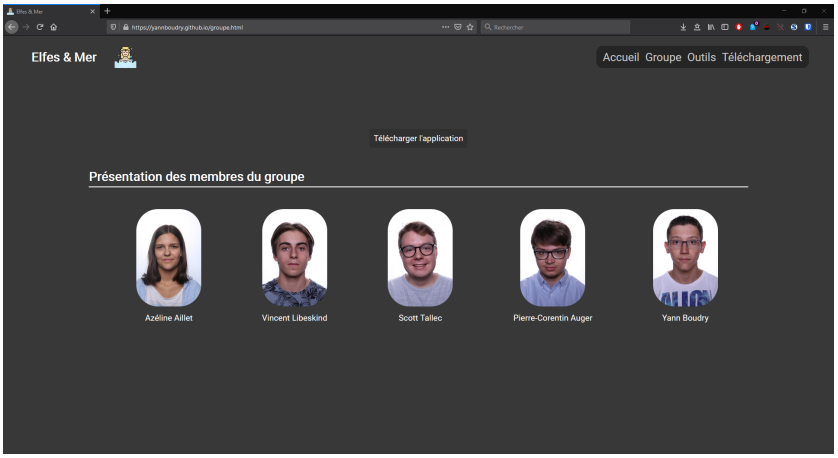
Grâce à ces fonctions, en maintenant les touches les touches de la souris, on peut modifier l'orientation de la caméra.

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 seconde soutenance. Nous avons donc bien avancé par rapport à notre programme initial. L'intégralité des parties prétraitement, attribution du relief d'une carte IGN et caméra 3D sont terminées. Le site Web, l'application graphique ainsi que la modélisation de la carte sont quant à eux également très bien avancés. Pour la dernière soutenance, nous pensons donc être dans les temps afin de finir le projet. Nous allons nous concentrer sur la partie modélisation et amélioration du modèle ainsi qu'à la recherche de nouvelle fonctionnalité et d'optimisation de notre programme.