ENSG Géomatique

ÉCOLE NATIONALE
DES SCIENCES
GÉOGRAPHIQUES

Rapport de projet géomatique

Ce projet a pour objectif de réaliser un modèle d'inondation sur une zone définie par un Modèle Numérique de Terrain.

The project's aim is to realize a flood model in a zone defined by a Digital Elevation Model

Réalisé par : Amina AIACHE,
Nisrine DRISSI
et Valentin SASYAN
Master 2 : Technologies des SI

Encadré par : M. Emmanuel BARDIERE

Table des matières

Liste des figures

Contexte du projet

Introduction générale

Notre projet consiste à réaliser un modèle d'inondation à partir d'un Modèle Numérique de Terrain (MNT). L'intérêt est de nous familiariser avec le langage de programmation C++ ainsi qu'à l'utilisation de librairies spécifiques à l'instar de QGLViewer pour l'affichage 3D.

Ce travail, dont le but est la création d'une application affichant un MNT, permet la manipulation de données brutes telles qu'un fichier contenant les coordonnées des points du MNT. Sur celles-ci, nous avons appliqué une série de calculs aboutissant à une représentation 3D d'une montée d'eau, où la vitesse et la hauteur sont définies par l'utilisateur.

Dans ce rapport nous proposons un descriptif détaillé des différentes étapes du projet, avec notamment des explications sur les choix réalisés qui permettent l'optimisation de notre programme.

Ainsi, quelles sont les contraintes liées à la modélisation d'une inondation sur un MNT ?

Dans ce rapport, nous vous présenterons dans un premier temps les besoins auxquels répond le projet, puis les données utilisées et les choix opérés.

Enfin, nous terminerons cette étude par une explication plus approfondie du code source de notre application.

Conception générale

Étude préalable

– *Existant : présentation des données*

Nous disposons d'un Modèle Numérique de Terrain, il s'agit d'un fichier de coordonnées (fichier texte avec une extension .xyz).

Afin de pouvoir localiser la zone d'étude, nous avons utilisé le SIG QGis. Nous avons ainsi importé notre MNT et nous avons ajouté un fond de carte Bing à l'aide du plugin « openLayers », ce qui nous a permis de déduire que le MNT recouvre le Sud-Est de la France, ainsi qu'une partie de l'Italie, dont une partie du Parc National du Mercantour.

Figure 1 : Localisation géographique du MNT

Nombre de points en X	1799
Nombre de points en Y	4000
Nombre total de points	7 196 000

Les coordonnées en X et en Y varient d'un pas de 25 mètres et les altitudes sont comprises entre 682,534 et 2617,4 mètres.

– *Faisabilité*

Tout au début du projet, nous avions prévu de faire notre propre algorithme permettant de simuler une inondation sur un MNT d'une manière réaliste et nous avions donc écrit une première version de l'algorithme. Cependant, après une réunion avec notre encadrant, nous avons réalisé que le temps consacré au projet ne nous permettait pas de faire marcher un tel algorithme, et nous avons donc décidé de faire un modèle moins réaliste, mais réalisable dans les temps.

La manière dont s'élève l'eau ne correspond pas à une inondation visible sur le terrain. Il s'agit d'une montée fictive, sur l'ensemble de la surface du MNT. Nous ne tenons pas compte des pentes, il n'y a donc pas de calculs concernant l'écoulement de l'eau.

Analyse

- *Besoins*

Les besoins de notre applications se résument en :

- la réalisation d'un modèle d'inondation en trois dimensions sur un MNT,
- le programme doit marcher avec n'importe quel MNT,
- à chaque instant de l'inondation, l'intersection entre le niveau d'eau et le MNT doit être calculable.

- *Contraintes*

Comme tous les projets informatiques, la première contrainte que nous avions était le temps limité, vu que nous avions 4 semaines de cours pour réaliser le projet.

La deuxième contrainte était la contrainte du matériel avec lequel nous devions travailler. En effet, les ordinateurs fournis par l'école n'étaient pas très performants et nous ne pouvions pas utiliser tous les outils que nous voulions.

- *Choix techniques*

  - Langage de programmation :

Nous avons utilisé le C++ comme langage de programmation, c'était une contrainte du projet.

- *Environnement de développement :*

Nous avons utilisé l'environnement de développement « Qt creator », qui non seulement nous a facilité le travail de développement, mais qui nous a aussi permis de créer et de gérer de manière simple des interfaces graphiques (Interfaces Homme-Machine).

  - Librairies

Pour faire de la 3D, nous avons utilisé la librairie OpenGL classique. Nous avons utilisé la librairie « libGQLViewer » de OpenGL, qui nous a facilité tout ce qui est gestion de la caméra et des Widgets.

  - Outil de versioning du code

Pour bien gérer notre travail d'équipe, nous avons choisi « GitHub » comme outil de versionnement du code.

  - Systèmes d'exploitation

Au début, nous avons travaillé sous Linux (distribution Debian installée sur nos ordinateurs), mais nous avons pensé par la suite à personnaliser notre application pour qu'elle fonctionne sur tous les systèmes, à savoir Linux, Windows et Mac OSX.

Conception détaillée

– *Diagramme des cas d'utilisation*

Il y a un seul type d'utilisateur de notre application : l'utilisateur qui souhaite effectuer des simulations d'inondations sur un MNT. Il pourra donc visualiser le MNT, simuler une inondation, calculer les isolignes d'intersection entre le plan d'eau et le MNT et exporter les données calculées pour les afficher dans un MNT.

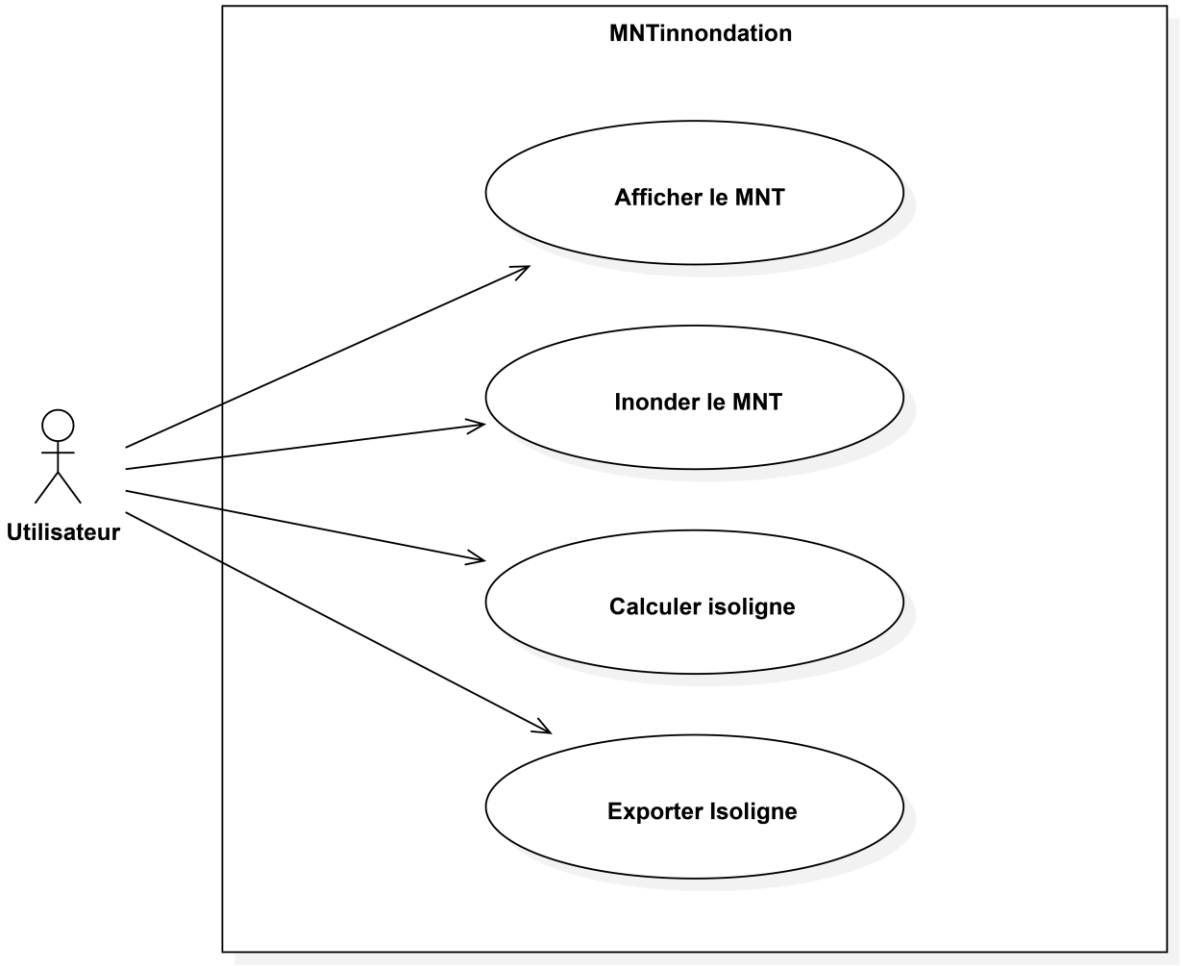
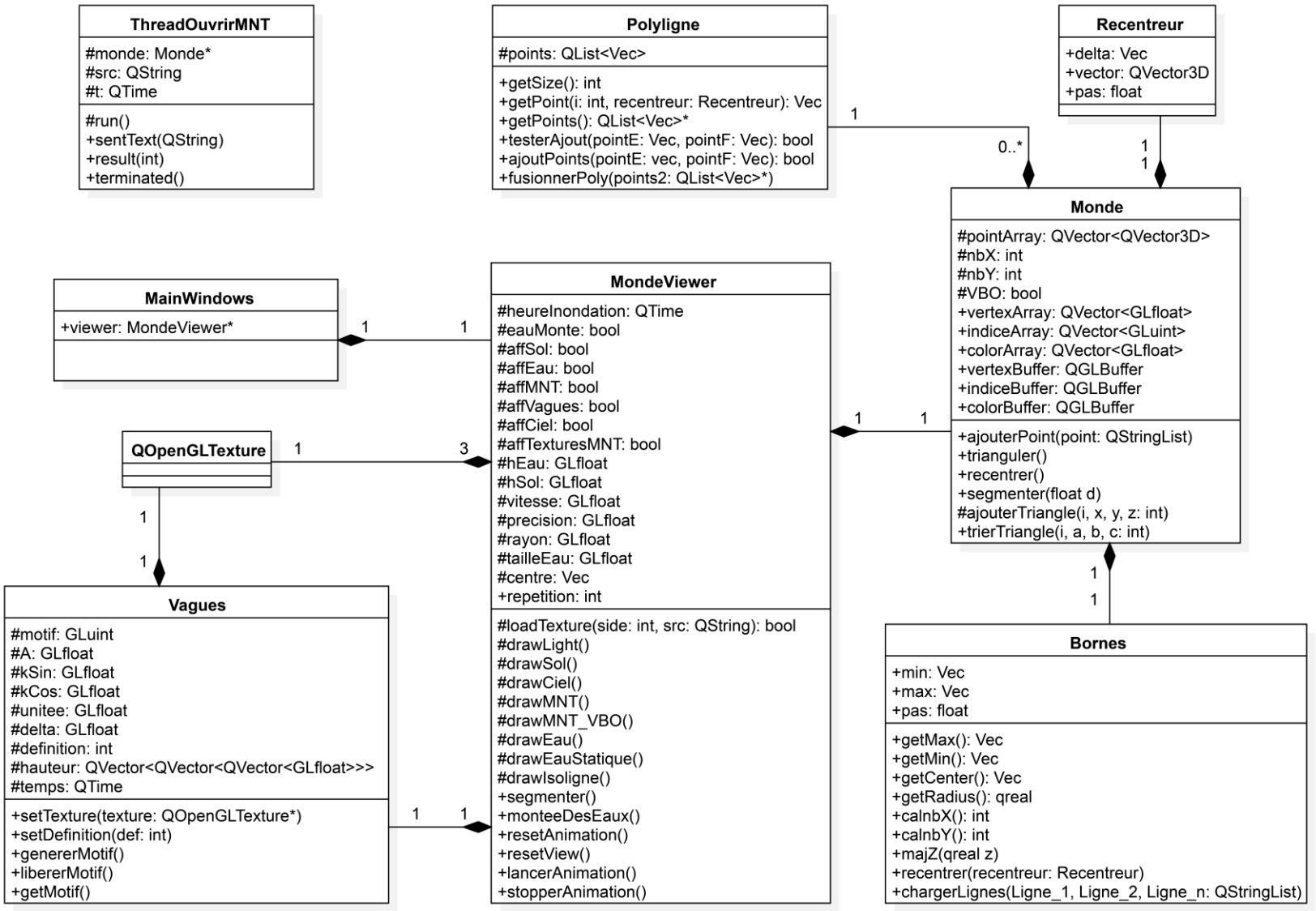


Figure 2 : Diagramme de cas d'utilisation

– *Diagramme de classes*



Les deux classes principales sont Monde et MondeViewer. Elles permettent de stocker toutes les informations relatives au MNT (Monde) et d'en gérer l'affichage (MondeViewer).

Classe Monde

Attributs :

- ses bornes via la classe Bornes ;
- les informations pour passer des coordonnées réelles aux cordonnées OpenGL recentrées via la classe Recentreur ;
- un tableau contenant les polylignes issues de l'intersection MNT/eau (polyligneArray) ;
- un tableau contenant les points du MNT (pointArray) ;
- la taille du MNT (nbX et nbY) ;
- les tableaux contenant les informations à passer à la carte graphique (les points, les indices des triangles et les couleurs des points, respectivement vertexArray, indiceArray, colorArray) ;
- un booléen pour savoir si on utilise des VBO[1] ou non ;
- l'adresse des données si on utilise des VBO (vertexBuffer, indiceBuffer, colorBuffer).

Méthodes :

- ajouterPoint : cette méthode prend en paramètre une ligne du fichier .xyz « parsée » sur le caractère blanc " " (donc un QStringList) et ajoute le point correspondant à la classe, elle met aussi à jour les Bornes du Monde en z (majZ) ;
- trianguler : cette fonction construit une liste d'entiers contenant les indices des triangles du MNT. On triangule en supposant un MNT rectangulaire régulier, on obtient 3 points par triangle, et il y a $2 \times (nbX - 1) \times (nbY - 1)$ triangles ;
- recentrer : cette fonction recentre tous les points du triangle autour de $\binom{X = 0}{Y = 0}$ en posant $Z_{min} = 0$. La fonction remplit les tableaux vertexArray et colorArray pour l'affichage ;
- segmenter : cette fonction calcule pour chaque triangle le segment d'intersection entre lui et le MNT s'il existe et l'ajoute dans le tableau polyligneArray en le regroupant par polyligne ;
- trierTriangle : cette fonction permet d'ajouter le triangle ABC créé en s'assurant qu'il soit indicé de sorte que : $H_A \leq H_B \leq H_C$. Le triangle est ensuite ajouté au tableau via la fonction ajouterTriangle.
- ajouterTriangle : ajoute le triangle.

[1] VBO : « Vertex Buffer Object », cette méthode permet de stocker les éléments à afficher directement dans la mémoire de la carte graphique à la place de les stocker dans la mémoire vive du système. Cela permet d'éviter les transferts et de gagner en performance.

Classe MondeViewer

Attributs :

- des booléens pour déterminer les éléments à afficher (affSol, affEau, affMNT, affVagues, affCiel, affTexturesMNT) ;
- des entiers pour stocker la hauteur de l'eau (hEau) et du point le plus bas (hSol) ;
- la vitesse de l'inondation (vitesse) en mètre recentré[2] par seconde, la hauteur d'eau minimale à monter pour redessiner l'eau (precision) ;
- la taille du monde à afficher (rayon), le nombre (repetition) et la taille (tailleEau) des motifs de textures à répéter pour le sol et l'eau ;
- le centre du Monde (centre) ;
- le statut de l'inondation (eauMonte),
- monde : pointeur vers l'instance de la classe Monde associée.

Méthodes :

- loadTexture : la fonction charge une texture et l'ajoute au tableau de textures ;
- draw : la fonction dessine tous les éléments. Chaque dessin est séparé par catégorie dans des sous fonctions parlantes (drawLight, drawSol, drawCiel, drawMNT, drawMNT_VBO, drawEau, drawEauStatique, drawIsoligne) ;
- segmenter : la fonction appelle la fonction segmenter sur l'instance Monde pointé par l'attribut monde ;
- monteeDesEau : la fonction gère la montée des eaux (augmenter hEau, arrêter l'augmentation si trop haut, etc) ;
- resetAnimation : remet $hEau = hSol$;
- lancerAnimation active l'augmentation régulière de $hEau$;
- stopperAnimation : met en pause l'augmentation de $hEau$ (sans remettre $hEau = hSol$).

Classe Polyligne

Attribut :

- points : contient la liste de tous les points de la polyligne dans le bon ordre.

Méthodes :

- getSize : retourne la taille (le nombre de points) de la polyligne ;
- getPoint : retourne le point demandé recentré ;
- getPoints retourne un pointeur vers la liste des points de la polyligne ;
- testerAjout : renvoie un booléen disant si les points passés en paramètre peuvent être ajoutés à la polyligne ;
- ajoutPoint : ajoute les deux points passés en paramètres à la polyligne ;
- fusionnerPoly : fusionne la polyligne passée en paramètre (il faut ensuite penser à la détruire).

[2] C'est-à-dire, en mètre divisé par le pas du MNT (voir la classe Recentreur).

Classe Recentreur

Attributs :

- delta : décalage pour recentrer le monde en Vec ;
- vector : décalage pour recentrer le monde en QVector3D ;
- pas : facteur d'échelle entre le MNT réel et les points affichés en OpenGL.

Classe Bornes

Attributs :

- min : point extérieur le plus bas dans les 3 directions ;
- max : point extérieur le plus haut dans les 3 directions ;
- pas : écart entre les points du MNT originel.

Méthodes :

- getMax : retourne max ;
- getMin : retourne min ;
- getCentre : calcule et retourne le centre du MNT ($centre = \frac{min+max}{2}$) ;
- getRadius : calcule le rayon minimal du MNT ;
- calnbX : calcule le nombre de points en X ($nbX = \frac{max.x-min.x}{pas}$) ;
- calnbY : calcule le nombre de points en X ($nbY = \frac{max.y-min.y}{pas}$) ;
- majZ : met à jour min.z et max.z en fonction du z passé en paramètre ;
- recentrer : recentre les bornes ;
- chargerLignes : définit les bornes de base d'après la première, la deuxième et la dernière ligne du fichier .xyz du MNT.

Classe ThreadOuvrirMNT

Attributs :

- monde : pointeur vers le Monde où charger le MNT ;
- src : adresse du fichier .xyz ;
- t : pour calculer le temps écoulé entre le début et la fin de l'ouverture du MNT.

Méthodes :

- run : cœur du processus, la fonction qui ouvre et charge le MNT dans la classe Monde ;
- sentText : signal d'actualisation de l'état d'ouverture du fichier ;
- result : signal de fin d'ouverture du fichier avec la durée d'exécution de la fonction ;
- terminated : signal de fin.

Classe Vagues

Attributs :

- motif : numéro de la DisplayList où est affiché le motif des vagues ;
- A : hauteur des vagues ;
- kSin : pulsation des vagues selon les sinus ;

- kCos : pulsation des vagues selon les cosinus ;
- unitee : constante de division du delta des millisecondes pour l'animation des vagues ;
- definition : nombre de points à calculer en x et en y ;
- hauteur : tableau des différences de hauteur et de la position des points ;
- temps : QTime pour mesure le temps et animer les vagues.

Méthodes :

- setTexture : définit l'adresse mémoire de la texture à utiliser ;
- setDefinition : définit la définition des vagues ;
- genererMotif : met à jour le motif des vagues ;
- libereMotif : libère la DisplayList du motif ;
- getMotif : renvoie le numéro de la DisplayList.

Réalisations

- *Ouverture du MNT*

La première tâche par laquelle nous avons commencé était de faire un algorithme qui nous permettrait d'ouvrir le fichier du MNT d'une manière optimale.

Pour connaitre les paramètres des bornes du Monde, il nous suffit de lire les deux premières lignes du fichier et la dernière :

- le pas est calculé grâce aux deux premières lignes (en X ou en Y et, comme on suppose le pas égal dans les deux dimensions, on a aussi le pas en Y ou en X),
- les bornes (X et Y minimaux et maximaux) sont calculées grâce à la dernière et la première ligne,
- la taille du MNT (le nombre de points en X et en Y) est calculée grâce à la distance entre les points, divisée par le pas.

Pour ne pas parcourir le fichier deux fois, nous stockons tous les points au fur et à mesure dès la première lecture. Ensuite, nous recentrons ces points : ils seront centrés en $\begin{pmatrix} X = 0 \\ Y = 0 \end{pmatrix}$ et l'altitude minimale est rapportée à $Z = 0$.

La mise en place pratique de tout ce qui a été décidé nous a obligés à faire quelques modifications de structures et certains choix techniques.

Isolation du processus d'ouverture

Au début, il n'était pas prévu d'isoler l'ouverture du fichier .xyz dans un processus séparé. Vu la longueur de la tâche (environ 40 secondes), nous avons décidé d'effectuer une séparation entre le processus de l'IHM et le processus d'ouverture. Cela permet de ne pas figer l'interface du programme, de tenir l'utilisateur au courant de l'avancée de l'ouverture (« Ouverture du MNY », puis « Triangulation » et « Recentrage »). De plus, cela évite à certains OS de considérer que le programme a planté.

– *Optimisation du dessin OpenGL*

La base du dessin en OpenGL est le point, on peut ensuite dessiner des objets plus complexes (triangles, carrés, etc) en envoyant des groupes de points.

– Le MNT

Nous pouvons toujours dessiner éléments par éléments en OpenGL. Par exemple, dans le cas de notre MNT, il est possible de dessiner les 14 millions de triangles un par un. Cependant cela n'est pas possible en pratique car cela n'optimise pas le transfert de mémoire entre le processeur et la carte graphique.

Il faut donc envoyer un tableau de tous les triangles. Cela permet de ne faire qu'un échange. On peut optimiser encore plus en envoyant :

- Un tableau contenant les points (donc les sommets des triangles, vertex en anglais).
- Un tableau contenant les indices des points composant chaque triangle.
- Un tableau contenant la couleur des points (optionnel).
- Un tableau contenant la texture des points (optionnel).

On envoie donc chaque point qu'une seule fois (cela prend moins de mémoire même si nous devons renvoyer plusieurs fois son indice dans le tableau car il appartient à plusieurs triangles). On peut ensuite ajouter des tableaux de couleur voire de texture, de la même manière.

– Les autres éléments

En plus du MNT, nous devons dessiner un « sol » sur toute la zone visible à $Z = 0$, une « skybox » pour imiter le ciel et de l'eau pour simuler la montée des eaux.

Les éléments du sol et du ciel sont dessinés de manière classique : pas besoin de tableau pour optimiser les transferts à la carte graphique car il n'y a que deux éléments à dessiner.

Pour l'eau, elle est normalement dessinée de la même manière, sauf quand l'utilisateur décide de l'animer.

Dans le cas de l'affichage d'eau animée, on utilise la classe Vagues. Elle permet de créer un petit carré d'eau animée (motif) qui sera ensuite répété sur tout le terrain. Cela permet de gagner en performances puisqu'on ne calcule le motif que sur une petite surface.

Le motif est stocké en mémoire graphique grâce au système des DisplayList, qui permet de sauvegarder un motif pour le redessiner ensuite.

Le motif est calculé via des sinus et des cosinus : on prend 101 points dans les deux dimensions, et on calcule, à chaque point, la différence de hauteur due à la vague selon la formule suivante :

$$h_{i,j} = A \times [sin(pos + j \times kSin) + cos(pos\ + j \times kCos)]$$

pos est un paramètre dépendant du temps et $kSin$ et $kCos$ sont les paramètres de fréquence de vagues dans les deux dimensions.

Cela permet de donner simplement un effet assez réaliste de mer, sans utilisation avancée (shader par exemple), et donc sans effet de lumière qui rendrait l'eau plus réaliste.

– *Génération des isolignes*

La fonction segmenter nous permet d'obtenir le segment de chaque triangle coupé par la surface d'eau.

Le paramètre d'entrée de la fonction segmenter est la hauteur d. Cette variable permet de tester si d est supérieur, inférieur ou égal à la hauteur des points du triangle. Les points des triangles sont classés par hauteur croissante (on a pour tout triangle ABC $A_z \leq B_z \leq C_z$), ce qui nous permet de tester facilement si le triangle est intersecté ou non par l'eau.

Par exemple, pour un triangle ABC, si la valeur de la hauteur d'eau d est supérieure à A mais inférieure à B, on effectuera des calculs plus précis pour récupérer les équations de droite de AB et de AC.

Pour cela, on calcule tout d'abord les vecteurs AB et AC :

$$AB_x = B_x - A_x \qquad AC_x = C_x - A_x$$

$$AB_y = B_y - A_y \qquad AC_y = C_y - A_y$$

$$AB_z = B_z - A_z \qquad AC_z = C_z - A_z$$

Puis on calcule les équations paramétriques d'AB et BC avec t qui est une constante pour obtenir les coordonnées x, y et z des points E et F :

$$E_t = \frac{-A_z + \mathrm{d}}{Z_{ab}} \quad E_x = A_x + E_t * X_{ab} \quad E_y = A_y + E_t * Y_{ab}$$

$$F_t = \frac{-A_z + \mathrm{d}}{Z_{ac}} \quad F_x = A_x + F_t * X_{ac} \quad F_y = A_y + F_t * X_{ay}$$

On estime que Ex et Fx sont égaux à la valeur de la hauteur d. Ainsi, nous obtenons les coordonnées x, y et z des points recherchés.

Ces deux points sont ensuite testés afin de savoir s'ils peuvent être ajoutés à une polyligne existante ou s'il faut créer une nouvelle polyligne dans notre tableau « polyligneArray ».

C'est grâce à ce tableau que l'utilisateur aura la possibilité d'exporter la courbe de niveau issue de l'intersection entre l'eau et le MNT.

Cette étape permet également d'améliorer les performances, puisque l'on cherche à mettre les points les uns à la suite des autres afin de les dessiner plus facilement sur OpenGL.

Pour savoir si les points E et F peuvent être ajoutés à une polyligne existante, on teste les extrémités des polylignes existantes avec ces deux points, ce qui nous donne quatre possibilités :

- Point E = début de la polyligne ;
- Point E = fin de la polyligne ;
- Point F = début de la polyligne ;
- Point F = fin de la polyligne ;

Si aucune de ses possibilités n'est vraie, il y a création d'une nouvelle polyligne.

Dans le cas où l'un des points peut être ajouté à 2 polylignes différentes, nous effectuons la fusion de ces 2 polylignes.

Ainsi, le QVector « polyligneArray » constitue un ensemble d'isolignes.

Recette de l'application

- *Résultats*

Étant donné que le vrai MNT sur lequel nous avons travaillé est très lourd, nous avons créé un petit MNT pour tester nos fonctionnalités rapidement avant de passer au plus grand.

Nous avons donc pu réaliser notre application, qui permet tout d'abord à l'utilisateur de parcourir son disque pour choisir le MNT sur lequel il veut simuler l'inondation.

Dès que l'utilisateur choisit le MNT, l'application affiche ce dernier, tout en laissant à l'utilisateur le choix de :

- Utiliser ou pas les VBO : si l'option VBO est cochée, le MNT sera affiché avec des couleurs qui n'ont aucun sens, car nous n'avons pas réussi à mettre des couleurs dans ce cas. Par contre, si cette option n'est pas cochée, le MNT sera affiché en dégradé de couleurs : noir pour les altitudes les plus basses, puis vert, puis cyan, et finalement le blanc pour les altitudes les plus hautes.

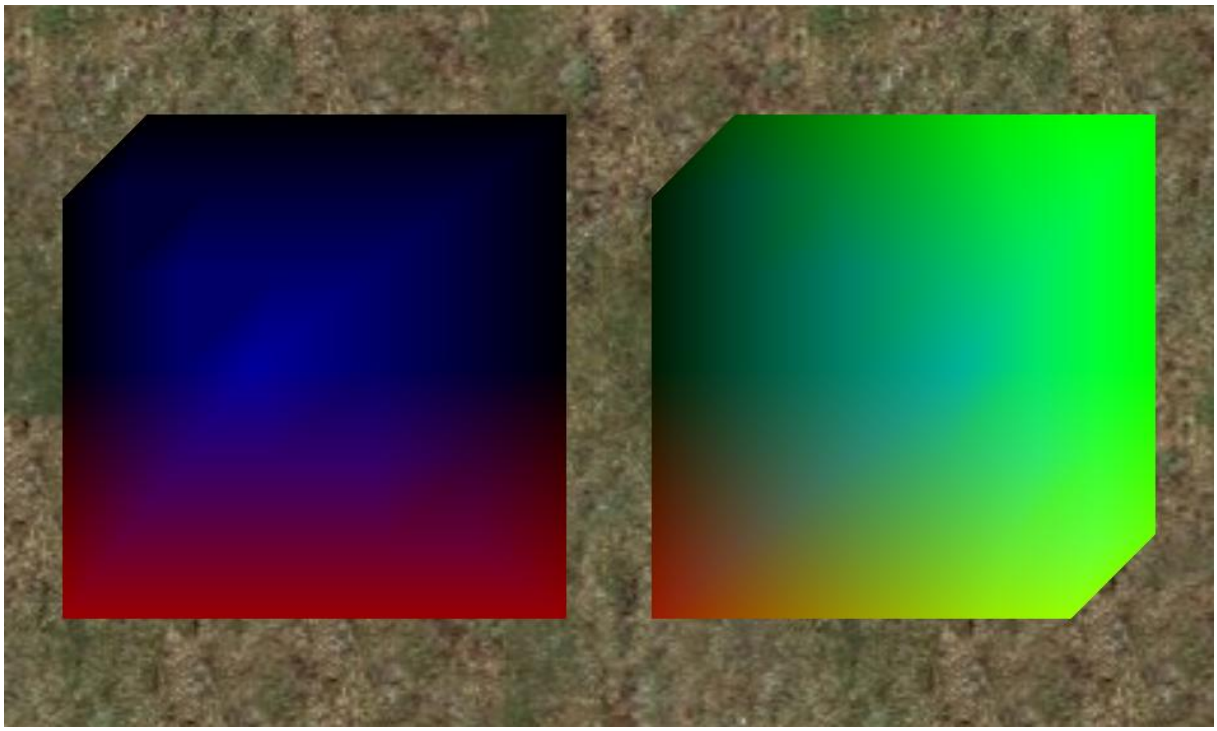


Figure 3 : Affichage du petit MNT en utilisant les VBO

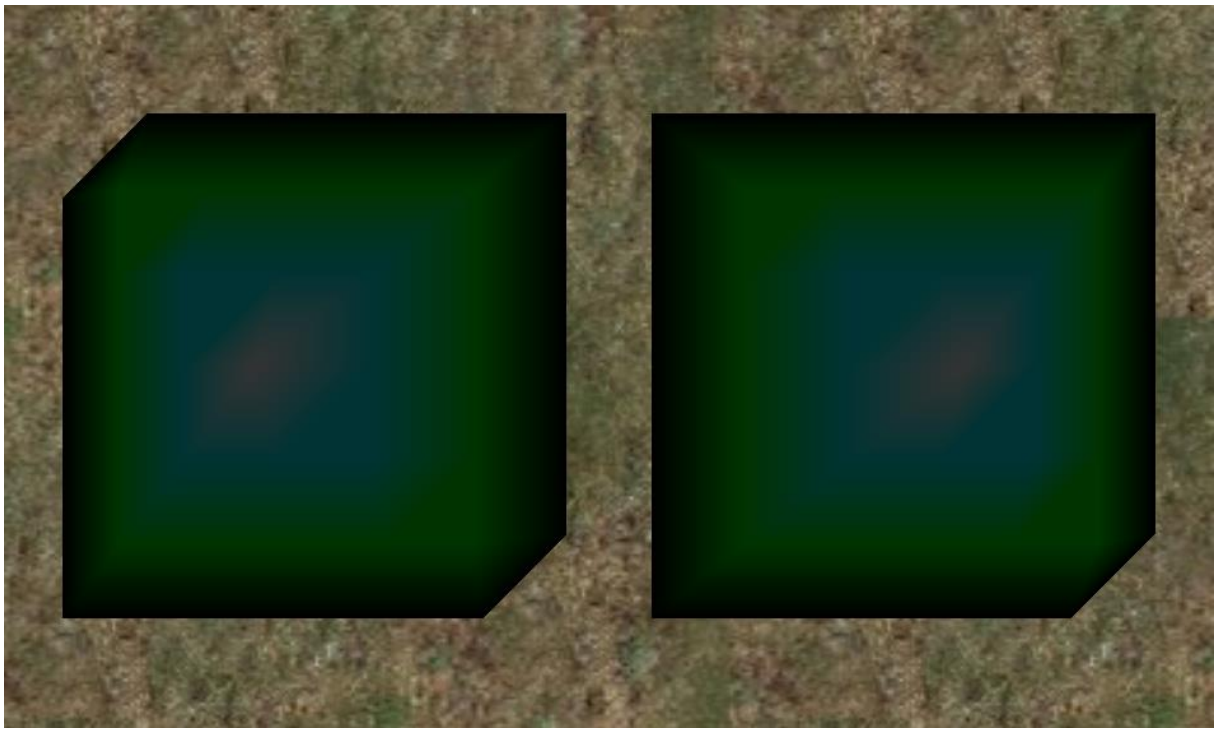


Figure 4 : Affichage du petit MNT en couleurs sans utilisation des VBO

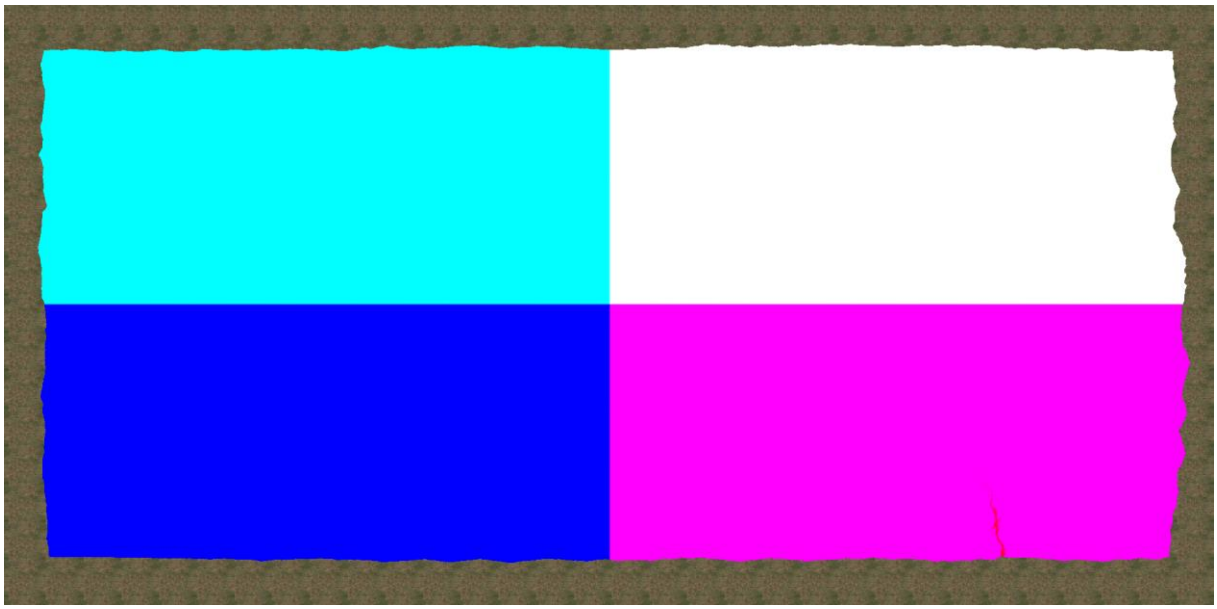


Figure 5 : Affichage du grand MNT en utilisant des VBO

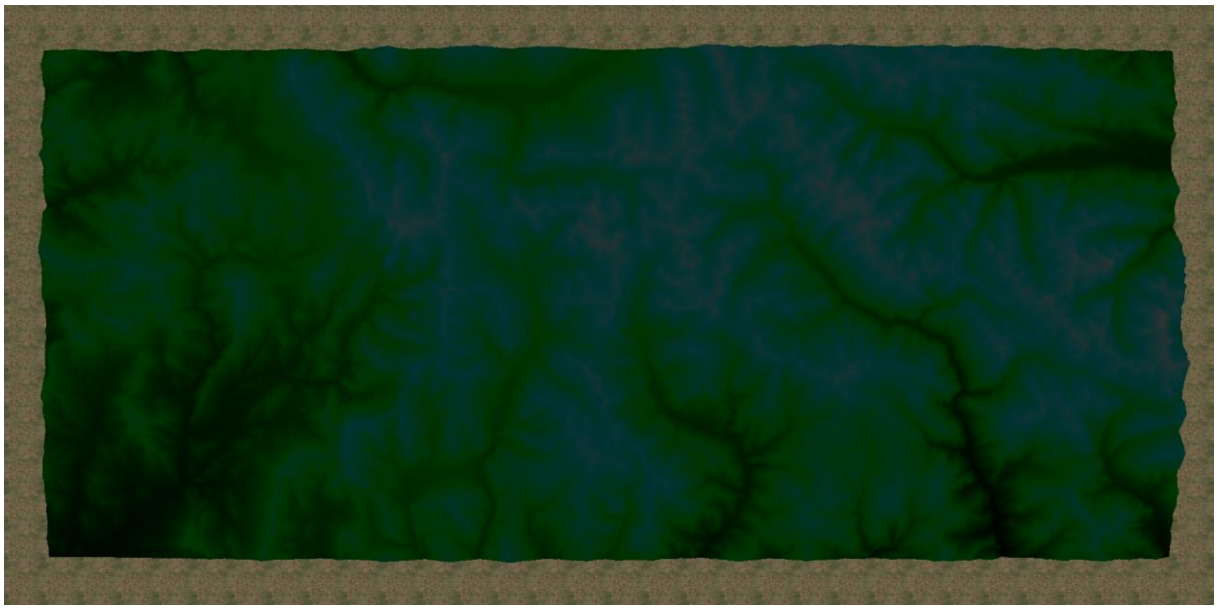


Figure 6 : Affichage du grand MNT en couleurs sans utilisation des VBO

- Mettre ou enlever le ciel et/ou le sol.
- Mettre ou enlever la texture : nous n'avons pas réussi à texturer notre MNT, car nous n'avons pas réussi à passer le tableau des coordonnées des points de prélèvement des textures (glTexCoord). Par contre nous avons bien réussi à texturer le sol, le ciel et l'eau.

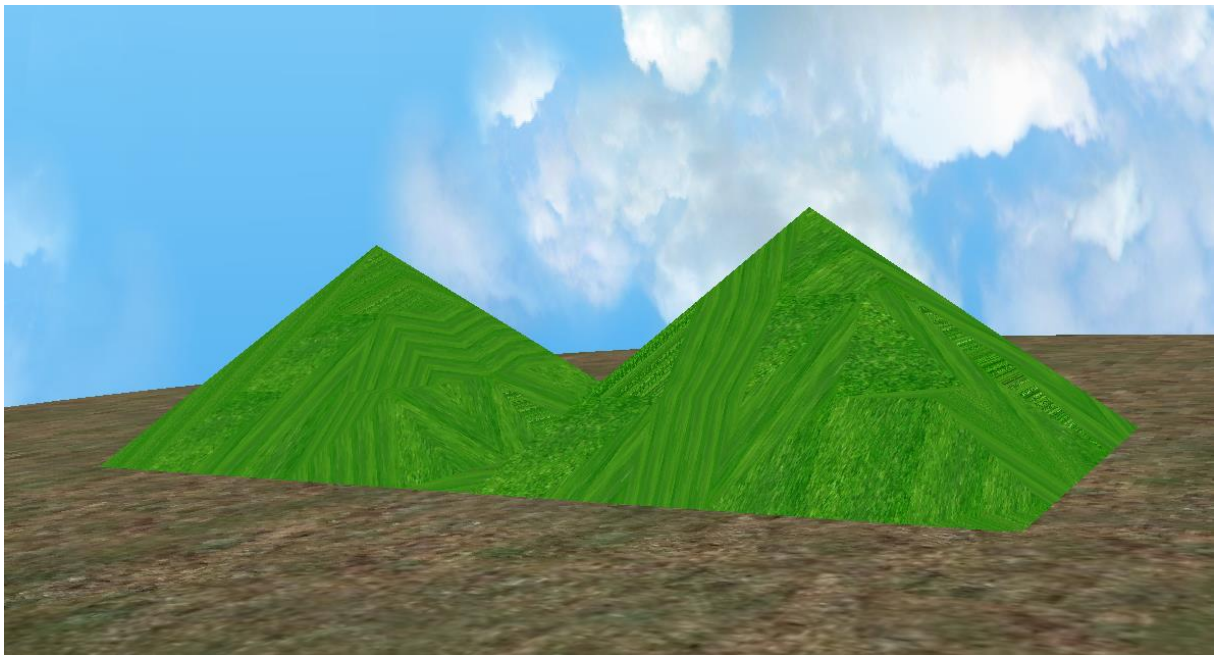


Figure 7 : Affichage du MNT avec la texture

– Effectuer l'inondation : l'utilisateur peut lancer l'inondation avec la vitesse et la précision qu'il souhaite.

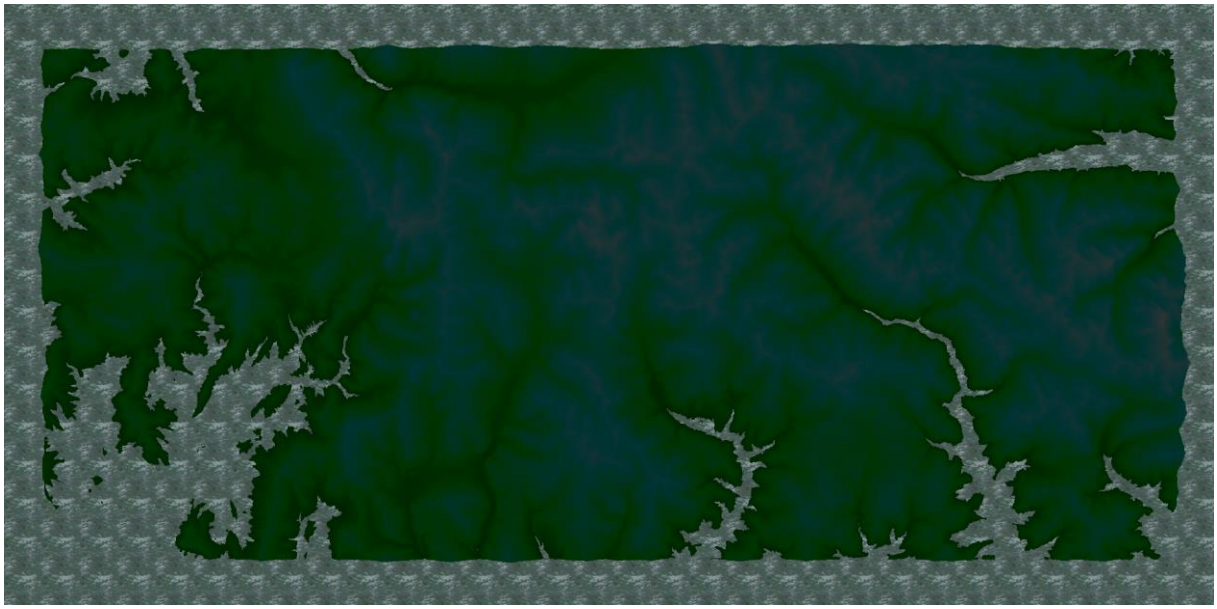


Figure 8 : Inondation sur le grand MNT sans vagues

– Mettre ou enlever les vagues :

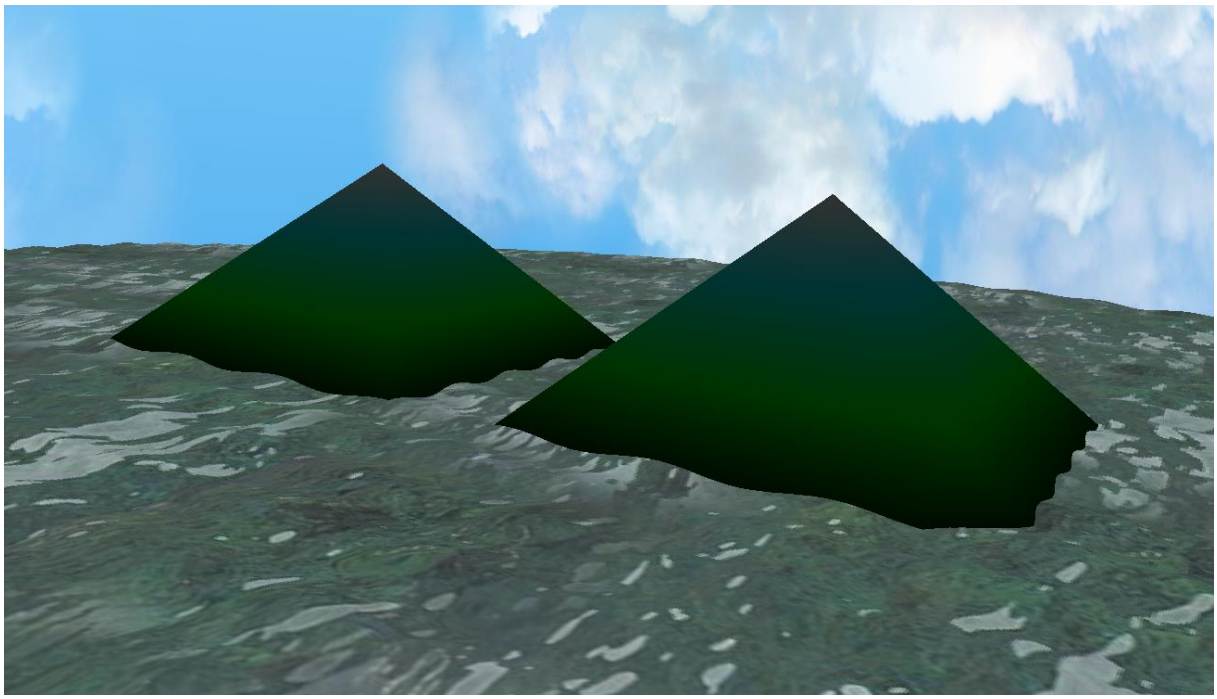


Figure 9 : Inondation sur le petit MNT avec les vagues

– L'utilisateur peut calculer les isolignes et remettre à zéro le niveau de l'eau pour qu'il puisse les voir (en rouge sur la figue ci-dessous) :

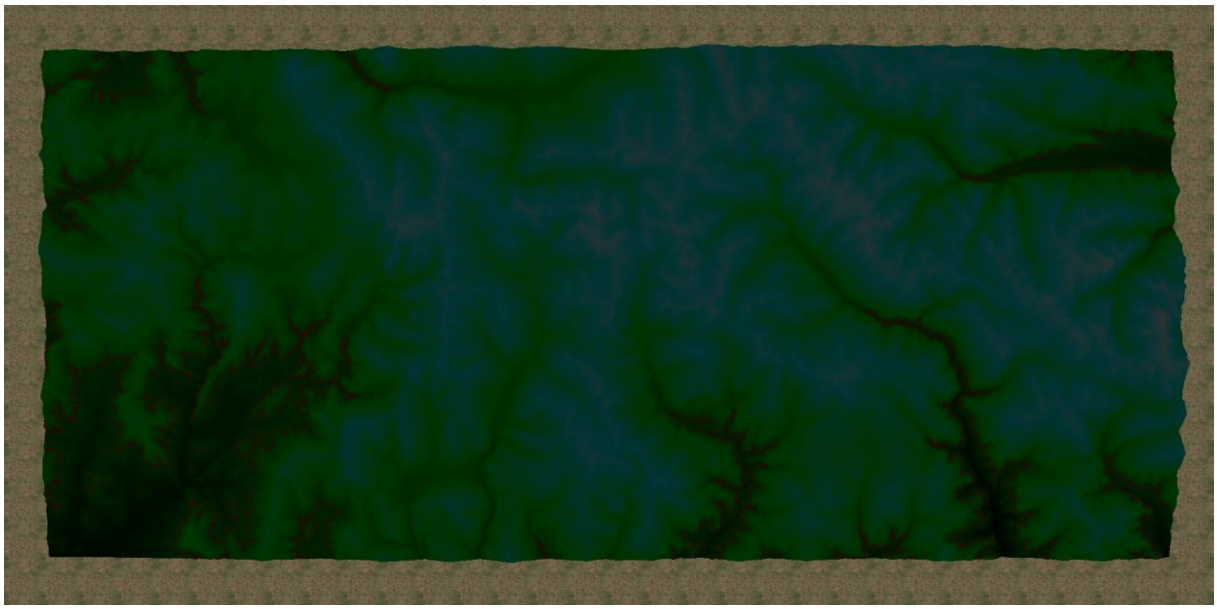


Figure 10 : Affichage des isolignes calculées à un niveau d'eau donné

– Finalement, l'utilisateur peut exporter les isolignes en fichier .csv où les polylignes seront écrites en format wkt (well-known text), et peut alors l'ouvrir sous n'importe quel SIG. Par exemple, nous l'avons fait sous QGis :

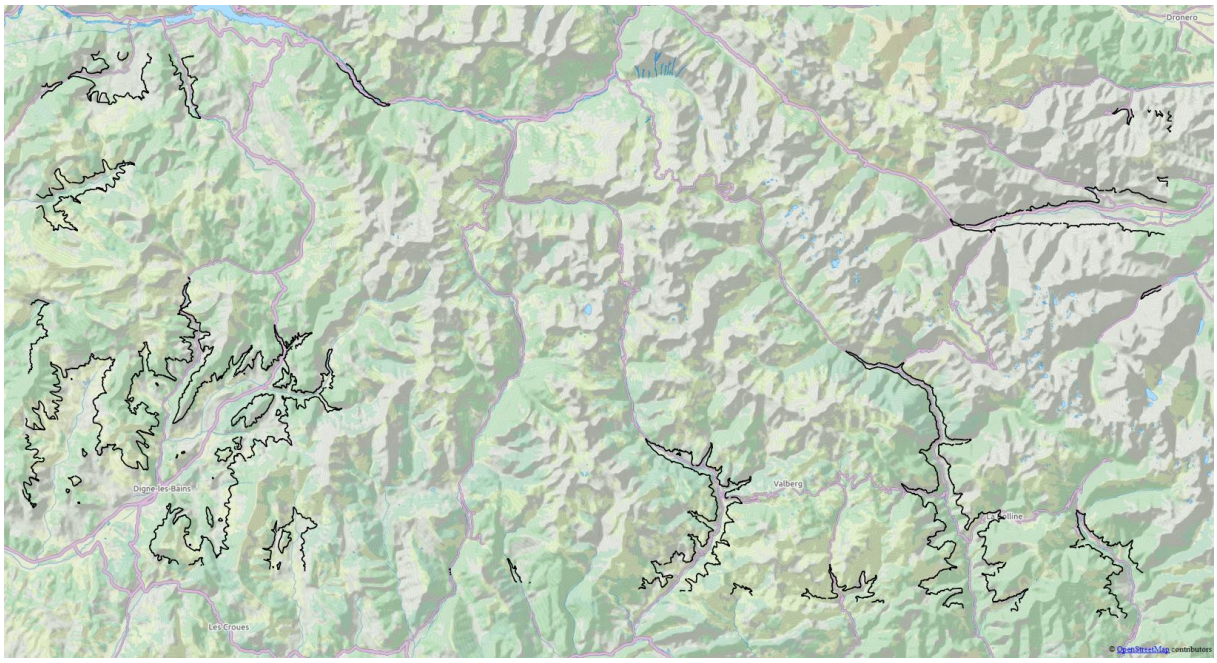
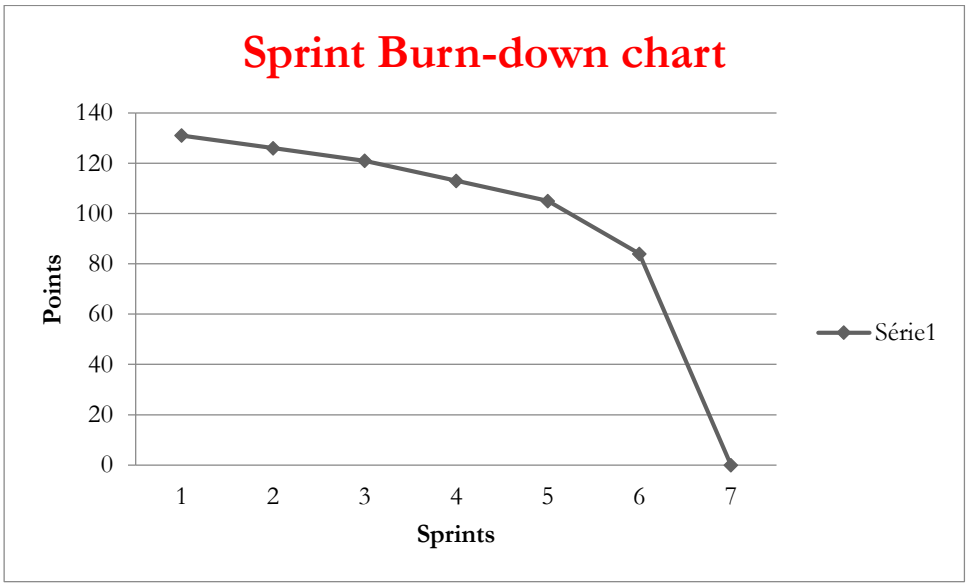


Figure 11 : Isolignes vues dans QGis avec un fond de carte OpenStreetMap

Gestion du projet

Pour bien gérer notre projet, nous avons utilisé la méthode Agile, en essayant d'appliquer ce que nous avons appris pendant le cours « Techniques de développement et méthode Agile » que nous avons eu juste avant de commencer le projet.

Nous avons donc défini toutes nos user stories, en les décomposant chacune en tâches et en leur attribuant une note. Tout cela nous a permis de faire notre Sprint burn-down chart que montre la figure suivante :



1	Installation de l'environnement de développement
2	Recherche bibliographique (algorithmes d'inondation)
3	Conception : diagramme des cas d'utilisation et diagramme de classes
4	Lecture du MNT
5	Intégration du module de lecture dans le modèle
6	Affichage du MNT
7	Réalisation du modèle d'inondation

Ainsi, toutes les user-stories ont été décomposées en tâches, de sorte que chacun des membres de l'équipe puisse travailler indépendamment.

Nous avons aussi, tout au long du projet, effectué des stand-up meetings pour suivre le travail réalisé la veille et celui à faire le jour même.

Il nous paraissait aussi indispensable de définir le « done » afin que nous puissions savoir à quel état d'avancement nous étions à chaque début de sprint, et savoir quand considérer que la fonctionnalité en question est terminée et que nous n'avons plus besoin de revenir pour la vérifier.

Conclusion et perspectives

Ce projet a été une véritable occasion d'apprentissage pour nous. En effet, en termes de compétences techniques, nous avons appris comment optimiser les transferts entre la mémoire vive et la carte graphique pour améliorer les performances. Aussi, nous avons pu découvrir l'utilisation des processus.

Nous avons également pu mettre en pratique la méthode Agile vue en cours.

Comme améliorations futures de notre projet, nous pensons qu'il faudrait mettre en place une gestion correcte des couleurs et des textures pour régler les problèmes d'affichage. Aussi, il serait bien de mettre le calcul des isolignes dans un processus séparé pour éviter de figer l'interface pendant le calcul.