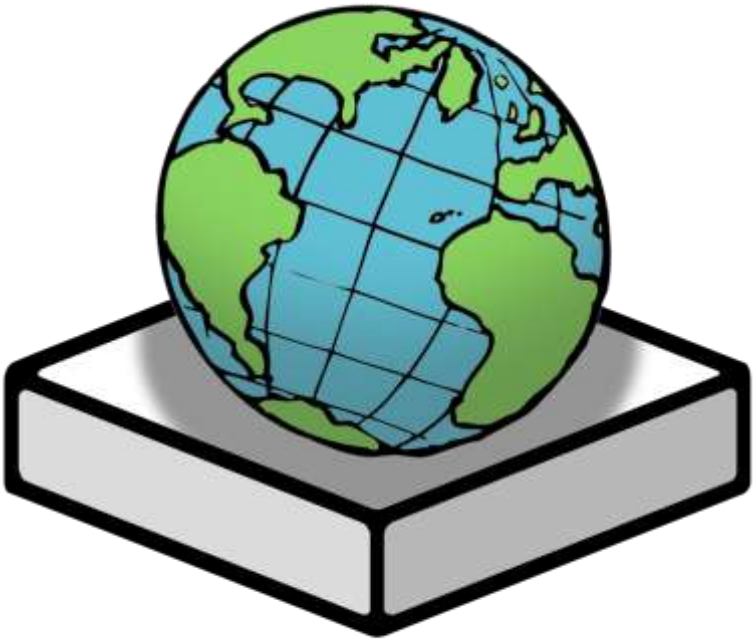
IUT DE CLERMONT-FD
Aurillac - Le Puy-en-Velay - Clermont-Ferrand
UNIVERSITÉ
Clermont Auvergne

3DMAPGEN



Année 2016-2017

Rapport de projet tuteuré

MEMBRES : DARDINIER Alexis
KLEIN Thomas
PETIT Pierre
ROUZE Timothé
VINCENT Mathieu

TUTEUR : FAVREAU Jean-Marie

3DMapGen

RAPPORT DE PROJET TUTEURE

AUTORISATION DE DIFFUSION

Nous autorisons la diffusion de notre rapport sur l'intranet de l'IUT.

REMERCIEMENTS

Nous tenons tout particulièrement à remercier notre tuteur Jean-Marie Favreau qui nous a guidé et orienté tout au long de notre projet, ainsi que les étudiants de l'ISILab qui nous ont permis d'imprimer nos pièces en 3D grâce à leurs imprimantes.

SOMMAIRE

INTRODUCTION

Notre projet consiste à générer des fichiers 3D à partir d'une image en niveaux de gris. Nous avons la charge de créer une application permettant de transformer une image classique en un maillage 3D imprimable. Les images à transformer seront plus spécifiquement des plans de quartier ou de ville en niveaux de gris. Ces images vont être conçues par l'Institut Géographique National, elles comportent des informations sur le lieu en aplats de noir. Après le traitement les niveaux de gris sur l'image auront une hauteur permettant, une fois imprimée, aux déficients visuel d'avoir un plan tactile avec des informations en braille. Le résultat final de ce projet est de permettre un accès à des cartes tactiles à des particuliers souhaitant imprimer une carte d'un lieu qu'ils veulent découvrir.

Dans la première partie de ce rapport, nous allons aborder le projet en général, son contexte et les technologies. Dans un deuxième temps nous allons étudier la structure de donnée que nous avons mise en place et le traitement de l'image. Enfin nous verrons comment nous avons implémenté les différentes fonctionnalités de l'application.

L'IDEE DU PROJET, DEFINITION DES TERMES ET DES TECHNOLOGIES

Contexte et méthodes de travail

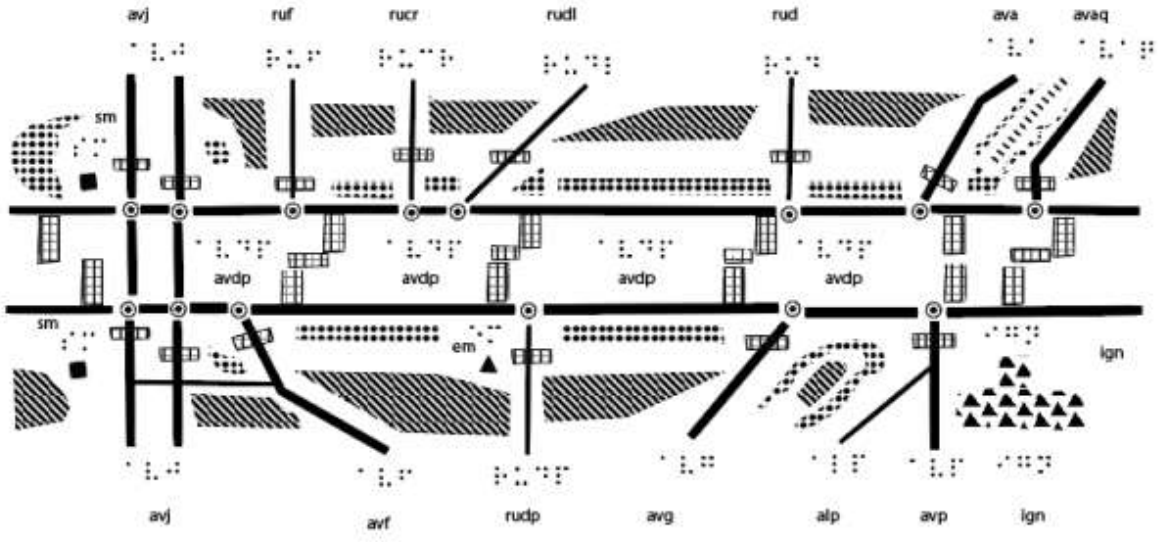
En partenariat avec un laboratoire de recherche de l'IGN, un travail de génération automatique de cartes tactiles pour les non-voyants et les déficients visuels est en cours de conception. Il s'agit de permettre à une personne non voyante d'imprimer en 3D une carte tactile afin de prendre connaissance de la configuration d'un quartier (carrefours, passages piétons, feux, obstacles sur le trottoir ou encore arrêts de bus ou de tramway).

Pour ce projet, nous nous plaçons à la fin de la chaîne de traitement quand le plan a été adapté, et qu'il est disponible sous forme d'une image en noir et blanc (voire en niveaux de gris).
L'objectif du présent projet tuteuré consiste à développer un logiciel qui va générer le ou les maillages prévus pour l'impression 3D volumique de la carte, à la manière de ce que propose le projet HaptoRender d'OpenStreetMap.

Nous avons été convié à différents événement en parallèle à notre projet. Le 8 décembre nous avons assisté à des conférences au zénith d'Auvergne sur la fabrication additive. Nous avons pu découvrir les différents type d'impression 3D, les matières utilisées (plastique, bois, poudre d'aluminium et même chocolat), les types d'imprimantes et les recherches en cours pour l'amélioration des matériaux.

Le 16 mars nous avons assisté à un séminaire du LIMOS où M. Guillaume Touya de l'IGN a expliqué les différentes méthodes de cartographie. Nous avons appris que pour les cartes numériques le traitement informatique peut être très complexe pour permettre une bonne lisibilité à l'utilisateur. En effet plusieurs traitements sont réalisés dans le but de mettre en valeur certains détails des informations du plan aux différentes échelles.

De plus nous avons discuté avec M. Touya qui nous a expliqué ce que l'étudiant en master avait réalisé pour fournir des cartes en noir et blanc pour que nous les utilisions dans notre application.



Carte en noir et blanc et braille réalisée par un étudiant en master.

Pour la réalisation du projet nous avons utilisé une méthode agile, qui nous a permis de d'appréhender cette méthode au sein d'un projet d'une plus grande importance.
Nous avons utilisé la méthode SCRUM. En effet cette méthode est souvent utilisée en entreprise, il nous a semblé intéressant de l'appréhender pour ce projet qui pouvait se découper en plusieurs objectifs. Cette méthode consiste à se fixer des objectifs à courts terme sous forme de sprint. À la fin de chaque sprint le résultat montré doit être fonctionnel pour être montré au maître d'œuvre et au maître d'ouvrage. Les objectifs pour le sprint suivant sont redéfinis à la fin de chaque réunion de fin de sprint. Les tâches sont attribuées sous forme de points selon la difficulté de celle-ci et le temps que nous souhaitons y consacrer. Chaque membre de l'équipe peut prendre le nombre de point qu'il souhaite pour le sprint. Ainsi, de sprint en sprint nous avons pu ajuster le nombre de tâche à prendre ainsi que la quantité de travail. Cela nous a poussé à travailler pour de nombreuses échéances (toutes les 2 semaines) plutôt qu'une seule à la fin du projet (Gantt réel en annexe).
Enfin, nous avons utilisé le logiciel gratuit Meshlab pour visualiser nos maillages 3D ainsi que pour vérifier s'il y a des doublons de points ou si le maillage est hermétique.

Structure en puzzle et prise en compte du braille

Plusieurs aspects ont été considérés. D'une part, la génération de l'élévation de terrain en fonction du niveau de gris, et la configuration du maillage correspondant devra être bien ajustée. En effet il existe des normes concernant la hauteur et les dimensions de cellule de braille.
Nous avons donc contacté l'association Braille & Culture qui nous a renseigné sur ces normes.
La hauteur était un paramètre que nous étions capables de gérer et manipuler, cependant, les dimensions de chaque cellule étaient celles sur les cartes tchèques utilisées pour nos tests de rendu. Un agrandissement ou une réduction de ces dimensions est possible lorsqu'on choisit nos paramètres, la longueur et la largeur finale de la carte vont impacter la taille des cellules.
Mais lorsqu'on choisit une taille "raisonnable", la déformation n'est que moindre et il est quand même possible de lire les inscriptions sur les cartes.

D'autre part, les imprimantes 3D personnelles ne permettent pas l'impression de grandes surfaces. Nous avons donc proposé une génération de maillage sous forme de pièces de puzzle assemblables permettant de produire des cartes de surfaces plus importantes à celles imprimables sans trop de complexité pour l'utilisateur.

Description d'un maillage 3D et des formats imprimables

Un maillage est un ensemble de formes géométriques disposées de manière à modéliser des objets. Les maillages sont particulièrement utilisés en infographie pour représenter des surfaces ou en modélisation.

Un maillage est constitué de sommets, connectés les uns aux autres par des faces. Lorsque toutes les faces sont des triangles, on parle de maillage triangulaire (trimesh), ou de triangulation selon les domaines. Les maillages par quadrilatères sont aussi très courants. En 3 dimensions, il est aussi possible

d'utiliser des maillages volumiques, qui relient les sommets par des tétraèdres, des hexaèdres (cuboïdes) et des prismes.

Afin de stocker ces maillages, il faut les écrire dans des fichiers 3D qui respectent certaines normes, les formats de fichiers 3D se comptent par dizaines mais pour les besoins de notre projet, nous avons choisi et utilisé le format OBJ qui utilise un maillage triangulaire (toutes les faces sont des triangles) dont la syntaxe est la suivante :

- Un sommet est défini de la manière suivante :

  v X Y Z // Pour un point de coordonnées (X;Y;Z)

- Une coordonnée de texture est définie de la manière suivante :

  vt X Y // Pour un vecteur de coordonnées (X Y)

- Un vecteur normal est défini de la manière suivante :

  vn X Y Z // Pour un vecteur normal de coordonnées (X Y Z)

- Puis, nous pouvons écrire nos faces, qui sont composées de 3 (ou plus) sommets puis, optionnellement, les vecteurs de textures et les vecteurs normaux.

  f V1/VT1/VN1 V2/VT2/VN2 V3/VT3/VN3

Étant donné que nous ne mettons pas de textures, ni de vecteurs normaux, l'écriture de nos faces se résument aux indices des 3 faces, comme ci-dessous :

f V1 V2 V3

Les indices que l'on insère dans les faces sont les numéros des lignes sur laquelle est écrite le sommet. Autrement dit, si on a 3 sommets comme ceci :

v 1.0 0.0 0.0 // Ligne 1

v 0.0 1.0 0.0 // Ligne 2

v 0.0 0.0 1.0 // Ligne 3

La face qui sera composée de ces trois points sera écrite comme ceci :

f 1 2 3

Donc, pour écrire nos fichiers en OBJ, on "instancie" tous les sommets avec v suivi des coordonnées puis les faces avec f suivie du numéro des lignes des points souhaités.

STRUCTURE DE DONNEE MISE EN PLACE ET ALGORITHME DE TRAITEMENT

Structure de données et implémentation

Après avoir étudié la structure théorique d'un maillage, nous avons pu procéder à son implémentation, celle-ci s'est faite en plusieurs étapes.

Dans un premier temps, nous avons modélisé les sommets :

```
1.  public class Sommet {
2.      private static int cpt = 1;
3.      private double x;
4.      private double y;
5.      private double z;
6.      private int id;
7.      public Sommet(double ligne, double hauteur, double colonne) {
8.          x = colonne;
9.          y = hauteur;
10.         z = ligne;
11.         id = cpt;
12.         cpt++;
13.     }...
14. }
```

Puis ensuite les faces qui sont composées de ces sommets :

```
1.  public class Face {
2.      public static int cpt = 1;
3.      private int id = 0;
4.      private int s1 = 0;
5.      private int s2 = 0;
6.      private int s3 = 0;
7.      public Face(int idSommet1, int idSommet2, int idSommet3) {
8.          this.id = cpt;
9.          cpt++;
10.         s1 = idSommet1;
11.         s2 = idSommet2;
12.         s3 = idSommet3;
13.     }...
14. }
```

En réalité, les faces ne sont pas réellement composées de sommets, vu que les faces n'ont pas besoin de connaître les coordonnées des sommets qui les délimitent mais on simplement besoin de l'identifiant qui fait référence à ces sommets.

Pour le maillage, les choses se complexifient, il faut en effet stocker l'ensembles des sommets ainsi que l'ensemble des faces qui se comptent en dizaines de milliers.

Étant donné que l'ordre dans lequel sont écrit les sommets dans le fichier est très important, le conteneur utilisé doit trier les sommets pour que les faces soient correctement agencées. Notre choix

d'implémentation s'est donc porté vers une *TreeMap* avec comme clé l'id du sommet et comme valeur l'objet l'instance du sommet correspondant : *TreeMap (<int id>,<Sommet s>)*

La classe *Maillage* contient également une *ArrayList* de sommet ou étaient présents une copie des points du socle, ce second conteneur était nécessaire car le socle nécessitait des traitements pour pouvoir le modeler et ces traitements se faisaient après que la pièce soit formée.

Les faces quant à elles n'ont pas besoin d'être triées quand elles sont écrites dans le fichier, c'est pourquoi elles sont contenues dans une simple *LinkedList*.

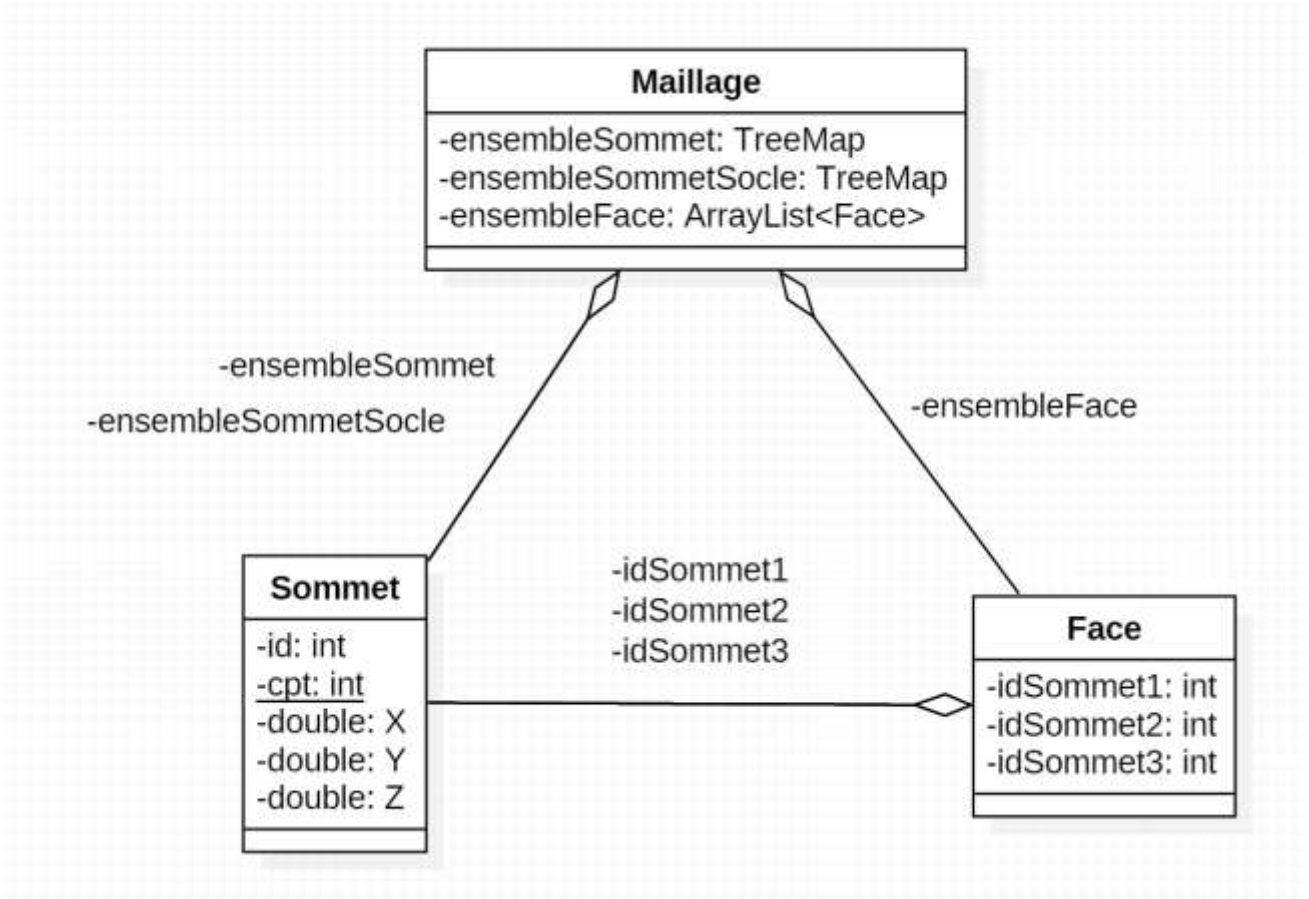


Diagramme UML de notre structure de données mise en place

Cependant cette structure, qui est assez minimaliste nous a posé par la suite un certain nombre de problèmes, en effet cette structure générait des points en plusieurs exemplaires et de fastidieux tests de créations n'étaient pas suffisant, de plus le traitement était relativement long (de l'ordre de 2 à 4 minutes).

Nous avons donc revu la structure de la classe *Maillage*. Les classes *Sommet* et *Faces* restent quant à elles inchangées.

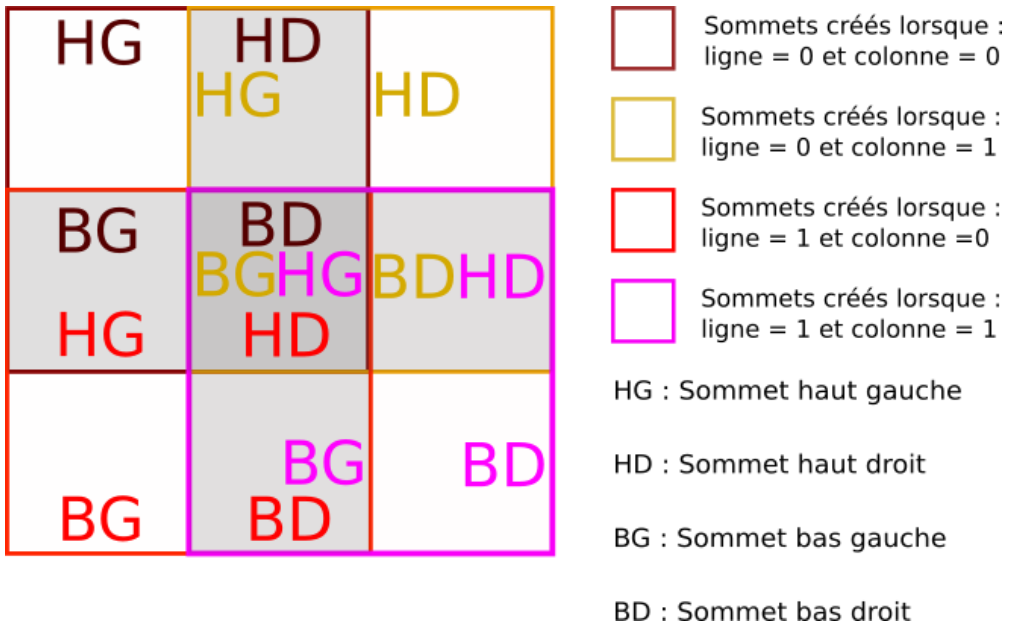
Nous avons proposé une structure qui évite les doublons et qui pourrait s'apparenter à une matrice puisque nos sommets sont rangés par lignes et colonnes. La structure est la suivante: *TreeMap<Double, TreeMap<Double, Sommet>>* où nous stockons un sommet comme ceci: *TreeMap<abscisse, TreeMap<ordonnée, "instance de Sommet">>*

Avec cette structure, le traitement est bien plus rapide et les tests de créations sont énormément simplifiés.

Lecture de l'image et traitements associés

Une image chargée en mémoire est sous forme de matrice. Nous parcourons donc l'image de type *BufferedImage* contenue la classe *Charger* par le biais de deux boucles for imbriquées. La première boucle permet de parcourir les lignes de l'image et la seconde permet de parcourir les colonnes de celle-ci.
Dans ces boucles for nous appliquons le traitement permettant de transformer chaque pixel de la *BufferedImage* en un sommet de notre structure. Ce traitement a subi de nombreux changements depuis les premières versions de ce projet. En effet, les traitements effectués au commencement permettaient de générer des points dans un espace à 3 dimensions et de transformer l'image d'entrée en un objet 3D. Cependant ces premières versions comportaient de nombreux défauts. Le premier étant la création du même point à plusieurs reprises car à chaque tour de boucle nous avions créé les quatre sommets permettant de construire un carré. Le problème avec cette implémentation du traitement est que les points se trouvant sur les bords étaient créés 2 fois et ceux se trouvant dans le milieu de l'image étaient créés 4 fois.

Ci-dessous, un schéma de la situation permettant de mieux la visualiser.



Ces doublons n'entraînaient pas seulement une prolifération de sommets de manière inutile mais posaient également un problème lors de la création des faces. En effet, Les deux faces composant un carré étaient créées à chaque tour de boucle et se retrouvaient donc, dès le deuxième tour de boucle à être composé des identifiant de sommets différents. Or, pour que les faces composent un maillage 3D

hermétique, celles-ci doivent être composées des mêmes points lorsqu'elles en ont en commun. Avec cette première version du traitement, nous avons donc été dans l'obligation d'y ajouter des tests avant la création de chaque sommet. Ceux-ci avaient pour but de vérifier si un sommet sur le point d'être créé n'avait pas une instance de *Sommet* qui avait été créée avec les mêmes coordonnées. Pour ce faire, nous devions récupérer tous les sommets de la ligne précédente afin de vérifier si le sommet courant n'avait pas été créé au tour précédent, lors de la création des sommets de coordonnées (colonne, ligne+1) et (colonne+1, ligne+1). Ces tests alourdissaient énormément le traitement, obligeant le parcours de la *TreeMap* contenant tous les sommets du maillage et le passage parmi les multiples conditions du test, et ce, à chaque tour de boucle. Au final nous nous sommes retrouvé avec un traitement qui durait plus de 3 minutes pour une image de taille relativement faible de 1000 pixels par 600.

Une refonte de la lecture de l'image

Le traitement étant beaucoup trop long et coûteux en ressources par rapport au résultat qu'il produisait (les maillages n'étaient pas entièrement hermétiques), nous avons suivis les conseils de notre tuteur qui étaient de revoir la structure du maillage.

Nous avons donc remplacé la *TreeMap* de *Sommet* qui regroupait l'intégralité des sommets du maillage par deux *TreeMap* de *TreeMap* de *Sommet*. Une pour les sommets de la surface et une deuxième pour les sommets du socle.

Pourquoi une *TreeMap* imbriquée ? Tout simplement pour recréer le fonctionnement d'une matrice contenant, en valeur finale, le sommet associé aux coordonnées ligne et colonne en guise de clés. Avec cette structure, nous n'avons plus besoin de tester si les instances de sommet ont déjà été créée car on sait exactement où l'on insère et ce qui a été créé. De plus, cela n'oblige pas à parcourir toute la *TreeMap* en testant sur chaque sommet ses coordonnées afin de trouver le sommet que l'on cherche. Nous gagnons donc en efficacité du code et en facilité d'utilisation de la structure. Les seuls tests restants sont ceux qui testent si la *TreeMap* se trouvant en valeur de la première est bien instancié.

Donner une hauteur à un pixel

Lors du parcours de l'image avec les techniques expliquées précédemment, nous associons à chaque pixel de l'image qui n'a que deux coordonnées, une troisième qui est sa hauteur dans l'espace à trois dimensions. Celle-ci est attribuée à chaque pixel en fonction de son intensité de gris, même si celui-ci est en couleur. La couleur d'un pixel étant divisée en trois, rouge, vert et bleu, pour obtenir son intensité de gris, nous faisons la moyenne de ces trois couleurs. Cela nous retourne alors un résultat entre 0 et 255 (valeurs possible pour un octet) qui est multiplié par un rapport calculé avant le lancement du traitement afin de créer des hauteurs correspondantes à la hauteur maximale désirée par l'utilisateur. Plus la valeur de l'intensité de gris est élevée, plus la hauteur retournée pour le sommet dans le maillage sera proche de la hauteur maximale défini par l'utilisateur et inversement.

Ci-dessous, la méthode permettant cette génération des hauteurs.

```
1.  public static double getHauteurPixel(double ligne, double colonne, double resolution, BufferedImage
     image) {
2.      int pixel, rouge, vert, bleu, moyenne;
3.      double hauteur;
```

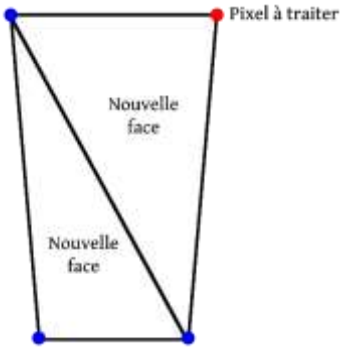
```
4.      int x = (int) Math.ceil(colonne);
5.      int y = (int) Math.ceil(ligne);
6.      pixel = image.getRGB(x, y);
7.      rouge = (pixel >> 16) & 0xff;
8.      vert = (pixel >> 8) & 0xff;
9.      bleu = (pixel) & 0xff;
10.     moyenne = 255 - (rouge + vert + bleu) / 3;
11.     hauteur = (resolution * moyenne) + 50;
12.     return hauteur;
13. }
```

Le traitement en lui-même

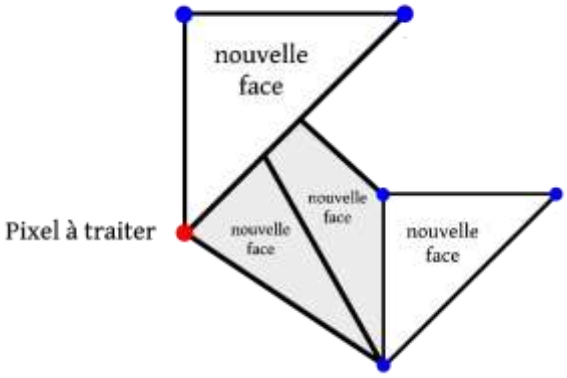
Pour ce qui est du traitement, nous avons procédé de cette manière :
Tout d'abord, nous créons les sommets dans l'ordre dans lequel ils seront écrits dans le fichier. Dans un premier temps, nous créons tous les sommets de la surface à la bonne hauteur. Dans un second temps, nous créons tous les points du socle, à une hauteur de 0.
Ensuite, nous devons créer les faces qui relient tous ses points. Nous parcourons toute l'image, pixel par pixel. Pour chaque pixel, plusieurs cas sont possibles :

- Cas n°1 : Le pixel est au centre,
  Dans ce cas-là, nous créons les faces de la surface et du socle de cette manière :

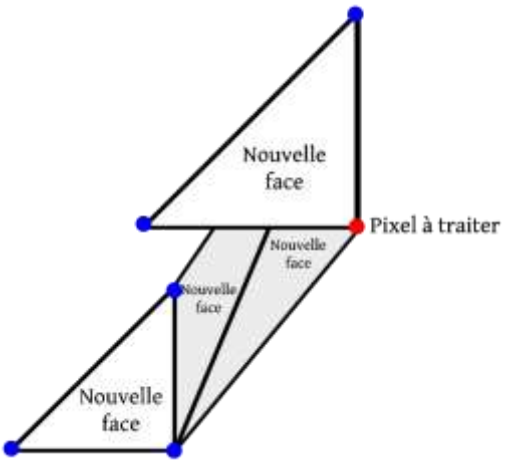
- Cas n°2 : En coordonnées (0;0),
  Aucune face n'est créé.
- Cas n°3 : Quand le pixel est tout en haut,
  Nous créons uniquement les faces du côté haut :



- Cas n°4 : Quand le pixel est sur le bord gauche,
  Nous créons les faces de cette manière :



- Cas n°5 : Quand le pixel est sur le bord droit,
  Nous créons les faces de cette manière :

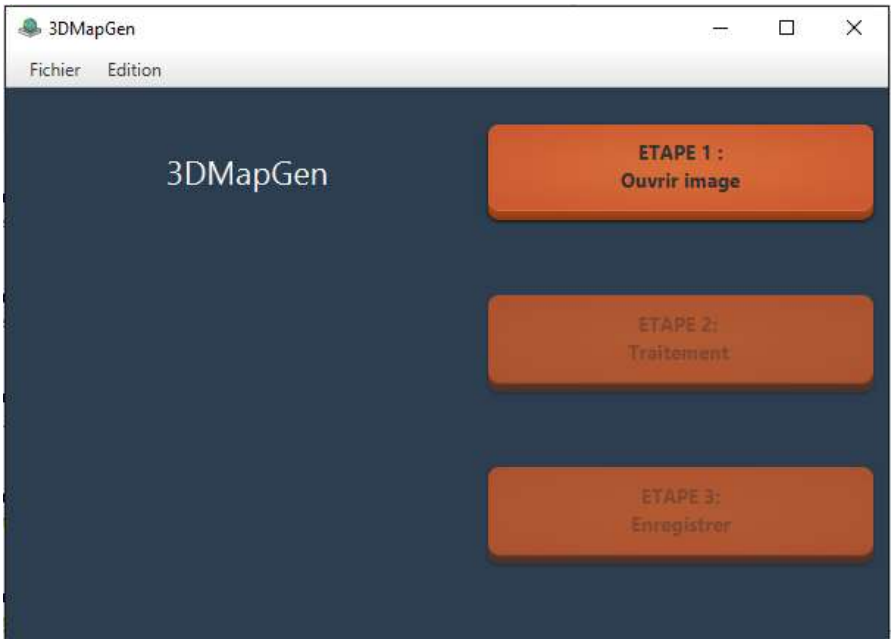


- Cas n°6 : Quand le pixel est tout en bas
  Nous créons les faces comme pour les pixels du centre et nous rajoutons le côté bas comme pour le côté haut.

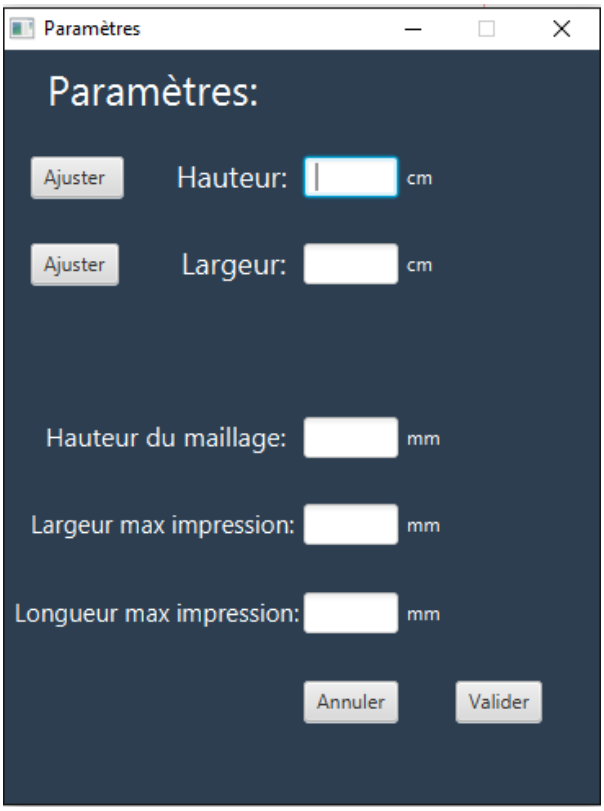
IMPLEMENTATION DES FONCTIONNALITES

L'interface graphique

L'interface graphique a été réalisé en Java FX. Nous avons choisi de faire une interface graphique assez simple d'utilisation pour que l'utilisateur puisse comprendre rapidement les étapes de la génération du maillage.



Etape 1: l'utilisateur peut choisir l'image qu'il veut traiter avec une fenêtre de dialogue.
Si une image a été sélectionnée une nouvelle fenêtre est ouverte, la fenêtre des paramètres.
Etape 2: quand les paramètres ont été sélectionnés on peut lancer le traitement.
Etape 3: l'utilisateur peut choisir où enregistrer le dossier avec les maillages.



Fenêtre de paramètre qui apparait à l'écran après avoir sélectionné une image

Elle permet à l'utilisateur de choisir ses propres paramètres pour le traitement de l'image selon la taille de son imprimante et de ses préférences au niveau de la taille de la pièce finale. Nous avons créé une classe *Parametres* avec toutes les valeurs. Elle va être instanciée lorsque le bouton Valider va être cliqué.

Pour que l'utilisateur puisse entrer la taille de la pièce finale nous devons contrôler ce qu'il rentre en paramètre afin que le ratio de l'image ne change pas pour ne pas déformer l'image ce qui peut entraîner un problème au niveau des attaches, de la taille de la police braille et une déformation du plan. Ainsi si le ratio hauteur/largeur entré par l'utilisateur n'est pas le même que le ratio de l'image de base un message est affiché et on ne peut pas valider les paramètres.
Les ratios sont calculés de la manière suivante:

```
1.  ratioL = image.getWidth() / image.getHeight();
2.  ratioH = image.getHeight() / image.getWidth();
```

L'utilisateur peut donc choisir d'ajuster la hauteur ou la largeur de l'image qui va être recalculée:

```
1.  largeur = Double.parseDouble(largeurField.getText());
2.  hauteur = largeur * ratioH;
3.  hauteurField.setText(String.valueOf(hauteur));
```

Le contrôleur des paramètres ne connaît que l'image à traiter et une instance de paramètres. Quand l'utilisateur valide ses paramètres ils sont récupérés de la façon suivante:

```
para.setHauteurImage(hauteur);
para.setHauteurMaillage(hauteurMaillage);
```

Ensuite le bouton Traitement permet de réaliser tous les traitements à la suite en récupérant les paramètres de cette instance, par exemple:

```
decouperImage(ch, para.getLargeurImage(), para.getHauteurImage(),
para.getLargeurMaxImpression(), para.getHauteurMaxImpression());
```

L'interface se veut simple d'utilisation car lorsqu'une étape est terminée l'utilisateur ne peut choisir que la suivante.
La partie gauche de l'interface est utilisée pour visionner l'image grâce à une *ImageView*. Elle apparaît une fois que l'utilisateur a ouvert son image pour vérifier qu'il s'agit bien de la bonne image.

De plus nous avons intégré dans le menu d'édition une sélection de thème pour les couleurs de l'interface. Un thème avec des couleurs plus sombres et un thème en noir et blanc et des polices d'écriture plus grosses. Ainsi cette interface peut être plus facilement utilisable pour une personne malvoyante. En effet les couleurs plus sombres et avec moins de contrastes peut être plus difficile à lire pour une personne daltonienne ou malvoyante. Nous avons donc dû décrire le style de la fenêtre dans un fichier CSS. Nous pouvons donc changer de style plus facilement en changeant de classe CSS utilisée de cette manière :

```
public void changerTheme1() {
    gridPane.setStyle("-fx-background-color:white");
    enregistrer.getStyleClass().remove("record-sales");
    enregistrer.getStyleClass().add("basic");
}
```

Découpage des pièces

Comme dit précédemment, les imprimantes 3D de nos jours ont des tailles d'impressions assez variables. Elles peuvent aller de 10 à 30 centimètres pour la plus grande partie des imprimantes mais peuvent atteindre des tailles d'impressions plus grandes avec des imprimantes professionnelles par exemple.

Donc, si on a une imprimante avec un plateau de 20 par 20 centimètres et que l'on veut obtenir une pièce finale de 40 par 40 centimètres, on aura besoin d'imprimer 4 pièces.
Nous avions donc besoin de nous occuper de la découpe des maillages. Pour cela, nous avons créé une classe *Decoupage* qui s'occupe de calculer le nombre de découpes nécessaires.

Pour connaître le nombre de découpes nécessaires dans la largeur (ou la hauteur), nous divisons la largeur voulue par la largeur maximale d'impression et nous récupérons l'entier supérieur :

```
Decoupage.nbDecoupeLargeur = (int) Math.ceil(largeurVoulue / (largeurMaxImpression / 10));
```

Où nbDecoupeLargeur est un attribut static de la classe *Decoupe*.

Ensuite, nous pouvons calculer la largeur et la hauteur d'une parcelle à partir des dimensions de l'image et du nombre de découpes :

```
1. Decoupage.largeurParcelle = (int) Math.floor(imageBase.getWidth() / getNbDecoupeLargeur());
```

Où imageBase représente l'image de base.

La méthode decouperImage() de la classe *Decoupe* retourne une liste de *BufferedImage* qui est, en fait, l'image découpée en parties égales selon les paramètres calculés précédemment :

```
1. for (int x = 0; x < nbDecoupeLargeur; x++) {
2.     for (int y = 0; y < nbDecoupeHauteur; y++) {
3.         listeImages.add(imageBase.getSubimage(x * getLargeurParcelle(), y * getHauteurParcelle(), g
   etLargeurParcelle(), getHauteurParcelle()));
4.     }
5. }
```

Où listeImage est la liste de sous-images qui est retournée.

De cette manière, on peut traiter les sous-images une par une et donc avoir autant de maillages que de sous-images.

Ecriture au dos

Dans le cas où l'image est découpée et que plusieurs impressions 3D sont requises, nous avons prévu d'inscrire au dos de la carte le numéro de celle-ci afin de rassembler les différentes parties. Par exemple pour une image découpée en 4 parties, l'écriture aurait été de la sorte:

A1	A2
B1	B2

Pour pouvoir écrire au dos de la carte, il fallait que le socle de celle-ci soit "creusé" afin de pouvoir écrire sans rehausser l'épaisseur totale de la pièce.
Afin de creuser ce socle, à la création d'un point du socle, nous faisions un test pour savoir si le sommet actuel était dans la zone du creux. La zone du creux a été délimitée arbitrairement comme couvrant 80% de la largeur de la pièce (afin d'avoir 10% de la largeur de chaque côté du creux) et pour un souci de dimensions, ce sont ces 10% de la largeur qui sont également reportés aux niveaux de l'épaisseur de la bordure du creux.

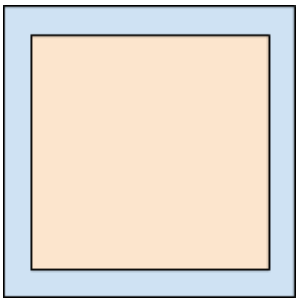


Figure représentant le dessous du socle, la partie rosée étant la partie creusée.

Le test se fait en comparant les coordonnées du point en train d'être créé et celles comprises dans la zone du creux. Si les coordonnées du point sont comprises dans la zone. La coordonnée Y du point est remontée, ce qui, point par point, forme ce creux.

La nécessité d'avoir la même largeur pour les bordures est dû au fait que les différentes pièces vont devoir être assemblées et nous avons pensé à un système d'attache qui permettrait de relier des pièces entre elles. Et comme on ne sait pas à l'avance les dimensions de chaque partie, nous avons normalisé la largeur de la bordure pour que l'attache qui sera générée en fonction des dimensions de la partie soit à la bonne taille.

Pour accueillir cette attache, le socle a été creusé sur les bords pour pouvoir y glisser l'attache

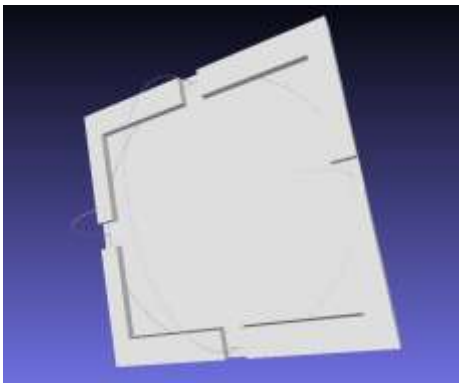


Figure représentant le dessous du socle, on peut apercevoir le creux et les emplacements pour les attaches

L'attache est générée en un seul exemplaire mais est à imprimer autant de fois que nécessaire. Ce nombre se détermine par nbAttache = (nbDecoupeL - 1) + (2*nbDecoupeL - 1) * (nbDecoupeH - 1) où nbDecoupeL est le nombre de parties découpées dans la largeur et nbDecoupeH est le nombre de parties découpées dans la largeur.

Par exemple pour une image découpée en 2 parties dans la largeur et 2 dans la hauteur (soit une image en 2*2), on a (2-1) + (2*2 - 1) * (2-1) = 1 + 3*1 = 4 attaches.

Attache qui permet de relier deux parties entre elles. Cette attache est proportionnelle à la taille de chaque partie.

Les attaches pour une image donnée sont toutes identiques c'est pour cela que l'épaisseur de la bordure est la même sur la largeur et la hauteur, pour que l'attache puisse être placée dans n'importe quel emplacement.
L'épaisseur de cette attache est la même que la hauteur du creux afin que ceux-ci s'emboîtent parfaitement.

Photo qui montre comment l'attache relie deux parties entre elles

Ce socle a donc été creusé afin d'écrire au verso les numéros des différentes parties, cependant après notre première impression, nous nous sommes rendu compte que le dessous du socle n'était pas lisse et par conséquent il est impossible d'imprimer un rendu propre. C'est pourquoi l'écriture au verso de la carte n'a pas été implémentée même si des pistes ont néanmoins été explorées (comme par exemple la conversion d'une chaine de caractère en image Bitmap qui aurait subi le même traitement (lecture des pixels et génération de la hauteur en fonction de la couleur)).

ABSTRACT

This project is part of our two-year university degree in Computer Science at the University Institute of Technology, UCA in Clermont-Ferrand.
We did a four-month project on the 3D meshes generation. There were five of us in the group, but we lost a member who repeated a year.

The objective of this project was to develop and design an application that would allow the user to generate 3D meshes for printing their own touch maps.
We developed a desktop application from scratch. For this project, we divided up the work into different tasks that we assigned ourselves according to our preferences and our skills.

The first stage of this project was to define the features of the application and to prioritize the tasks to be carried out.
Secondly, we designed the data structure allowing making meshes corresponding to the images loaded by the users in our applications.
Then, we have developed all the features of our application.
At the same time, we have designed a graphical user interface adapted to end-users who are visually impaired.

To carry out this project, we followed a SCRUM-like agile method according to the advice of our supervisor M. Favreau.
After a number of unsuccessful attempts, we did not have time to implement the writing on the back of the touch maps.

Even though we have not developed a couple of features, we can deliver a working application to our supervisor.

CONCLUSION

Lors de ce projet nous avons pu découvrir de nouveaux outils et technologies. Il s'agissait de notre premier projet de grande envergure et avec un objectif plus concret et important. Nous avons pu développer nos connaissances en langage Java et Java FX en les utilisant dans des contextes que nous ne connaissions pas forcément bien.

Nous avons donc dû utiliser de nouvelles méthodes de travail pour collaborer au sein d'une plus grande équipe. Autant au niveau du partage des tâches que de l'organisation et la gestion du projet dans le temps qui nous était donné.

Nous avons pu appréhender la création 3D qui est une technique assez rigoureuse.
Dans le cadre du projet nous avons également eu l'occasion de nous intéresser à l'impression 3D qui est un secteur qui nous semble très innovant et aux possibilités infinies, ce qui nous a motivé tout au long de notre travail. Aussi, le résultat final est concret et a une utilité réelle, nous avons eu l'honneur d'initier le développement d'un projet qui va évoluer dans le futur. En effet, la suite de notre travail va être reprise par des étudiants en master et ensuite être intégré à l'outil de développement des cartes de l'IGN. Dans cette optique nous avons essayé d'être le plus clair possible dans notre réalisation et de fournir un travail fonctionnel.

De plus nous avons eu l'occasion de fournir ce travail pour des personnes malvoyantes, ce qui nous a motivé car nous avons développé un outil utile et inédit dans ce domaine. Cependant, quelques contretemps nous ont empêchés de mettre en place certaines fonctionnalités que nous voulions intégrer à notre projet.

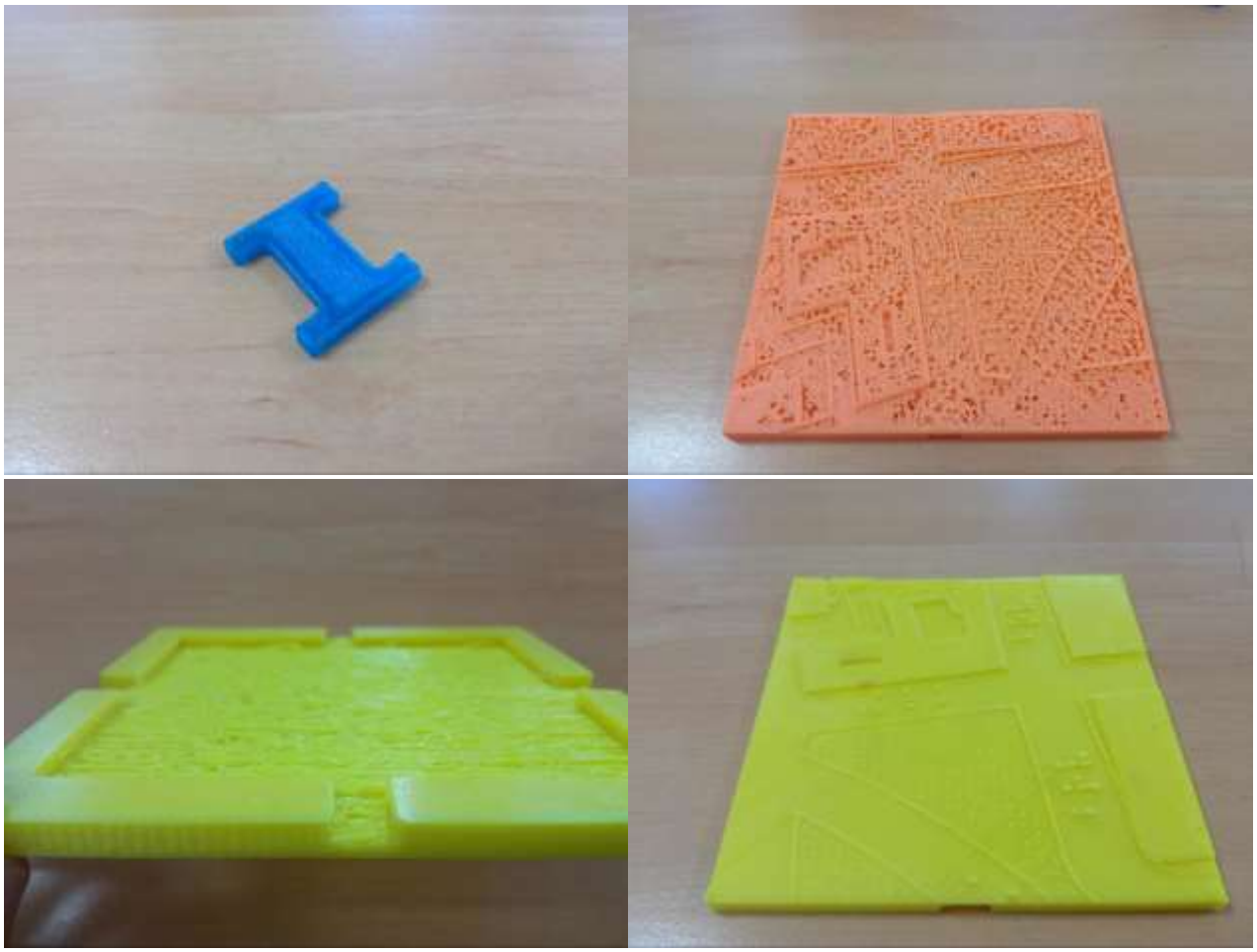
Enfin, ce projet nous a tous apporté à titre personnel de la méthodologie et de l'approfondissement dans certaines matières que nous avons étudiées en cours. De plus nous avons trouvé l'aspect culturel du thème très enrichissant de par les personnes que nous avons pu rencontrer ainsi que les recherches que nous avons effectuées sur un sujet que nous n'avions jamais abordé.

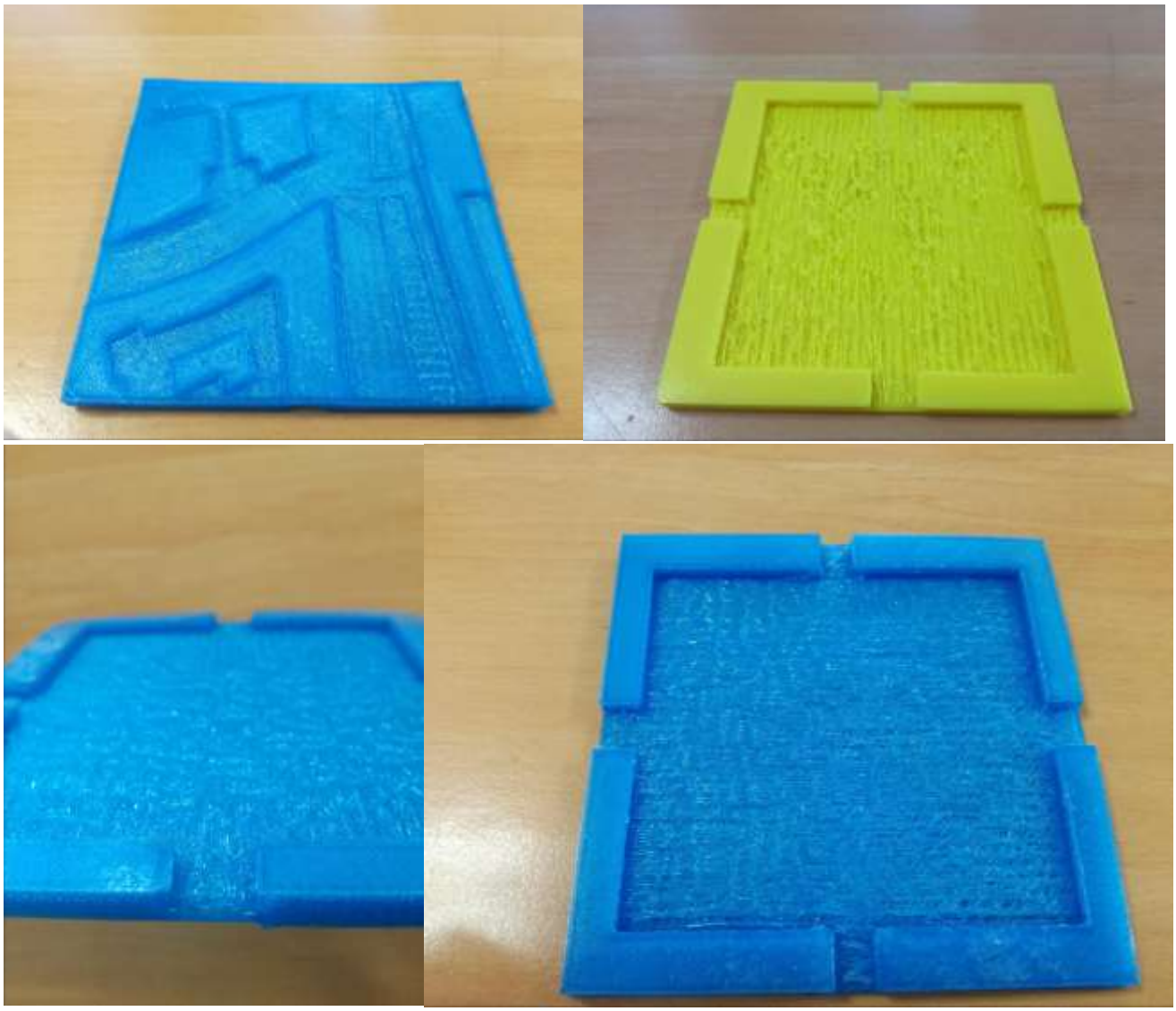
WEBOGRAPHIE

Wikipedia : https://fr.wikipedia.org/wiki/Objet_3D_(format_de_fichier) (Novembre 2016)
Developpez.com : http://slim-boukettaya.developpez.com/tutoriels/traitement-images-java/ (Novembre 2016)
Hapticke : http://hapticke.mapy.cz/?x=14.4573210&y=50.0744370&z=11 (Tout au long du projet)
Javadoc : http://docs.oracle.com/javase/8/docs/ (Tout au long du projet)
Topotopo.io : http://topotopo.io/ (Janvier 2017)

ANNEXES

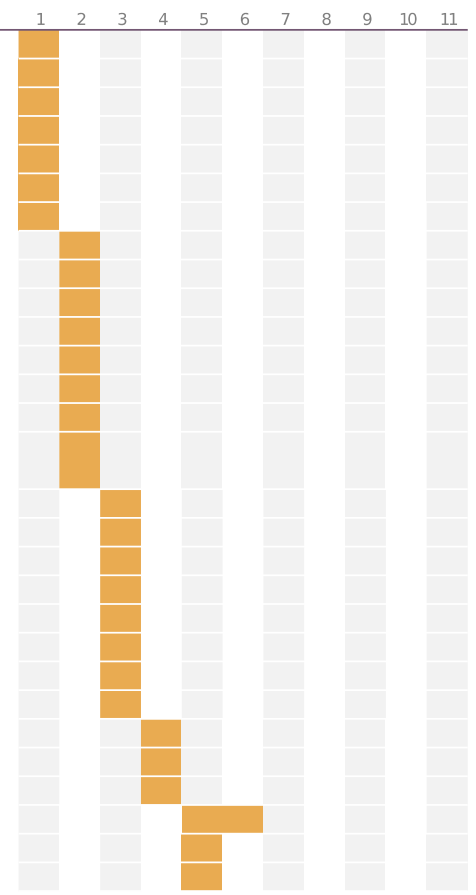
Photos des différentes pièces d'une carte que l'on a imprimée à partir de l'un de nos maillages :



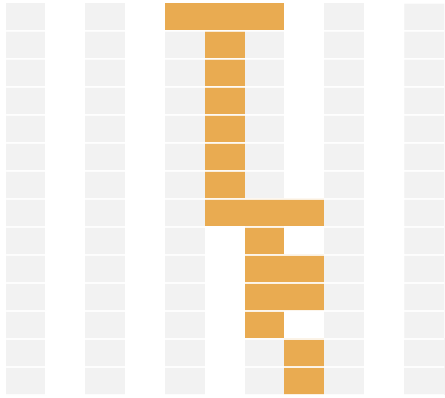


Gantt réel du projet

ACTIVITÉ	RÉEL DÉBUT	RÉEL DURÉE	POURCENTAGE ACHEVÉ
Réunion de présentation du projet	1	1	100%
Importation d'une image dans l'application	1	1	100%
Conception de la structure de donnée	1	1	100%
Création de la structure de donnée	1	1	100%
Documentation sur les formats de maillage 3D	1	1	100%
Traitement de l'image dans l'application	1	1	100%
Exporter un maillage dans un format .obj	1	1	100%
Terminer l'exportation du fichier	2	1	100%
Création d'une interface graphique	2	1	100%
Modifier la structure du maillage	2	1	100%
Conception du socle et des côtés	2	1	100%
Création du socle du maillage	2	1	100%
Dessin de conception de l'interface graphique	2	1	100%
Reprise de la conception du maillage	2	1	100%
Participation à la conférence "Fabrication additive : Opportunités et innovations"	2	1	100%
Création des côtés	3	1	100%
Gantt prévisionnel P3	3	1	100%
Rédaction d'une charte de codage	3	1	100%
Commenter le code	3	1	100%
Vérifier les noms de variables et de classes	3	1	100%
Conception des pièces du puzzle	3	1	100%
Pseudo code du découpage	3	1	100%
Extrusion du rectangle au verso de la carte	3	1	100%
Inversion du traitement des hauteurs	4	1	100%
Refonte graphique	4	1	100%
Création du répertoire à l'enregistrement du maillage	4	1	100%
Modification du traitement	5	2	100%
Modification du découpage de l'image	5	1	100%
Création des slots pour les attaches	5	1	100%



Mise en place de test sur la création des sommets	5	3	100%
Ajout d'une classe regroupant les paramètres	6	1	100%
Ajout d'une fenêtre de saisie des paramètres	6	1	100%
Finalisation des attaches	6	1	100%
Ajout d'une icône	6	1	100%
Mise à l'échelle des attaches par rapport aux slots des pièces	6	1	100%
Mise à l'échelle des maillages en mm	6	1	100%
Rédaction du rapport de projet	6	3	100%
Ajout du réglage automatique de la taille de l'image et conservation du	7	1	100%
Changement de la structure du maillage	7	2	100%
Refonte du traitement en conséquence du changement de structure	7	2	100%
Correction du ratio de l'image lors du redimensionnement	7	1	100%
Ajout d'une sélection de thèmes sur l'interface graphique	8	1	100%
Vérification interactive avec l'utilisateur pour la saisie des paramètres	8	1	100%



Diagrammes de cas d'utilisation de notre application :

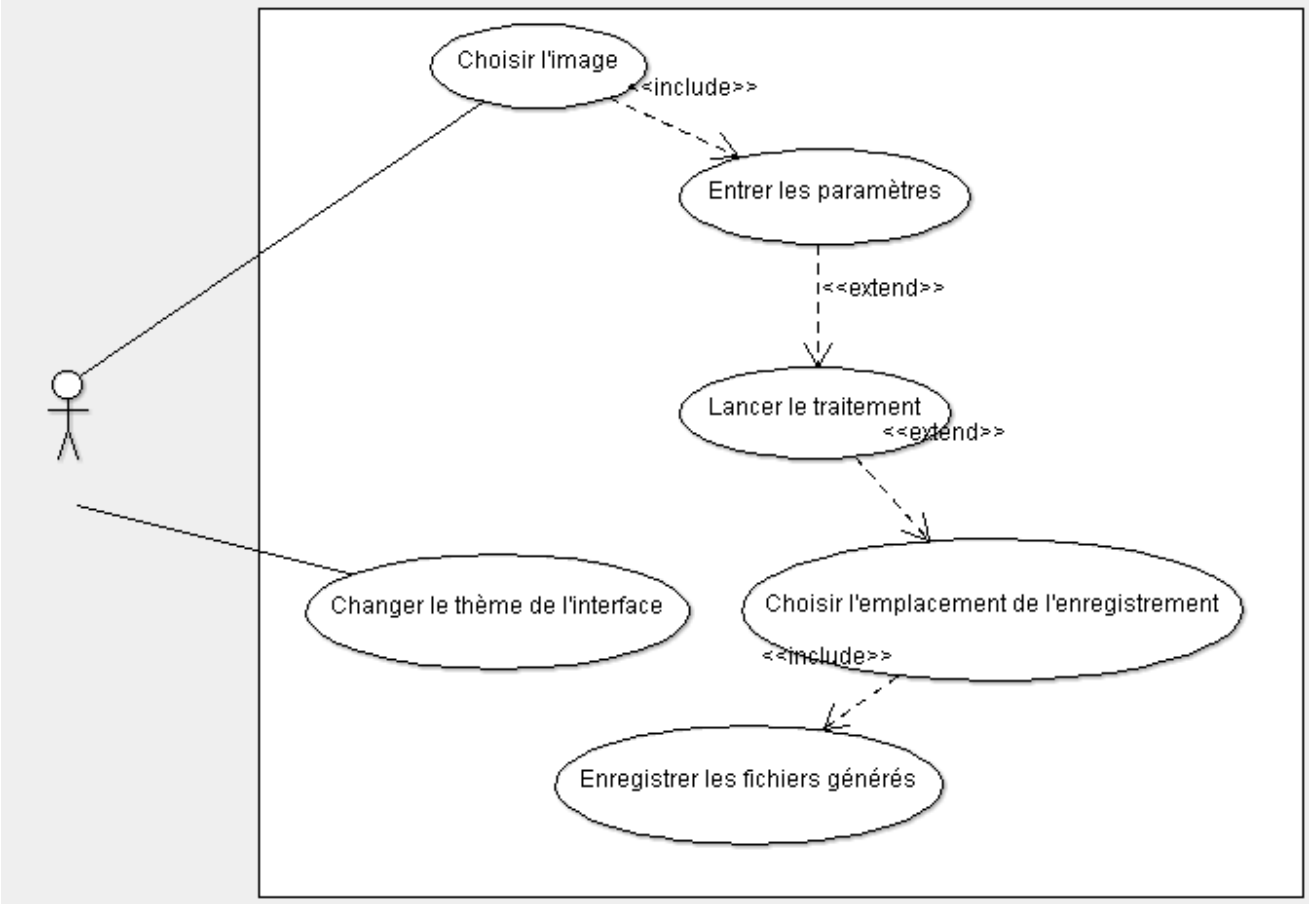


Diagramme de classes de l'application :