Rapport CGDI

Corto Cristofoli et Max Royer

4 avril 2025

1 Introduction

Cet article est réalisé dans le cadre de notre cours de CGDI. Il s'agit de la présentation de notre implémentation de l'article de recherche *Curved PN triangle*, par Alex Vlachos, Jorg Peters, Chas Boyd et Jason L. Mitchell.

L'article présente un algorithme de lissage de modèles 3D de basse résolution. Pour cela, l'idée est d'utiliser la méthode des « Curved PN triangles ». Un modèle 3D de basse résolution n'a que peu de polygones et donc une alure nécessairement anguleuse. Pour le rendre plus organique un lissage est nécessaire : il faut rajouter un grand nombre de triangle afin de faire disparaître ces angles. Ce rajout se fait algorithmiquement en suivant des surfaces de Bézier triangulaires.

Si l'article décrit l'algorithme de manière théorique, nous nous sommes penchés sur sa compréhension et son implémentation en C++ à l'aide de la bibliothèque **geometry-central**.

2 Principe de l'algorithme

Dans tout le rapport nous considérons qu'un modèle 3D est un ensemble de face triangulaires. Il est facile de se ramener à ce cas ci en triangulant simplement les faces non triangulaires.

L'algorithme vise à transformer chaque faces triangulaires en une multitude, dépendant d'un facteur : le *level of detail* ou `lod`.

Concrètement, le `lod` correspond au nombre de sommets à ajouter sur une arête du triangle d'origine (voir 1).

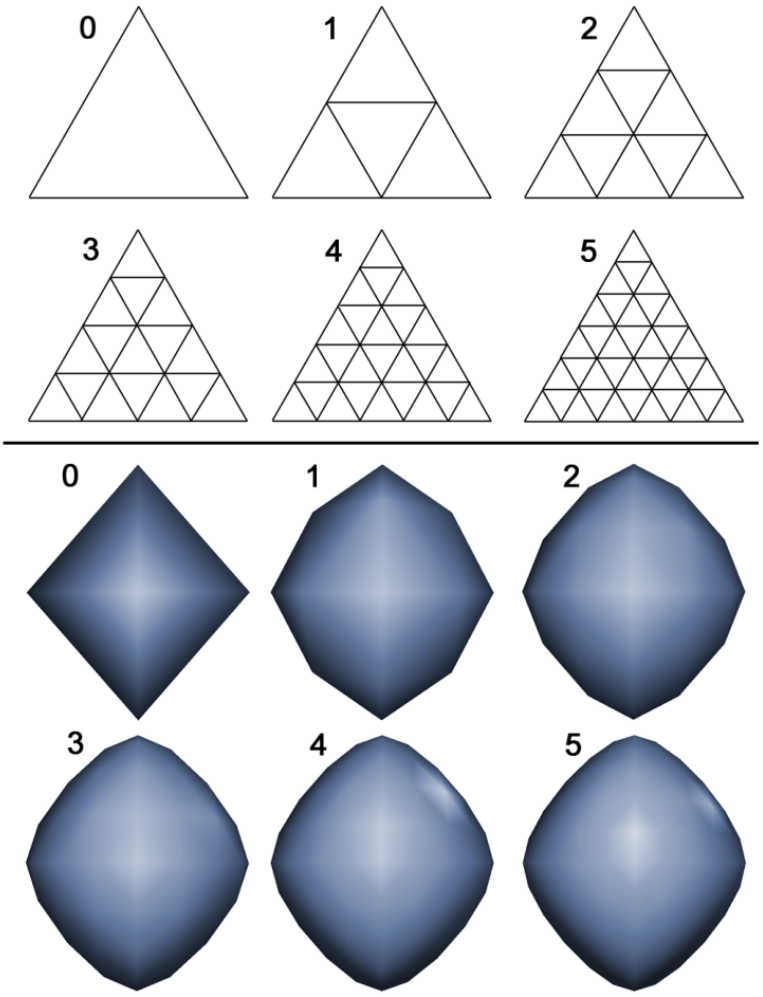

FIGURE 1 – Sous-triangulation en fonction du `lod` (de 0 à 5)

3 Enjeux d'implémentation de *Geometry Central*

4 Indexation des sommets du modèle 3D

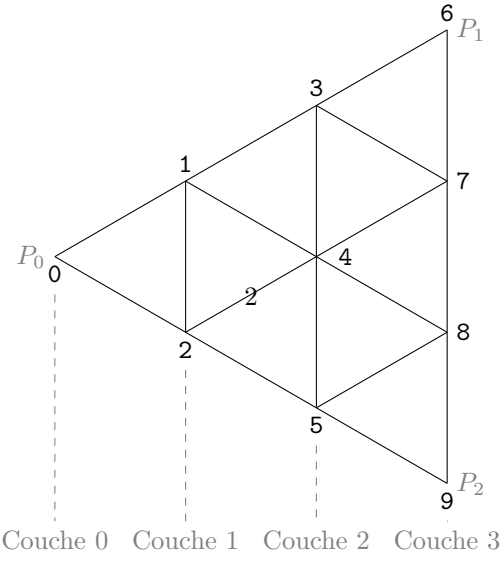

FIGURE 2 – Indexation de la triangulation d'une face pour un `lod` de 2

On numérote la discrétisation de la face de cette manière pour avoir facilement les nouvelles faces. En effet on remarque que pour un sommet i dans une couche c, on a que ses deux voisins dans la couche $c+1$ sont les sommets $i+c+1$ et $i+c+2$. Cela permet de déduire rapidement les nouvelles faces à ajouter en disant que pour une discrétisation avec un `lod` l, on ajoute pour chaque sommets i dans une couche c avec $c \leq l$ la face $(i, i+c+1, i+c+2)$ et la face $(i, i+c+2, i+1)$ si $i+1$ est aussi dans la couche c (*i.e.* $i+1 < \frac{(c+2)(c+1)}{2}$)

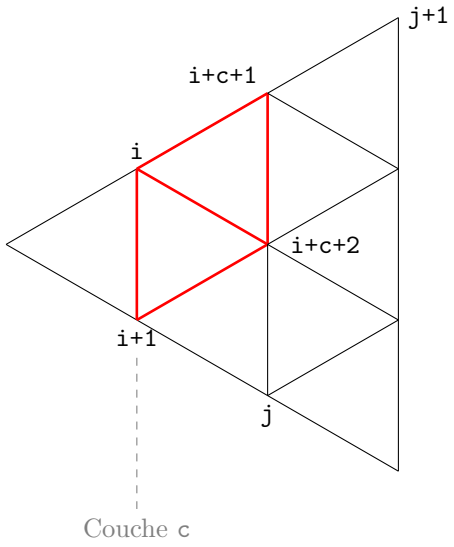

FIGURE 3 – Exemple de faces ajoutées pour un sommet `i` dans la couche `c` ainsi que le cas où `j` et `j+1` ne sont pas dans la même couche

Chaque sommets est alors identifié par ses coordonnées barycentriques en fonction de (P_0, P_1, P_2), étant donné que ces coordonnées sont rationnelles et sont en fait des multiples de $\frac{1}{\texttt{lod}+1}$ nous avons décider de garder des coordonnées entières pour identifier les sommets tout en gardant le `lod` lorqu'il s'agit de calculer la position du sommet dans la tuile de Bézier.

Ainsi un sommet est représenté par un `vector` de `pair(int,int)` de taille 3, tel que pour chaque pair, le premier élément est l'indice du point P_i et le second est la coordonnée barycentrique associée au point P_i.

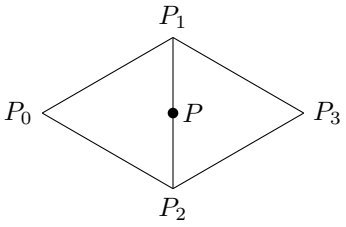

FIGURE 4 – Exemple de double représentation pour un seul et même point

Ici le sommet P peut être représenté par : $\{(P_0, 0), (P_1, \frac{1}{2}), (P_2, \frac{1}{2})\}$ mais aussi par $\{(P_1, \frac{1}{2}), (P_2, \frac{1}{2}), (P_3, 0)\}$. Au départ nous avons pensé naïvement à directement calculer la valeur du points dans la tuile de Bézier associé

car lui pour le coup sera "le même" que pour les autres représentations. Malheureusement cette approche n'est pas concevable car les arrondis sur les `double` ne nous garantiraient pas que l'identification d'un point soit unique. Nous allons voir comment régler ce problème après avoir décris la classe `Obj`.

Pour recréer un nouvel objet nous avons créé une petite classe `Obj` ayant les primitives pour construire un tel objet. Le problème est qu'un point identique peut avoir plusieurs représentation possibles. Par exemple pour :

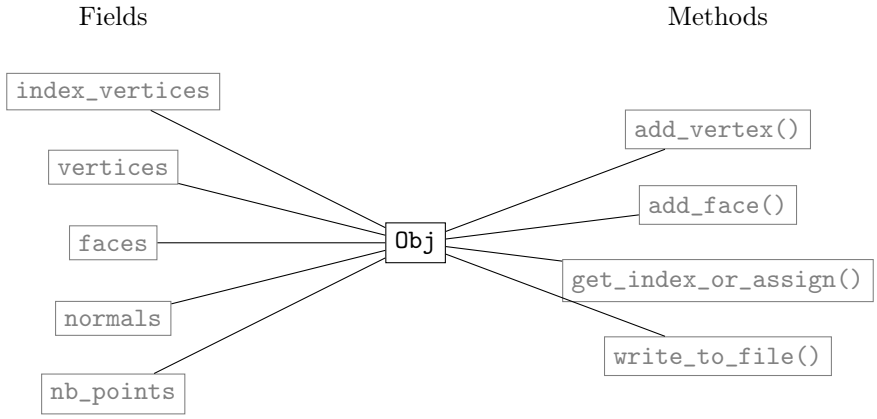

FIGURE 5 – Description de la classe `Obj`

Ici on a `Obj.index_vertices : map<vector<pair<int,int»,int>`, cela veut dire que l'on doit être capable d'ordonner les représentations des points en coordonnées barycentrique tout en conservant leur équivalence.

Pour cela nous effectuons d'abord un tri sur le `vector` des coordonnées de manières à trier selon l'ordre lexicographique inversé sur l'ordre et sur les coordonnées, *i.e.* :

$$(i,j) <_{pair} (k,l) \Leftrightarrow l < j \text{ ou } (l = j \text{ et } k < i) \tag{1}$$

Cela permet alors d'ordonné facilement les représentations triées en considérant l'ordre lexicographique sur la partie ou la représentation a des coordonnées strictement positives. Pour exemple reprenons les deux représentations de la figure 4.

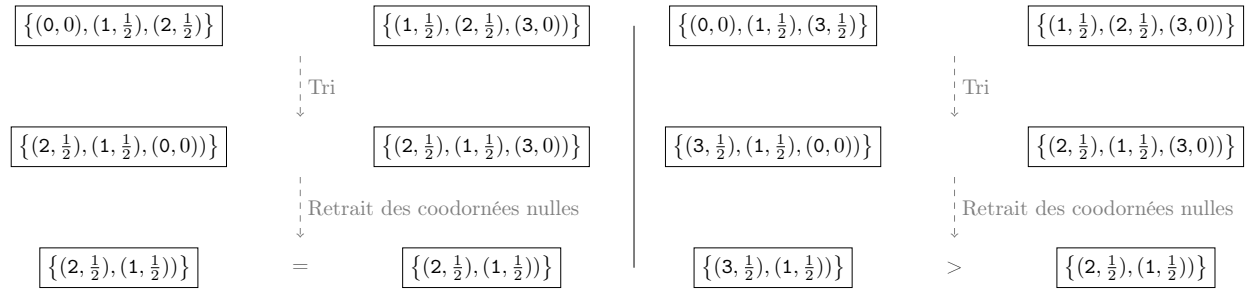

FIGURE 6 – Exemples d'ordres sur des représentations

Ainsi on peut passer à la `map` la structure de comparaison adéquate pour savoir si on a déjà ajouté le point dans `Obj.vertices`.