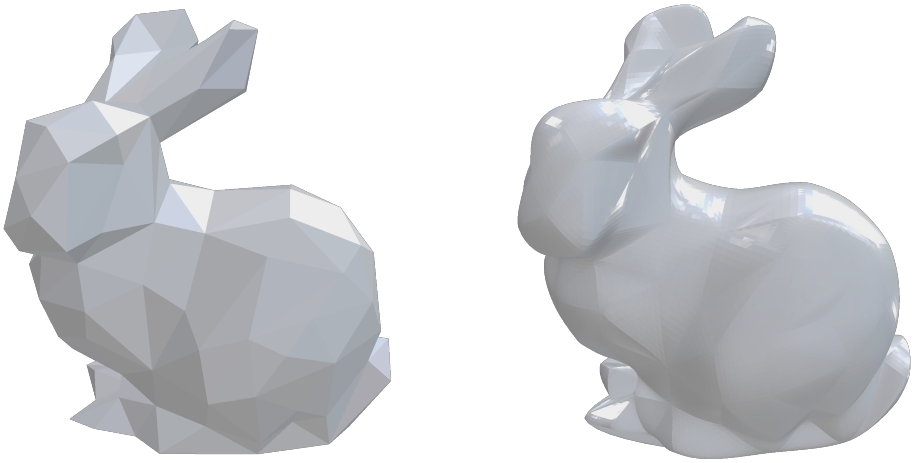
CURVE MY TRIANGLE
Rapport CGDI

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 diviser chaque faces triangulaires en une multitude de plus petites faces, dépendant d'un facteur : le *level of detail* ou `lod`.

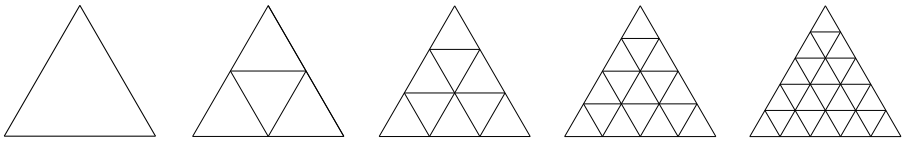


FIGURE 1 – Sous-triangulation en fonction du `lod` (de 0 à 4)

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

Une fois la subdivision faites pour chaque face (on verra plus tard que cette subdivision pose des soucis d'implémentation), il faut moduler la position des nouveaux sommets afin de parvenir à un lissage convainquant. Pour se faire, on utilise une surface de Bézier triangulaire avec 10 points de contrôle. Les différents sommets des nouvelles faces seront ensuite placés de manière à suivre la surface de Bézier produite. Si l'on nomme b_{ijk} avec $i + j + k = 3$ et $i, j, k \geq 0$, alors la position d'un des points du triangle aux coordonnées barycentrique (u, v) est définie dans l'espace par le patch de bézier b :

$$\text{Avec } w = 1 - u - v,\ b(u, v) = \sum_{i+j+k=3} b_{ijk} \frac{3!}{i!j!k!} u^i v^i w^i \tag{1}$$

Voyons maintenant comment calculer les différents b_{ijk}.

3 Calcul des points de contrôle du patch de Bézier

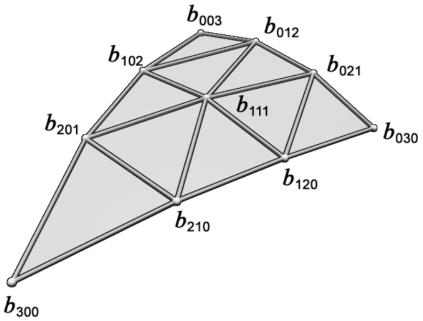


FIGURE 2 – Points/coefficients de contrôle d'un patch de Bézier triangulaire

En s'appuyant sur la figure 2, on peut distinguer 3 catégories de coefficients :

— *coefficients de sommets* : $b_{300}, b_{030}, b_{003}$
— *coefficients tangents* : $b_{210}, b_{201}, b_{120}, b_{021}, b_{102}, b_{012}$
— *coefficient central* : b_{111}

3.1 Coefficients de sommets

Afin de conserver la forme primaire du modèle, les sommets originaux ne change pas de position dans l'espace : la position des coefficients de sommets est donc la même que celle de leur sommet respectif (si l'on applique b à $(1,0)$, $(0,1)$ ou $(0,0)$ on voit bien que les positions sont respectivement b_{300}, b_{030} ou b_{003}).

3.2 Coefficients tangents

Afin de calculer les coefficients tangents introduisons la notion de normale à un point.

Définition 1 *Vecteur normal à un point*

Les modèles 3D que nous étudions sont des modèles orientés, c'est-à-dire que chaque face du modèle est définie par un unique vecteur unitaire, normal à la surface et orienté vers l'extérieur du modèle.

Soit s un sommet du modèle et f_a, f_b, f_c les faces auxquelles il appartient (si il y a un bord cela ne change pas grand chose). Le vecteur normal au sommet s est la moyenne des vecteurs normaux de chaques faces : $N_s = \frac{N_a+N_b+N_c}{||N_a+N_b+N_c||}$.

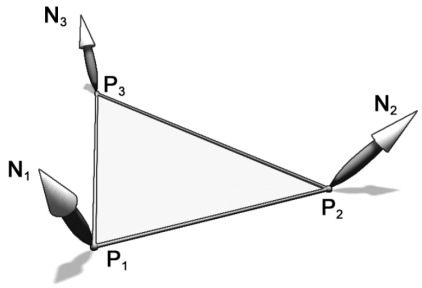


FIGURE 3 – Données en entrée : points P_i et leur normale N_i

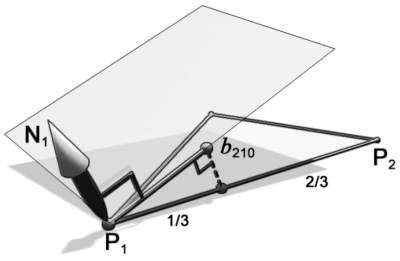


FIGURE 4 – Construction du coefficient tangent $b_2 10$ en projettant dans le plan de N_1

Pour chaque sommets, le vecteur normal induit un plan normal. On choisit donc de projeter chaque coefficient tangent dans le plan normal du sommet le plus proche (voir figure 4). Il n'y a pas d'ambiguïté sur le sommet le plus proche puisque les arêtes sont divisées chacune en trois et non en deux.

3.3 Coefficient central

Pour le coefficient central, on effectue une translation jusqu'à la position $E = (b_{210} + b_{120} + b_{021} + b_{012} + b_{201} + b_{102})/6$ (c'est la moyenne de tous les coefficients tangents). Si l'on note $V = (P_1 + P_2 + P_3)/3$ la position de départ de b_{111} on a donc effectué une translation de vecteur $E - V$. On ajoute ensuite la moitié de cette même translation.

Au final on aura $b_{111} = E + \frac{1}{2}(E - V)$. Il est important de noter que ce choix de facteur $\frac{1}{2}$ est arbitraire et qu'on peut varier cette valeur. En effet la formule pour calculer la position de b_{111} serait plutôt une formule du type $b_{111} = (1 - \alpha)E + \alpha V$. Le choix d'un $\alpha = -\frac{1}{2}$ est expliqué plus amplement dans le papier.

4 Enjeux d'implémentation de *Geometry Central*

Geometry Central utilise une structure de « manifold surface meshes ». Il s'agit d'un ensemble de faces triangulaires orientées (vers l'extérieur). Cet objet est très strict sur la manière de le transformer car il faut pouvoir maintenir la structure d' *halfedge* qui permet de naviguer à travers le modèle 3D. Cette contrainte permet une navigation très simple dans tout le modèle (via les sommets, les faces, les arêtes **ou** les *halfedges*) mais rend impossible l'ajout d'un simple point de manière arbitraire. La seule manière d'ajouter un point est de l'ajouter à une arête ou à une face, conduisant à une triangulation par ce point. Comme nous souhaitons ajouter de toutes nouvelles faces pour remplacer une ancienne face il faudrait pouvoir supprimer cette ancienne face, ce qui n'est pas faisable directement si ça n'est pas une face sur un bord du modèle.

Il est donc plus simple de reconstruire le modèle depuis zero. Un axe d'amélioration de notre projet serait donc de réussir à réaliser ces opérations « en place ». Dans la partie suivante nous expliquons comment gérer le soucis d'indexation de tous les nouveaux sommets à ajouter.

5 Indexation des sommets du modèle 3D

On indexe la discrétisation de la face de la manière décrite dans la figure 5 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.* $i + 1 < \frac{(c+2)(c+1)}{2}$) voir figure 6.

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.

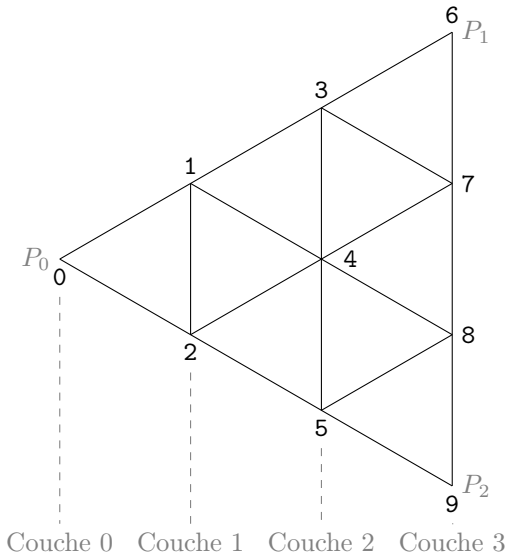


FIGURE 5 – Indexation de la triangulation d'une face pour un `lod` de 2

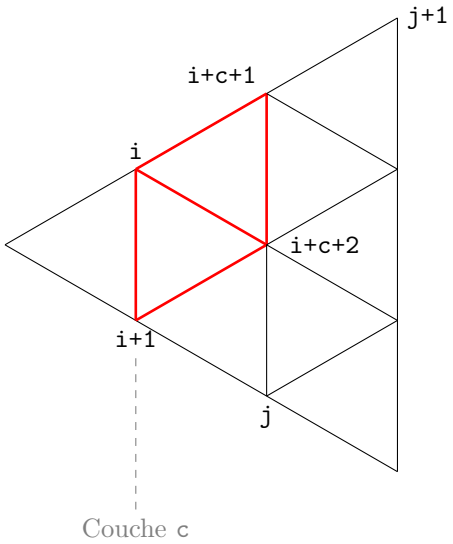


FIGURE 6 – Exemple de faces ajoutées pour un sommet `i` dans la couche `c` ainsi que le cas où `j` et `j+1` ne sont pas dans la même couche

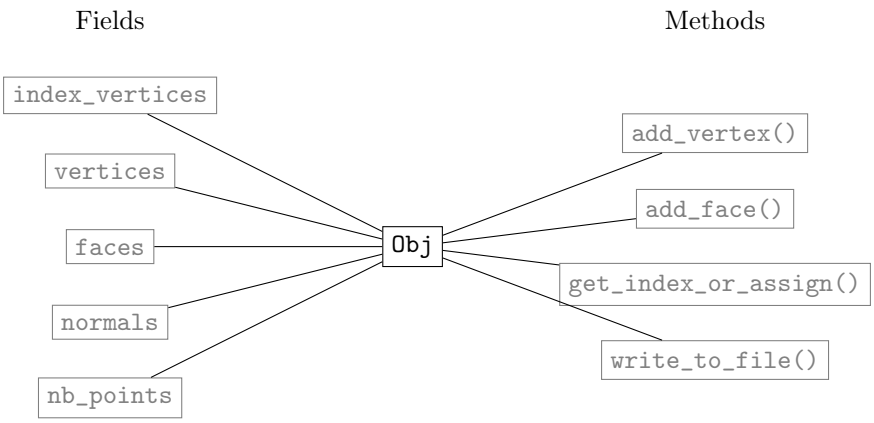


FIGURE 8 – Description de la classe `Obj`

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.

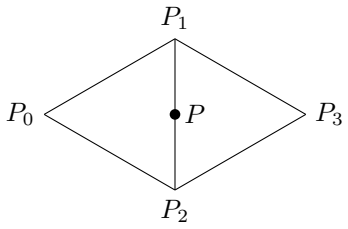


FIGURE 7 – Exemple de double représentation pour un seul et même point

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 associée 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. Voir la figure 8 qui récapitule les champs et les méthodes de la dite classe :

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.* :

$$(i, j) <_{pair} (k, l) \Leftrightarrow l < j \text{ ou } (l = j \text{ et } k < i) \tag{2}$$

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 7.

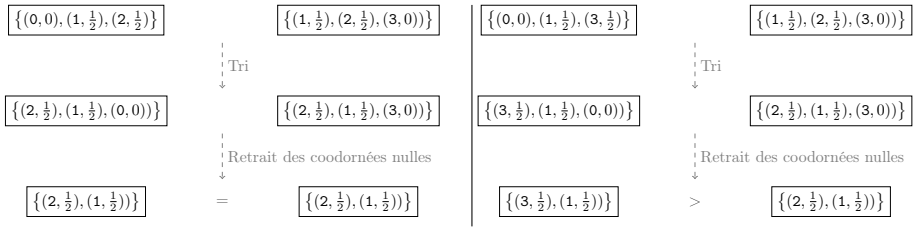
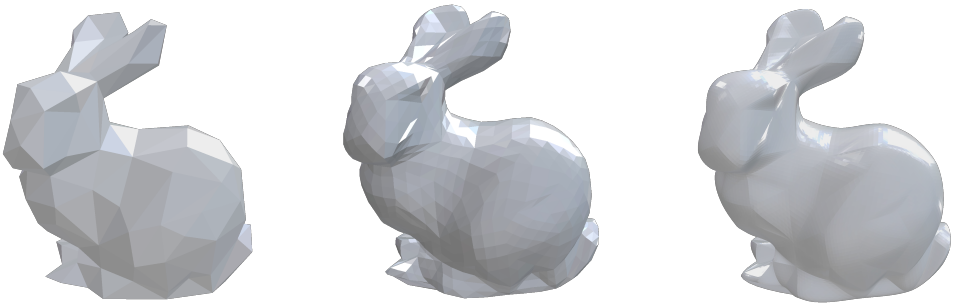


FIGURE 9 – Exemples d'ordres sur des représentations

Ainsi on peut passer à la `map` la structure de comparaison adéquate pour savoir si on a déjà ajouté le point dans `Obj.vertices`. Cela permet ainsi d'avoir une identification unique des nombreuses représentations d'un point possible.

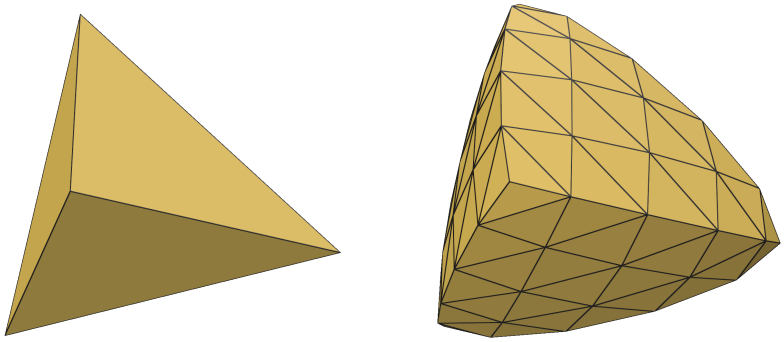
6 Résultats

Les résultats obtenus sont les suivants :



`lod` $= 0$ `lod` $= 2$ `lod` $= 8$

FIGURE 10 – Application de l'algorithme sur le fichier `bunny_low_poly` avec $\alpha = -\frac{1}{2}$



Avant Après

FIGURE 11 – Application de l'algorithme sur le fichier `tetrahedra` avec $\alpha = -\frac{1}{2}$ et `lod` $= 4$