

Géométrie algorithmique

Rapport de TP5

Baptiste DANIEL LAMAZIERE & Lilian HERTEL

28 mars 2023

Table des matières

Partie 1	Boîtes englobantes	1
Partie 2	Octree	1
Partie 3	Création de l'octree à partir d'un maillage	2
Partie 4	Visualisation de l'octree	3
Partie 5	Simplification du maillage	4

Partie 1 Boîtes englobantes

Pour commencer, nous avons dû réaliser une fonction pour calculer la boîte englobante d'un maillage. Pour cela, on calcule le maximum et le minimum des coordonnées des sommets du maillage sur chaque dimension. On construit ensuite 2 points qui seront deux coins opposés de notre boîte englobante. Il suffit donc de ces deux points pour construire une boîte englobante alignée sur les axes du maillages. On stocke aussi pour chaque boîte englobante, son point central pour simplifier certains traitements par la suite.

Partie 2 Octree

Pour modéliser l'octree, on commence par définir une structure représentant un noeud de l'arbre. Chaque noeud va donc contenir un vecteur de points pour stocker les sommets contenus dans le noeud, un vecteur de noeuds pour stocker les enfants du noeud, une boîte englobante, et un entier représentant le degré de profondeur. Nous avons aussi stocké dans un noeud deux informations qui servent à la simplification de maillage : le centre du noeud (calculé en faisant la moyenne des coordonnées de tous les points contenus dans le noeud), et un numéro qui permettra d'identifier ce noeud.

Partie 3 Création de l'octree à partir d'un maillage

Pour créer un octree à partir d'un maillage, on parcourt les sommets du maillages et pour chaque sommet on applique l'algorithme suivant qui prend en paramètre un sommet et la racine de l'octree:

```
if le noeud a des enfants then
    On cherche le noeud enfant dans lequel est contenu le sommet;
    On fait un appel récursif de la fonction en donnant ce noeud enfant en argument;
end
else
    if le noeud a plus de sommets que le maximum autorisé ET le niveau du noeud est
    inférieur à la profondeur maximum then
        On ajoute 8 fils au noeud;
        On cherche le noeud enfant dans lequel est contenu le sommet;
        On fait un appel récursif de la fonction en donnant ce nouveau noeud enfant en
        argument;
        for tous les sommets du noeud courant do
            On cherche le noeud enfant dans lequel est contenu le sommet;
            On fait un appel récursif de la fonction en prenant le noeud enfant en
            argument, et le sommet en question, pour descendre le sommet dans l'arbre;
        end
        On supprime tous les sommets du noeud courant;
    end
    else
        On ajoute le sommet au noeud;
    end
end
```

Algorithm 1: Algorithme d'ajout d'un sommet dans un octree

Pour nous aider à réaliser la création de l'octree nous avons recours à plusieurs fonctions auxiliaires.

Tout d'abord lors du parcours de l'arbre, nous utilisons une fonction de recherche de la boîte englobante fille associée à un sommet ; cette fonction utilise l'indice que nous avons donné à chacune des 8 boîtes filles lors de leur création. Il nous suffit donc de récupérer un vecteur de 3 booléens correspondant aux comparaisons entre les coordonnées du sommet avec celles du centre de la boîte englobante mère :

```
1  std::vector<bool> compareWithCenter(Polyhedron::Vertex_handle &vert, AABB
   ↳ boundingBox) {
2
3      std::vector<bool> result;
4
5      result.push_back(vert->point().x() > boundingBox.center.x());
6      result.push_back(vert->point().y() > boundingBox.center.y());
7      result.push_back(vert->point().z() > boundingBox.center.z());
8
9      return result;
10 }
```

En fonction de ces booléens on peut définir rapidement l'indice de la boîte fille qui contient le sommet.

```

1  OctreeNode& findBoundingBoxOfVertex_Optimized(OctreeNode &root,
   ↳ Polyhedron::Vertex_handle &vert) {
2      int index = 0;
3
4      std::vector<bool> decisions = compareWithCenter(vert, root.boundingBox);
5
6      if (decisions[0]) index++;
7      if (decisions[1]) index += 4;
8      if (decisions[2]) index += 2;
9
10     return root.enfants.at(index);
11 }

```

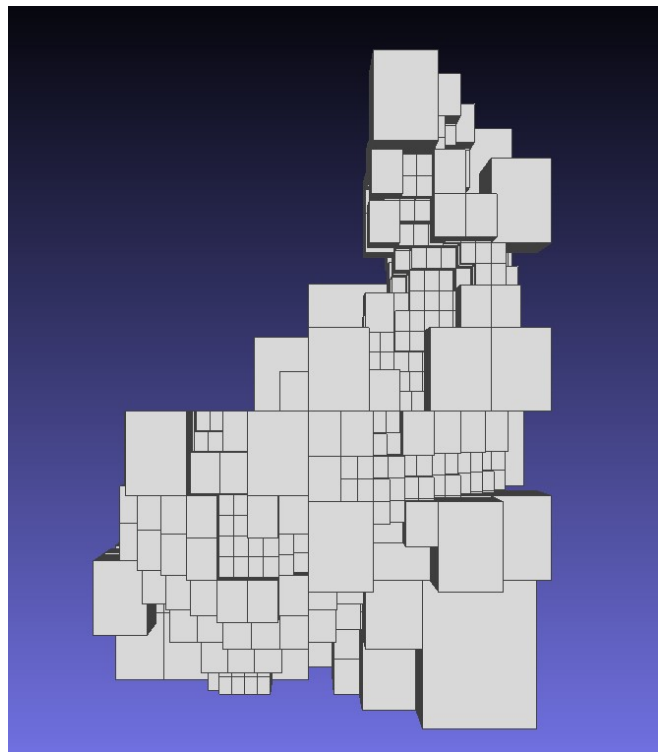
Partie 4 Visualisation de l'octree

Pour visualiser l'octree on peut simplement représenter les feuilles de l'arbre. Pour cela on fait un parcours en profondeur de l'octree. On récupère ainsi toutes les feuilles de l'octree et surtout leurs boîtes englobantes associées. L'étape suivante consiste à retrouver à partir des deux coins opposés de la boîte englobante les 8 sommets qui la composent, une fonction utilitaire effectue cela : `std::vector<Point3> getPointsOfAABB(AABB boundingBox)`. Il suffit ensuite d'écrire dans le fichier OFF résultat les 8 sommets de chaque boîte englobante ainsi que les 6 faces qui permettent de représenter la boîte englobante.

Cela nous permet d'afficher pour un maillage donné, l'ensemble des boîtes englobantes des feuilles de l'octree associé au maillage (Figure 1), et ainsi de visualiser notre octree.



(a) Mesh original



(b) Représentation de l'octree

Figure 1: Comparaison d'un maillage original avec la représentation des feuilles de son arbre

Cette représentation nous a permis de remarquer que la qualité de l'octree dépendait fortement de la profondeur max. et du nombre max. de points par feuille choisis. Par exemple, sur les images ci-dessus, le nombre max. de points est 35. Si nous utilisons ce même paramètre sur un maillage qui contient très peu de points, il risque de ne pas y avoir assez de subdivisions dans l'octree pour que cela soit utile.

Partie 5 Simplification du maillage

Pour simplifier le maillage, l'idée est de ne conserver qu'un seul sommet par feuille de l'octree. Toutefois pour pouvoir afficher un maillage simplifié il faut aussi savoir quelles étaient les faces du maillage original que l'on veut conserver. Il est important de noter que les faces sont soit modifiées et conservées soit supprimées du maillage. Afin de déterminer quelles sont les faces à conserver il faut donc parcourir les faces du maillage original et identifier quelles facettes ont des sommets répartis dans 3 ou 4 feuilles différentes, selon l'algorithme suivant :

```

for chaque face du maillage do
    On récupère les 3 points formant la face;
    for chaque point do
        | On récupère l'index du noeud de l'octree contenant ce même point;
    end
    if la face est un carré then
        On récupère le 4è point;
        On récupère l'index du noeud de l'octree contenant ce même point;
        if au moins 3 noeuds sont différents then
            | On ajoute la face (carré si 4, triangle si 3) aux faces à conserver;
        end
    end
    else
        if les 3 noeuds sont différents then
            | On ajoute la face (qui est un triangle) aux faces à conserver;
        end
    end
end

```

Algorithm 2: Algorithme d'identification des faces à conserver dans la simplification

Ainsi, seules les facettes ayant leurs 3 sommets dans 3 feuilles différentes seront conservées dans le maillage simplifié, puisque nous ne pouvons pas créer une face avec 2 sommets ou moins.

Ensuite, nous écrivons dans le fichier OFF le sommet barycentre des sommets de chaque feuille de l'arbre. Puis nous écrivons chacune des faces conservées.

Nous avons testé notre programme sur plusieurs maillages, et constaté qu'il ne fonctionnait finalement pas comme prévu. La figure 2 montre le résultat obtenu avec l'éléphant, qui... ne correspond pas du tout à un éléphant.

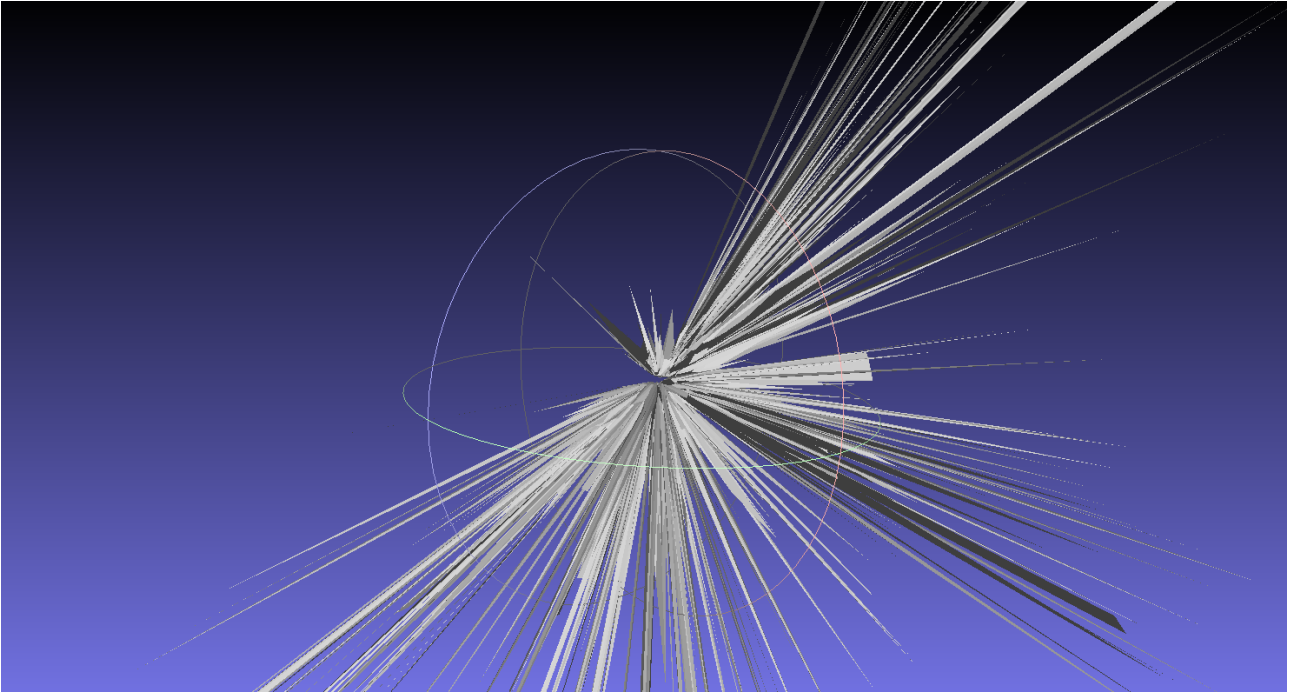


Figure 2: Exemple de simplification de maillage incorrecte

Les paramètres sont : $\text{MAX_POINTS} = 35$ et $\text{MAX_DEPTH} = 10$. Cela ne correspond pas du tout au résultat attendu. D'après nos recherches, il semblerait que pour chaque face nous ayons un ou plusieurs points incorrects (parmi les 3 ou 4 points constituant la face), mais nous n'avons pas trouvé exactement quelle partie du code posait problème.

Nous avons entre autres cherché à confirmer que cela ne venait pas de l'octree, qui serait lui-même incorrect. Pour tester cela, nous avons simplifié un maillage mais avec un nombre de points max. à 1, ce qui devrait donner exactement le même maillage que celui d'origine. la figure 3 illustre cela avec le champignon. On remarque que, mis à part les 3 trous (dont nous n'avons pas identifié l'origine), le maillage est exactement celui donné en entrée au programme. Les octrees générés semblent donc corrects (malgré les trous) et nous ne savons finalement pas pourquoi la simplification ne fonctionne pas.

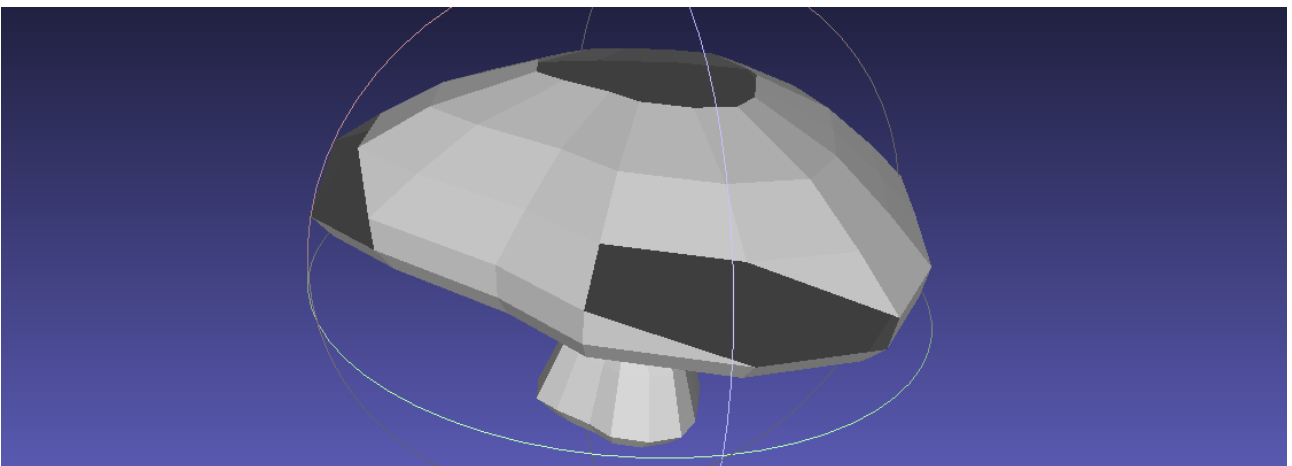


Figure 3: Exemple de simplification avec 1 point par feuille : retourne le maillage original