Projet de Géométrie Numérique

Romain PERRIN - romain.perrin@etu.unistra.fr
Maxime SEYER - maxime.seyer@etu.unistra.fr

Remarques générales

Le code a été développé en C++ et compilé avec QtCreator 5.9.x.
Le code ainsi que ce rapport ont été gérés à l'aide du gestionnaire de version **git** et le dépôt est disponible ici : http://github.com/Romain96/M1S2_GN.git

Deux bibliothèques ont été utilisées pour le projet et ont été directement incluses directement dans le projet qui ne nécessite de fait pas de dépendance externe. Les bibliothèques utilisées sont ***GLM*** pour la manipulation de vecteurs et de matrices et ***Eigen*** pour le calcul des vecteurs propres.

Description des classes du projet

Classe	Description
DisjointSets	Représente des ensembles disjoints utilisé pour la création d'un l'arbre couvrant minimal (algorithme de Kruskal)
Edge	Représente une arête dans le modèle *HalfEdge*
Face	Représente une face dans le modèle *HalfEdge*
Graph	Représente un graphe non orienté (et a fortiori un arbre)
HalfEdge	Représente une demi-arête dans le modèle *HalfEdge*
Mesh	Représente un modèle avec sommets, faces et demi-arêtes dans le modèle *HalfEdge*
MeshReconstructor	Effectue l'intégralité de l'algorithme de reconstruction de surfaces de Hoppe et al.
Node	Représente un nœud du graphe
Octree	Représente un octree (structure de données de type arbre dans laquelle chaque nœud compte exactement huit fils)
Plane	Représente un plan tangent et sa normale associée (contient les trois vecteurs propres du plan)
Vertex	Représente un sommet dans le modèle *HalfEdge*

Choix d'implémentation

Nous avons choisi de stocker les plans tangents ainsi que les normales dans une seule structure appelée Plane. Dans la mesure où l'analyse par composantes principales réalisée à l'aide de la bibliothèques ***Eigen*** fourni les trois vecteurs propres pour chaque centroïde et que deux de ces vecteurs propres représentent le plan tangent et le troisième représente la normale à ce plan, il est aisé de les stocker ensemble.

Afin de rendre la recherche de plus proches voisins plus optimale en temps, nous avons décidé de partitionner l'espace à l'aide de la structure d'***octree***. L'octree est utilisé une première fois lors du calcul des plans tangents par ACP. Cet octree est défini sur les points chargés par le programme. Pour la suite de l'algorithme, un second octree est généré mais cette fois-ci sur les centroïdes précédemment calculés. Il sert à la recherche des k plus proches voisins lors de la réorientation des plans tangents.

En réalité, c'est la classe ***MeshReconstructor*** qui fait le plus gros du travail de reconstruction. La classe ***HalfEdge*** fait partie du projet car celui-ci est basé sur le travail effectué en TP. De fait, la classe ***HalfEdge*** n'est jamais utilisé lors de la reconstruction. Les graphes sont simplement représentés comme deux listes : une liste de nœuds (contenant dans notre cas un pointeur sur un centroide) et une liste d'arêtes (contenant deux pointeurs sur des nœuds du graphe et un poids).

Pour réaliser l'algorithme des ***marching cubes***, nous avons utilisé une table de correspondance permettant de connaître la liste des triangles à former avec les arêtes associées en fonction des sommets de cube ayant une valeur négative (ie qui sont à l'intérieur de la surface). La table utilisée est celle décrite pas Paul Bourke (http://paulbourke.net/geometry/polygonise).

Difficultés rencontrées

L'algorithme est structuré en quatre étapes :

1) Calcul de centroïdes, plans tangents et normales associés
2) Réorientation des plans tangents
3) Définition d'une fonction de distance
4) Calcul d'une isosurface avec l'algorithme des Marching Cubes

L’étape 1 ne nous a pas posé de problème. Après codage de la structure d’octree, le calcul des centroides est simple et les plans tangents et normales sont donnés par Eigen.

L’étape 3 est une simple implémentation de la fonction de distance présentée dans l’article.

L’étape 4 était assez aisée après utilisation de la table de correspondance permettant d’obtenir directement la liste des triangles à créer. Il suffisait juste de traiter les triangles obtenus et d’ajouter les points (à créer) dans un liste et de créer des faces dans une autre liste. Enfin la structure Mesh permettait déjà de stocker et d’écrire le fichier final au format OFF.

L’étape 2 s’est révélée être la plus ardue et ce pour diverses raisons. La première raison est d’ordre théorique. En effet, l’article n’est pas totalement explicite du point de vue de la méthode à utiliser. La principale difficulté était de comprendre que les graphes/arbres ne représentent pas tous la même information (les poids codent pas la même chose). Nous avions initialement compris que les graphes/arbres codaient la relation de « proximité de parallélisme » des plans tangents. Ce n’est que plus tard que nous avons réalisé que les premiers graphes (le graphe initial est l’arbre couvrant minimal Euclidien ou EMST) représentaient en réalité l’information de distance des centroïdes tandis que les graphes suivants (le graphe Riemannien et son arbre couvrant minimal) représentait eux la relation de « proximité de parallélisme » des plans tangents.

La deuxième difficulté majeure de cette partie était la gestion de la mémoire notamment pour stocker les graphes. En effet, le graphe initial est un graphe de distance sur les centroïdes et a la particularité d’être entièrement connecté. Afin de réduire la quantité de mémoire disponible, nous avons choisi de ne créer que les arêtes dans un sens (puisque notre structure représente des graphes non orientés). Cependant cette structure rend le parcours de l’arbre couvrant minimal crée difficile et nous nous sommes trouvés dans la situation où le MST résultant ne permettait pas de parcourir tous les sommets (malgrès le fait qu’il comportent bien n-1 arêtes). De ce fait, la réorientation des plans tangents n’est pas effectuée. Les plans ne sont pas réorientés et le programme ne fournit pas un modèle correct en sortie.

Critique de l'algorithme

Comme dit précédemment, le principal défaut de cet algorithme est la quantité de mémoire disponible pour réorienter les plans.

Si l'on considère lors de la création du graphe totalement interconnecté que l'on ne crée que les arêtes dans un sens et que l'on élimine les nœuds produisant un cycle sur eux-mêmes, le nombre d'arêtes nécessaires au graphe est :

$$(n-1)+(n-2)+...+(n-(n-1))+(n-n)=\sum_{i=0}^{n-1}(n-i)=\frac{(n-1)*(n-2)}{2}$$

Le nombre d'arêtes nécessaires est de l'ordre de $\frac{n^2}{2}$ ce qui est conséquent même avec le modèle « block.off » utilisé pour nos tests. Le nombre d'arêtes est de l'ordre de 2,5 millions alors que le nombre de points du nuage n'est que de 2132.

Un autre défaut est que l'algorithme demande deux paramètres : un coefficient de densité et un coefficient de bruitage. De ces deux coefficients résulte la paramétrisation de la fonction de distance signée. En effet, celle-ci positionne les points dont la distance (non signée) est supérieure à la somme des deux coefficients. Fixer ces coefficients n'est pas du tout intuitif…

Avis sur le projet

Quoiqu'il en soit ce projet était très intéressant pour plusieurs raisons :

- C'est le seul projet que l'on nous ait donné à ce jour (TER exclu) qui demande d'implémenter un algorithme d'un papier de recherche.
- Le projet demande de comprendre un algorithme inconnu.
- Le projet est en rapport avec le cours/les TPs et avec la formation axée image.

Bien que certains points négatifs subsistent :

- Il n'est pas possible de visualiser le résultat sans avoir terminé correctement toutes les étapes
- Les paramètres de l'algorithme ne sont pas très intuitifs