Martin Jachimowicz
X96

Enseignement d'approfondissement :
Analyse et Synthèse d'Images

Simplification de Maillages

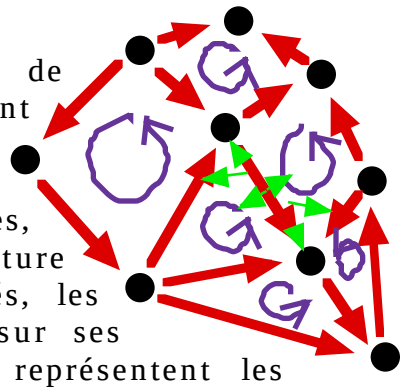
d'après les travaux de Hugues Hoppe.

Présentation :

La simplification de maillages, c'est-à-dire de représentations polyédriques de surfaces tridimensionnelles, est une opération consistant à chercher un maillage proche (nous préciserons plus tard ce terme) d'un maillage que l'on estime trop complexe, et donc coûteux en ressources mémoires et en temps de calcul ou de transmission.

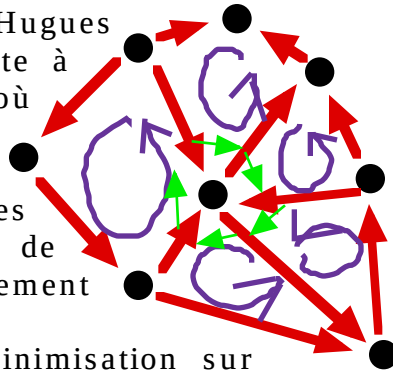
Elle peut s'avérer très utile dans plusieurs cas :

- transmission d'environnements virtuels via un réseau (pour réduire le temps de transfert)
- simplification d'objets générés par des scanners tridimensionnels à résolution fixe (pour réduire la taille de stockage)
- utilisations de plusieurs niveaux de détail d'un même objet en fonction des conditions d'observation, par exemple de la distance (pour réduire la charge de calcul).



Un maillage est représenté par un ensemble de points et un ensemble de polygones (généralement orientés) dont les sommets sont des points de l'ensemble précédent. Pour disposer d'informations utiles sur le voisinage de points et l'adjacence de faces, on introduit de manière redondante une structure supplémentaire, l'arête, qui "connait" ses extrémités, les polygones voisins, et deux autres arêtes donnant sur ses extrémités. Ci-contre en haut, les flèches vertes représentent les pointeurs contenus dans l'arête centrale. Les informations des arêtes permettent d'agir sur leur voisinage sans avoir a tester tous les points ou polygones (qui se comptent parfois par millions).

Méthode :



La méthode employée ici, mieux décrite par Hugues Hoppe dans son article “Progressive Meshes”, consiste à itérer une transformation, dite effondrement d’arête, où l’on fusionne les deux extrémités d’une arête en un point arbitrairement placé, entraînant la disparition d’un point et de jusqu’à deux arêtes et faces, si les faces voisines sont des triangles. La difficulté est de mettre à jour les informations d’adjacence, fortement perturbées par l’opération.

Le choix de l’arête à effondrer résulte de la minimisation sur l’ensemble des arêtes d’une fonction d’énergie que l’on se donne, et qui va servir à caractériser la fidélité du maillage simplifié au maillage initial. On peut également minimiser cette fonction sur l’espace des positions acceptables du point destination, ce qui conduit a de meilleures simplifications.

Maillages Progressifs :

L’utilisation de cette méthode permet également, l’effondrement d’arête étant réversible si l’on dispose des informations nécessaires, un codage progressif des maillages. En effet, en partant d’un maillage simplifié et en appliquant la “fission” réciproque du dernier effondrement opéré, affiner progressivement le maillage. Ce point est particulièrement intéressant pour la transmission par réseau, permettant au récepteur de visualiser l’objet avant qu’il ne soit entièrement reçu.

Choix personnels

J’ai choisi en entamant ce projet le langage de programmation C++, car un langage orienté objet me semblait bien adapté aux types de données que j’avais à décrire et manipuler. En outre, la très grande popularité du C++ et sa modularité, liée à l’héritage, m’ont permis d’avoir recours à un cadre d’application pour gérer les ressources système. J’ai également décidé de programmer sous MacOS, par facilité personnelle.

Enfin, j’ai retenu comme format de fichier d’entrées/sorties le format VRML (spécifiquement 1.0C), avec l’avantage de pouvoir utiliser un navigateur pour visualiser les résultats.

Structures de données

les principales structures de données utilisées reflètent l’organisation “naturelle” des différents éléments.

La classe Scene (Objects.h) contient un ensemble d’Objets. Un Objet (Objects.h) contient un repère et une collection de points.

La classe Object a un descendant utile : la classe Polyedron (Polyedron.h), qui contient en sus une collection de _3DPolygon. Un _3DPolygon (_3DPolygon.h) est essentiellement composé d’un tableau de pointeurs sur ses sommets, qui doivent appartenir à son Polyedron racine.

La classe Polyedron a elle-même un descendant : la classe Mesh (Mesh.h) qui contient en outre un tableau d’Edges et des informations liées à

l'effondrement. un Edge (Edges.h) contient essentiellement des pointeurs sur ses extrémités, ses faces voisines et sur une arête à chaque extrémité.

Compte tenu du grand nombre de suppressions de points, facettes et arêtes, il aurait peut-être été plus judicieux de stocker ces objets sous forme, non pas de tableau, mais de tableau de pointeurs.

Les principales fonctions

La création des arêtes (Mesh::make_edges) :

Pour créer les arêtes de manière efficace, on parcoure successivement les faces du Polyedron considéré, et pour chaque côté on vérifie s'il existe déjà une arête entre les deux points considérés. Si ce n'est pas le cas on la crée, sinon on complète cette arête en lui apportant les informations manquantes. Chaque arête est, en pratique, créée en deux passages.

Cette procédure ne marchant qu'avec les surfaces orientées (c'est-à-dire où tous les polygones ont la même orientation), je lui ai fait une petite soeur (Mesh::make_oriented_edges) qui vérifie si la surface est orientable, l'oriente et crée les arêtes.

L'effondrement d'arête (Mesh::collapse)

Cette fonction procède en plusieurs étapes : supprimer le point arrivée de l'arête à effondrer, supprimer l'arête, corriger les faces adjacentes (dans le cas d'un triangle, cela conduit à le supprimer, ainsi que l'une des deux autres arêtes), puis corriger les faces aux points éventuellement non coplanaires. La difficulté essentielle est la bonne gestion des informations d'adjacence, c'est-à dire la sauvegarde, avant chaque étape, des informations utiles à sa réalisation, et la mise à jour des informations compte tenu des modifications apportées.

L'évaluation de la fonction d'énergie (Mesh::compute_collapse_energy) :

Pour évalue de manière pas trop coûteuse l'énergie d'effondrement d'une arête, on extrait du maillage initial un sous-maillage composé des facettes touchant l'une ou l'autre des extrémités de l'arête dont on veut évaluer l'énergie d'effondrement. On duplique et simplifie ensuite ce sous-maillage.

La fonction d'énergie se compose de deux contributions distinctes. La première est égale à la somme des carrés des distances d'un certain nombre (déterminé par le paramètre order) de points du sous-maillage simplifié au sous-maillage initial. Elle est destinée à pénaliser les maillages s'éloignant trop du maillage initial. La seconde contribution est la différence entre l'énergie moyenne de ressort entre les points du sous-maillage simplifié et celle du sous-maillage initial. Elle est destinée à assurer une densité spatiale de points la plus uniforme possible.

Un point particulièrement sensible est la pondération des différentes contributions. J'ai décidé d'utiliser une pondération dépendant du nombre de points dans le voisinage, favorisant la première contribution lorsque le nombre de points est élevé.

Le choix de l' arête à effondrer (Mesh::find_which) :

Pour déterminer l'arête la plus intéressante à effondrer, on évalue au départ l'énergie d'effondrement de chaque arête. Il suffit ensuite d'effondrer celle qui réalise le minimum., puis de recalculer les énergies des arêtes touchant celle qui vient d'être effondrée, et de réitérer.

La lecture des fichiers VRML (SimpFileStream::parse)

Le format VRML étant plus riche que mon format interne de travail, je ne m'intéresse dans ce format qu'aux objets de type IndexedFaceSet, et aux objets de type Coordinate3 dont ils dépendent. La fonction trouve et lit dans un premier temps les ensembles de points, puis trouve les définitions des faces, remonte la hiérarchie jusqu'à trouver l'ensemble de points sur lesquels elle sont définies, puis crée le Mesh correspondant. Cette fonction ne gère pas encore les noeuds DEF et USE du format VRML, rendant certains fichiers illisibles.

L'écriture des fichiers VRML (SimpDoc::DoSave)

La grande majorité du fichier initial n'étant pas reconnue à la lecture, cette fonction recopie ce fichier jusqu'à arriver à un sommet IndexedFaceSet, à partir d'où elle insère un séparateur contenant les points et les facettes correspondantes avant de recopier le fichier initial de la fin du noeud au début du IndexedFaceSet suivant, et ainsi de suite jusqu'à la fin du fichier. Ceci conduit malheureusement à des fichiers redondants, parce que la fonction ne peut effacer les points originaux sous peine de rendre invalide tout noeud ultérieur de type IndexedLineSet ou PointSet.

Cette fonction ne marche pas parfaitement à l'heure actuelle, et les fichiers écrits sont parfois illisibles.

Non-Résutats :

Je n'ai malheureusement pas de résultats concrets à présenter, des problèmes de mémoire limitant le nombre de simplifications à une dizaine au plus. Spécifiquement, des erreurs de type "unmapped memory" se produisent lorsque j'essaie d'allouer de l'espace à l'aide de la fonction new. Je pense que ceci est du à la présence en mémoire d'objets alloués par new et non correctement détruits, mais je ne parviens pas à déterminer exactement l'origine du problème.