Mini rapport sur les TPs Synthèse d'Image – IGR202

Modélisation géométrique

TP1 : Geometry Filtering

Dans une nouvelle méthode void laplacienFilter(), on implémente le principe décrit dans le TP. On calcule d'abord les barycentres de chaque 1-voisinage de chaque sommet, on bouge les dits sommets en ces points barycentriques, puis on recalcule les normales de ces nouveaux sommets pour obtenir un résultat satisfaisant.

Dans cette même fonction, on ajoute un paramètre : bool cotangentWeights, qui active, s'il est true, le filtrage laplacien *géométrique* , qui prend en compte avec des « poids » (weights), en fonction des angles formés autour de, pour pondérer le nouveau sommet ainsi que sa normale.

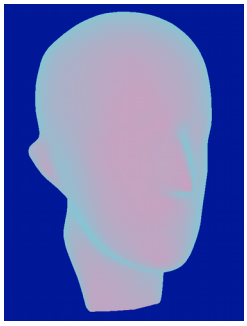
On obtient avec les 2 méthodes des résultats assez similaires, mais si cotangentWeights == true, on obtient normalement un résultat un peu meilleur, car un peu plus précis à chaque sommet.

On ajoute un second paramètre alpha dans la méthode qui permet de pondérer le résultat d'un coefficient alpha, pour moduler le déplacement de chaque sommet du au filtrage.

On ajoute aussi des shortCut Keyboard : L permet d'appliquer le filtrage Laplacien avec les paramètres définis dans le main de la fonction, 1 permet d'appliquer le filtrage avec un alpha de 0.1, 2 d'un alpha de 0.5, et 3 d'un alpha de 1.0.

F2 permet de recharger le modèle initial pour changer les paramètres.

On obtient ce genre de résultats avec le modèle initial « max_50K.off » :



Les résultats sont du même type avec cotangentWeights true ou non.

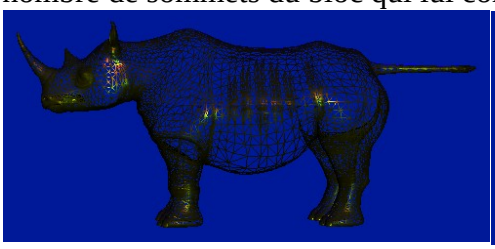
J'avais d'abord implémenté une fonction trop longue dans ses boucles pour avoir un résultat viable, que j'ai mis en commentaire en début de méthode, j'ai donc réalisé la fonction en me plaçant dans chaque triangle directement.

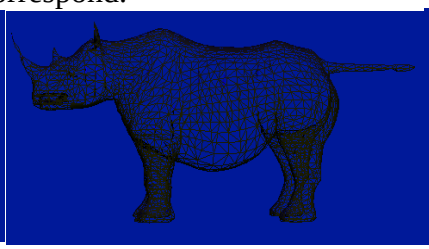
Lorsque j'init la scène dans le main, je calcule les poids cotangents initiaux (qui ne change alors plus) dans cotW, en faisant appel à computeCotangentWeights de Mesh.cpp (grâce à un std::vector de map). Avec ces derniers je peux donc calculer le laplacien géométrique.

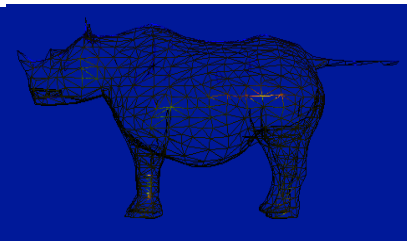
TP2 : Simplification

Pour la simplification, il s'agissait tout d'abord d'implémenter une fonction void Mesh::simplify (unsigned int resolution).

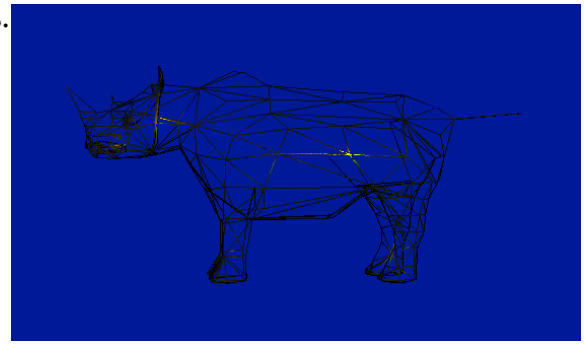
Dans celle là, je simplifie par clustering, en définissant une boite englobante autour de mon modèle. A partir de là, je crée une grille G, de résolution définie en appel de la fonction (paramètre entier). Puis je fais tel que décrit dans le TP : pour chaque sommet j'ajoute sa position dans le bloc de la grille correspondant, et je compte le nombre de sommet que j'ajoute dans chaque bloc en parallèle. Ensuite, j'élimine les triangles s'ils ont tous leurs sommets dans le même bloc, et ne garde que les triangle dont les sommets ne sont pas dans la même zone, en les déplaçant chacun au barycentre de chaque bloc. Enfin, on normalise chaque bloc en divisant la position de chaque représentant par le nombre de sommets du bloc qui lui correspond.







Par la suite, on réalise l'adaptative simplification, qui correspond à une grille (graphe) d'Octree qui donc s'adapte en fonction de présence ou non de sommet en son sein. Pour réaliser cette fonction on prend l'exemple donné dans la slide 12 du cours sur les simplifications, et on implémente de façon récursive la fonction buildOctree. Mais il faut aussi des objets de type OctreeNode, c'est pourquoi on inclus dans notre fonction adaptativeSimplify(unsigned int numOfPerLeafVertices) un Struct, qui permet de définir une classe de type OctreeNode, qui contient la fonction buildOctree. C'est en sortant de ce struct qu'on appelle alors buildOctree.
Dans mon cas, j'ai un souci d'implémentation que je n'ai pas réussi à résoudre : il semblerait que peu importe la valeur de mon paramètre unsigned int numOfPerLeafVertices cela me donne le même résultat, peu importe la version de l'Octree que j'utilise (j'ai pu comparer avec des codes de mes camarades qui fonctionnent chez eux). J'ai actuellement dans mon fichier ligne pour ligne le code de Julien Hage, et pourtant cela renvoie toujours le même résultat : un problème subsiste ailleurs mais je n'ai pas trouvé d'où il venait malgré tous mes efforts. J'ai le même rendu (exactement) avec d'autres codes que je sais correctes. Malheureusement, ayant passé énormément de temps à chercher le problème en essayant différentes méthodes, en codant moi même différentes versions, la meilleure solution m'a semblé être d'être tout à fait honnête dans mon rapport. Il semble que numOfPerLeafVertices soit initialisé ou explicite à une valeur quelque part, mais je n'ai pas trouvé où. J'ai tout de même un résultat de ce type :



Les shortCuts sont S pour simplify et Q pour adaptativeSimplify. J'ai ajouté un booléen qui ne permet d'appliquer les transformations qu'une seule fois à chaque fois (si l'on veut voir les différents rendus il faut changer à la main dans le main les valeurs des fonctions). F2 permet de recharger le modèle original.

TP3 : Subdivision

Pour cette partie, j'ai suivi le papier fourni en annexe, et ai crée une méthode void Mesh ::subdivide(), dans laquelle j'utilise une map afin de trouver les arrêtes en commun entre les triangles, puis un vector de map afin de stocker les oddVertex (calculés en fonction de ces arrêtes), dans les identifiants correspondants aux bons sommets. Par la suite, je recalcule les evenVertex sachant tous les odd autour de ces derniers, et puis je les ajoute tous dans le bon sens (ordre trigonométrique), dans mes m_vertexPositions et mes m_triangleIndices, puis je recalcule les normales.
Malheureusement, j'ai essayé de debugger mon code pendant 5 jours jusqu'à aujourd'hui, car mon code compilait en entier, mais lorsque j'appuyais sur P (mon shortCut), un abort apparaissait parfois (ou segmentation fault) (elle avait lieu une fois sur 2). Lorsqu'elle n'apparaissait pas, le modèle disparaissait de la fenêtre. Même avec des std::cout / endl, que j'utilisais pour voir là où le abort avait lieu, je sortais de la fonction et la segmentation fault avait lieu après. Je n'avais donc pas moyen de voir d'où venait l'erreur. Après plusieurs modifications différentes, j'ai réussi à décaler cette segmentation fault dans ma fonction computeOdd, et j'ai réglé le problème qui était du à mon indice i+1 qui sortait de liste et qui donc n'avait pas de vecteur équivalent dans ma liste m_vertexPosition. J'ai donc seulement ce matin (mercredi jour de rendu du TP), réussi à visualiser ma subdivision. Elle n'est pas encore correcte, car je déplace les bons sommets au bon endroit mais j'ai un souci dans mes indices de triangles ce qui fait que je les relie mal. Malheureusement je ne vois pas l'erreur, et je n'ai pas le temps de refaire tout le code dans le temps imparti (car j'ai cours jusqu'à 18h30 cet après midi).