Informatique graphique

Baptiste GENEST
MIF02

November 11, 2022

Projet à compiler avec QtCreator par le .pro, en mode Release pour de meilleures performances. Pour que les imports de fichier fonctionnent, il faut que l'éxécutable soit lancé depuis un dossier de build placé à côté du dossier "ComputerGraphicsM1_GENEST"

1 Tronc Commun

1.1 Mathématiques

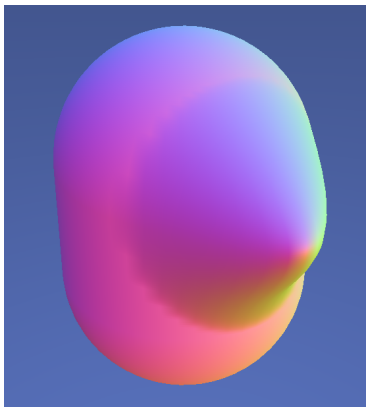
1.1.1 Manifolds

Avant d'être discrétisées, la plupart des primitives géométriques communes sont décrites par une classe héritant de **Manifold2D**. Cette classe abstraite définit ce qui est nécessaire pour discrétiser une variété lisse, i.e., une paramétrisation lisse de la surface ainsi que des normales. **Manifold2D** décrit également le type homotopique de chacun des deux paramètres de la paramétrisation: par exemple, les **HomotopyTypeParameters** d'un cylindre paramétré par $(cos(2\pi u), sin(2\pi u), v)$ sont $\mathbb{S}^1$ et **line** car la paramétrisation est périodique sur la première coordonnée et pas sur la deuxième. Cette donnée permet l'identification des bords de la surface (voir TopologicalMesh). L'intérêt de cette couche d'abstraction est qu'il n'y a qu'un seul algorithme de discrétisation de maillage[1] pour les classes en héritant. Pour les objets qui ne sont pas des variétés lisses (paralélogramme et disque par exemple) des fonctions de discrétisations spécifiques sont implémentées (souvent simples de par leur structure).

1.1.2 Déformations locales

Un ensemble de fonctions noyaux est également implémenté pour permettre plus de liberté pour les déformations lisses par des mécanismes de programmation fonctionnelle. Par exemple, la fonction **SphereWarping** prend en paramètres la sphère associée à la déformation, la direction, et le noyau qui donne l'intensité de la déformation en fonction de la distance au centre.

Enfin, chaque déformation donne également sa jacobienne[2] en chaque point de l'espace pour transformer les normales du maillages de manière adéquate par $\mathbf{n}' = (\mathbf{J_x}^T)^{-1}\mathbf{n}$



[1] sauf pour la sphère et l'hémisphère car les types homotopiques de leurs paramétrisations ne sont pas des produits de types 1D, l'algorithme peut s'appliquer à eux mais les poles seront dupliqués. L'appel est alors surchargé pour produire une discrétisation sans duplications des poles.

[2] par défaut, on fournit une approximation par différences finies de la jacobienne

1.2 Structure du code

1.2.1 Hiérarchie de classes Mesh et généricité

D'une manière générale, la structure du code a été pensé d'abord pour la généricité et l'abstraction (majoritairement par le polymorphisme) plus que pour la performance. La génération et l'interaction avec un maillage se fait par 3 couches de rafinement: La classe Mesh telle que fournie dans le code initiale est principalement vouée à être affichée, avec peu de processing directe, à l'exception de l'**application de déformations** (voir section dédiée) et de la **concaténation de maillages**.

La classe TopologicalMesh ajoute à la classe Mesh des informations topologiques utiles à certaines opérations. Elle ajoute la donnée des arrêtes et des faces, stockées de manière uniques, ainsi que la liste des chaines de vertex formant les bords du maillage. Les arrêtes et les faces servent par exemple à la **subdivision de maillages** (qui sert elle même par exemple à générer une icosphère). La donnée des bords du maillages sert à fusionner deux **TopologicalMesh**, en spécifiant une chaine de vertex d'un bord qu'il ont en commun (et supprime donc les sommets en double) et permet une modélisation plus complexe par association (par exemple une capsule en collant un cylindre et deux hémisphère).

Enfin, la classe **ManifoldMesh** est principalement chargée de discrétiser les **Manifold2D**. On exploite le caractère lisse des surfaces par le fait que chaque point à une unique normale associée ce qui simplifie la construction d'un maillage par rapport à **Mesh**.

2 Approfondissements

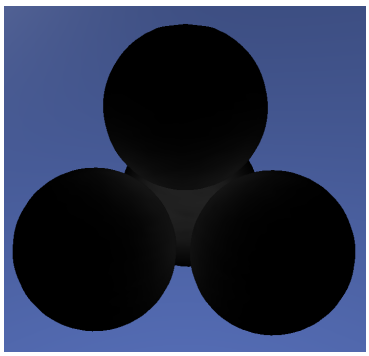
2.1 Ambiant Occlusion

2.1.1 GeometricQueryInterface

Afin de calculer des intersections de rayons avec la scène une grande partie des objets vouée à la composer hérite de l'interface **GeometricQueryInterface** qui vise à spécifier l'ensemble des query géométriques qu'il peut-être nécessaire de réaliser: comme justement l'intersection avec un rayon, un test d'appartenance volumétrique, la distance avec la structure, le point moyen etc... Ainsi, l'algorithme naif d'intersection loop sur un emsemble de shared ptr de **GeometricQueryInterace**[3] et prend l'intersection la plus proche. A noter que cet héritage accélère en soit considérablement le raytracing car par exemple la sphère hérite de **Manifold2D** et de **GeometricQueryInterface** afin de pouvoir maintenir la dualité entre la discrétisation de l'objet qui sera affichée et sa représentation abstraite dans la scène avec laquelle il est trivial et rapide de calculer l'intersection avec un rayon. A noter également que la classe Mesh spécifie aussi GQI, il est donc possible de calculer de l'AO sur n'importe quel maillage.

2.1.2 Intégration Quasi-MonteCarlo

Pour approximer l'intégrale d'AO on a d'abord employé une **RandomLookupTable** de tirages sur l'hémisphère canonique (ensuite tournés pour s'adapter aux normales en chaque point) mais le bruit de la variance de l'intégration MonteCarlo étant disgracieux entre deux points voisins, il a été choisi d'employer une méthode de QuasiMonteCarlo en fixant les tirages par la suite de fibonnaci sur la sphère, méthode qui assure une cohérence locale.



[3] on passe par des pointeurs pour bénéficier du polymorphisme

2.2 Diffusion Limited Aggregation

La génération d'une structure de DLA a été accélérée par deux idées:

2.2.1 Amélioration de l'algorithme par sphere walking

Une remarque importante, inspirée de l'algorithme de Sphere Walking pour le MonteCarloGeomtryProcessing, est qu'à chaque instant, si on regarde la sphère centré sur le point courant et la structure actuelle, je sais qu'aucune intersection n'aura lieu en son sein, et comme on simule un mouvement brownien, ce dernier ayant la même probabilité de sortie partout sur la sphère, on remplace le déplacement à pas fixe par la suite de tirages aléatoires:

$$x_{n+1}^{(i)} \sim \mathcal{U}(x_n^{(i)} + \mathbb{S}^2 d(x_n^{(i)}, \Gamma_n))$$
$$x_{N_i}^{(i)}, \text{ si } d(x_{N_i-1}^{(i)}, \Gamma_n) < \epsilon$$

où

$$\Gamma_n = \bigcup_i x_{N_i}^{(i)} + \mathbb{S}^2$$

2.2.2 Structures accélératrices

Pour accélérer la requête de distance à la structure, omniprésentes dans l'approche présentée au-dessus, il a été choisi d'utiliser un octree, qui permet à la fois d'ajouter à chaque itération la nouvelle sphère dans la structure sans avoir à recalculer, et qui permet une implémentation simple de la recherche de plus proche voisin. L'interface de structure accélératrice présente que ces dernières sont vouées à contenir des **GQI**, tout en en étant elles même une, (les querys géométriques sont alors dépendantes des objets contenus), on peut donc évisager des entremellements de structures.

2.2.3 Rayon extérieur d'apparition

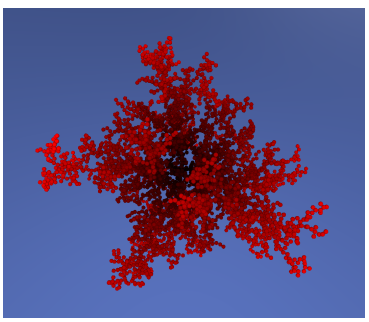
On initialise la position initiale de la particule courante par un tirage aléatoire sur une sphère centrée d'un rayon proportionnel à $(1+k)\max_{x\in\Gamma_n}||x||$ pour garantir l'abscence de collision à l'initialisation et le paramètre k ($k = 0.2$ dans l'implémentation), controle la probabilité de contact avec une sphère éloignée de son point initial. A noter qu'une étude empirique indique que $\mathbb{E}(\max_{x\in\Gamma_n}||x||) = \mathcal{O}(\sqrt{n})$ mais ne permet pas de garantie contrairement au suivie du diamètre de la structure.

2.2.4 Présence d'obstacles

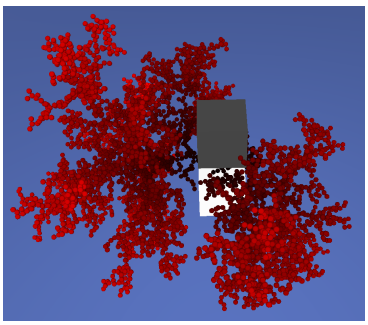
l'intégration d'obstacle dans la structure par l'approche de sphere walking est simple,
on prend $\min(d(x, \Gamma_n), d(x, \text{Obstacles}))$ pour la taille du pas.

2.2.5 Coloration par distance topologique au noyau

Pour colorier la structure on garde la trace des collisions entre les sphères pour retracer l'historique de l'aggrégation. La couleur de chaque sphère est donc proportionnelle au nombre de sphères mimimum qu'il faut parcourir pour revenir au noyau originel.



(a) DLA sans obstacle, (5000 sphères en 4 sec)



(b) DLA avec obstacles, (5000 sphères en 13 sec)

2.3 Terrain editing and local characteristics

2.3.1 Interpolation bilinéaire et bicubique

La classe **Interpolator2D** est une classe abstraite qui représente un ensemble de valeur sur une grille discrète 2D on fait ensuite le mapping par héritage par interpolation bilinéaire ou bicubique $[0,1]^2$ et les valeurs de la grille. Il est possible de charger une image au format PGM P2 sur la grille.

2.3.2 Classe Terrain

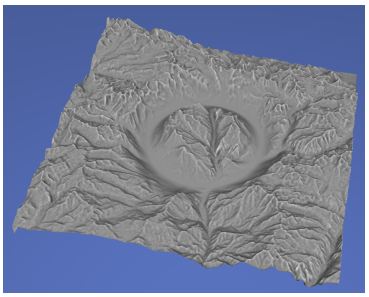
La classe **Terrain**, hérite de **BicubicInterpolator**, de **Manifold2D** et de **ImplicitVolume** tout en ajoutant les méthodes dédiées aux traitement de terrains.

2.3.3 Fonction de terrassement

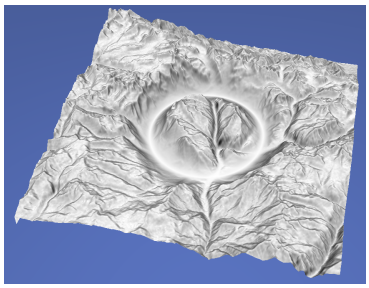
On spécifie une fonction de distance à une primitive abstraite, le terrassement fait l'interpolation entre la valeur du tableau et une hauteur donnée,on controle le profil de la déformation par une fonction de noyau. La fonction de terrassement n'est pas considérée comme une déformation car elle agit directement sur les valeurs du tableau de la heightmap.

2.3.4 Coloration par caractéristiques différentielles locales et AO

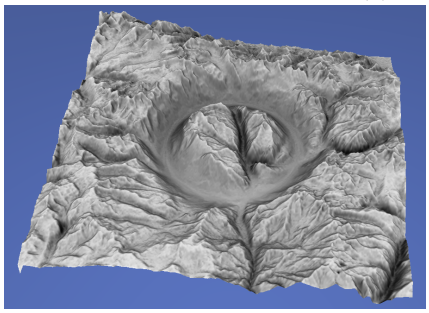
La slope du terrain est calculée par $||\nabla h||$, le laplacien et le gradient sont calculés par différences finies classiques. Ces opérateurs sont appliqués à chaque noeud de la grille puis stockés dans des **Interpolator2D** pour faciliter le plaquage de la coloration sur le maillage. Pour le calcul de l'ambient occlusion, on passe par du raymarching par la sdf[4]: $(\mathbf{x}_z - h(\mathbf{x}_{xy}))/\lambda$, où, $\lambda = ||\nabla h||_{L^\infty}$, la constante de lipschitz du terrain.



(a) slope



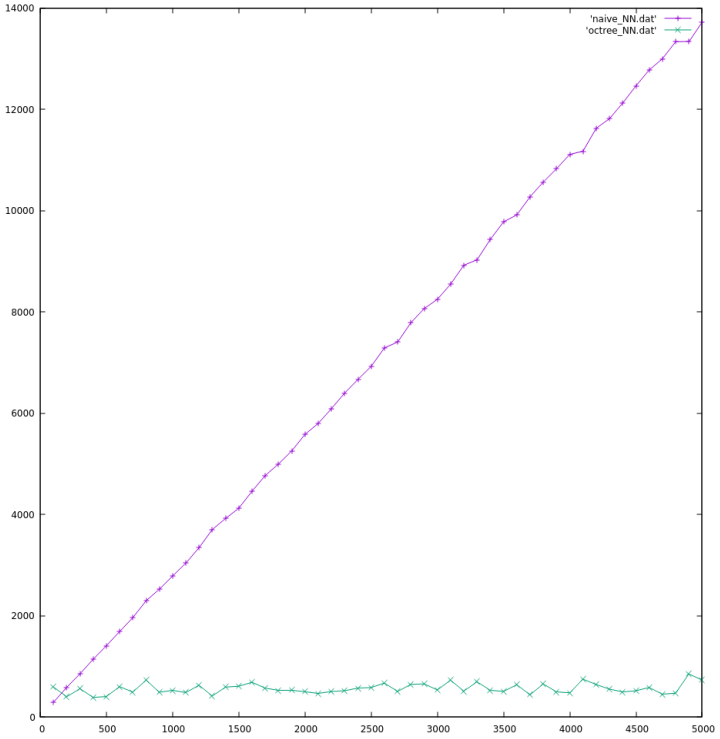
(b) laplacien



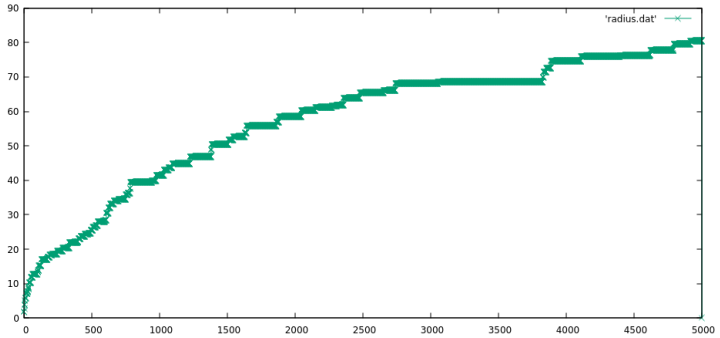
(c) AO

3 Annexe

[4]La classe **ImplicitVolume** spéçialise la class **GeometricQueryInterface** pour les primitives ayant une sdf connue.



(a) comparaison complexité structure accélératrice (naive vs octree)



(a) évolution du diamètre d'un DLA