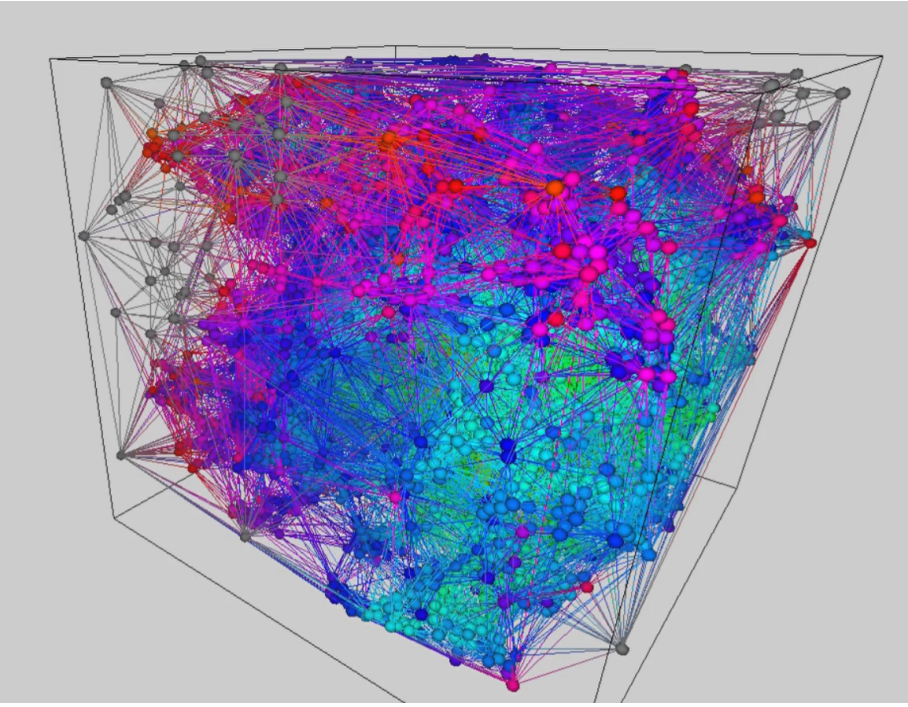
Rapport de stage: Recherche de voisin par tétraédralisation de Delaunay

Hugo Bec, Claire Morin

Juillet 2022



Introduction

Mise en situation

Ce projet a consisté à expérimenter des méthodes de recherche des points les plus proches en utilisant la tétraédralisation de Delaunay. Cette tétraédralisation consiste à lier par des tétraèdres les particules de façon à ce que lorsque l'on trace la sphère circonscrite d'un tétraèdre, aucune autre particule ne soit comprise dedans. Cette méthode permet de lier directement chaque particule avec leurs voisins les plus proches.

Pour tester nos algorithmes, nous avons créé deux simulations :

- Une simulation de mouvement pseudo-brownien de particules en 3D avec une agrégation par diffusion limité.
- Une simulation de boids en 3D.

Les objets sont initialisés dans un espace fini selon une distribution uniforme et ne peuvent pas sortir de cet espace. Si une particule touche un bord de l'espace, elle rebondit sur la paroi. Si un Boid sort de l'espace, sa vélocité se redirige lentement vers l'intérieur pour garder un mouvement fluide.

L'idée va être de chercher pour chacun des points, l'ensemble des autres points ayant une distance inférieure à un rayon donné en utilisant une tétraédralisation de Delaunay.

Ces expérimentations permettent de tester leur efficacité concrète séquentiellement et de les rendre parallélisables pour une future adaptation sur GPU.

Paramètres

Général

- DSTRUCTURE_VERBOSE: Active/Désactive les détails d'exécution dans le terminal
- TYPE_SIMULATION: Choisit entre le mode Aggregation par Diffusion Limité (0) ou le mode Boids (1)
- NB_POINT: Choisit le nombre de Particules / Boids
- SPEED_POINTS: Choisit la vitesse des Particules / Boids
- ATTRACT_RADIUS: Choisit leur rayon d'attraction, pour le DLA il est préférable de l'initialiser à $2 \times SIZE\_OBJECTS$, pour le Boids détermine son rayon d'influence
- SIZE_OBJECTS: Choisit la taille du Mesh 3D
- TETRA_REFRESH_RATE: Détermine le nombre de frames avant de recalculer la tétraédralisation de Delaunay
- CAGE_DIM: Détermine la dimension de la cage
- NB_INIT_FIXED_POINTS: Choisit le nombre de point fixe au départ (pour les Boids doit être mis à 0)

Boids

- BOIDS_SEPARATION_DISTANCE: Détermine la distance moyenne entre chaque Boid
- BOIDS_BOX_MARGIN: Détermine à partir de quelle distance du bord les Boids initient un virage pour retourner dans la cage
- BOIDS_COHESION_FACTOR : Détermine le facteur de cohésion ajouté à chaque frame
- BOIDS_ALIGNMENT_FACTOR: Détermine le facteur d'alignement des Boids à chaque frame
- BOIDS_SEPARATION_FACTOR: Détermine le facteur de séparation des Boids à chaque frame
- BOIDS_TURN_FACTOR: Détermine le facteur de virage au bord de la cage qu'un Boid effectue à chaque frame.

Les librairies

Pour ce projet, nous avons utilisé les librairies :

- `tetgen` pour calculer la tétraédralisation de Delaunay;
- `OpenGL` pour représenter l'espace fini, la tétraédralisation et les particules en 3D en temps réel;
- `Assimp` pour lire les fichiers .mesh;
- `ImGUI` pour l'interface graphique.

L'ensemble est exécutable depuis Visual Studio 2022.

1 Démarches

1.1 Tétraédralisation

Pour réaliser la tétraédralisation en trois dimensions, nous avons utilisé la librairie `tetgen` permettant, à partir d'un nuage de points de générer une tétraédralisation de Delaunay sous le format `tetgenio`. L'idée est de lire cette structure de données afin de créer une visualisation (Voir partie 1.2 Visualisation) mais aussi d'opérer des optimisations sur la recherche de points les plus proches. Pour se faire, nous avons créé une classe `DelaunayStructure`. Dans cette classe nous génèrons la tetraédralisation de Delaunay en appelant la librairie `tetgen`, nous l'affichons, puis nous cherchons les points les plus proches.

1.2 Visualisation

Avant de développer des solutions d'optimisations, nous souhaitions pouvoir visualiser les solutions de façon intuitive et en temps réel. Pour se faire nous nous sommes basés sur un squelette de code `OpenGL` de M.MARIA que nous avons adapté pour pouvoir afficher les particules, leur tétraédralisation ainsi que l'espace fini en trois dimensions. Plusieurs modes de visualisation sont disponibles avec l'interface `ImGUI` ou grâce aux touches du clavier.

- ZQSDRF + souris: caméra
- +/- : visualiser le point suivant/précédent ainsi que ses points attractifs (dépend du rayon d'attraction)
- E : afficher les arêtes associées de la tétraédralisation de Delaunay
- T : afficher l'ensemble des points de la simulation
- P : pause/joue la simulation
- O : affiche uniquement la frame suivante
- M : change entre la visualisation des points attractifs et la visualisation des points fixes
- B: active/désactive la visualisation 3D des points (dépend de la taille d'un point)

Nous avons ajouté une visualisation sphérique aux particules afin de mieux visualiser l'agrégation par diffusion limité. Pour ce faire nous avons utilisé un mesh d'iso-sphère (de précision 1, 2 et 3 généré sur Blender) que nous chargeons dans le programme pour l'afficher avec `OpenGL`. L'idée a été d'ensuite effectuer une copie de la structure de mesh pour chaque particule pour éviter de lire le fichier plusieurs fois. Nous souhaitions au départ utiliser la technologie d'instanciation d'OpenGL pour gagner en performance d'affichage mais nous n'avons pas pu la mettre en place par manque de temps.

2 Comparaison des versions

2.1 Fonction compute_attract_by_double_radius

Connaissant la vitesse maximale des particules, il est possible de déduire la distance maximale qu'une particule peut parcourir sur un nombre N de frames avec la formule suivante: $speed \times N \times 2$, fois deux car nous considérons que les deux particules s'éloignent en direction opposée, cela représente une zone finie que l'on nomme le rayon de Verlet. L'idée est alors de recalculer une tétraédralisation uniquement toutes les N frames au lieu de toutes les frames, et de calculer le rayon de Verlet ainsi que toutes les particules comprises dans ce rayon pour chaque particule. L'idée va être ensuite de calculer les particules attractives des N frames suivantes en utilisant uniquement l'ensemble des particules comprises dans le rayon de Verlet.

Ce procédé évite de recalculer la tétraédralisation toutes les frames et permet, pour une vitesse de particule et un nombre N de frames raisonnable, de gagner en performance. Un inconvénient cependant est qu'il crée des "freezes" (des ralentissements sur une seule frame) à intervalles réguliers (au moment de recalculer la tétraédralisation ainsi que pour parcourir le graphe et enregistrer l'ensemble des particules comprises dans le rayon de Verlet courant).

En réalité, parcourir une liste de N particules prend moins de temps que de parcourir N particules dans un graphe. Le parcours Breadth-First Search a une complexité plus grande que le simple parcours de tableau.

2.2 Fonction compute_attract_by_flooding

Une autre méthode a été expérimentée, elle consiste, à chaques frames, à parcourir le graphe de tétraédralisation sans pour autant la recalculer. L'idée est de faire un parcours en largeur du graphe partant d'un point jusqu'à une profondeur fixée. La même problématique revient par rapport au coût de parcours du BFS et cette méthode s'avère plus lente que la précédente.

Plus la tétraédralisation est ancienne, plus il est nécessaire de parcourir l'arbre de plus en plus en profondeur pour être sûr de récupérer tous les points attractifs. Il peut arriver que si on ne parcourt pas le graphe assez loin certains points ne soient pas trouvés. Ainsi pour s'assurer de récupérer tous les points attractifs, il est nécessaire de choisir une profondeur assez grande et donc coûteuse en temps de calcul (dû au BFS), surtout quand la tétraédralisation n'a pas été mise à jour depuis longtemps.

Nous pouvons constater qu'avec un rayon d'attraction et un refresh frame faible la méthode double rayons est plus performante que celle du parcours en profondeur.

Graphiques

Pour ce graphique nous avons 10 000 points avec une vitesse de 0,1. Le calcul de temps est fait sur 20 frames.

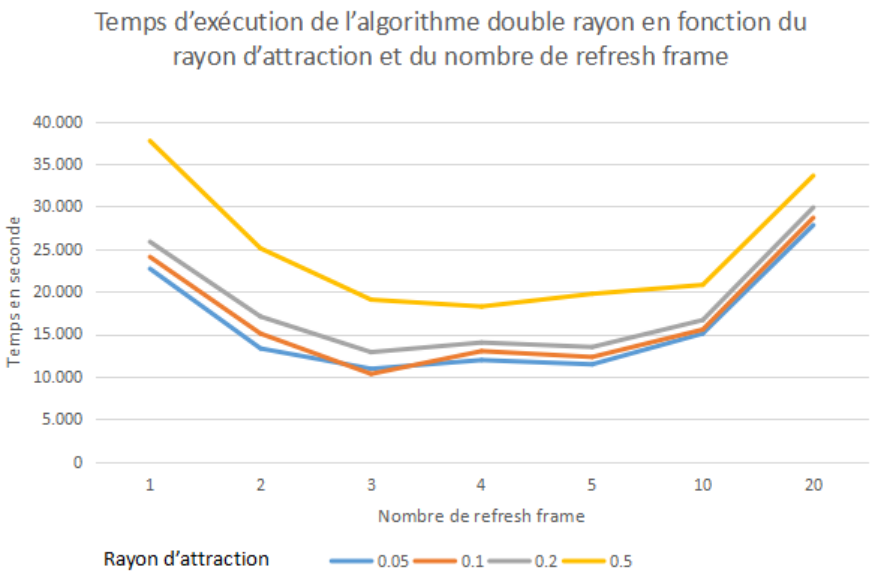


Figure 1: Graphique sur la fonction `compute_attract_by_double_radius` pour 10 000 particules

A l'aide de ce graphique, nous pouvons constater que la méthode `compute_attract_by_double_radius` est plus efficace quand le refresh frame est à 3 ou 4.

Pour ce graphique nous avons 1000 points avec une vitesse de 0,01 et des rayons de 2,3,4 et 5. Le calcul de temps est fait sur 100 frames.

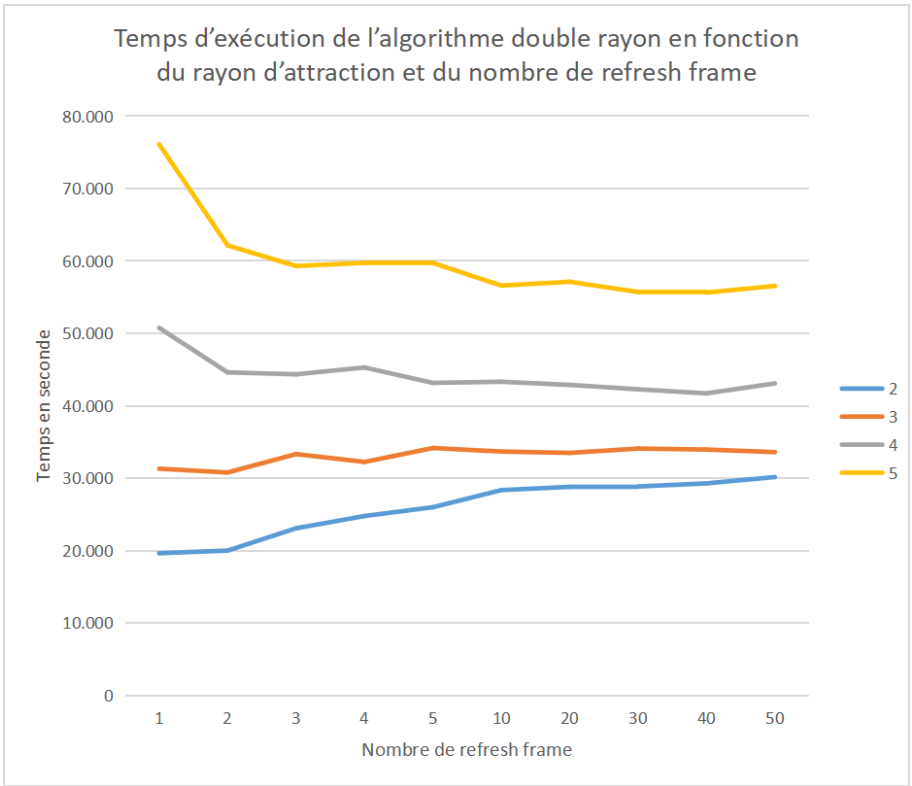


Figure 2: Graphique sur la fonction `compute_attract_by_double_radius` pour 1000 particules

Nous pouvons constater que plus le rayon est grand plus le refresh frame doit être grand pour être efficace.

3 Algorithme "Diffusion Limited Aggregation" : `compute_diffusion_limited_aggregation`

Pour utiliser cet algorithme il faut que le nombre de particules fixes, `NB_INIT_FIXED_POINTS`, soit différent de 0.

L'objectif est que toutes les particules soient fixées. Elles se fixent lorsque qu'elles entrent dans le rayon d'attraction d'une particule fixée.

Pour que le rendement soit plus rapide, nous cherchons les particules attractives des particules fixes tant que le nombre de particules fixes est inférieur à la moitié du nombre total de particules. Sinon nous recherchons les particules attractives des particules non fixes. Ainsi nous ne faisons la recherche que pour au maximum la moitié des particules.
Pour trouver les particules attractives nous utilisons la liste `_possible_futur_attract` avec la méthode `compute_attract_by_double_radius`.

La coloration se fait en fonction du nombre de frames durant lesquelles la particule est restée en mouvement. Elle est colorée au moment où elle devient fixe.

Pour que les particules se fixent quand elles se touchent, nous mettons un rayon d'attraction qui est le double de la taille d'une particule, qui est le rayon de la particule.

Nous avons réalisé un timelapse visible sur Youtube (Timelapse d'une agrégation)

4 Résultat visuel de l'algorithme `compute_diffusion_limited_aggregation`

Les particules rouges sont les plus anciennes à avoir été fixées, ensuite ce sont les couleurs : orange, jaune, vert, bleu, violet et rose. Ensuite nous recommençons ce cycle de couleurs. Nous pouvons observer une structure d'arbre sur la Figure 4.

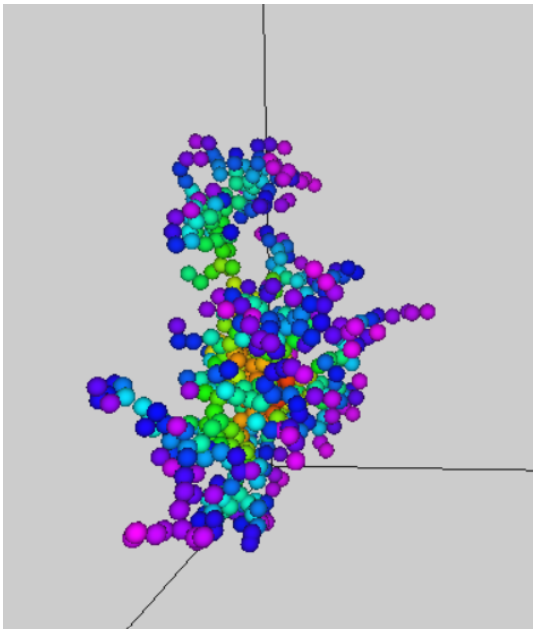


Figure 3: Agrégation colorée 1 avec 3000 points

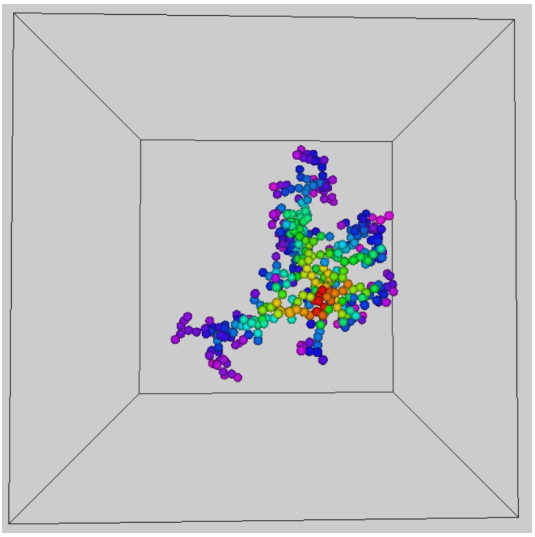


Figure 4: Agrégation colorée 2 avec 3000 points

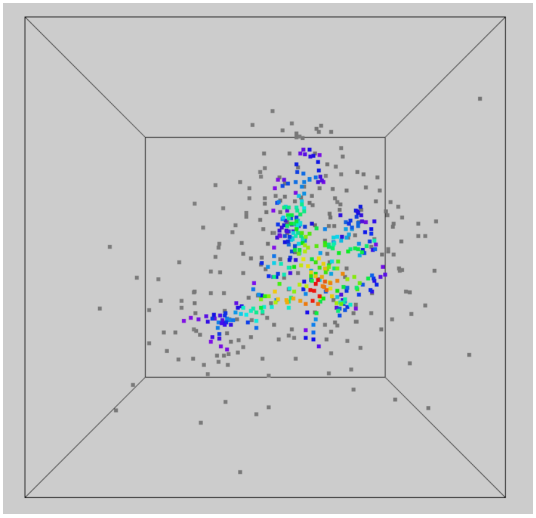


Figure 5: Agrégation colorée 2 avec 3000 points sans les meshs

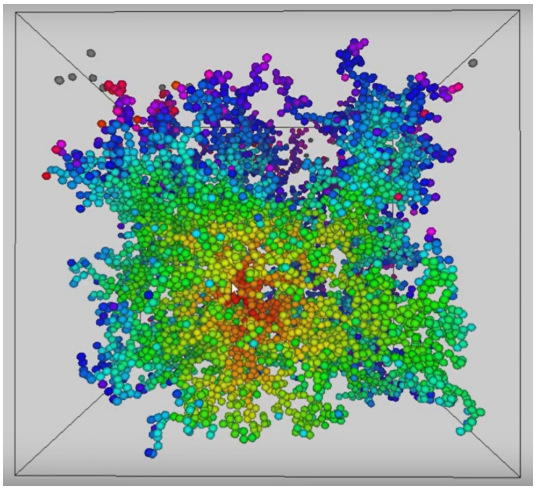


Figure 6: Résultat final agrégation colorée avec 10 000 points du (Timelapse d'une agrégation)

5 3D Boids

Pour tester nos méthodes nous avons reproduit le comportement des Boids de Craig Reynolds (Site de référence) paramétrable. Un boid est l'équivalent d'une particule avec simplement un mouvement différent décrit par l'algorithme de Craig Reynold. Ainsi l'avantage est d'utiliser `compute_attract_by_double_radius` pour récupérer les voisins de chacun des Boids.

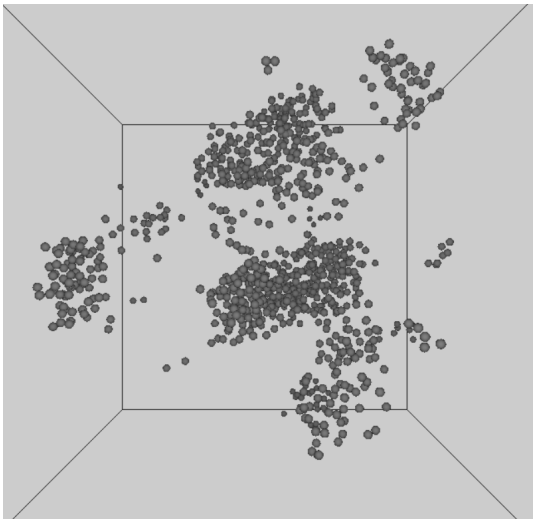


Figure 7: Boids 3D