Université Sorbonne Paris Nord

Rapport de projet Structure de données avancée

Les personnes qui ont participé à la réalisation du projet :

Youssef JEBBOURI 12013048
Ismaïne MESSAOUD 11923798
Amine AMZAL 12011737

1-Introduction :

En s'inspirant de l'algorithme ForceAtlas2, on a développé une implémentation qui permet de visualiser un graphe en 3D en partant d'un fichier .dot, un nombre de points égal au nombre de sommets dans le graphe sont créés aléatoirement au début et qui exercent des forces les uns sur les autres.
Cela est possible en utilisant une structure de Octo-Tree (Arbre à 8 fils) qui contiendra nos points dans chacun de ses nœuds.

2- Bibliothèques :

numpy: on utilise la bibliothèque numpy pour manipuler les tableaux dans le projet

matplotlib: générer le graphe de complexité final

random: pour générer les points aléatoirement selon une loi uniforme

graphviz: pour générer les visualisations 3D selon les coordonnées entrées

PyOpenGL: pour afficher notre graphe dans des fenêtre ciblées

PyQt5: pour pouvoir afficher deux fenêtres en même temps

3-Choix d'implémentation :

Pour réaliser ce travail nous avons choisi de considérer les points comme des objets placés dans l'espace à des coordonnées (x,y,z) qui auront une masse, un label et un nombre de voisins stockés sous forme d'une liste. Et pour connaître la distance entre un point et un autre nous avons ajouté une méthode "distance_to". Ceci est l'implémentation de notre class "Point3d".

Chacun des points sera à l'intérieur d'une structure(Class) de noeud qui aura des coordonnées (X0,Y0,Z0) faisant référence au centre du

noeud, une largeur, une hauteur, une profondeur, une liste de points, une liste de ses fils, une masse et un attribut “leaf” qui désignera si le noeud est une feuille ou pas(si le noeud est divisé ou pas) . Ainsi que des guetteurs qui vont nous permettre de récupérer les valeurs des attributs des nœuds. Ceci est l’implémentation de notre class “Node3d”.

On a utilisé le Octo-tree car ce dernier nous permet de diviser nos points en plusieurs petits groupes en utilisant leur coordonnées , ceci est fait par l'opération “subdivide” qui est implémenté avec des octo-tree et qui a comme complexité n*log (n).

Et finalement pour représenter notre graphe on utilise un octo-tree en implémentant une class “OTree” constitué d’un seuil de points “threshold” permis par noeud(le nombre de points qu’un noeud peut avoir avant qu’il soit divisé), d’un nombre “n” de points générés aléatoirement selon une loi uniforme afin d’éviter une concentration sur une seule zone et une racine qui est une instance de la class “Node3d”.

On a les fonctions “add_point” et “get_points” qui nous permettront respectivement d’ajouter un points à notre “OTree” et renvoyer la liste des points de l’OTree

Chaque noeud de l’arbre sera représenté comme un cube et quand le nombre de points dans le noeud dépasse le seuil autorisé (threshold) on le divise en 8 sous cube qui seront les fils du premier, et ainsi de suite pour chaque fils (on vérifie récursivement si les fils doivent être subdiviser).

Maintenant que les points sont bien représentés, on doit calculer la nouvelle position de chaque point après un certain lapse de temps (dt) en prenant en compte les forces qu’ils exercent les uns sur les autres, ce qu’on fait avec la fonction “new_position3d” qui prend en paramètre les forces, les points, un delta t(interval de temps) et la vitesse initiale, avec ces derniers on calcule l'accélération qui nous permet de calculer la vitesse, qui nous permet à son tour de calculer les nouvelles positions, la fonctions retourne la liste des points initiale avec leur

nouvelles positions.

Ensuite pour visualiser notre structure nous avons besoin d'une bibliothèque qui permet d'afficher notre OTree sous forme d'un cube et notre graphe à l'intérieur, cette bibliothèque est OpenGL qui combinée avec PyQt5 nous permet de coder les fonctions "draw_cube"(affiche le cube) et "scatter_points"(affiche notre graphe à l'intérieur).

Modèle d'énergie:

On a utilisé un modèle d'énergie simple à partir du papier de recherche donné et qui représente la force de répulsion et d'attraction comme suit:

Force de répulsion(p1,p2) = k/(distance(p1,p2)
avec 'k' une constante à définir (dans notre cas k=1)

Force d'attraction(p1,p2) = -kr * distance(p1,p2)
avec 'kr' une constante à définir (dans notre cas kr=1)

Les formules utilisées :

Accélération = F/M (Force/masse)

Vitesse = Vitesse_initiale + accélération *dt (différentiel de temps)

Nouvelle_position = ancienne_pos + vitesse *dt (différentiel de temps)

4-Analyse théorique et empirique:

On va faire l'analyse théorique des fonctions "rep_force" et "recursive_subdivide3d"

"rep_force":
Pour calculer la force de répulsion appliquée sur un point, il faut calculer la sommes de toutes les forces répulsives exercées par les autres points sur notre point (et répéter ce processus 'n' fois pour avoir la liste des forces de répulsion appliquées sur tous les points)
et c'est pour cette raison que la complexité de ce calcul pour tous les points est d'ordre $O(n^2)$.
Pour optimiser ces calculs on va considérer les points qui sont assez loins et dans une zone assez petite comme un seul point, représentés par leur centre

de gravité, et qu'on utilise pour calculer la force de répulsion entre notre point et ce groupe de points.
L'octo-tree nous facilite cette opération parce que l'ensemble des points contenus dans ce dernier sont répartis en plusieurs zones d'une hauteur, longueur et largeur prédéfinis, donc il suffit de parcourir notre octo-tree et à chaque fois comparer le ratio(distance/volume) avec une constante qu'on choisit (dans notre cas c'est 1 car on travail sur des molécules)
En utilisant cette approche de type arbre, la fonction peut rapidement calculer la force répulsive pour un grand nombre de points en parcourant efficacement l'arbre et en limitant le nombre de calculs effectués pour chaque point.

"recursive_subdivide3d":
La fonction recursive_subdivide3d prend en entrée un noeud de l'octree et une valeur seuil (k).

La fonction commence par vérifier si le nombre de points dans le noeud est inférieur ou égal à la valeur seuil k. Si c'est le cas, la fonction retourne et arrête les subdivisions ultérieures. Si le nombre de points est supérieur à k, la fonction poursuit les subdivisions. La propriété "feuille" du noeud est définie à False, indiquant qu'il ne s'agit pas d'un noeud feuille.

La fonction calcule ensuite la largeur, la hauteur et la profondeur à moitié du noeud, et appelle une fonction séparée appelée "contains" pour déterminer les points qui appartiennent à chacun des 8 sous-noeuds. 8 nouveaux sous-noeuds sont alors créés et initialisés avec les propriétés appropriées, et la fonction s'appelle récursivement sur chacun de ces sous-noeuds avec la même valeur seuil k. Ce processus se poursuit jusqu'à ce que le nombre de points dans un noeud soit inférieur ou égal à k, moment où la fonction cesse de diviser davantage et retourne.

Enfin, la liste des enfants pour le noeud d'origine est mise à jour pour inclure les 8 nouveaux sous-noeuds qui ont été créés, et la fonction retourne. Ce processus se poursuit jusqu'à ce que l'intégralité de l'octree soit construit et subdivisé, permettant une organisation et une recherche efficace des points dans l'espace 3D.

calcul:

$T(n) = T(n/8) + 8*O(n) \Rightarrow T(n) = 8T(n/8*8) + O(n/8) + 8*O(n) \Rightarrow$

$T(n) = 8*T(n/8*8*8) + 8*O(n/8*8) + O(n/8) + 8*O(n) \Rightarrow \ldots.$

$T(n) = 8*T(n/8^k) + 8*O(n/8^{(k-1)}) + O(n/2^{(k-2)}) + \ldots + O(n/8) + O(n)$

T(n) = O(1) + O(n/8^1) + O(n/8^2) + ... + O(n/8^(log8(n)-1)) + O(n/8^(log8(n)))

en combinant tous les terme de forme O(n/8^i)

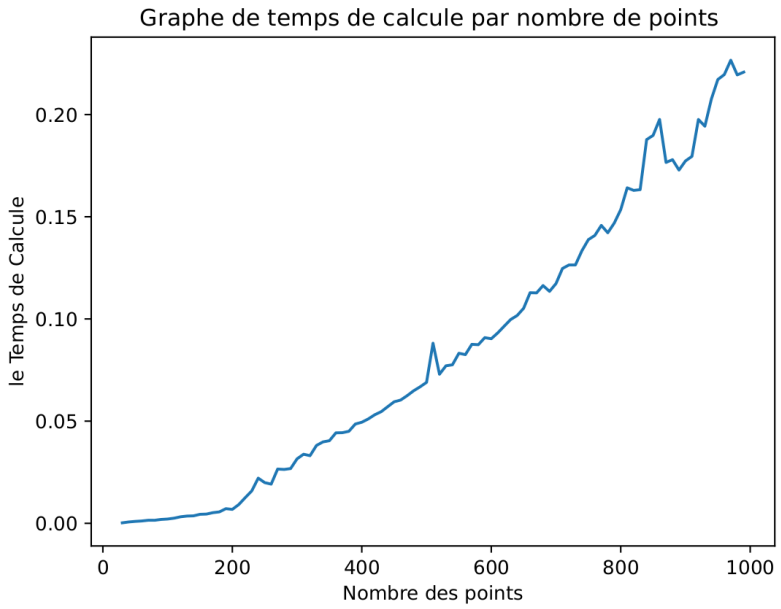
T(n)= O(1) + O(n(1/8^1 + 1/8^2) + ... + 1/8^(log8(n)-1)) + 1/8^(log8(n)))

En utilisant la formule de la progression géométrique infinie:

…

T(n) = n*log8(n) => la complexité de la fonction recursive_subdivide3d est: n*log(n)

Analyse empirique:



On a calculé le temps nécessaire pour calculer les forces et les nouvelles positions pour un nombre de points qui varient entre 30 et 1000, afin d'avoir une idée sur l'évolution de la durée d'exécution en fonction du nombre de points.