Notice technique

Sommaire

1 - Méthode de construction du diagramme de Voronoï

Les classes Point et Segment représentent respectivement les coordonnées (x, y) et les arêtes.

La classe Triangle va gérer la géométrie centrale en trois étapes clés : Calcul du centre de son cercle circonscrit (futur sommet de Voronoï), vérification mathématique si un point est dans ce cercle (critère de Delaunay) et extraction de ses arêtes.

Pour **Delaunay** l'algorithme démarre avec un "Super-Triangle" géant (sommets à 5000 et -5000) englobant tous les points. À chaque ajout d'un point, on identifie et supprime les "mauvais triangles" (ceux dont le cercle circonscrit contient ce point). Ce "trou" polygonal est ensuite retriangulé en reliant ses bords au nouveau point. Enfin, le Super-Triangle initial et ses connexions sont nettoyés.

Pour **Voronoï** l'algorithme analyse comment les triangles se partagent les arêtes, Si l'arête appartient à 2 triangles : La frontière de Voronoï relie simplement les centres circonscrits des deux triangles. Si l'arête appartient à 1 seul triangle, il est sur le bord extérieur, la frontière trace une ligne fuyant vers l'infini. Sa longueur est calculée dynamiquement (10 fois la taille du dessin) pour s'adapter à toute échelle de données. La fonction calculer_diagramme sert d'interface. Elle convertit les coordonnées d'entrée en objets Point, exécute les calculs, et formate le résultat dans un dictionnaire standardisé ({"sommet": [...], "aretes": [...]}).

2 - Choix techniques

2.1 - Langages utilisés

Pour pouvoir créer notre application, nous avons décidé d'utiliser le langage de programmation Python pour plusieurs raisons.

D'abord, nous avons fait le point sur les compétences techniques de chacun de l'équipe et tout le monde était à l'aise d'utiliser Python pour le projet.

Ensuite, après avoir analysé les exigences du projet, nous avons conclu que l'application serait simple en termes de complexité des fonctionnalités. De ce fait, il n'y avait pas besoin d'utiliser un langage de programmation complexe comme Java ou C, donc Python était le choix idéal pour programmer simplement tout en restant efficace.

Pour finir, comparé à d'autres langages de programmation comme Java ou C, Python est beaucoup plus lisible et compréhensible permettant ainsi de faciliter la collaboration au niveau du développement avec Git et GitHub.

2.2 - Bibliothèques utilisées

À travers notre application, nous avons utilisé plusieurs bibliothèques Python :

- **pathlib** : Permet de manipuler des chemins de fichiers.
- **sys** : Permet d'interagir avec le système où l'application est exécutée.
- **json** : Permet de lire le fichier JSON contenant les points du diagramme de Voronoï.
- **math** : Permet d'utiliser des fonctions mathématiques dans l'algorithme calculant le diagramme de Voronoï.
- **matplotlib** : Permet de créer le diagramme de Voronoï selon le résultat de l'algorithme calculant le diagramme de Voronoï.
- **streamlit** : Permet de créer l'interface web de l'application affichant le résultat du diagramme de Voronoï.
- **traceback** : Permet d'afficher certaines erreurs de l'application.
- **pytest** : Permet de tester chaque fonctionnalité de l'application.

3 - Architecture

Pour créer notre application, nous avons décidé de créer un seul programme Python pour chaque fonctionnalité de l'application afin d'avoir des responsabilités séparées.

Voici une explication sur ce que fait chaque programme Python :

- **app/streamlit_app.py** : Programme central de l'application appelant dans l'ordre les fonctions des autres programmes Python pour à la fin permettre d'afficher ou d'exporter en PNG ou en SVG le diagramme de Voronoï à travers une interface web.
- **data/tab.txt** : Fichier contenant une liste de points au format TXT.
- **data/test.json** : Fichier contenant une liste de points au format JSON.
- **src/domain/point.py** : Programme contenant une classe représentant un point par ses coordonnées x et y.
- **src/domain/segment.py** : Programme contenant une classe représentant un segment par deux points.
- **src/domain/triangle.py** : Programme contenant une classe représentant un triangle par trois points.
- **src/delaunay_bw.py** : Programme implémentant l'algorithme de triangulation de Delaunay en utilisant des points et des triangles.
- **src/io_points.py** : Programme lisant un fichier TXT ou JSON afin d'y extraire une liste de points.
- **src/render.py** : Programme dessinant le diagramme de Voronoï.
- **src/voronoi.py** : Programme calculant le diagramme de Voronoï en utilisant des points, des segments et l'algorithme de triangulation de Delaunay.
- **test/test_algo.py** : Programme de test sur le programme implémentant l'algorithme de triangulation de Delaunay et sur le programme calculant le diagramme de Voronoï.
- **test/test_domain.py** : Programme de test sur les classes représentant un point, un segment et un triangle.

4 - Limites

À la fin du développement de l'application, nous avons constaté plusieurs limites.

Pour commencer, notre application ne peut extraire les données des points que dans des fichiers dans le format TXT ou JSON. Ainsi, il n'est pas possible de lire des fichiers sous d'autres formats comme par exemple CSV, XML…

De plus, plus le nombre de points dans le diagramme de Voronoï est élevé et plus l'application prendra du temps à la construire selon la puissance du système où l'application est exécutée.

Pour finir, les coordonnées des points sont représentées par des float ce qui veut dire que dans le calcul du diagramme de Voronoï, il peut y avoir des petites erreurs d'arrondi donc le résultat sera correct d'un point de vue mathématique mais sera légèrement imprécis.