

SAÉ – S6

Notice Technique : Générer un Diagramme de Voronoï



Membres :

Hocine Zared – Lacine Sanogo – Hichem Zenaini – Yacine Sellaoui

Critères d'évaluations de la Notice Technique :

Méthode de construction du diagramme de Voronoï

Choix techniques

Architecture

Limites

1. Comprendre la logique

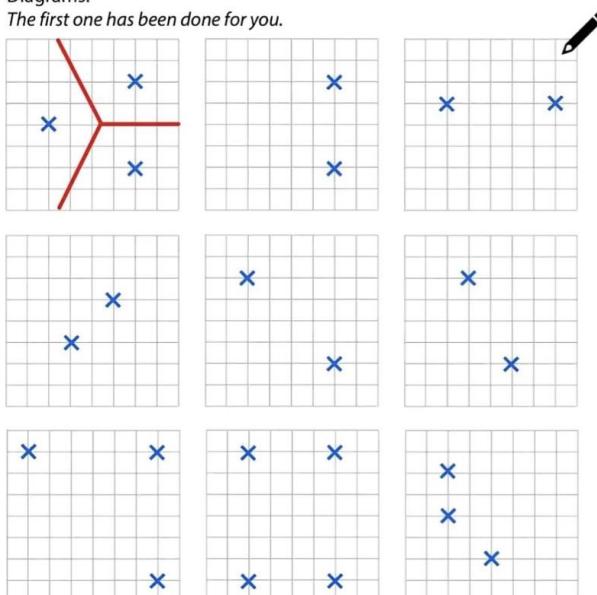
Pour être tout à fait transparent, le vrai défi de ce TP n'était pas l'algorithme pure ou le langage Python. Le plus dur, ça a été de visualiser ce qu'était concrètement un diagramme de Voronoï. Les définitions mathématiques qu'on trouve en ligne parlent d'espaces métriques et de partitionnement, et j'avoue que j'ai complètement bloqué dessus au début.

Je suis resté bloqué un moment, je suis allé sur des forums reddit, j'ai regardé de multiples vidéos YouTube d'abord dans un premier temps qui expliquaient le concept de Diagramme de Voronoï, allé sur Wikipédia, ensuite des vidéos qui expliquaient comment coder ça... jusqu'à tomber un peu au hasard sur une vidéo TikTok d'un prof de maths. Il montrait un exercice de géométrie niveau collège sur une grille quadrillée : il y avait des croix bleues, et il fallait tracer des lignes rouges entre elles. C'est cette image exacte qui m'a débloquée.

Voronoi Diagrams

1

1. The following grids show coordinates of Voronoi Diagram.
Draw the perpendicular bisectors on the grids to complete the Voronoi Diagrams.
The first one has been done for you.



(Légende : Image trouvée sur Tiktok qui m'a permis de comprendre le concept du diagramme de Voronoï)

J'ai compris qu'un diagramme de Voronoï, c'était simplement tracer la médiatrice entre chaque paire de points pour délimiter des territoires.

J'ai donc d'abord essayé de coder ça : un programme qui calcule des équations de droites et cherche leurs intersections. Grosse erreur. C'est facile sur le papier, mais dire à la machine de gérer des droites infinies et d'effacer les segments qui dépassent... Je me suis vite rendu compte que c'était compliqué...

En repensant au cours d'Introduction à la programmation multimédia, j'ai changé d'approche. Pourquoi s'acharner à tracer des lignes (en vectoriel) quand on peut raisonner en pixels ? L'idée était de parcourir l'image comme une grille et de demander à chaque pixel : "Tu es le plus proche de quel point ?". En coloriant simplement chaque pixel avec la

couleur du point gagnant, les frontières allaient se dessiner toutes seules au niveau des zones de contact. C'était beaucoup plus accessible et logique à coder.

2. Choix techniques et architecture

Pour mettre ça en place sans alourdir le projet, j'ai dû choisir les bons outils :

Numpy :

Numpy m'a permis de créer un tableau vide (`np.zeros`) et de le manipuler facilement par ses coordonnées. Et c'est une bibliothèque de base de Python, donc tant mieux

Matplotlib :

C'était le module parfait pour transformer mon tableau Numpy en image grâce à la fonction `imshow`

Tkinter :

Le sujet demandait une interface "conviviale". Je ne voulais pas m'embarquer dans l'installation de grosses librairies tierces, et puis sur Internet on tombe direct sur cette bibliothèque avec une documentation assez "simple". Mais la vraie galère ici, ça a été d'intégrer le graphique Matplotlib directement dans la fenêtre Tkinter pour éviter d'avoir une fenêtre pop-up qui s'ouvre à côté du menu. J'ai dû fouiller un peu la documentation et des forum pour trouver et utiliser `FigureCanvasTkAgg`.

J'aimerais afficher le résultat de mon dataframe sous forme de graphique matplotlib mais que ce résultat s'affiche sur une fenêtre tkinter et non sur plt.show de matplotlib. J'ai besoin de votre aide pour trouver la solution. Merci pour votre aide!

anonyme, samedi 04 décembre 2021 à 15h58

Salut,

Il faut utiliser le backend `FigureCanvasTkAgg`. Il y a un exemple [dans la documentation](#). Ce qui t'intéresse est surtout le début, jusqu'au `canvas = FigureCanvasTkAgg(fig, master=root)` qui prend une figure matplotlib et une fenêtre tkinter. `canvas.draw()` permet ensuite de tracer le plot lui-même.

Côté architecture, j'ai fait attention à ne pas tout mélanger. D'un côté, j'ai mes fonctions de calcul pur (lecture du fichier texte, calcul de la distance Euclidienne, recherche du point le plus proche). Elles tournent de manière totalement indépendante.

De l'autre, j'ai l'interface graphique, que j'ai encapsulée dans une classe (`AppVoronoi`). Faire de l'objet pour l'interface m'a évité de me retrouver avec des variables globales partout ce qui devient vite un cauchemar à débugger. J'ai aussi factorisé le code d'exportation : les boutons PNG et SVG utilisent la même fonction en arrière-plan pour éviter la duplication de code.

3. Limites de mon code et autocritique

Même si le résultat final correspond aux attentes visuelles et que l'interface est fluide, je suis conscient que mon code a plusieurs défauts, principalement à cause de cette approche "pixel par pixel".

La lenteur de la "force brute" : Mon algorithme calcule la distance pour *chaque* pixel avec *chaque* point du fichier. Avec une dizaine de points sur une petite fenêtre, ça passe inaperçu. Mais si on lui donne 1 000 pixels par exemple, je pense que l'interface Tkinter va se figer pendant plusieurs secondes.

La gestion artisanale de l'échelle : Pour l'instant, mon code regarde la coordonnée la plus grande dans le fichier texte et crée une grille de cette taille (en ajoutant 10 pixels de marge). C'est du bricolage. Si un utilisateur donne un point aux coordonnées ([272727](#), [272727](#)), le programme va essayer de créer une matrice Numpy immense et va très certainement crasher. Il faudrait implémenter une vraie fonction de normalisation pour ramener n'importe quelles coordonnées sur un Canvas d'une taille d'affichage fixe (par exemple 600x600).

Au final, ce TP m'a forcé à contourner un blocage mathématique par une solution algorithmique et visuelle (la matrice de pixels). L'intégration entre Matplotlib et Tkinter m'a pris du temps, mais le rendu est propre, et j'ai bien identifié les optimisations qu'il faudrait apporter si le programme devait traiter de vraies données à grande échelle. Nous sommes à présent prêt à aborder la phase 2 avec tout ce bagage et en étant conscient des limites du code et de la logique derrière les diagrammes de Voronoï.