Notice Technique : Diagramme de Voronoï (Phase 1)

Kelvin UTHAYAKUMAR, Edmilson DA COSTA SA, Yann DIARRASSOUBA,
Rayan EL OUAZZANI, Ilyes MEDJDOUB, Leelian SERRANT

Introduction

Ce document s'inscrit dans le cadre de la Situation d'Apprentissage et d'Évaluation du semestre 6 du BUT Informatique. L'objectif global de ce projet est de concevoir une application capable de déterminer et de visualiser un diagramme de Voronoï à partir d'un ensemble de points fournis sous forme de coordonnées (x, y) dans un fichier texte. Afin d'expliquer le travail effectué, cette notice détaillera dans un premier temps la méthode mathématique et algorithmique que nous avons retenue pour la construction du diagramme. Dans un second temps, nous justifierons les différents choix techniques et technologiques, notamment en matière de langage, d'outils et de stratégie de test pour fiabiliser le code. Ensuite, nous présenterons l'architecture logicielle mise en place pour garantir la modularité et la maintenabilité du code. Enfin, nous conclurons par une analyse objective des limites de notre implémentation, ouvrant ainsi la voie aux futures évolutions du projet. Afin d'expliciter le travail accompli, ce rapport détaillera dans un premier temps la méthode mathématique et algorithmique retenue pour la construction du diagramme. Dans un deuxième temps, nous justifierons les différents choix techniques et technologiques opérés, notamment en matière de langage, d'outils et de stratégie de test. Ensuite, nous présenterons l'architecture logicielle mise en place pour garantir la modularité et la maintenabilité du code. Enfin, nous conclurons par une analyse objective des limites inhérentes à notre implémentation actuelle, ouvrant ainsi la voie aux futures itérations du projet.

1 Méthode de construction du diagramme

Pour cette première phase du projet, la méthode de construction repose sur un algorithme de découpage d'espace par force brute. Cet espace est défini par des bornes allant de x_{min} à x_{max} et de y_{min} à y_{max} et est divisé en une grille régulière de dimension $N \times N$. Tout d'abord, l'algorithme commence par évaluer la taille des cellules qui composeront le diagramme finale. En divisant la largeur et la hauteur totales du repère par la résolution N, il calcule les pas d'avancement Δx et Δy. Ces deux valeurs conditionnent directement les dimensions de chaque cellule dessiné par l'interface graphique, selon les rapports : $\Delta x = \frac{x_{max} - x_{min}}{N}$ et $\Delta y = \frac{y_{max} - y_{min}}{N}$.

Une fois cette grille établie, le programme effectue un balayage complet à l'aide d'une double boucle itérative. Pour chaque cellule, l'algorithme génère les coordonnées exactes de son coin inférieur gauche. Ces coordonnées, définies par $((x_{min} + i \cdot \Delta x), (y_{min} + j \cdot \Delta y))$, sont ensuite transmises au moteur de calcul géométrique pour déterminer quel est le germe le plus proche de cette zone spécifique.

Ensuite, la recherche du germe le plus proche est exécutée de manière séquentielle. L'algorithme initialise la distance minimale de référence avec le premier germe de la liste, puis évalue les distances

avec tous les autres germes un par un. Pour déterminer ce point le plus proche, il s'appuie sur le carré de la distance euclidienne plutôt que sur la formule exacte avec sa racine carrée. En effet, l'objectif de l'algorithme n'est pas de calculer une longueur réelle absolue, mais uniquement de comparer des distances entre elles pour isoler un minimum. La fonction racine carrée étant strictement croissante, l'ordre de grandeur est toujours conservé et comparer des distances réelles donne mathématiquement le même classement que comparer leurs carrés. Le programme se contente donc d'évaluer l'inéquation suivante pour mettre à jour l'index du germe de référence :

$$(x - x_{res})^2 + (y - y_{res})^2 > (x - x_i)^2 + (y - y_i)^2$$

Enfin, une fois la vérification terminée pour la cellule courante, l'algorithme récupère l'index du germe identifié comme étant le plus proche. L'interface graphique utilise ensuite cet index pour attribuer à la cellule actuelle la couleur correspondante à ce germe, piochée dans une palette préalablement mélangée. Afin d'éviter tout dépassement si le nombre de germes s'avère supérieur aux couleurs disponibles, l'association s'effectue de manière sécurisée grâce à une opération mathématique de modulo appliquée sur cet index. Ce processus d'évaluation et de coloration se répète ensuite itérativement pour chaque cellule jusqu'à recouvrir l'intégralité de la grille, construisant ainsi progressivement le rendu visuel final.

2 Choix techniques

Les technologies et les approches de développement ont été sélectionnées dans le but de répondre aux exigences de clarté, de maintenabilité et d'efficacité de la Phase 1. Tout d'abord, nous avons choisi le langage Python pour sa flexibilité et sa grande pertinence dans la manipulation de données ainsi que pour l'exécution de calculs géométriques simples. Concernant la visualisation des résultats, le choix s'est porté sur la bibliothèque Matplotlib plutôt que sur une interface graphique lourde et complexe. L'utilisation du module `matplotlib.pyplot` et de l'objet `Rectangle` nous a permis de tracer très rapidement la grille de cellules colorées. De plus, cette bibliothèque intègre nativement l'exportation du rendu final via la commande `savefig`, répondant ainsi parfaitement au cahier des charges. Afin d'évaluer l'efficacité de cette solution, le module natif `time` a été intégré pour chronométrer précisément la durée de calcul et de dessin, offrant une métrique de performance claire.

Sur le plan de la validation logicielle et de la fiabilité du code, nous avons mis en place des de tests unitaires automatisés grâce au framework Pytest. Ces tests ont été fait pour valider la robustesse de la logique métier une fois celle-ci implémentée. Cette suite de tests vérifie d'une part l'intégrité de la lecture des fichiers, notamment la gestion des fichiers vides ou contenant du texte invalide, et d'autre part l'exactitude des calculs mathématiques de la classe `CalculerGerme`, y compris face à des cas limites tels que des points équidistants ou des coordonnées négatives. L'utilisation de la fixture `tmp_path` a également permis de simuler dynamiquement des fichiers texte, garantissant que les tests s'exécutent en isolation totale sans dépendre de fichiers locaux statiques et préviennent ainsi toute régression.

3 Architecture Logicielle

L'architecture du projet a été pensée pour respecter le Principe de Responsabilité Unique, garantissant ainsi une séparation stricte des fonctionnalités. L'application est structurée autour de trois classes distinctes, isolées les unes des autres et orchestrées par un point d'entrée centralisé.

La classe `LirePoints` agit comme le module d'extraction des données. Elle a pour rôle exclusif de lire le fichier de coordonnées, d'en extraire le texte brut, de le nettoyer, puis de convertir ces informations en structures de données exploitables par le programme, à savoir une liste de tuples contenant des nombres flottants.

De son côté, la classe `CalculerGerme` constitue le véritable moteur mathématique de l'application. Initialisée avec les coordonnées des germes, elle expose une méthode de calcul pour déterminer le point le plus proche. Elle est conçue de manière totalement indépendante, ignorant tout des aspects liés à l'affichage graphique ou à la provenance initiale des données.

Ensuite, la classe `Visualisation` se concentre exclusivement sur la présentation. Elle gère la création de la fenêtre graphique, le dimensionnement de la grille de résolution, le brassage des couleurs disponibles, et le dessin itératif des rectangles. Pour colorier sa grille, elle se contente d'interroger la logique métier à la volée.

Enfin, l'ensemble de ces composants est coordonné par le script `main.py`. Ce dernier agit comme un contrôleur principal, il instancie le lecteur avec le chemin d'accès au fichier source, récupère les points, initialise le calculateur et déclenche le processus de visualisation. Cette architecture modulaire garantit la maintenabilité et l'évolutivité de l'application. En effet, Les différentes responsabilités étant séparées, il sera très facile de remplacer l'algorithme de calcul actuel par des approches mathématiques plus complexes à l'avenir. Pour aller plus loin, et comme envisagé lors de la conception initiale, l'implémentation d'interfaces permettrait de formaliser cette structure en définissant un contrat strict entre la logique métier et l'affichage, facilitant ainsi l'intégration de nouvelles méthodes.

4 Limites de l'implémentation actuelle

Bien que la méthode employée produise un résultat visuellement correct et fonctionnel, elle comporte plusieurs limitations techniques qui sont intrinsèques à l'approche par découpage en grille.

La première limite majeure concerne la complexité spatiale et temporelle de l'algorithme. La résolution du rendu final dépend directement de la variable de maillage N. La complexité temporelle est proportionnelle à $O(N^2 \times G)$, où G représente le nombre de germes. Par conséquent, si l'on souhaite augmenter la netteté de l'image, par exemple en passant d'une résolution de $N = 100$ à $N = 1000$, la quantité de calculs à effectuer est multipliée par cent. Cela entraîne une hausse quadratique du temps de génération et peut rapidement saturer les ressources de la machine.

La seconde limitation est d'ordre visuel. Les frontières délimitant les différentes régions de Voronoï ne sont pas tracées à l'aide de véritables vecteurs mathématiques continus. Elles sont approximées par une succession de blocs rectangulaires. Cela provoque inévitablement un effet d'escalier, communément appelé aliasing, qui est particulièrement visible sur les arêtes obliques séparant les différentes zones de couleur.

Enfin, la dernière limite concerne l'absence de création d'informations topologiques. Le programme actuel génère uniquement une matrice visuelle, mais il ne construit aucune structure de données décrivant formellement les relations géométriques entre les points de l'espace. L'application ne connaît ni les sommets, ni les arêtes, ni les graphes d'adjacence des polygones. Il est par conséquent impossible d'utiliser cette implémentation pour répondre à des requêtes analytiques complexes, telles que l'identification immédiate des voisins directs d'un germe spécifique. Ces différentes limites justifient pleinement l'intérêt d'explorer des algorithmes plus performants et mathématiquement exacts lors des expérimentations prévues dans la deuxième phase de ce projet.