Rochedreux Hugo

Drouvot Melody

Fenard Dorian

Klingler Emma

SAE Optimisation

Travail réalisé

Après une première analyse du sujet, notre première étape a été de trouver quels algorithmes de clustering étaient les plus adaptés à notre cas. La solution qui nous à paru la plus évidente a été d’utiliser l’algorithme de Kmeans pour la recherche de biomes, puis DBSCAN pour différencier les écosystèmes.

Nous avons choisi Kmeans pour plusieurs raisons. En premier, il est rapide. Nos images ont une taille de 1400\*1400 pixels. On à donc au total quasiment 2 millions de pixels à traiter. Avec sa complexité de O(N), Kmeans permet de les traiter plutôt rapidement. Aussi, il est rapide à mettre en place. Cependant, nous avons dû essayer de trouver le nombre idéal de centroïde à utiliser dans notre implémentation, les résultats peuvent différer en fonction des conditions initiales.

Le problème de Kmeans, c’est qu’il détecte mieux les clusters plutôt arrondis, ce qui n’est pas toujours le cas des biomes.

Pour la recherche d’écosystème, nous avons utiliser DBSCAN. C’est un algorithme plutôt lent, avec une complexité de O(N²), mais vu que l’image traitée ne contient que certains des biomes, le nombre de pixel a traité est réduit. Il résiste aussi très bien au bruit et supporte des clusters de forme variable, il est donc adapté pour les écosystèmes.

Pour être plus agréable d’utilisation, on a créé une interface graphique permettant de rapidement parcourir les images générées et de facilement choisir la planète à analyser.

Nous avons aussi implémenté deux manières de calculer la distance entre les couleurs, soit avec la distance euclidienne, soit avec la distance ∆E∗94. Avec la distance ∆E∗94, te temps de calcul est plus grand, mais la perception des couleurs est plus semblable à celle de l’œil humain.

Analyse des résultats

Avec l’algorithme de Kmeans, certains biomes n’étaient pas détectés. Aussi, avec le temps qui restait on a essayé d’implémenter encore plus d’algorithmes pour voir les différents résultats.

A chaque utilisation de notre programme, le résultat est un peu différent car les clusters ne commencent pas au même endroit pour Kmeans. Et vu que la recherche d’écosystème se base sur les biomes, eux aussi différent un peu.

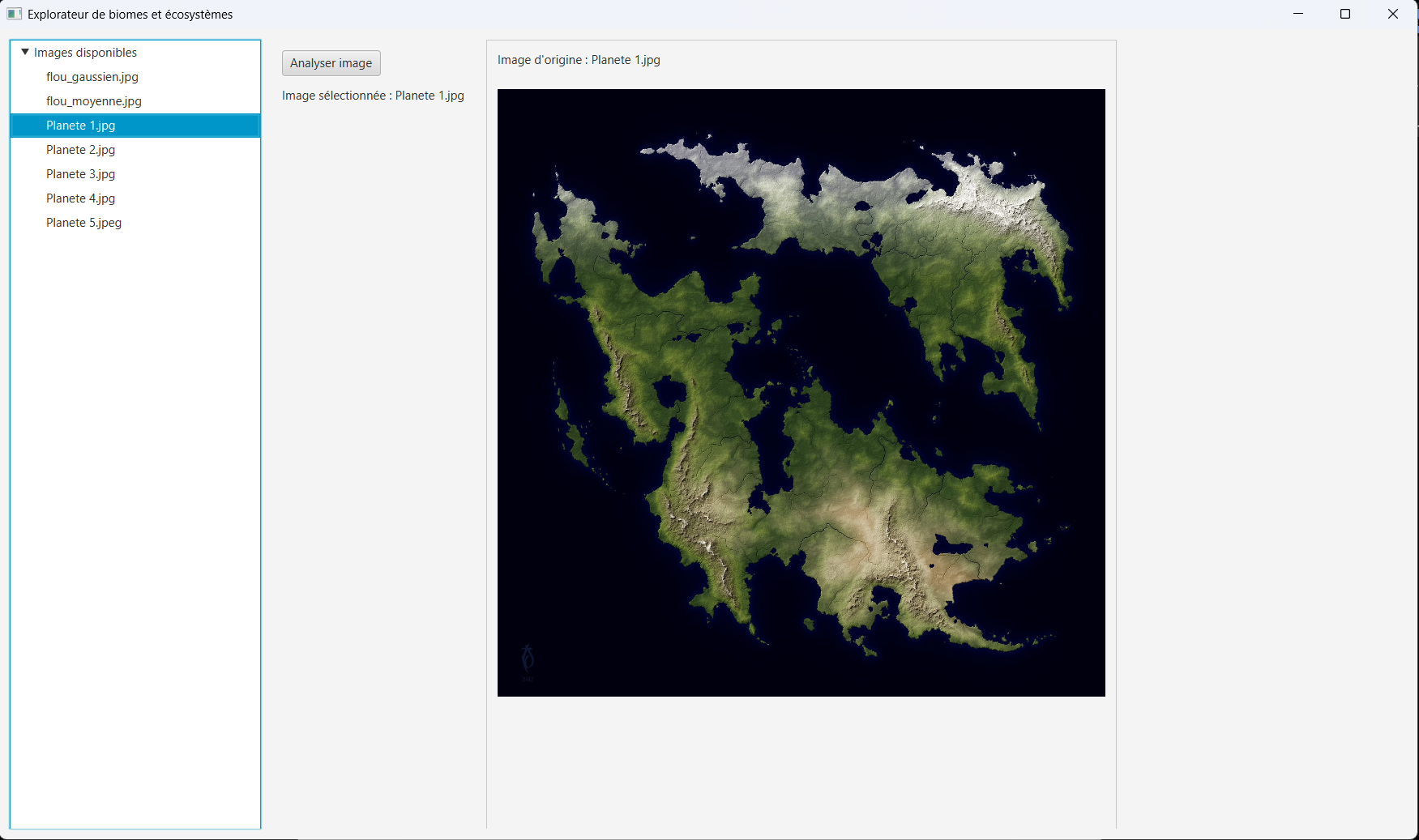


Figure - Menu de l'application

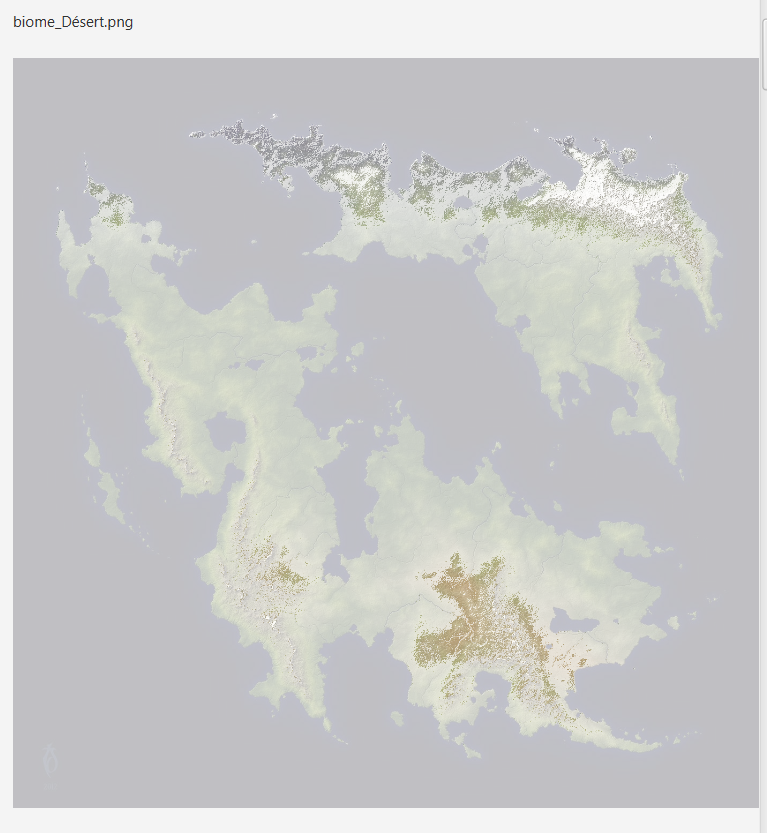


Figure - Exemple de résultat pour le désert sur la planète 1

Présentation de l’algorithme DBSCAN

DBSCAN (pour Density-Based Spatial Clustering of Applications with Noise) est un algorithme de regroupement de données basé sur la densité. Contrairement à d'autres méthodes comme K-Means, il n’a pas besoin de connaître le nombre de clusters à l’avance. Il détecte automatiquement les zones où les points sont suffisamment proches les uns des autres, et les regroupe. Les points trop isolés sont considérés comme du bruit et exclus des clusters.

**L’algorithme repose sur deux paramètres :**

* **ε (epsilon)** : la distance maximale pour que deux points soient considérés comme voisins ;
* **minPts** : le nombre minimum de voisins qu’un point doit avoir pour être considéré comme un “point cœur”.

Un cluster est ensuite formé en reliant entre eux les points qui sont accessibles à partir d’un point cœur, via une chaîne de voisins directs.

Limites de l’implémentation naïve

Dans une version classique, DBSCAN compare chaque point à tous les autres pour trouver ses voisins. Cette méthode simple devient vite très lente dès que le nombre de points augmente, car sa complexité est en O(n²). Pour des jeux de données volumineux, comme des images haute résolution ou des cartes géographiques, ce temps de calcul devient un vrai problème.

**Optimisation par grille spatiale**

Pour améliorer les performances, on utilise une grille spatiale. L’idée est de découper l’espace en petites cases carrées (dont la taille est liée à ε) et d’associer chaque point à une case. Ainsi, lorsqu’on cherche les voisins d’un point, on n’a plus besoin de vérifier tous les autres points, mais seulement ceux qui se trouvent dans la même case ou dans les cases voisines. Cela réduit énormément le nombre de comparaisons.

Grâce à cette méthode, on passe d’une complexité en O(n²) à quelque chose de beaucoup plus efficace, souvent proche de O(n), surtout si les points sont bien répartis dans l’espace.

**Résultats**

Avec cette optimisation, DBSCAN devient beaucoup plus rapide, sans perdre en précision. Il détecte les mêmes clusters que dans la version classique, mais en un temps beaucoup plus court. On peut donc l’appliquer à des jeux de données très larges, comme des images entières ou des cartes de biomes, tout en gardant une bonne réactivité.

Présentation de l’algorithme OPTICS

OPTICS est un algorithme de clustering hiérarchique basé sur la densité. Contrairement à DBSCAN, il ne produit pas directement des clusters, mais une structure ordonnée des points (une sorte de "profil de densité") qui permet ensuite d’extraire des clusters à différentes échelles de densité.

L’algorithme repose sur deux concepts clés :

* **La distance de portée** (*reachability distance*), qui mesure à quel point un point est accessible depuis un autre selon la densité locale.
* **La distance cœur** (*core distance*), qui est la plus petite distance à laquelle un point possède un minimum de voisins requis (paramètre minPts).

OPTICS permet d’identifier des clusters de formes complexes et de densités variables, mais son principal inconvénient est **son coût computationnel élevé**, notamment dû au calcul de nombreuses distances.

L’implémentation standard implique de parcourir l’ensemble des points pour calculer les distances entre voisins, ce qui a un coût en **O(n²)** en l’absence de structures d’indexation.

**Optimisation via une grille spatiale**

Pour améliorer les performances, nous avons introduit une **structure de grille spatiale** pour réduire le nombre de comparaisons nécessaires lors de la recherche des voisins d’un point.

L’idée consiste à :

* Diviser l’espace en **cellules de taille fixe** (déterminée par la valeur de ε, la distance maximale pour considérer un voisinage).
* Associer chaque point à une cellule.
* Lors de la recherche de voisins, ne considérer que les points dans **la cellule courante et ses cellules adjacentes**.

Cette approche permet de réduire drastiquement le nombre de comparaisons, en passant d’une complexité théorique en **O(n²)** à une complexité proche de **O(n)** dans le cas de données bien réparties.

**Résultats et bénéfices**

Grâce à cette optimisation :

* Le temps d’exécution de l’algorithme a été considérablement réduit, surtout sur des jeux de données volumineux.
* L'optimisation n’a pas d’impact négatif sur la qualité des clusters détectés, car les relations de voisinage sont conservées dans la grille.