Modélisation procédurale de terrain avec végétation et routes

Par Julien Fleckinger et Cassandra Breton

Plan

But du projet

Le projet consistait à générer un terrain en travaillant sur 2 des 3 sujets qui étaient l'érosion, la végétation et les routes. Nous avons choisi de travailler sur la végétation et les routes.

Classes de bases

Tout d'abord nous avons créé les classes Box (boite englobante) et Array (permettant de créer une grille) afin de pouvoir créer notre classe ScalarField. Nous avons aussi fait une classe image (qui charge ou génère des images), Maillage (gestion de maillage), Render (pour le rendu 3D), Vector (pour tous ce qui est gestion de points) et Terrain (regroupant tous les résultats des différents calculs).

Notre classe ScalarField nous permet de calculer des gradients, d'interpoler avec les méthode triangulaire et bilinéaire, de lire des images et de normaliser des valeurs.

La classe HeightField a été implémenté sur la base de la classe ScalarField afin de pouvoir créer le maillage résultat, de pouvoir modifier la taille de la grille, mais surtout de pouvoir calculer la slope, le drainage area, la wetness index et le power stream.

Noise

Nous avons développé une fonction permettant de créer un terrain grâce à une fonction noise.

Le terrain qui suit a été créé avec les fréquences [5000, 2450, 1220, 550], avec les [300, 140, 65, 30] et une grille de 100 par 100 qui a pour taille réelle 10.000 x 10.000.

Notre fonction noise suit le calcule d'un ridge-noise.

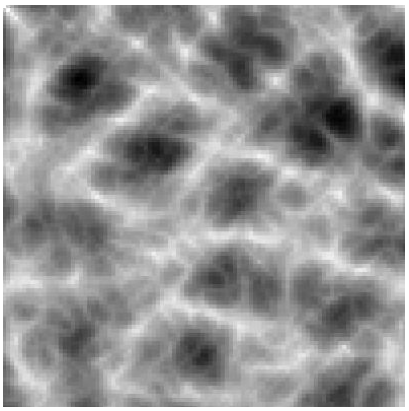


Figure 1 – Terrain généré par notre Fonction noise

Slope

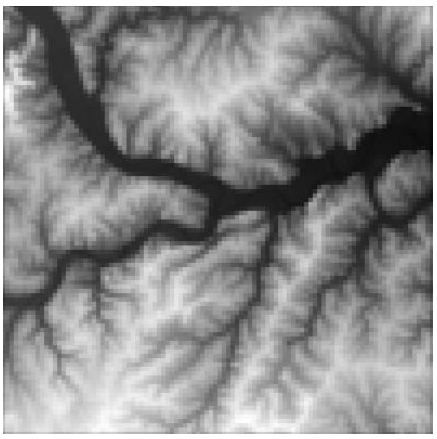


Figure 2 – Terrain de base

Figure 3 – Slope calculée à partir du terrain donné

Afin de calculer la slope on calcule simplement la norme du gradient en chaque point de notre terrain en prenant garde aux points se trouvant sur des extrémités.

Calcul de gradient :

$$\vec{\nabla} T(x,y,z) = \left(\frac{\partial T}{\partial x}(x,y,z), \frac{\partial T}{\partial y}(x,y,z), \frac{\partial T}{\partial z}(x,y,z) \right)$$

Drainage area

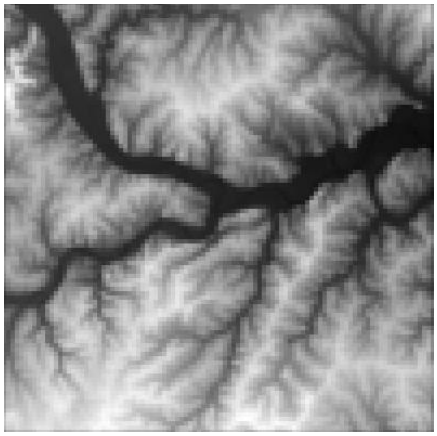


Figure 4 – Terrain de base

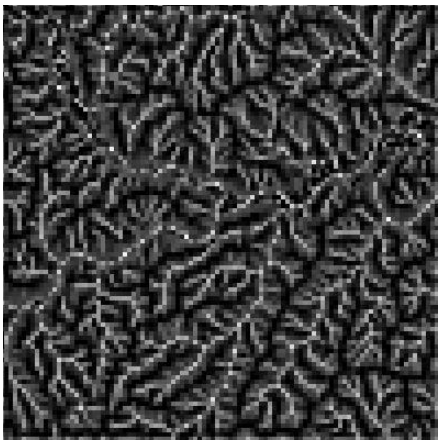
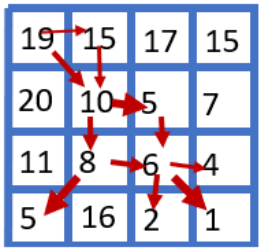


Figure 5 – Drainage area calculée à partir du terrain donné

Le drainage area est calculé en fonction d'une valeur d'écoulement et d'une position. Ensuite, on distribue cette valeur afin de savoir où ira l'eau.



Distribution de l'eau à tous les voisins dont l'altitude est inférieure avec des coefficients de pondération liés à la différence

Wetness index

Figure 6 – Slope

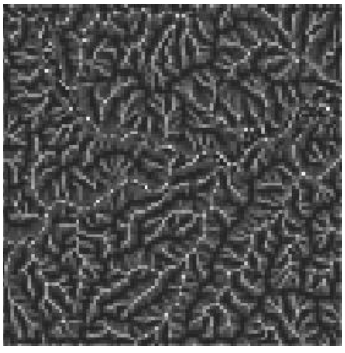


Figure 7 – Drainage area

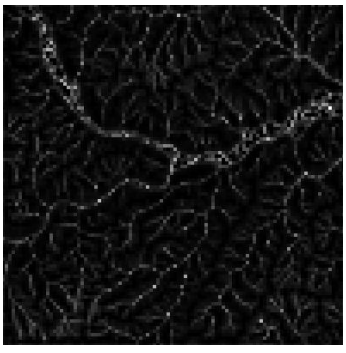


Figure 8 - Wetness index calculé

Le wetness index est calculé grâce au ScalarField contenant les valeurs de slope et celui contenant les valeurs de drainage area et une valeur k qui permet de choisir l'importance de la slope.

On fait donc pour chaque point p :

$$\text{Wetness (p)} = \frac{\log(drainage(p))}{1+k*slope(p)}$$

Power stream

Figure 9 – Slope

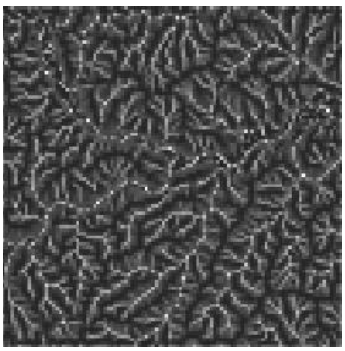


Figure 10 – Drainage area

Figure 11 - Streampower calculé

Tout comme le wetness index, on a besoin du ScalarField contenant les valeurs de slope et celui contenant les valeurs de drainage area.

Ensuite, on fait pour chaque point :

$$\text{Powerstream (p)} = slope(\,p\,) * \sqrt{drainage\,(\,p\,)}$$

Végétation

Notre simulation de végétation nécessite le ScalarField contenant la slope, celui contenant la hauteur et celui avec le wetness index.

Figure 12 – Slope

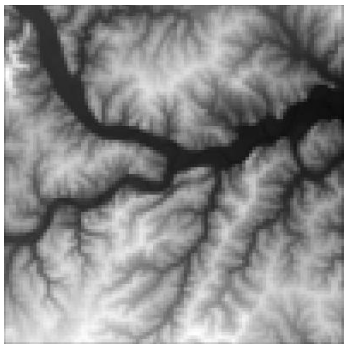


Figure 13 – Hauteur

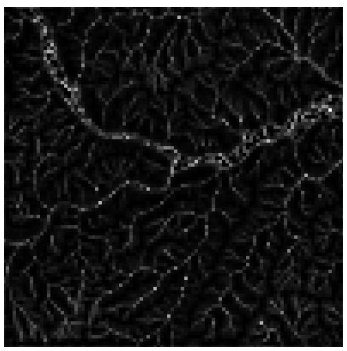


Figure 14 - Wetness index

Figure 15 – Végétation calculée à des hauteurs,
des slopes et du wetness index

Les classes crées pour la simulation de végétation sont Veget (dans laquelle est les données de l'arbre sont stockée : l'âge, l'espérance de vie, la position, la canopée, la valeur de développement, celle de maturité, la hauteur, le type de l'arbre et toutes les valeurs de contraintes d'existence, telles que les valeurs min et max de wetness, de slope, d'altitude, de taille et de canopée) et Foret (qui stocke les arbres, la densité de végétation et permet de lancer la simulation).

Nous avons deux types d'arbres : des sapins (en vert) et des pommiers (en rouge). Le sapin pouvant pousser sur des slopes plus pentues que le pommier, mais vie aussi plus longtemps et est plus haut. De même, il vit à des altitudes plus élevées et est plus grand. Enfin, sa canopée est plus grande.

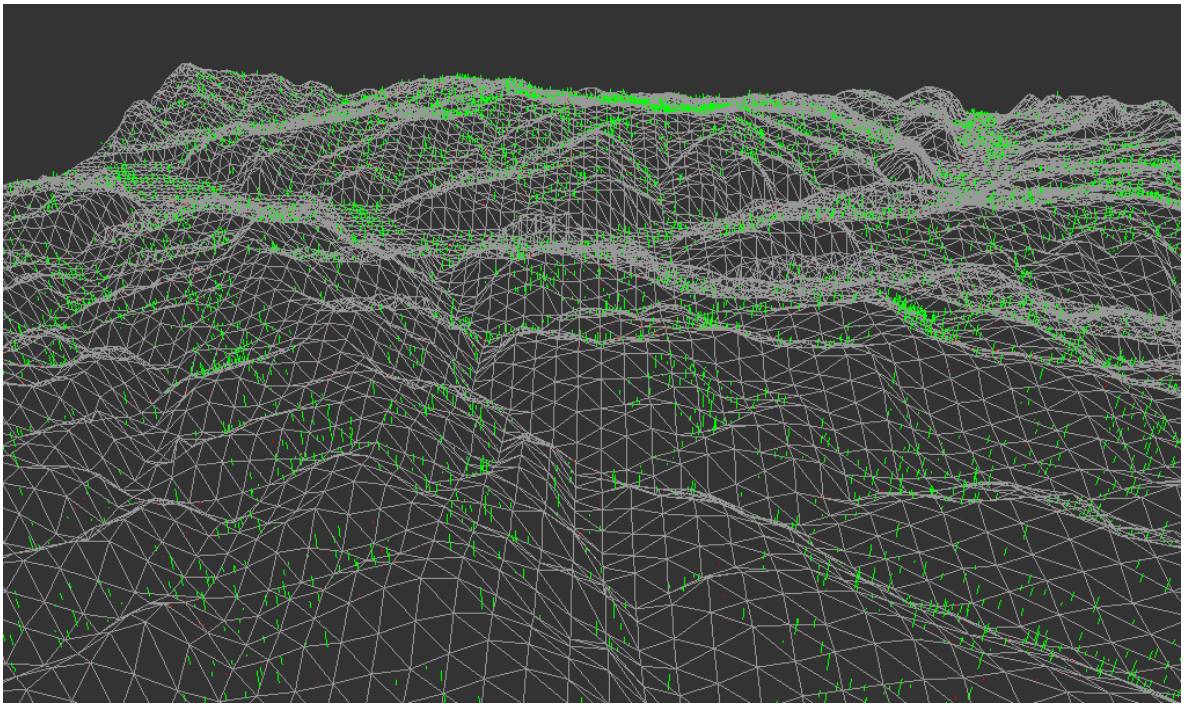


Figure 16 – Végétation initiale (vert = sapin, rouge = pommier)

Notre génération de forêt débute avec un nombre de naissance et de mort à 0.

On commence alors par regarder pour chaque arbre s'il doit mourir, pour ce faire, on regarde si son âge est supérieur à la moitié de son espérance de vie. Dans ce cas, on lance un jet de probabilité qu'il meurt et si tel n'est pas le cas, on diminue sa chance de survivre la prochaine fois. Ensuite, s'il est assez développé on l'agrandie et on met à jour sa canopée.

Par la suite, on cherche maintenant à savoir s'il y a eu des naissances, pour ce faire, si l'arbre est à maturité, on lance un aléatoire pour savoir s'il y a une naissance. Et si tel est le cas, on ajoute un arbre.

On doit maintenant placer ces nouveaux arbres en fonction de la canopée des voisins et de l'affinité du nouvel arbre en termes de valeurs de slope, wetness index et hauteur aux positions visées par ces naissances.

On peut donc boucler sur un certain nombre d'années pour voir l'évolution de la région.

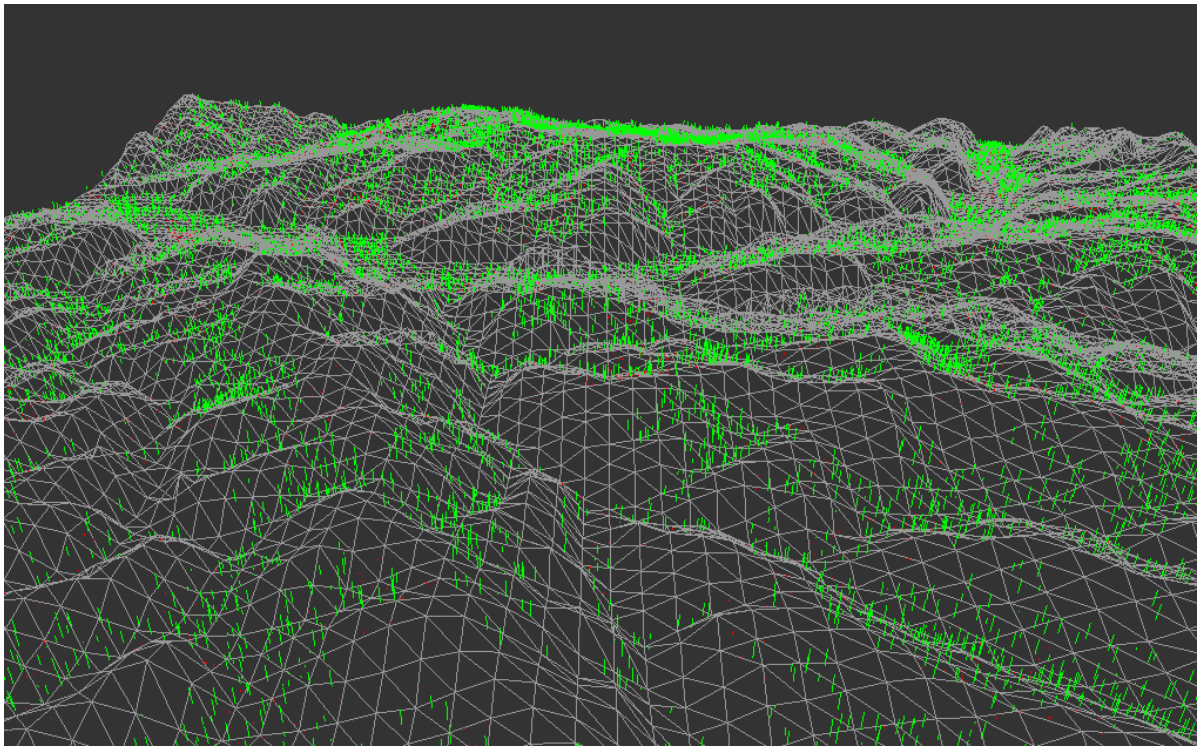


Figure 17 – Végétation après une simulation de 10 ans

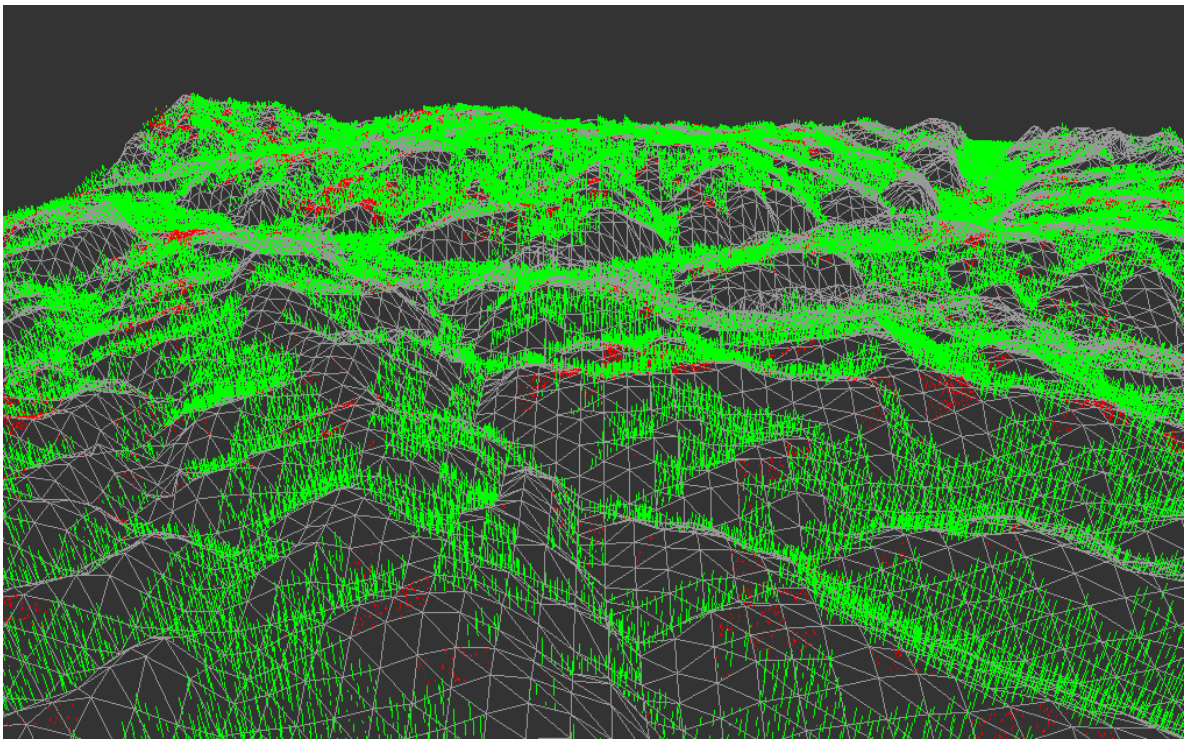


Figure 18 – Végétation après une simulation de 50 ans

Routes

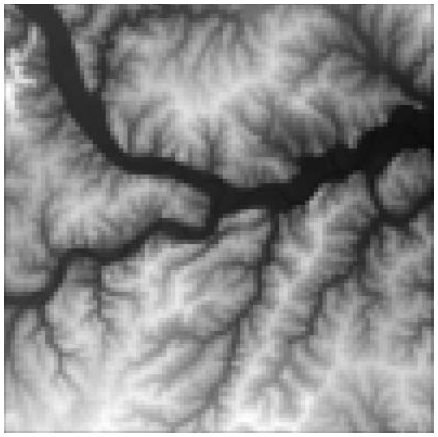
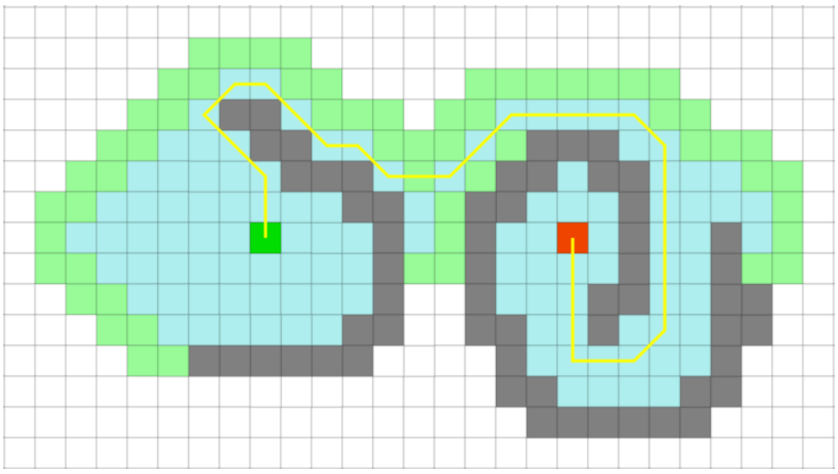


Figure 19 – Terrain de base

Figure 20 – Routes calculées (la position départ est en blanc à gauche et les positions d'arrivées en blanc à droite)

Pour la génération de routes, on s'est basée sur la slope et sur A*. Pour ce faire, nous avons créés deux classes : Case (contenant les informations nécessaires en chaque point pour le calcul de l'heuristique et du cout dans A*) et Route (qui lance A* sur le ScalarField de slope du terrain et qui donnera le chemin résultat et le cout de celui-ci).

Schéma (sans prise en compte de la hauteur de A*) :



Pour ce faire, on donne à A* nos deux positions (départ et objectif) et on applique un simple A* avec comme heuristique la distance et la valeur de slope (les valeurs de slopes trop extrêmes auront comme cout la valeur INFINITY).