Université Claude Bernard Lyon 1

Génération Procédurale de Terrain

Auteur :
Thomas Richard
Aymen Litim

5 mai 2021

Table des matières

Introduction

Ce projet a pour objectif le développement de classes et d'algorithmes nécessaire à la modélisation d'un terrain de façon procédurale.Ce projet va décrire et expliquer le fonctionnement de nos algorithmes ainsi que nos choix d'implémentation.

Chapitre 1

Génération du terrain

1.1 Définition

Un terrain peut-être modelisé comme suit :

$$T = (t_{ij}) \in \mathbb{R}^{h \times w}$$

L'objectif de la génération est de remplir T de valeurs $t_{i,j}$ représentant la hauteur du terrain aux coordonnées i, j, avec comme paramètres la hauteur h et longueur w, ainsi que l'altitude max alt_max. T doit remplir les propriétés suivantes :

$$\forall_{ij},\ t_{ij} >= 0 \wedge t_{ij} < alt\_max$$

Ressemble a un terrain

Il est à noter que cette modélisation ne permet pas de représenter le terrain comme un *volume*, nous ne traiterons donc que la surface.

1.2 Génération de la roche

Pour générer de la roche, nous allons utiliser un perlin noise, une fonction pseudo-aléatoire à 2 dimensions : $\mathcal{P}(x, y) = v$ avec $v \in [0, 1]$.

1.2.1 Simple

Pour chaque valeur $T_{i,j}$, nous avons commencé par lui assigner directement la valeur de $\mathcal{P}$.

$$\forall i, j\ \in\ \mathbb{R}\ i, j >= 0 \wedge i, j < t\ ,\ p_{i,j} = \mathcal{P}(i, j)$$

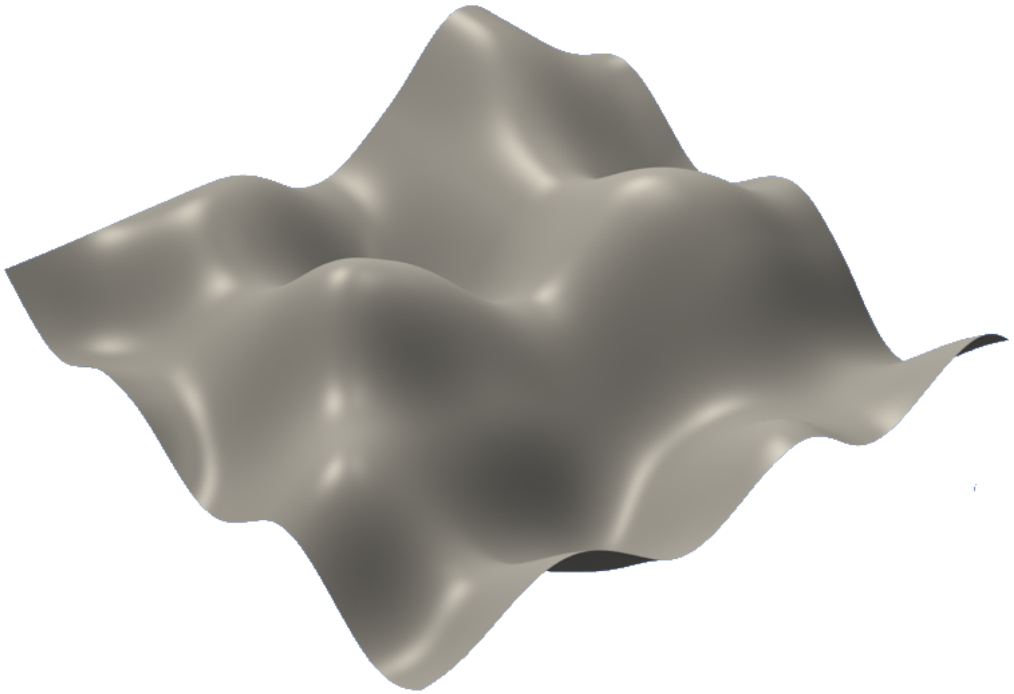


FIGURE 1.1 – Perlin noise simple

1.2.2 Somme

Nous avons ensuite sommé des perlin noise de fréquences différentes, ce qui nous a permis d'avoir un terrain moins homogène et légérement plus réaliste.

$$\forall i, j\ \in\ \mathbb{R}\ i, j >= 0 \wedge i, j < t\ ,\ t_{i,j} = \sum_{x=0}^{n} \mathcal{P}_x(i, j)$$

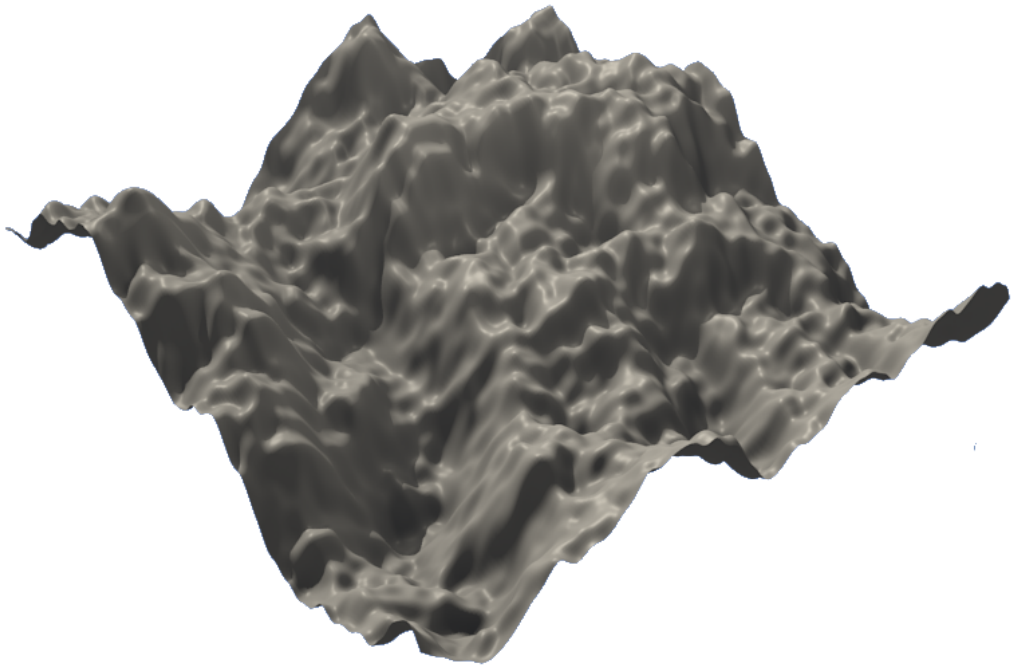


FIGURE 1.2 – Somme de perlin noise

1.2.3 Double Somme Ridged

Pour obtenir des crêtes, nous avons généré deux sommes de perlin noise avec des paramètres différents. Puis, nous avons récupéré le minimum des deux. Ainsi, à l'intersection des deux sommes, l'une des deux valeurs *annulera* l'autre et nous obtiendrons des crêtes.
Soit $\mathcal{P}s_1$ et $\mathcal{P}s_2$ deux sommes de perlin noise avec des paramètres différents.

$$\forall i,j \ \in \ \mathbb{R} \ i,j >= 0 \wedge i,j < t \ , \ t_{i,j} = min(\mathcal{P}s_1(i,j), \mathcal{P}s_2(i,j))$$

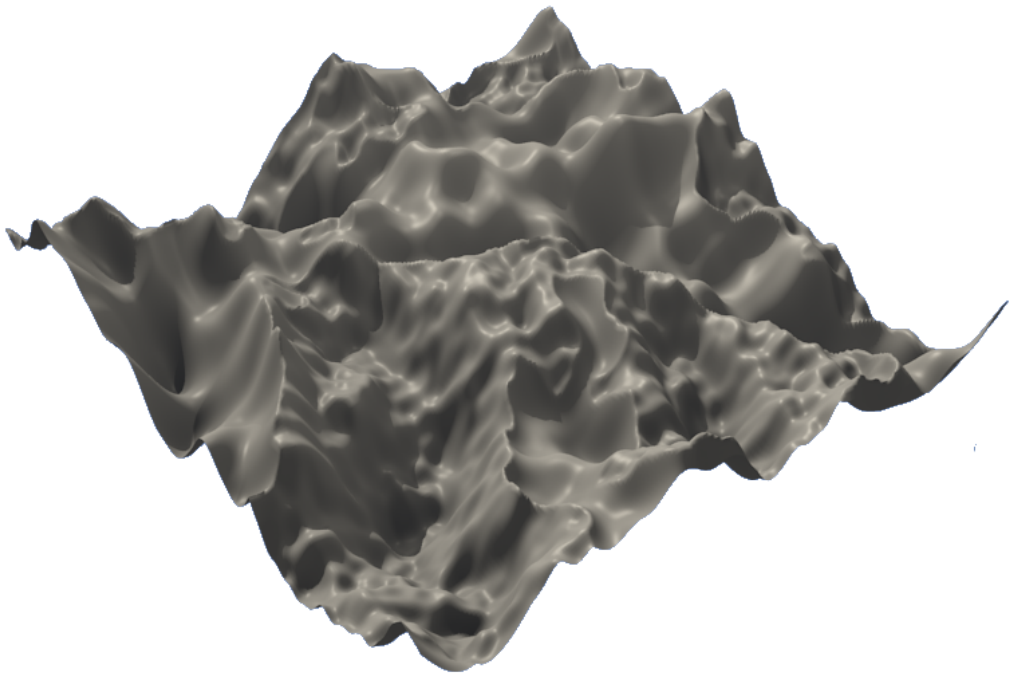
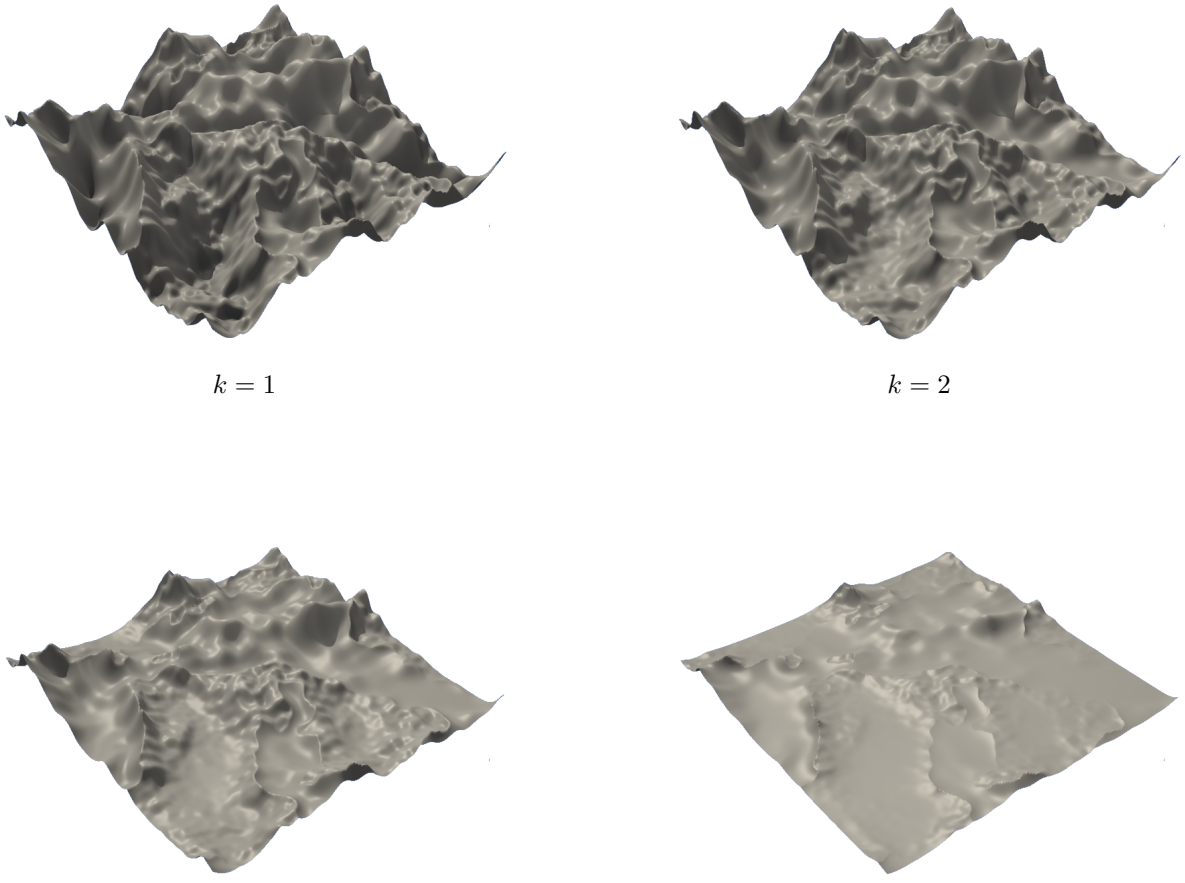


FIGURE 1.3 – Somme de perlin noise avec ridge

1.2.4 Attenuation

Nous avons ensuite ajouté un coefficient d'attenuation k : plus k augmente et plus les petites valeurs diminuent vite par rapport aux grandes. On obtiendra donc un terrain plus plat, tout en conservant des reliefs là où les valeurs étaient les plus hautes.

$$\forall i, j, k \ \in \ \mathbb{R} \ i, j >= 0 \wedge i, j < t \ \wedge k >= 1, \ t_{i,j} = (min(\sum_{x=0}^{n} \mathcal{P}s_1(i,j), \sum_{x=0}^{n} \mathcal{P}s_2(i,j)))^k$$



$k = 1$ $k = 2$

$k = 3$ $k = 5$

FIGURE 1.4 – Modification de k

1.3 Génération des sédiments

A ce stade, nous avons un terrain à un instant t, nous allons maintenant le faire évoluer. Le terrain subira deux types de phénomènes : l'érosion thermale et l'érosion hydraulique.

L'érosion est la transformation de roches en sédiments, il est donc nécessaire d'ajouter un concept de couches pour modéliser différentes matières. Nous avons choisi le modèle bi-couche suivant :

— $\mathcal{B}_{i,j}$, la carte de hauteur de la roche (*bedrock*)
— $\mathcal{S}_{i,j}$, la carte de scalaire de sédiments

La surface finale du terrain T est donc redéfinie comme suit :

$$T_{i,j} = \mathcal{B}_{i,j} + \mathcal{S}_{i,j}$$

Toutes les valeurs de T calculées précédemment deviennent ainsi les valeurs de $\mathcal{B}$.

1.3.1 Initialisation érosion thermique

Nous allons utiliser la carte d'illumination $\mathcal{I}$ et la carte des pentes $\mathcal{S}l$ pour éroder plus ou moins le terrain en fonction de son exposition et de sa pente.

$$\mathcal{B}_{i,j} = \mathcal{B}_{i,j} - \mathcal{T}\mathcal{S}l_{i,j}\mathcal{I}_{i,j}$$
$$\mathcal{S}_{i,j} = \mathcal{S}_{i,j} + \mathcal{T}\mathcal{S}l_{i,j}\mathcal{I}_{i,j}$$

avec

$\mathcal{T}$ = taux d'érosion

1.3.2 Initialisation Erosion hydraulique

Nous allons utiliser la carte d'humidité $\mathcal{H}$ et la carte des pentes $\mathcal{S}l$ pour éroder plus ou moins le terrain en fonction de son aire de drainage et de ca pentes.

$$\mathcal{Q}a_{i,j} = \mathcal{T}\mathcal{H}_{i,j}\mathcal{S}l_{i,j}$$

$SI\ \mathcal{S}_{i,j} \leq \mathcal{Q}a_{i,j}\ ALORS$

$$\mathcal{R}este = \mathcal{Q}a_{i,j} - \mathcal{S}$$
$$\mathcal{S}_{i,j} = 0$$
$$\mathcal{B}_{i,j} = \mathcal{B}_{i,j} - \mathcal{R}este$$

$SINON$

$$\mathcal{S}_{i,j} = \mathcal{S}_{i,j} - \mathcal{Q}a_{i,j}$$

1.3.3 Stabilisation

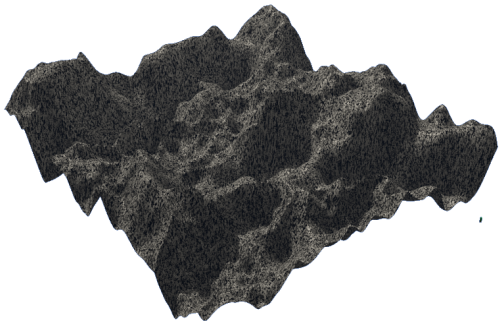
$\mathcal{S}$ est maintenant initialisée. Néanmoins, la couche de sédiments est une matière friable soumise à des phénomènes d'écoulement. Il faut donc passer par une étape de stabilisation.
On vérifie si une pile de sédiments est stable de la manière suivante :

Algorithm 1 isStable

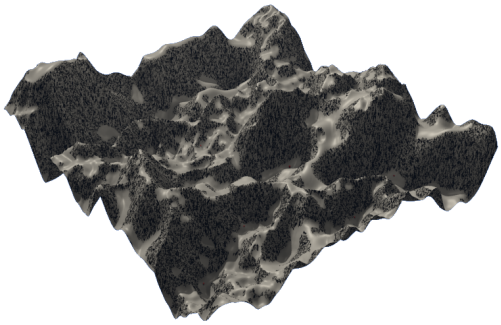
Require: $x, y >= 0 \wedge x, y < size, NeighhborN$
  if $Sem_{x,y} \leq 0$ **then**
    return $true$
  end if
  $value \leftarrow Bm_{x,y} + Sem_{x,y}$
  if $nbLowerNeighbor(value, N) = 0$ **then**
    return $true$
  end if
  for $LowerNeighborv : Neighbor$ **do**
    $diff \leftarrow heightDiff(x, y, v)$
    $angle \leftarrow \frac{diff - angle}{sizeCells_y}$
  end for
  if $angle \geq \tan 45$ **then**
    return $false$
  end if
  return $true$

Algorithm 2 Stabilisation

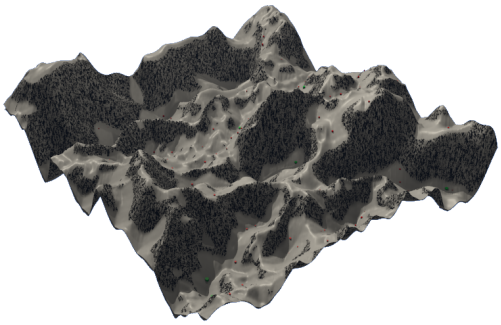
```
while Q is not empty do
  x ← random value
  y ← random value
  p ← Q_{x,y}
  delete Q_{x,y}
  if p is not stable then
    Q.addNeighbours(x,y)
    fall(x,y)
  end if
end while
```



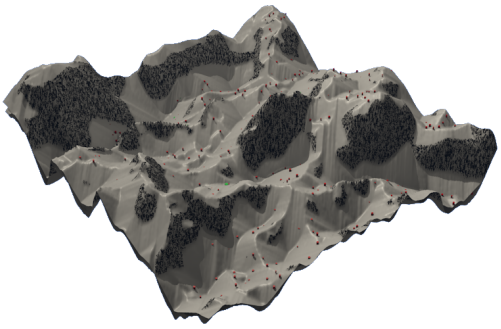
$\mathcal{T} = 0$



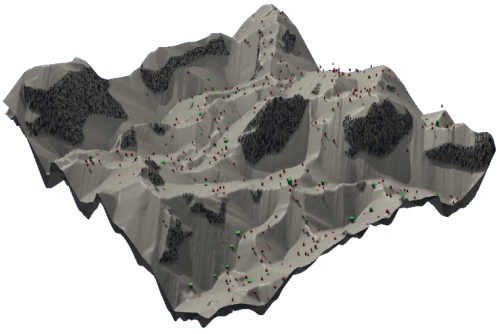
$\mathcal{T} = 0.01$



$\mathcal{T} = 0.1$

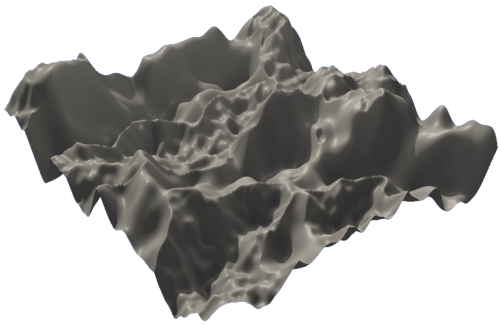


$\mathcal{T} = 0.5$

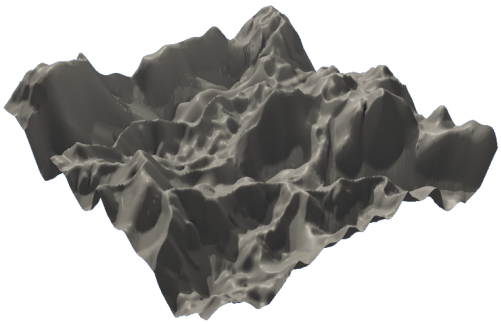


$\mathcal{T} = 1$

Résultats érosion thermique



Avant érosion hydrolique



Après érosion hydrolique

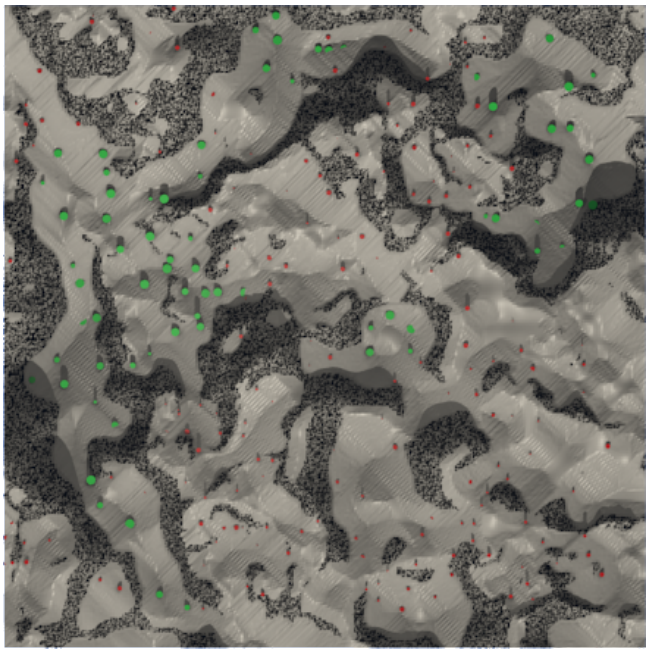
Résultats érosion hydrolique

1.4 Calcul des maps

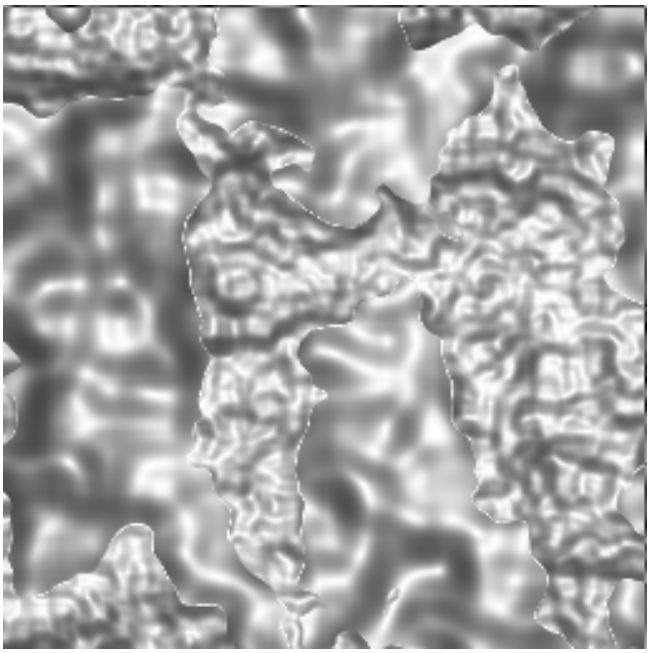
La suite des traitements qui sera appliquée sur le terrain implique le calcul des différentes composantes physiques de l'environnement en un point p. Cette partie traite de la méthode de calcul de ces valeurs.

1.4.1 Pentes

Une donnée essentielle est la pente en un point. Soit $\vec{n}$ la normale au terrain et $\vec{up}$ l'axe y, la pente $\nabla = n \cdot up$, avec ∇ un angle en radian.
Une autre méthode est de calculée le gradient des valeurs



Rendu



Carte de pente

1.4.2 Illumination

L'illumination $\mathcal{I}$ représente l'exposition en un point p, autrement dit, la quantité de lumière que reçoit le point. Soit $\mathcal{N}$ le nombre de rayons lancés depuis p et $\mathcal{L}$ leurs longueurs, on a :

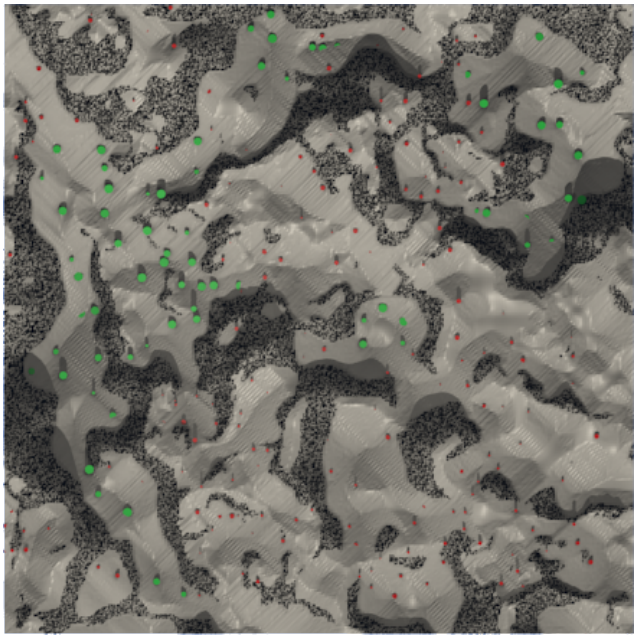
$$\mathcal{I} = \frac{\sum_{i=0}^{\mathcal{N}-1} \hat{n} \cdot (p + \mathcal{L} * \vec{d})}{\mathcal{N} - 1} \tag{1.1}$$

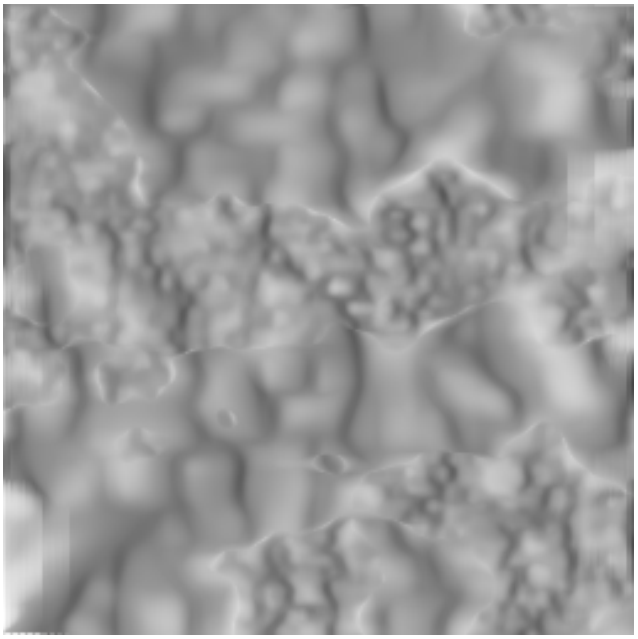
avec

$$\vec{d} = sin(\frac{2i\pi}{\mathcal{N}}), y, cos(\frac{2i\pi}{\mathcal{N}}) \tag{1.2}$$

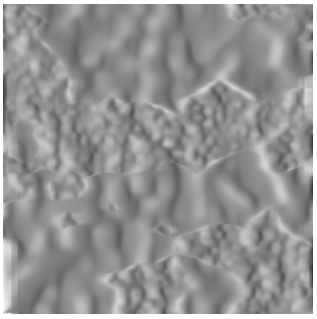
d_y dépend du terrain, on cherche la coordonnée qui garde le rayon tangent au terrain, donc pas forcément le plus haut, mais celui qui a le plus grand angle avec l'axe z.

Les deux paramètres $\mathcal{N}$ et $\mathcal{L}$ ont un fort impact sur la qualité du résultat. Avec $\mathcal{L}$ trop forte, l'illumation est calculée sur une trop grande zone, et un $\mathcal{N}$ insuffisant entraine un mauvais échantillonage.

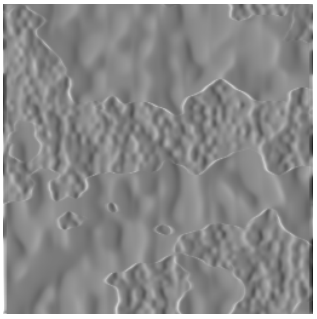




Rendu final
$\mathcal{N} = 10$ $\mathcal{L} = 2$

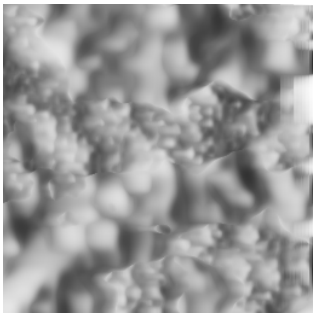


$\mathcal{N} = 10$ $\mathcal{L} = 1$



$\mathcal{N} = 10$ $\mathcal{L} = 0.5$

FIGURE 1.5 – Rayons trop court, calcul trop local



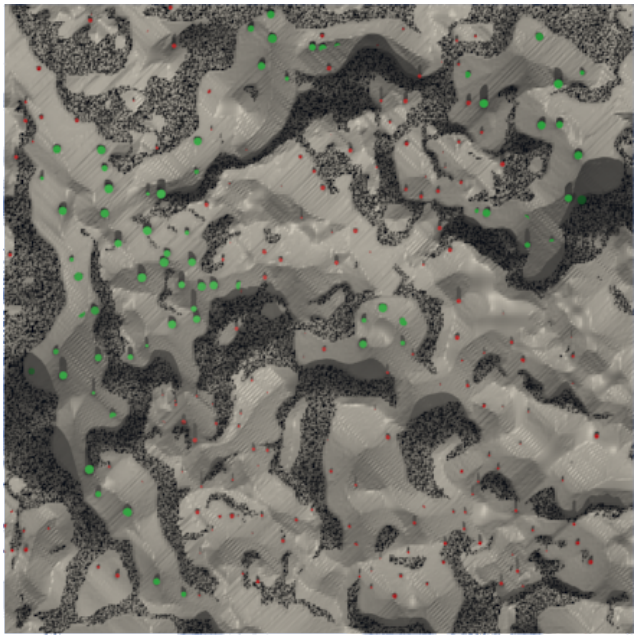
$\mathcal{N} = 4$ $\mathcal{L} = 2$

$\mathcal{N} = 1$ $\mathcal{L} = 2$

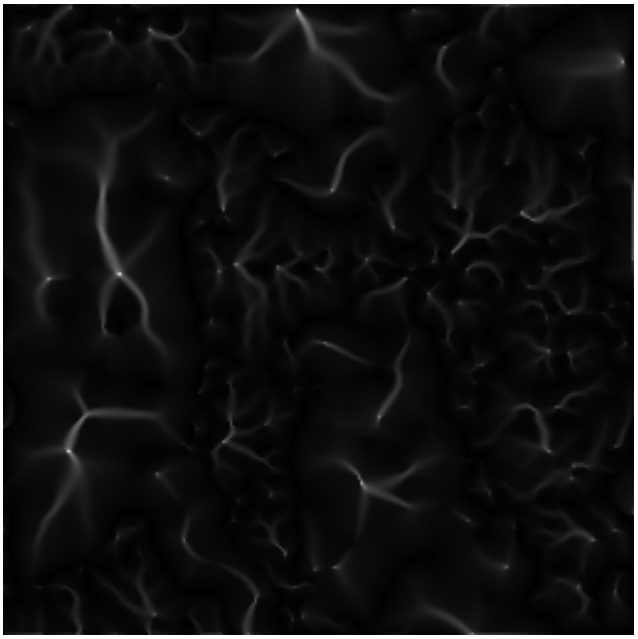
FIGURE 1.6 – Trop peu de rayons, lumière *oriente* car trop peu de directions

1.4.3 Humidité

Pour calculer $\mathcal{H}$ la carte d'humidité, on initialise toutes ses valeurs à 1, puis on la parcours par hauteur décroissante. Puis, pour chaque voisin, on lui ajoute une partie de la valeur courante, pondéré par sa différence de hauteur. On a ainsi un *ecoulement* de valeurs qui modélise l'humidité.



Terrain



Carte d'humidité

Chapitre 2

Ecosystème

L'enjeu de cette partie est de parvenir à simuler le développement d'espèces végétale dans un environnement numérique.

2.1 Répartition

La génération de densité est faite grâce aux courbes de réponse $\mathcal{C}$ de chaque espèce e. Il y a une courbe de réponse pour chacun des paramètres physiques du terrain.

$$\mathcal{D}_{i,j} = \mathcal{C}_{\mathcal{S}}(\mathcal{S}_{i,j}) * (\mathcal{C}_{\mathcal{I}}(\mathcal{I}_{i,j}) + \mathcal{C}_{\mathcal{H}}(\mathcal{H}_{i,j})) \tag{2.1}$$

avec :

$\mathcal{C}_\lambda$ = courbe de réponse tel que $\mathrm{C}_\lambda(x) \in [0,1]$

$\mathcal{I}_{i,j}$ = carte d'illumination

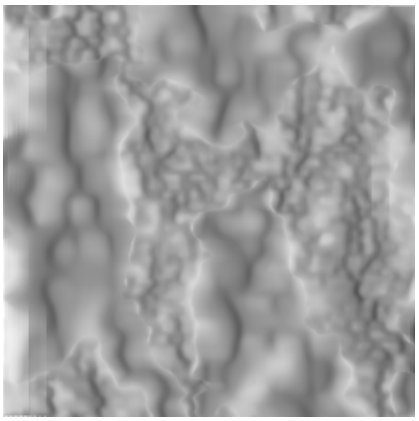
$\mathcal{H}_{i,j}$ = carte d'humidité

$\mathcal{S}_{i,j}$ = carte de sédiment

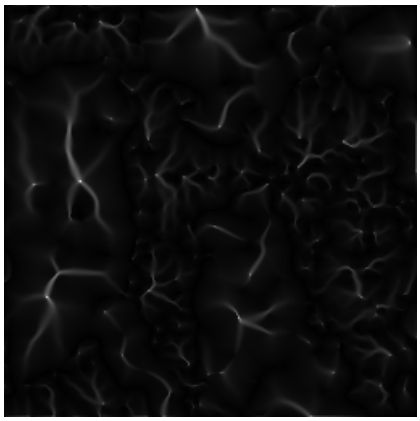
Autres méthodes :

Moyenne : $\mathcal{D}_{i,j} = (\mathcal{C}_\mathcal{S}(\mathcal{S}_{i,j}) + \mathcal{C}_\mathcal{I}(\mathcal{I}_{i,j}) + \mathcal{C}_\mathcal{H}(\mathcal{H}_{i,j}))/3$
Addition : $\mathcal{D}_{i,j} = \mathcal{C}_\mathcal{S}(\mathcal{S}_{i,j}) + \mathcal{C}_\mathcal{I}(\mathcal{I}_{i,j}) + \mathcal{C}_\mathcal{H}(\mathcal{H}_{i,j})$
Problème : Pas de valeurs *critiques*, $\forall i, j \ D_{i,j} > 0$.

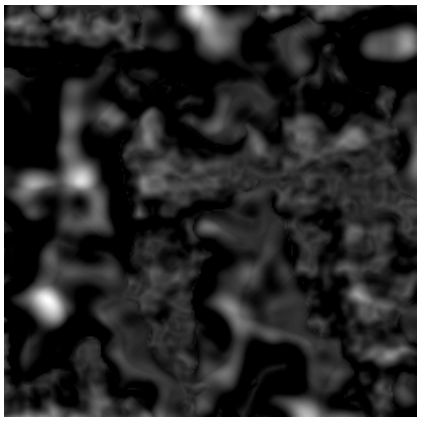
Min : $\mathcal{D}_{i,j} = \mathcal{C}_\mathcal{S}(\mathcal{S}_{i,j}) + \mathcal{C}_\mathcal{I}(\mathcal{I}_{i,j}) + \mathcal{C}_\mathcal{H}(\mathcal{H}_{i,j})$
Problème : Pas d'inter-influence des courbes.



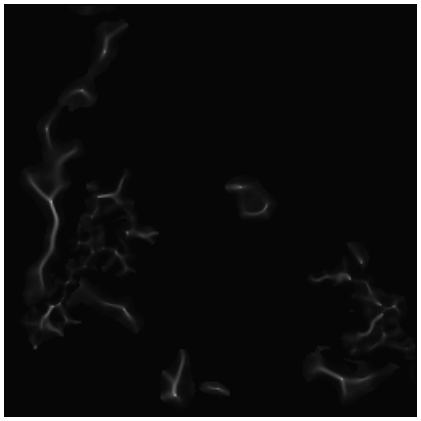
$\mathcal{C}_\mathcal{I}(\mathcal{I}_{i,j})$ +



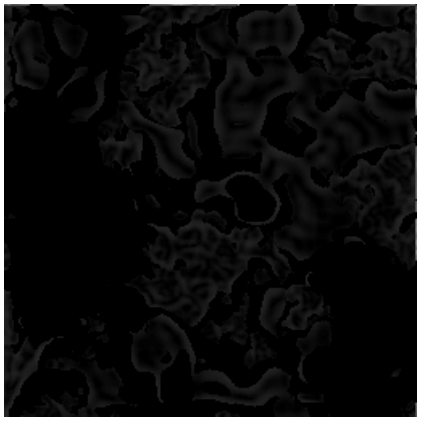
$\mathcal{C}_\mathcal{H}(\mathcal{H}_{i,j})$ +



$\mathcal{C}_\mathcal{S}(\mathcal{S}_{i,j})$



$\mathcal{D}_{e1}$



$\mathcal{D}_{e2}$

La gestion de toutes ces fonctions de réponses ainsi que la génération d'une carte de densité concentrée dans la classe abstraite *vegetation*. Il suffira ensuite de réinstancier une classe par espèce incluant ses fonctions de réponse unique. Notre projet contient deux instanciations de *vegetation* : *tree* et *bush*.

2.2 Distribution

La distribution des individus en fonction des cartes de densités se fait de façon classique : avec P mon ensemble de positions de végétaux d'une espèce e, et $D_{i,j}$ sa densité.

Algorithm 3 Ajoute arbre

while $nbBeforeRejection > 0$ **do**
  $xRand, yRand \leftarrow randomPosition$
  $rand \leftarrow randoValue \in [0\ ,\ 1]$
  if $D_{xRand,yRand} > rand$ **then**
    if $!is\_Valid(D_{xRand,yRand})$ **then**
      $P \bigcup (xRand, yRand)$
    end if
  end if
  $nbBeforeRejection- = 1$
end while

Algorithm 4 isValid

Require: pos
for $Chaquevoisin$ **do**
  $dist \leftarrow vegetal.pos - voisin$
  if $|dist| < vegetal.radius$ **then**
    return $False$
  end if
end for
return $True$

Une structure de données naïves entraîne une complexité en $\Theta(n^2)$, car sans mécanisme d'accès aux voisins on doit reparcourir toutes les positions.

La structure $Vegetation.h$ d'une espèce e contient donc 2 attributs, une liste de t positions de vegetaux p_i, et une structure accélératrice $\mathcal{A}_{i,j}$ qui découpe l'espace en grille et place un indice de p dans chaque case ce qui permet un accès direct aux voisins. $\mathcal{A}_{i,j}$ respecte la propriété suivante :

$$\forall i, j \quad diagonal(\mathcal{A}_{i,j}) = \lambda r \wedge \mathcal{A}_{i,j} \in [-1, t]$$

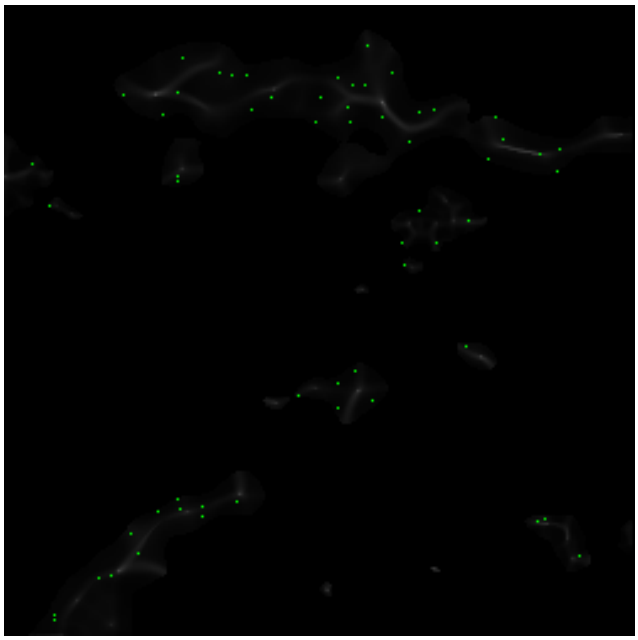
avec

$r =$ le radius de l'espèce e

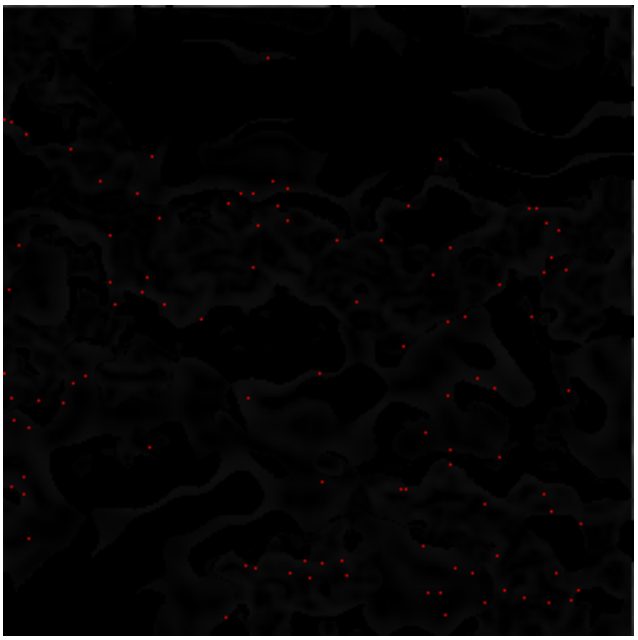
$\lambda =$ une valeur $\in [0,\ 1]$

Une augmentation de ε du radius est aussi gérée. Il suffit de fixer $diagonal(\mathcal{A}_{i,j}) = \varepsilon$, et de parcourir les $\frac{r}{\ }$ cases adjacentes pour accéder à tous les potentiels voisins.

Cette structure est cependant dans l'incapacité de rétirer une position déjà insérée dans la liste : toutes les valeurs de la grille accélératrice étant des indices, la suppression d'une valeur entraîne le décalage de toutes les suivantes.



Répartition arbres



Répartition buissons

2.3 Evolution

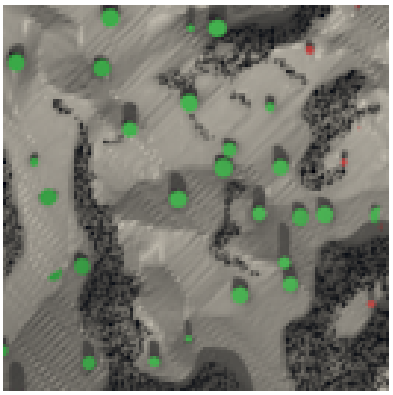
2.3.1 Croissance

Les données supplémentaires suivantes ont été ajoutées pour gérer la croissance d'un végétal :

- r_i : données jumelles à p_i qui indique le rayon du végétal
- ϵ : la valeur d'augmentation du radius à chaque pas de simulation
- a : un âge limite

Algorithm 5 croissance

Require:
for Chaque végétal d'index i **do**
if $!mort(i)$ **then**
$r_i += \epsilon$
if $r_i/\epsilon > a$ **then**
$i = mort$
end if
end if
end for



Différent rayons d'une même espèce

2.3.2 Compétition

Notre écosystème implémente la forme la plus simple de compétition :

$$r_1 > d_{v_1,v_2} \wedge r_1 > r_2 \implies v_2 = mort$$
$$r_2 > d_{v_1,v_2} \wedge r_2 > r_1 \implies v_1 = mort$$

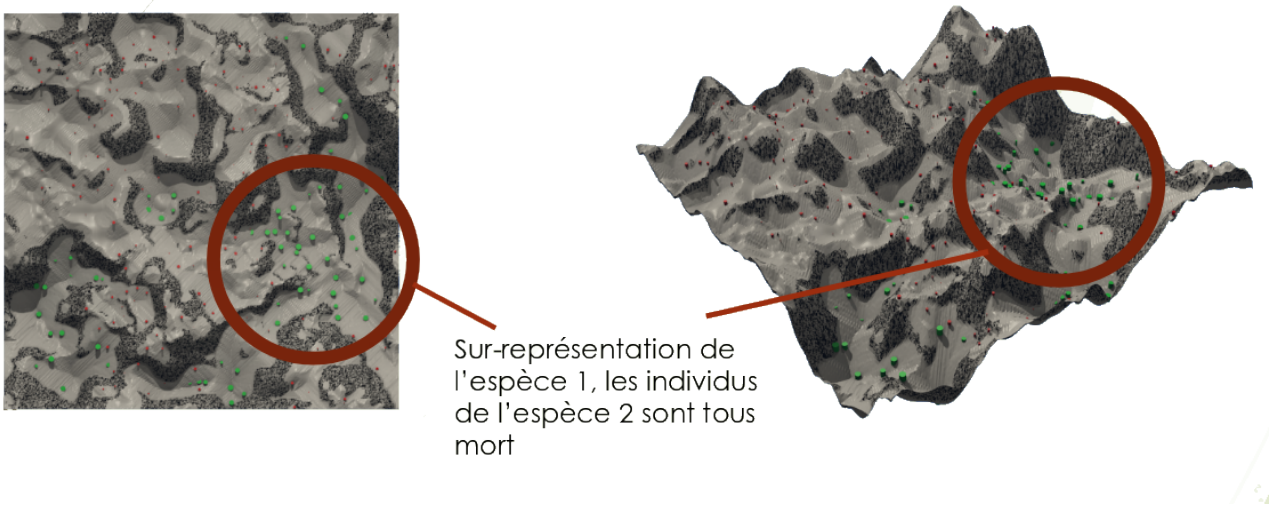
avec

$d_{i,j}$ = distance entre i et j

v_i = végétal i

r_i = le rayon du végétal i

Une autre forme de compétition implicite est également présente. La densité d'une espèce augmente autour d'un individu déjà présent, ce qui réduit les chances de présence d'une autre espèce



2.3.3 Mort

Comme décrit précédemment, la structure ne peux pas retirer de position. Un végétal mort est donc simplement défini comme $r_i = -1$.

La mort d'un individu peut provenir des deux causes décrites précédemment :
- Vieillesse
- Mort par "écrasement" d'un autre individu