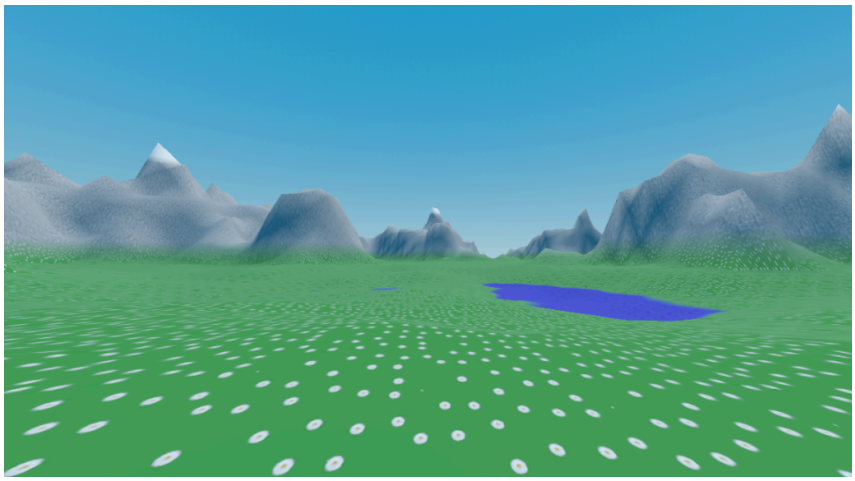


Étude bibliographique : Génération procédurale de monde 3D

Floriane Payen

2025-03-17

3A-SN-IMM - master PSMSC

Abstract

Ce rapport présente différentes méthodes dans le but de générer un monde 3D : de la création de la topologie du terrain à l'ajout de modèle 3D, sur ce même terrain pour le rendre plus réel. La topologie du terrain sera principalement définie par des bruits : le bruit de Perlin et le bruit de Voronoï. En quête d'un aspect plus réaliste qu'un simple terrain 3D, de la végétation, des bâtiments, des routes, etc. seront ajoutés afin de rendre le monde 3D généré plus vivant.

Table des matières

1. Introduction

La génération procédurale de monde 3D s'avère utile dans la création de jeux vidéo ou de simulation ou encore dans l'animation. Ce rapport va donc présenter comment créer une carte 3D en ajoutant des éléments pas à pas.

La première étape est de générer un terrain 3D. Les étapes suivantes sont l'ajout de textures, de végétations, de villages et de routes à ce terrain. Dans toutes les étapes de la génération, que ce soit pour la création du terrain ou pour l'ajout de végétations ou de bâtiments, un des défis majeurs est de réussir à créer un résultat cohérent, réaliste et varié. La prise en compte de l'érosion, l'ajout de végétation sur le terrain contribue grandement à le rendre plus réaliste.

Cette étude bibliographique va donc présenter différentes méthodes existantes pour générer un monde 3D cohérent et réaliste. Dans un premier temps, nous verrons comment générer la topologie du terrain, notamment grâce aux bruits de Perlin et de Voronoï. Ensuite, nous ajouterons de la couleur (par simple plaquage sur le terrain). Puis, nous verrons comment rendre ce monde plus vivant en ajoutant notamment de la végétation, des bâtiments et des routes.

Afin d'illustrer ce rapport, des exemples ont été généré à l'aide de Godot, un moteur de jeu.

2. Génération de la topologie du terrain

2.1. Champ de hauteur

Une des manières de générer procéduralement un terrain se fait grâce à la création d'un champ de hauteur (*heightmap*). Pour cela, on définit un plan, découpé selon une grille qui forme le maillage du plan et qui corresponds au champ de hauteur. En chaque point de cette grille, une altitude lui est attribuée, généralement déduit d'un bruit comme détaillé dans la suite du rapport. Le champ de hauteur correspond donc à une matrice regroupant pour chaque point du maillage son altitude, il est souvent représenté sous la forme d'une image avec l'utilisation des niveaux de gris pour symboliser la hauteur (Fig. 3).

Deux bruits seront principalement présentés : le bruit de Perlin et le bruit de Voronoï. Bien qu'ils semblent tout deux avoir été inventé dans un premier temps pour répondre à une problématique de création de texture, ils s'adaptent parfaitement à la création d'une *heightmap* et de manière plus générale à la génération d'un monde 3D.

2.1.1. Bruit de perlin

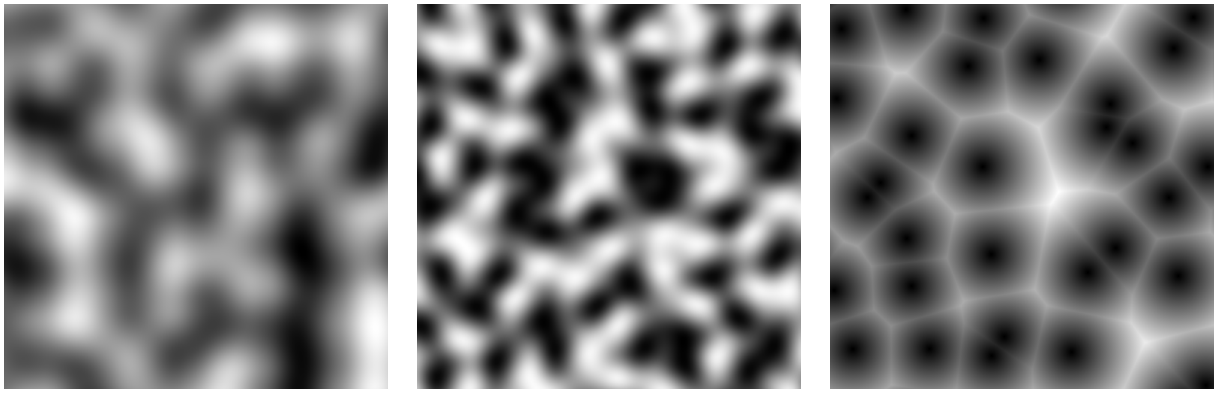
Dans le papier originel sur le bruit de Perlin [1], son auteur présente tout un tas de solution pour créer des textures réalistes : que ce soit de l'eau, des nuages, du feu ou encore des textures imitant le marbre. Cependant, ce bruit est aussi très adapté à la génération d'un champ de hauteur.

Concept de base

On considère une grille, par exemple de taille $n * n$, où un gradient orienté dans une direction aléatoire est associé à chaque point de la grille. Lorsque l'on souhaite calculer le bruit de Perlin en un point quelconque de cette grille, on calcule le produit scalaire entre son vecteur suivant x et son vecteur suivant y avec chacun des 4 points de la grille qui l'entoure. On réalise ensuite une interpolation des résultats trouvés pour chacun des 4 points [2]. C'est cette interpolation qui permet d'avoir des changements doux entre les valeurs de points dans un même voisinage. Cette propriété est pratique dans le contexte de génération de monde 3D. Pour pouvoir répéter d'une fois à l'autre le même bruit, les gradients ne sont pas choisis entièrement de manière aléatoire : ils sont sélectionnés parmi une liste de gradients définis (les valeurs conseillées sont 8 ou 16 en 2D). De plus, pour des raisons d'optimisations

de calculs, les gradients ont en général leurs composantes qui prennent uniquement leur valeur dans {-1;1;0} [2].

En associant à chaque valeur un niveau de gris, on peut représenter le bruit de Perlin sous la forme d'image (Fig. 3a).



(a) Bruit de Perlin. (b) *Simplex Noise.* (c) Bruit de Voronoï.

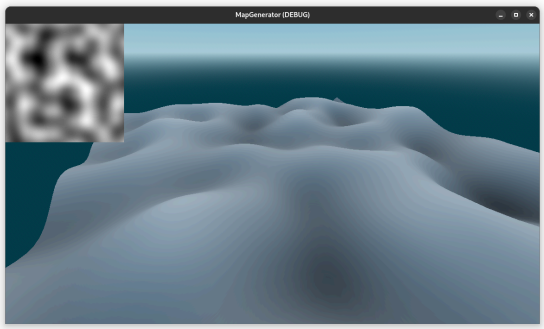
Fig. 3. – Exemples de bruit

Utilisations possibles

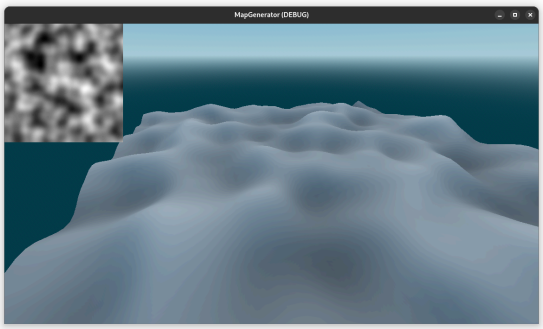
Utiliser un bruit pour générer le champ de hauteur du monde 3D semble donc prometteur. Afin de pouvoir modéliser des détails à différentes échelles, Perlin introduit dans [1] la formule suivante :

$$\sum_{f} \frac{\text{Noise}\left(\text{point} * 2^{f}\right)}{2^{f}}$$

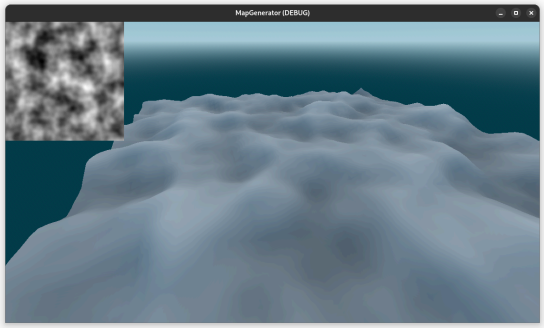
avec f un réel choisi pouvant s'apparenter à la fréquence du bruit et Noise la fonction appliquant le bruit de Perlin classique. Chaque élément de cette somme est appelé une *octave* : les différentes octaves vont servir à définir des niveaux de détails différents. Chaque octave a pour amplitude $\frac{1}{2^f}$ et pour fréquence 2^f. Ainsi, la topologie globale du terrain est plutôt représentée par la première octave, car elle possède la fréquence la plus faible et l'amplitude la plus élevée. Les octaves supérieures servent plutôt à définir des détails plus fins sur le terrain. Les plaines, les montagnes sont plutôt définies par la première octave, la granularité du terrain est plutôt définie par les dernières octaves. Inigo Quilez définit cette somme d'octaves comme un mouvement brownien fractionnaire (*Fractional Brownian Motion, fBm*) dans [3]. L'effet de l'ajout progressif d'octaves pour modéliser le terrain peut être visualisé sur la Fig. 4. On peut y voir en haut à gauche l'image représentent la somme d'octaves et sur le reste de l'image l'effet que cela peut avoir sur un terrain.



(a) octaves = 1



(b) octaves = 2



(c) octaves = 3

Fig. 4. – Bruits de Perlin.

Cependant, définir linéairement une altitude en fonction de la valeur générée par bruit de Perlin mène à des résultats qui ne sont pas très convaincants (Fig. 4). Comme expliqué par Sebastian Lague dans [4], une solution pour pallier ce problème consiste à appliquer, sur ce bruit, une fonction permettant de moduler les altitudes comme on le souhaite, comme par exemple la fonction carrée.

Ainsi, le bruit de Perlin peut permettre de définir des types de terrains : les zones d'eau, les zones montagneuses, les zones de plaines. Puis, on associe une plage des valeurs renvoyées par le bruit de Perlin à chacun des types de terrains que l'on veut définir. Suivant la taille de ces plages, les types de terrains seront plus ou moins prépondérant sur la carte générée (même si ce n'est pas proportionnel, car la répartition des valeurs prises par le bruit de Perlin n'est pas uniforme).

On peut ensuite imaginer affecter chaque type de terrain différemment : moduler l'altitude, rajouter des octaves et un peu plus tard, y appliquer des textures différentes. Dans les zones montagneuses, il est probablement préférable d'avoir des grandes variations d'altitudes entre le point le plus bas et le plus haut et de rajouter des octaves pour modéliser des cailloux. Au contraire, pour les zones avec de l'eau (lac et mers), la variation d'altitude est quasiment nulle mais on peut chercher à représenter les petites vaguelettes de l'eau. Enfin, pour les plaines, la variation d'altitude sera plus élevée que pour les zones d'eau mais plus faible que pour les zones montagneuses. On peut donc modifier la topologie de chaque type de terrain séparément.

L'utilisation du bruit de Perlin dans le cadre de la génération de monde 3D peut donc permettre de définir en chaque point du monde à quel type de terrain il appartient : libre à l'utilisateur d'ensuite modeler chaque type de terrain à sa convenance.

Simplex Noise

Le *Simplex Noise* est une réécriture du bruit de Perlin par Perlin lui même dans [5]. Au lieu d'utiliser une grille régulière comme précédemment, il choisit un maillage triangulaire (dans le cas 2D). Comme expliqué dans [2], le but est d'avoir la forme unitaire la plus compacte possible, avec le moins de sommet, permettant de remplir un plan, en 2D cela correspond au triangle équilatéral. Cela permet de limiter le nombre de calcul à faire pour l'interpolation des points qui ne sont pas sur la grille. L'interpolation entre les différents points est elle aussi simplifier : c'est désormais une somme de 3 termes (un pour chaque sommet) qui est réalisée. Utiliser le *Simplex Noise* au lieu du bruit de Perlin, permet de réduire le temps de calcul d'un facteur 2. Le gain de temps est d'autant plus notable avec l'augmentation des dimensions car, en notant n la dimension, le bruit de Perlin a une complexité de $O(n2^n)$ et le *Simplex Noise* de $O(n^2)$ [5].

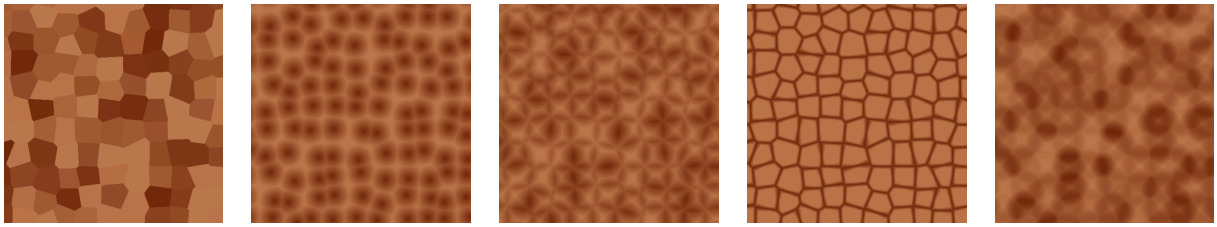
Comme on peut le voir sur la Fig. 3, le *Simplex Noise* donne un résultat assez similaire au bruit de Perlin et peut donc être utilisé à sa place.

2.1.2. Bruit de Voronoï

Le bruit de Voronoï est très répandu, notamment pour la création de texture. Il permet de créer des pavages de cailloux, des grillages, etc. Le bruit de Voronoï est, comme son nom le laisse entendre, basé sur les cellules de Voronoï. Il est décrit par Steven Worley dans [6] : il en donne sa définition et quelques exemples d'application pour la création de texture.

Pour créer ce bruit, il répartit de manière aléatoire des points qui vont servir de *points clés* suivant une densité défini grâce à une distribution de Poisson. La valeur du bruit en un point quelconque correspond à sa distance au *point clé* le plus proche, ce bruit est appelé F1 dans [6]. Le bruit F2 est défini en prenant la distance au second point le plus proche. F1 et F2 peuvent aussi être combinées pour donner des aspects encore différents. Des exemples visuels de ces différents bruits peuvent être visualisés sur la Fig. 5.

En Godot (donc sur les images présentes dans ce rapport), le bruit de Voronoï n'est pas créé à l'aide d'une distribution de Poisson pour les points clés mais avec une grille de points clés que l'on a « agité ».



(a) Valeur cellule (b) F1 (c) F2 (d) F1 - F2 (e) F1 + F2

Fig. 5. – Bruit de Voronoï

On pourrait imaginer utiliser ce bruit pour altérer le champ de hauteur comme sur la Fig. 6. Il peut aussi permettre de reproduire certaines merveilles naturelles comme les orgues basaltiques.

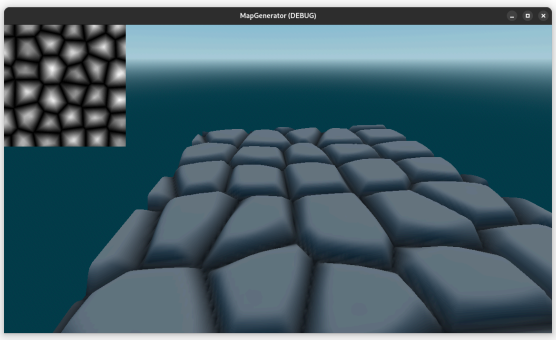


Fig. 6. – Bruit de Voronoï appliqué comme une carte de hauteur

Les cellules de Voronoï peuvent aussi servir à définir les différents biomes qui vont peupler le monde 3D.

Les bruits de Perlin et de Voronoï sont différents et complémentaires, et sont tout deux utiles dans le cas de génération procédurale de monde.

2.2. De très grandes cartes

Comment l'accentue Steven Worley dans [6], l'avantage primaire d'utiliser un bruit est que l'on n'a pas besoin de les calculer à l'avance, on peut calculer et recalculé une portion de la carte à chaque fois que l'on veut l'afficher.

Il n'est en général pas possible de générer d'un coup une très grande carte car cela serait soit trop coûteux en place mémoire soit trop coûteux en calcul. Il faut donc générer petit bout par petit bout. Spécifiquement pour les mondes sans fin, il faut pouvoir générer suffisamment de terrains pour que l'utilisateur ne soit pas bloqué car arrivé au bout du monde. Pour cela, on peut créer plusieurs entités individuelles, des portions de terrain, appelées *chunk* que l'on regroupera ensemble. Si l'on conserve simplement les informations utilisées pour les générer, cela sera moins coûteux que de garder tous les *chunks* en mémoire.

De plus, pour limiter l'impacte de grandes cartes sur les performances, on peut modifier le niveau de détail de chaque *chunk* en fonction de sa distance à la caméra (ou à l'utilisateur). Certains détails seront inutiles car trop petits, autant les simplifier.

Dans le cas d'une carte généré grâce à des bruits, une façon de créer ces différents niveaux de détails est de modifier le nombre d'octaves utilisées : pour un niveau de détails élevé, on peut garder toutes les octaves mais pour un niveau de détails faible, on peut supprimer celle de moindre amplitude.

Dans la dernière partie de ce rapport, des modèles 3D seront rajoutés sur la carte. Il peut être intéressant de les avoir avec différents niveaux de compression pour correspondre aux différents niveaux de détails.

2.3. Créé des sites naturelles cohérents

Le bruit de Perlin et le *Simplex Noise* permettent de générer des champs de hauteur qui semble plutôt correcte, cependant certaines problématiques apparaissent :

- il n'y a généralement pas de fines lignes dessinées, qui permettrait de représenter les chaînes de montagnes, ou des rivières.
- l'utilisation de champ de hauteur peut apporter des obstacles : on ne peut pas créer de grotte, de tunnel, de terrier ou de cavité en général.

Dans le cas des chaînes de montagnes, on pourrait imaginer se servir des cellules de Voronoï pour représenter les plaques en utilisant une densité faible pour que les *points clés* soit assez espacés, et ainsi

placer des montagnes aux frontières de ces plaques. Dans sa présentation [7] Suzanne Baxter décrit une autre technique pour créer une chaîne de montagne cohérente. Elle applique un masque sur les parties suffisamment hautes du monde, puis elle le décale légèrement dans une direction aléatoire. En superposant les deux masques A et B, et en prenant A\B, on obtient une zone géographique que l'on peut élever pour former une chaîne de montagnes. Dans [8], les montagnes sont créés en générant le terrain avec la fonction $(1 - |\sin(\text{Noise})|)^2$. La couche de montagne est rajoutée par-dessus le terrain existant (et multiplier par celui-ci pour ne pas avoir de montagne au milieu de l'eau).

En ce qui concerne les cavités, il faudrait donc plus utiliser une approche avec des Voxels (***Vo** lume-**El** ement*, l'équivalent des pixels en 3D), ce qui permettrait pour un même (x, y), d'obtenir différents points à des altitudes différentes, ce qui n'est pas le cas avec une approche par champ de hauteur. Utiliser des Voxels serait une approche volumique tandis qu'utiliser un champ de hauteur est une approche surfacique.

2.4. Créé une île

Comme expliqué par Suzanne Baxter dans [7], pour créer une île entourée d'eau à partir d'un champ de hauteur généré comme précédemment, on peut appliquer une fonction décroissante en fonction de la distance au centre voulu de l'île sur la matrice des champs de hauteur. Ainsi, plus un point est loin du centre de l'île, plus son altitude est affectée et diminuée, il a donc plus de chance de se retrouver sous le niveau de l'eau.

2.5. Créé une planète !

L'approche par champ de hauteur permet de générer un monde 3D avec des élévations différentes sur toutes la cartes, cependant cela reste un monde « plat ». On peut donc vouloir que notre monde soit plus proche de la forme d'une sphère que d'un plan, se rapprocher de la forme d'une planète. Une manière simple de construire une sphère à partir des terrains créés précédemment, décrite par Sebastian Lague dans [9], est de combiner 6 plans pour former les faces d'un cube puis de les déformer pour former une sphère.

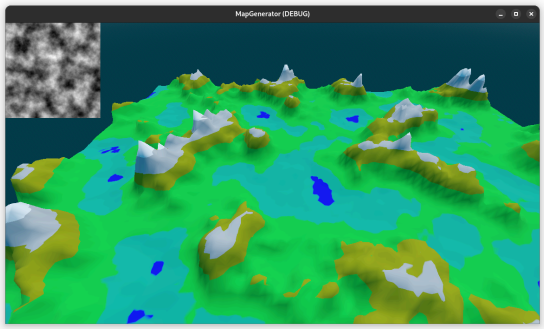
Afin que tous les triangles composants le maillage est la même taille, on peut utiliser la formule défini dans [10] pour effectuer la conversion des coordonnées des points du cube aux coordonnées d'une sphère :

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} x\sqrt{1 - \frac{y^2}{2} - \frac{z^2}{2} + \frac{y^2z^2}{3}} \\ y\sqrt{1 - \frac{x^2}{2} - \frac{z^2}{2} + \frac{x^2z^2}{3}} \\ z\sqrt{1 - \frac{x^2}{2} - \frac{y^2}{2} + \frac{x^2y^2}{3}} \end{bmatrix}$$

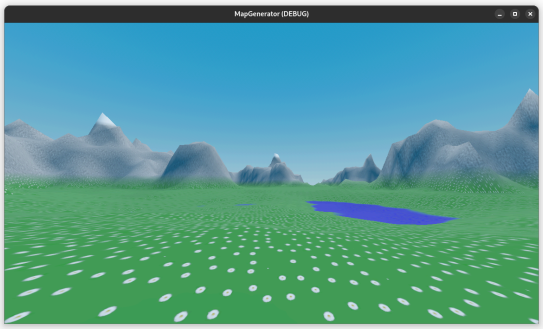
3. Ajout de la couleur

Dans la partie précédente, nous avons vu comment générer la topologie d'un monde 3D, sans aucune couleur, sans aucune texture, juste des variations d'altitudes. Il est désormais temps de rajouter un peu de couleur à ce monde. Avec les bruits décrits précédemment, on peut aussi créer tout un tas de texture ! C'est ce que nous allons voir dans cette partie.

Dans un premier temps, on peut tout simplement appliquer une couleur spécifique en fonction de l'altitude ou du type de terrains : par exemple bleue comme la mer à l'altitude la plus basse et blanche comme la neige en haut des montagnes, à l'altitude la plus haute. Cette approche donne un résultat visuel plutôt satisfaisant (Fig. 7a) mais les limites de son potentiel pour un rendu réaliste sont vite atteintes.



(a) Application de couleur



(b) Application de texture

Fig. 7. – Exemples colorisations d'une carte

Afin d'avoir un rendu réaliste, on peut appliquer des textures réalistes au lieu de simple couleur. Sur Fig. 7b, des textures ont été appliquées (même si elles ne sont pas réalistes du tout). On peut très bien créer ces textures procéduralement grâce au bruit de Perlin et au bruit de Voronoï (exemple Fig. 8). De même, on peut imaginer que la texture utilisée ne dépendra pas seulement de l'altitude (du type de terrain) mais aussi de la pente du terrain.

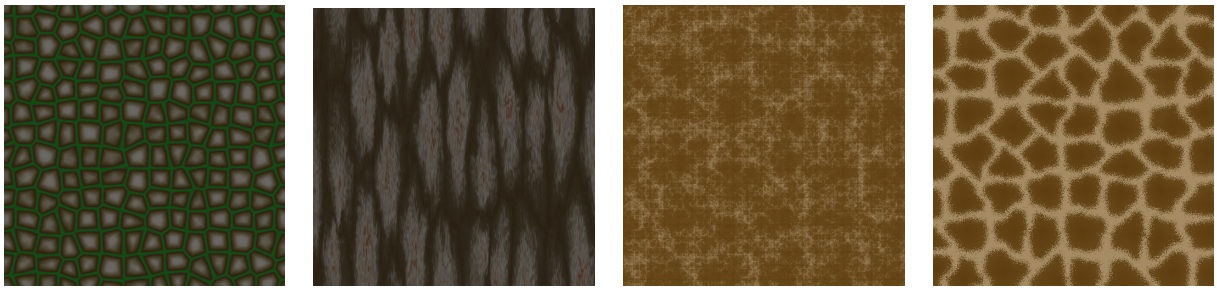


Fig. 8. – Exemples de texture

En appliquant des couleurs et des textures, on ne fait que définir la couleur « brute » de chaque pixel appelée l'albedo. Cependant, pour obtenir un rendu réaliste, il faut aussi tenir compte des interactions de la lumière avec les surfaces : elle peut être plus ou moins absorbée, diffusée ou réfléchie, et des ombres peuvent se former en fonction des conditions d'éclairage. Par exemple, le sable réfléchit très peu la lumière tandis que la neige la réfléchit fortement. Pour cela, il est important de connaître la normale en chaque point.

4. Comment faire prendre vie à cette carte 3D ?

Maintenant que le sol du monde 3D a été généré et coloré, il faut le rendre plus vivant : en prenant exemple sur la Terre, elle évolue au cours du temps. Tout un tas de phénomènes géologiques et naturels ont lieu : les roches subissent l'érosion, des volcans explosent, l'eau des océans monte, etc. De même, les être vivants peuplent la planète et la font évoluer. Cette partie va donc se concentrer sur ces aspects.

4.1. Comment imiter l'érosion ?

Pour rendre le terrain plus réaliste, il faut, comme dans la réalité, laisser l'érosion s'appliquer. La mer doit pouvoir creuser la roche, les rivières doivent pouvoir creuser le sol sur leur chemin. Pour cela, comme décrit dans [7], une carte de hauteur n'est pas toujours adaptée.

Une approche par Voxel peut mieux correspondre (notamment pour la création de grotte, impossible avec une simple carte de hauteur). La technique utilisée dans ce cas est l'utilisation de *gouttelettes* : une gouttelette à une certaine capacité à déplacer des sédiments, lorsque celle-ci a atteint la capacité maximum, elle dépose les sédiments qu'elle avait emmenés avec elle. Ainsi, des bouts de terrain sont déplacés d'un endroit à un autre. Pour ses déplacements, une gouttelette dispose d'une certaine vitesse et d'une certaine direction, qui pourront varier au cours du temps. Cependant, cette approche peut être très coûteuse en terme de ressource de calcul. On pourrait adapter, la même approche par gouttelettes à un champ de hauteurs : la gouttelette va creuser le champ des hauteurs aux endroits où elle passe et l'élever à l'endroit où elle devrait déposer les sédiments.

Le papier [11] définit une autre manière d'imiter l'érosion en partant d'un champ de hauteur généré à l'aide d'un bruit rose (ou *fractal noise* ou *1/f noise*). Le bruit rose a pour caractéristique d'avoir le spectre des puissances qui suit approximativement la fonction 1/f : $P(f) = \frac{1}{f^a}$. Pour le générer, il utilise l'algorithme Diamand-Carré (*Diamond-Square*). Cet algorithme consiste à alterner entre les *carrés* et les *diamants*. On part d'un carré avec 4 sommets. Les valeurs initiales de ces quatre sommets sont choisies aléatoirement. Lorsque l'on est à une étape de type *diamants*, on trouve la valeur du centre de ce carré en faisant la moyenne des 4 sommets du carré et en ajoutant un léger décalage. Lorsque l'on est à une étape de type *carrés*, on calcule cette fois-ci les valeurs au milieu des arêtes du carré à l'aide de ses 4 voisins, on décale aussi légèrement cette valeur [11]. Le papier présente plein d'astuce pour limiter le temps de calcul : le léger décalage en théorie devrait suivre une loi gaussienne, mais pour des questions de rapidité, il utilise une simple loi uniforme. Il utilise aussi les cellules de Voronoï (F2 - F1) en simplifiant le calcul des distances, encore une fois pour réduire le temps de calcul. À chaque fois, il essaye de maximiser au mieux un certain score représentant l'érosion. Après avoir généré ce terrain, des algorithmes d'érosion peuvent être appliqués :

- *thermal erosion*
- *hydraulic erosion*

C'est deux algorithmes se rapprochent un peu de l'approche par gouttelette défini dans [7].

4.2. Ajout de verdure

Pour l'ajout d'herbes, de fleurs, d'arbre ou plus généralement de végétaux, l'échantillonnage par disques de Poisson (*Poisson disk sampling*) est plutôt adapté. Comme décris dans [12] et par Sebastian Lague dans [13], l'échantillonnage par disques de Poisson consiste à placer un ensemble de points en espaçant d'une distance minimale tous les points entre eux : dans un rayon r autour d'un point, il ne pourra y avoir aucun autre point. Ils définissent pour cela une méthode plutôt optimisée pour y parvenir. On peut utiliser ainsi chaque point généré par l'échantillonnage par disques de Poisson comme centre pour placer un objet (exemple Fig. 9 avec des pâquerettes et des arbres).

L'exemple Fig. 9 a été réalisé grâce à l'extension Godot créé par Udit Parmar [14] (basé sur l'algorithme décrit dans [12]).

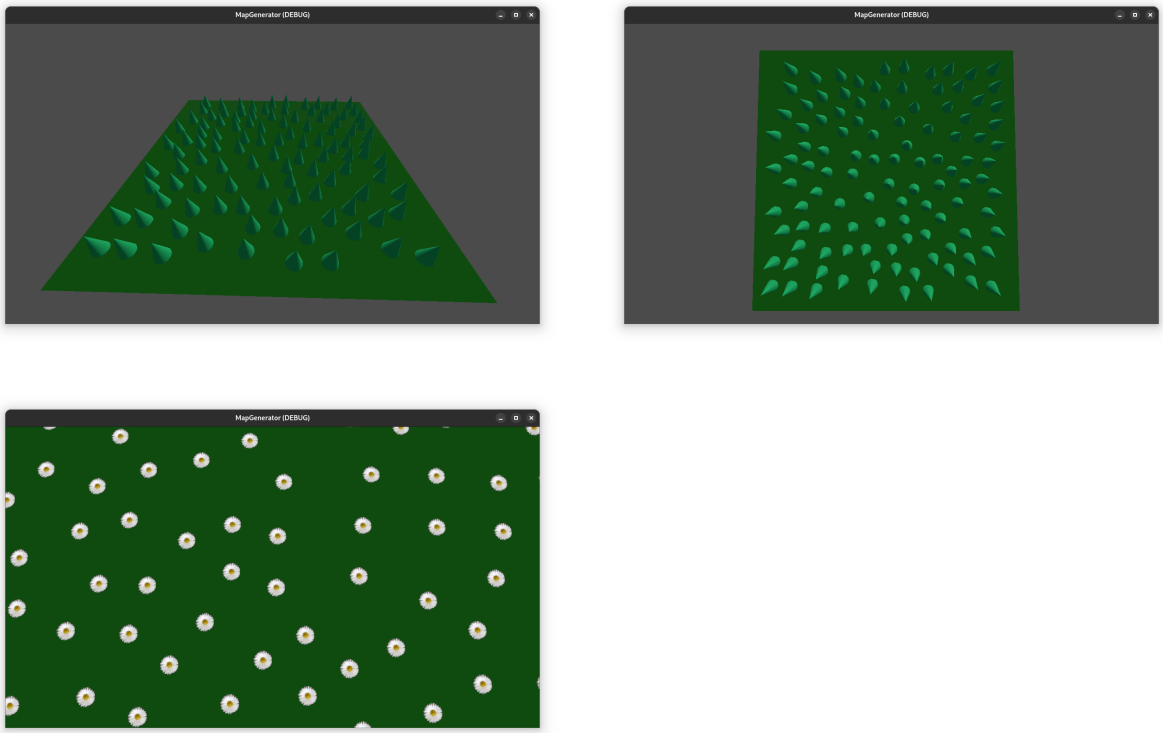
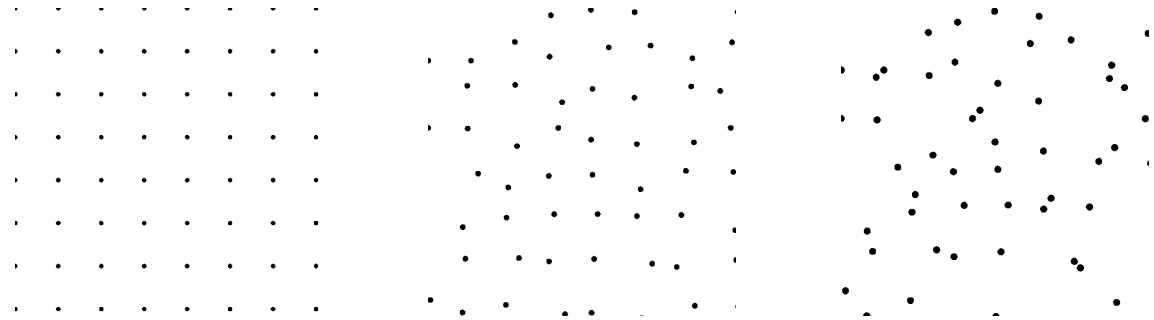


Fig. 9. – Exemples d'application d'échantillonnage par disques de Poisson

Les résultats obtenus par cette méthode sont plus convaincants que lorsque l'on utilise plus simplement une grille pour laquelle on met un point positionné aléatoirement dans chaque case de cette grille (*Jitter*). Dans ce cas, il n'y a pas de distance minimale établie, les points peuvent donc être trop proches les uns des autres pour deux cases voisines (Fig. 10).



(a) jitter = 0 (b) jitter = 0.5 (c) jitter = 1.0

Fig. 10. – Exemples méthode de placement par grille et *jittered*

La méthode de positionnement par *Poisson Disk Sampling* semble plus convaincante. Cela peut servir à définir les principales forêts de la carte.

Pour ce qui est de la création des modèles 3D d'arbres et de végétaux en général, il est possible d'utiliser une approche comme décrite dans [15] avec des L-systèmes (*Lindenmayer Systems*). Les L-systèmes permettent de définir des règles de développement pour les plantes. On peut ainsi définir des règles pour les branches, les feuilles, les fleurs, etc. En définissant les bonnes règles, les plantes vont se développer de manière cohérente, elles essaieront au maximum d'être exposées à la lumière par exemple. Cette approche permet de générer un ensemble de plantes variées, mais qui se ressemble tout de même.

4.3. Urbaniser la carte

Le monde 3D décrit précédemment s'approche de plus en plus d'un monde 3D réaliste. Il manque désormais la civilisation humaine !

Pour cela, il faut désormais ajouter des constructions humaines, des villages, des villes et des routes. Les papiers [16] et [17] aident beaucoup pour cette étape.

4.3.1. Placement des bâtiments

Pour placer des bâtiments, de nombreuses contraintes doivent être respectées : il faut faire attention à ce qu'ils s'adaptent bien à la forme du terrain : les portes doivent bien être au niveau du terrain, ou à défaut être accessible par un escalier, les fenêtres ne doivent pas entrer en collision avec le sol.

De plus, les bâtiments doivent être placés à des endroits cohérents. Il est beaucoup moins probable d'avoir un village sur un terrain montagneux que sur une zone plutôt plate à côté d'une source d'eau. Les châteaux (du moins ceux servant à la défense), ont plutôt tendance à être en hauteur avec une vue sur les contrées alentour. L'eau douce est vitale, les habitations, les villages ont de grandes chances dans la réalité d'être à proximité d'une source. Certaines constructions sont susceptibles d'en attirer d'autres : une église sera souvent entourée de maisons, un moulin à eau ou un cimetière sera souvent à proximité d'un village. Quand plusieurs habitations sont regroupées, elles peuvent se transformer en village et continué d'attirer plus de population et de grandir.

Pour chaque bâtiment à placer, il y a donc des contraintes à respecter et certains lieux sont plus propices que d'autre à accueillir certains bâtiments. Le papier [16] introduit donc une fonction d'intérêt défini en chaque point, spécifique aux différents types de bâtiment. Une église n'aura pas la même définition de fonction d'intérêt qu'une ferme ou un moulin à eau par exemple. La fonction d'intérêt permet de définir le respect de certains critères et de trouver le lieu le plus propice à la construction d'un bâtiment. Elle permet aussi notamment de forcer un espacement minimal entre les différents bâtiments.

La fonction d'intérêt défini dans [16] est en réalité la somme de plusieurs fonctions : certaines peuvent être calculées à l'avance (proximité d'un cours d'eau par exemple) et d'autres doivent être constamment recalculé (espacement minimal entre 2 bâtiments). Chaque fonction composant la fonction d'intérêt va correspondre au respect d'un critère au point choisi. Si un critère n'est pas respecté, suivant son importance, la fonction correspondant à ce critère aura, en ce point, une valeur plus ou moins négative. Ainsi, cette localisation sera pénalisée si elle ne respecte pas certains critères et aura peu de chances d'être choisi. La position choisie correspond au point en lequel la fonction d'intérêt est maximale.

Lorsque l'on veut construire un bâtiment, il faut aussi lui attribuer une parcelle de terrain lui appartenant. Le papier [16] a essayé plusieurs approches dont celle des diagrammes de Voronoï mais celle-ci n'était pas convaincantes. La méthode choisie finalement consiste à d'abord affecter les portions de routes au bâtiment le plus proche de celle-ci (tant que la distance est inférieure à une valeur maximale). Ensuite, de la propagation, partant de ses portions de route, est utilisée pour définir la parcelle. Pour conquérir chaque portion de terrain, cela a un certain coût (plus ou moins important suivant la pente du terrain, si c'est de l'eau ou une route, etc). Afin d'éviter d'avoir des parcelles trop grandes, il y a un coût maximal à ne pas dépasser en faisant la somme de tous les coûts individuels des portions de terrain déjà prise. Ce coût maximal est défini pour chaque type de bâtiments. Pour avoir des frontières de terrain un peu plus réaliste, les frontières sont ensuite approximées par un polygone.

4.3.2. Construction des bâtiments

Une fois que l'on connaît la localisation du futur bâtiment, il faut donc le construire. Deux solutions s'offrent alors. La première consiste à avoir un ensemble de modèle 3D de bâtiment suffisamment grand

pour que la répétition soit discrète et qu'on ait au moins un bâtiment qui s'adaptera aux différents terrains. La seconde solution, décrite légèrement dans [16] et plus précisément dans [17], consiste à créer le bâtiment à partir de composant plus élémentaire : les fenêtres, les portes, etc. Pour cela, une grammaire a été créée précisément pour la génération procédurale de bâtiments : *CGA shape* [17]. Cette grammaire s'est inspiré du L-système qui existait déjà pour la génération de plantes. La grammaire *CGA shape* se fonde sur :

- des axiomes, qui vont définir la forme globale du bâtiment (un ensemble de cubes, cylindres, etc.).
- des règles, qui vont permettre de faire évoluer la forme du bâtiment, ajouter des détails sur les façades comme des fenêtres, des balcons ou des portes, définir des étages au bâtiment.

La grammaire ajoute aussi des contraintes pour éviter les occlusions et faire en sorte que les différents détails soient alignés entre eux. Des vérifications peuvent être faites avant de placer les fenêtres par exemple pour être sûr qu'elle ne sera pas bloqué par un mur. Grâce à cette grammaire, on peut aussi aligner les différentes fenêtres entre elles. Pour utiliser cette grammaire, on part donc d'un axiome puis on choisi aléatoirement une règle à appliquer, sachant que chaque règle à une certaine probabilité d'être choisi. On itère plusieurs fois avec des règles différentes pour obtenir le bâtiment final. L'utilisation d'une grammaire pour générer les bâtiments permet d'en générer un grand nombre rapidement, à la fois différent et à la fois en conservant une certaine cohérence entre eux (si on utilise le même ensemble de règles pour les construire). [16] ont étendu cette grammaire afin qu'elle prenne en compte la topologie du terrain et les autres bâtiments déjà construit.

Les bâtiments sont souvent (si ce n'est systématiquement) reliés à des routes. Il est donc important de les ajouter elles aussi à notre monde 3D.

4.3.3. Routes

Le papier [16] décrit aussi, comment ajouter les routes. Elles sont placées grâce à la résolution d'un problème du plus court chemin, d'un coût à minimiser. Afin d'avoir des routes cohérentes, des contraintes sont ajoutées :

- les routes ne doivent pas traverser des bâtiments déjà existants.
- lorsque les pentes sont trop abruptes, le coût de construction de la route sera beaucoup plus important.
- traverser d'autres routes est favorisé afin d'avoir un rendu plus proche de la réalité (et éviter d'avoir beaucoup de routes très proches les unes des autres sans se croiser).

Enfin, lorsque l'on créé une route pour relier un bâtiment à une route déjà existante, il faut aussi regarder si la route créée peut être prolongée dans la même direction pour la relier à une autre route existante afin de faire une boucle. Cela n'est pas détaillé dans l'article, mais on pourrait aussi ajouter une contrainte permettant de placer des « routes » (ponts) sur l'eau, à un coût plus coûteux qu'une route classique.

Si l'on veut être pointilleux sur la véracité des routes, il faut faire attention lors des virages à utiliser des clothoïdes (ou spirale de Cornu ou spirale d'Euler) entre la portion de ligne droite et la portion de virage comme décrit dans [18]. Les clothoïdes sont très couramment utilisé par les ingénieurs notamment pour concevoir des routes ou des rails car elles permettent une augmentation progressive de la force centrifuge au moment où le véhicule entame le virage, ce qui est plus confortable et moins dangereux qu'une augmentation brusque. De même, les routes goudronnées ont une forme un peu arrondie sur les côtés en général.

4.4. Rivières

En utilisant le bruit de Perlin, tout comme les chaînes de montagne, aucune rivière n'était créée. Les rivières sont particulières, car elles doivent en général rejoindre 2 points d'eau. On peut peut-être utiliser la même méthode que celle pour créer les routes afin de créer les rivières, c'est-à-dire un

algorithme du plus cours chemin. On pourrait aussi utiliser de la propagation comme pour la création des parcelles de terrain. Cependant, les contraintes utilisées seront différentes. Pour une rivière, se déplacer dans le sens de la pente descendante doit avoir un coût très faible, tandis que se déplacer dans le sens de la pente montante doit avoir un coût très élevé. Cela permettrait donc aussi de favoriser l'émergence de cascade.

5. Conclusion

Nous avons vu quelques aspects basiques pour la génération de monde 3D. De la création de la topologie globale du terrain à l'ajout de détails beaucoup plus précis comme les bâtiments.

Plein d'autre aspect pour la génération de monde 3D pourrait être abordé comme le calcul des normales en chaque point du maillage pour pouvoir gérer la lumière correctement, l'utilisation de réseaux de neurones pour générer la topologie de la carte ou le placement des bâtiments.

De plus, un monde 3D réaliste ne se limite pas à son relief et à son urbanisation. Il faudrait aussi y intégrer des phénomènes climatiques, des variations météorologiques (la pluie, le vent, etc.), des nuages, etc. On pourrait aussi ajouter du mouvement avec des êtres vivants comme les animaux.

Bibliographie

[1] K. Perlin, « An image synthesizer », in *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, in SIGGRAPH '85. New York, NY, USA: Association for Computing Machinery, 1985, p. 287-296. doi: 10.1145/325334.325247.

[2] S. Gustavson, « Simplex noise demystified », p. , 2005.

[3] I. Quilez, « Inigo Quilez ». Consulté le: 14 mars 2025. [En ligne]. Disponible sur: https://iquilezles.org/articles/morenoise/

[4] S. Lague, « Procedural Landmass Generation (E06: LOD) ». Consulté le: 14 mars 2025. [En ligne]. Disponible sur: https://www.youtube.com/watch?v=417kJGPKwDg&list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3&index=6

[5] K. Perlin, « Chapter 2 Noise Hardware », [En ligne]. Disponible sur: https://api.semanticscholar.org/CorpusID:16050677

[6] S. Worley, « A Cellular Texture Basis Function », in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1996, New Orleans, LA, USA, August 4-9, 1996*, J. Fujii, Éd., ACM, 1996, p. 291-294. doi: 10.1145/237170.237267.

[7] S. Baxter, « Procedural Generation of Terrain ». Consulté le: 10 mars 2025. [En ligne]. Disponible sur: https://2020.pycon.org.au/program/KLALFC/

[8] S. Lague, « [Unity] Procedural Planets (E04: multiple noise filters) ». Consulté le: 15 mars 2025. [En ligne]. Disponible sur: https://www.youtube.com/watch?v=H4g-TC__cvg&list=PLFt_AvWsXl0cONs3T0By4puYy6GM22ko8&index=4

[9] S. Lague, « [Unity] Procedural Planets (E01 the sphere) ». Consulté le: 15 mars 2025. [En ligne]. Disponible sur: https://www.youtube.com/watch?v=QN39W020LqU&list=PLFt_AvWsXl0cONs3T0By4puYy6GM22ko8

[10] P. Nowell, « Mapping a Cube to a Sphere ». Consulté le: 15 mars 2025. [En ligne]. Disponible sur: https://mathproofs.blogspot.com/2005/07/mapping-cube-to-sphere.html

[11] J. Olsen, « Realtime Procedural Terrain Generation Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games », 2004. [En ligne]. Disponible sur: https://api.semanticscholar.org/CorpusID:18949321

[12] R. Bridson, « Fast Poisson disk sampling in arbitrary dimensions », in *ACM SIGGRAPH 2007 Sketches*, in SIGGRAPH '07. San Diego, California: Association for Computing Machinery, 2007, p. 22–es. doi: 10.1145/1278780.1278807.

[13] S. Lague, « [Unity] Procedural Object Placement (E01: poisson disc sampling) ». Consulté le: 11 mars 2025. [En ligne]. Disponible sur: https://www.youtube.com/watch?v=7WcmyxyFO7o&list=PLFt_AvWsXl0cbBEB7Z0QSeTCXDRL9IIH0

[14] U. Parmar, « Poisson Disc Sampling ». Consulté le: 12 mars 2025. [En ligne]. Disponible sur: https://github.com/udit/poisson-disc-sampling

[15] R. Měch et P. Prusinkiewicz, « Visual models of plants interacting with their environment », in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, in SIGGRAPH '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 397-410. doi: 10.1145/237170.237279.

[16] A. Emilien, A. Bernhardt, A. Peytavie, M.-P. Cani, et E. Galin, « Procedural Generation of Villages on Arbitrary Terrains », *The Visual Computer*, vol. 28, p. , 2012, doi: 10.1007/s00371-012-0699-7.

[17] P. Müller, P. Wonka, S. Haegler, A. Ulmer, et L. Van Gool, « Procedural Modeling of Buildings », *ACM Trans. Graph.*, vol. 25, p. 614-623, 2006, doi: 10.1145/1141911.1141931.

[18] R. Levien, « The Euler spiral: a mathematical history », sept. 2008. [En ligne]. Disponible sur: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-111.html