

PSC : RAPPORT FINAL

Génération des fonds marins
INF 15

Avril 2023

Nicolas Welti, Mohamed Amine Bouguezzoul, Félix Houdouin



Table des matières

1	Introduction	2
2	Étude des reliefs océaniques	3
3	Stratégie de génération	5
3.1	La démarche suivie	5
3.2	Explicitation des étapes de la génération procédurale	6
4	Classification hiérarchique	7
4.1	Les données réelles	7
4.2	Calculs d'indicateurs	8
4.3	Classification supervisée et création du jeu de données	12
5	Génération des différents types par GAN	17
5.1	Motivations	17
5.2	Principes	17
5.3	Quelques exemples	18
5.4	La convolution	20
5.5	La perte de Wasserstein	21
5.6	Architecture et implémentation	22
5.7	Génération du profil d'altitude	28
5.8	Génération d'un large profil d'altitude	31
6	Génération du profil local	34
6.1	L'utilisation d'un bruit pour le profil local	34
6.2	Algorithme de l'érosion hydraulique	37
7	Présentation du résultats	40
7.1	Utilisation de Unity pour le rendu graphique	40
8	Conclusion	41
9	Répartition du travail dans le groupe	41
10	Liens Github des codes	42

1 Introduction

L'objectif de notre projet est de générer procéduralement des fonds océaniques. Nous souhaitons obtenir un système permettant de générer les reliefs sous-marins d'une planète de type terrestre à partir des données disponibles pour la partie émergée de la planète. Pour atteindre cet objectif, notre algorithme doit être capable de compléter les informations manquantes et de créer un profil d'altitude cohérent pour les fonds marins en vue d'obtenir un rendu graphique satisfaisant. Il est essentiel que les fonds marins générés soient cohérents les uns avec les autres et avec la partie émergée de la planète, afin de garantir une génération de reliefs cohérente à l'échelle planétaire. De plus, il est crucial que le profil généré soit réaliste à petite échelle.

Notre algorithme pourrait être utilisé dans divers contextes tels que la génération de terrains dans des jeux vidéos ou des simulateurs. Grâce à ce générateur, les utilisateurs pourront bénéficier d'un terrain réaliste, cohérent et bien détaillé, offrant ainsi une expérience plus immersive.

Nous avons donc établi les objectifs suivants : lire les données terrestres puis les exploiter en extrayant les différents reliefs existant et leurs caractéristiques. Puis, à partir de ces données, développer des algorithmes de génération de ces différents types. Enfin, réussir à générer de manière cohérente ces différents types sur toutes la planète et obtenir un rendu graphique avec Unity.

Plus précisément, nos objectifs intermédiaires sont :

1. L'acquisition des données terrestres pour obtenir un profil d'altitude le plus précis possible des parties émergées et immergées.
2. La réalisation d'une classification des reliefs marins pour permettre de visualiser ces reliefs et donc d'évaluer la qualité des terrains générés. Cela permet donc d'affiner les générateurs, et est indispensable pour certaines approches (cGAN).
3. La génération de chacun des types de relief marin identifié à une résolution correspondant à celle des données (un réseau de pas 500m).
4. La génération d'une texture permettant d'avoir un rendu graphique avec la résolution souhaitée (nettement inférieure à celle des données, de l'ordre du mètre) et réaliste.
5. La génération d'une carte des fonds marins, une « carte des types », représentant à grande échelle la répartition des différents types de terrains avec une cohérence globale, précédant la génération procédurale de chacune des parcelles de terrains ($250\text{km} \times 250\text{km}$) d'un type donné.
6. Trouver un système permettant la génération continue des différentes parcelles de terrain adjacentes.
7. Visualiser les terrains obtenus sur Unity.

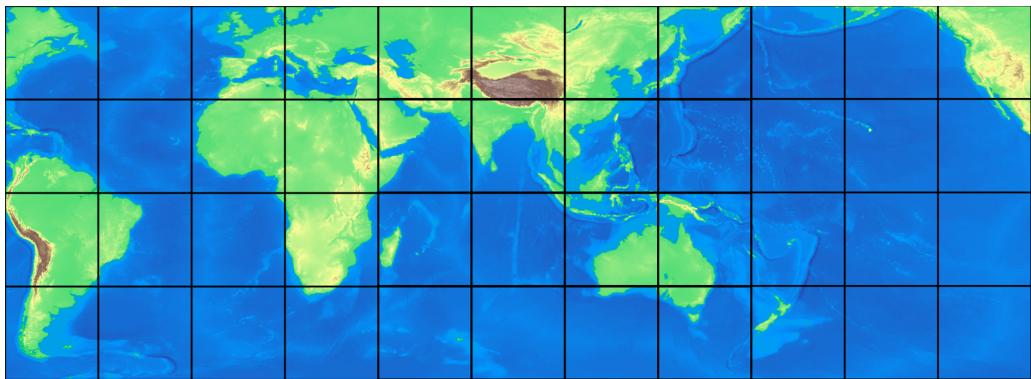


FIGURE 1 – Quadrillage de la carte terrestre

2 Étude des reliefs océaniques

Nous avons étudié superficiellement les différents reliefs océaniques existants, leurs caractéristiques et leur répartition sur la planète. Cette étude est un point de départ indispensable permettant d'évaluer la pertinence de nos algorithmes. La connaissance de ces reliefs est également importante pour comprendre ce qui suit. Nous avons retenu les reliefs océaniques suivants :

1. **La plaine abyssale** : Étendue plate qui couvre le fond des océans à une profondeur moyenne de 4 000 mètres (entre 3 000 à 5 000 mètres de profondeur et de 2 000 à 2 500 mètres en Méditerranée). La plaine abyssale correspond à 80% de la surface océanique totale.
2. **Le plateau continental** : Le plateau continental est le prolongement du continent sous la surface de l'océan. La zone submergée possède toutes les propriétés d'un continent et est d'une profondeur de 100 à 200 mètres. La largeur moyenne du plateau continental est de 74 km, avec des fourchettes variant de 8,5 km à 500 km.
3. **Le talus continental** : Zone en pente, qui assure la liaison entre le plateau continental et la plaine abyssale. Le talus continental présente un angle de pente généralement compris entre 4 et 5° mais qui peut être nettement plus accentué comme au large de la Côte d'Azur.
4. **Le Glacis continental** : Il s'agit de la zone bombée située au pied du talus continental, elle est formée par les sédiments qui descendent avec des vitesses variables le long du talus continental.

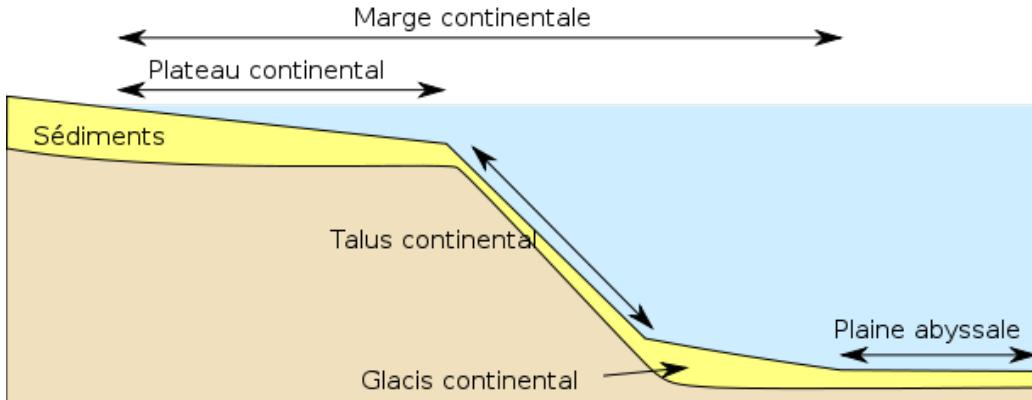


FIGURE 2 – Quelques reliefs océaniques

5. **La dorsale Océanique :** Chaîne de montagnes sous-marine, qui se caractérise par des profondeurs aux alentours de -2 000 m. Une dorsale océanique est une frontière de divergence entre deux plaques tectoniques. Le réseau de ces dorsales est continu et s'étend sous les océans sur près de 80 000 km.
6. **Rift Océanique :** Fossé d'effondrement se localisant le long de l'axe des dorsales uniquement.
7. **Les failles transformantes :** Les failles transformantes sont des limites de plaques lithosphériques où il n'y a ni subduction ni création de lithosphère. Elles sont situées en bordure de plaques tectoniques et découpent les dorsales perpendiculairement.
8. **Les fosses océaniques :** Dépression sous-marine profonde présente dans les zones de subduction. Elles sont engendrées par la collision ou l'expansion de plaques tectoniques.

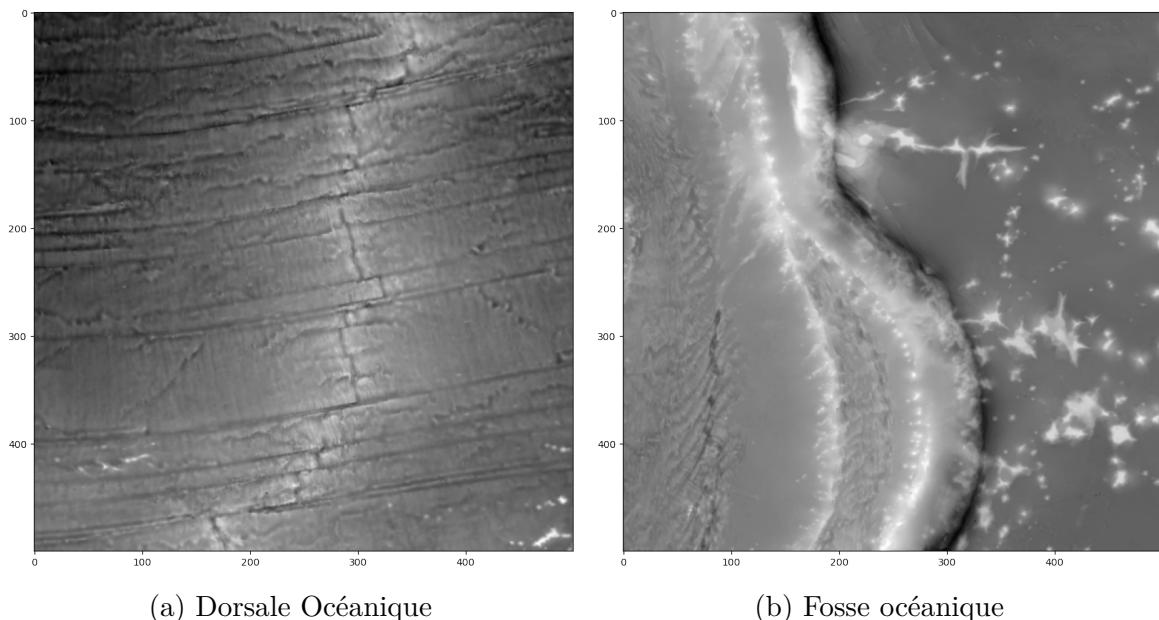


FIGURE 3 – Images brutes issues de nos données

Il existe de nombreuses autres structures sous-marines caractéristiques que nous n'avons pas cherché à générer. Par exemple, les chaînes de montagnes sous-marines peuvent également se former suite au déplacement des plaques au-dessus d'un point chaud (Hawaii par exemple).

Les différents reliefs identifiés ne sont pas réparties au hasard sur les fonds océaniques et il ne suffit pas de pouvoir générer chacun des reliefs séparément. Par exemple les dorsales forment un réseau continu délimitant les plaques tectoniques.

3 Stratégie de génération

3.1 La démarche suivie

L'objectif de notre algorithme est de générer un profil géographique réaliste des parties immergées d'une planète de type terrestre à partir des données des parties émergées. Afin d'atteindre cet objectif, nous avons identifié plusieurs sous-problèmes. Schématiquement, notre stratégie de génération est la suivante :

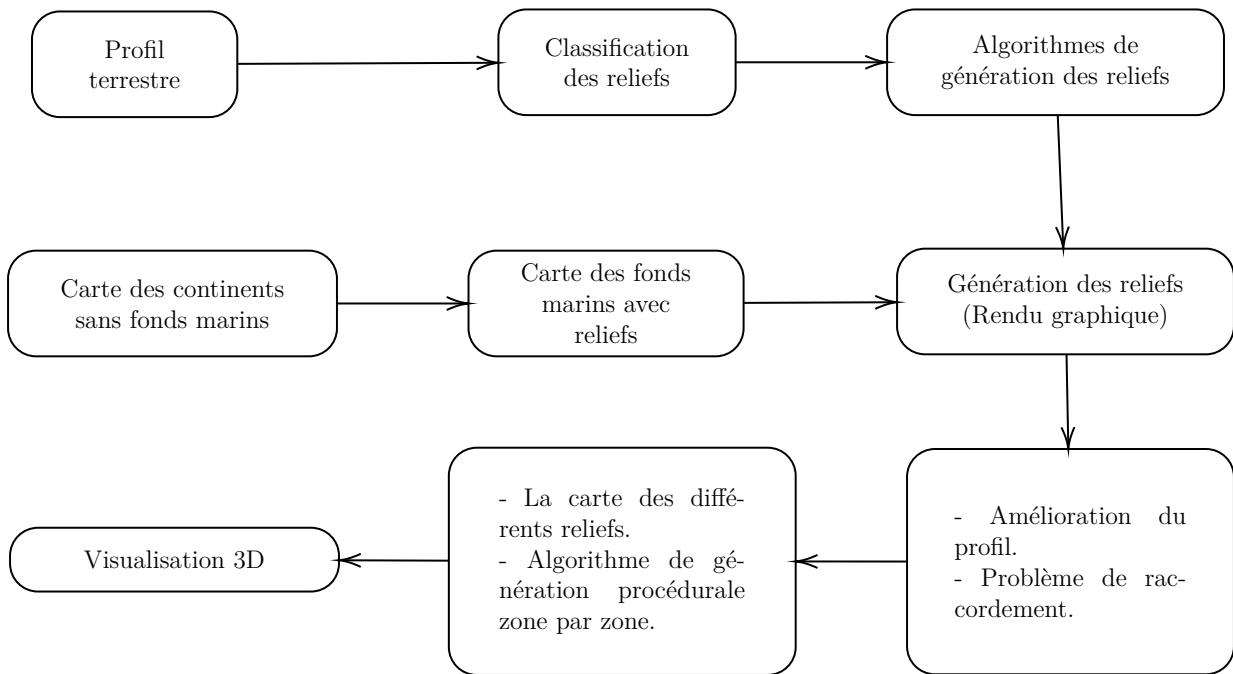


FIGURE 4 – Schéma expliquant la stratégie de génération

Nous disposons d'un relevé d'altitude complet de la terre avec une résolution correspondant à un réseau de 500m de coté. L'exploitation de ces données est importante car a pour but de nous permettre de valider les profils d'altitudes générés. Nous avons essayé plusieurs approches, le clustering dans un premier temps qui s'est révélé peu fructueux. Nous avons finalement opté pour une classification hiérarchique supervisée qui donne de bien meilleurs résultats.

Nous avons ensuite cherché à utiliser la catégorisation de ces données pour adapter au mieux nos générateurs de profils. Si nous souhaitons générer les fonds des océans avec une résolution en dessous du mètre (ce qui est cohérent pour un jeu par exemple), il est nécessaire de ne pas générer tous les fonds marins mais uniquement des petites zones où un joueur se déplacerait. Nous avons décidé de découper la zone à générer en petit carré de 250km de coté et de générer un terrain (« une tuile ») avec la même précision que nos données. Nos premiers générateurs reposaient sur du bruit de Perlin, qui donne des textures réalistes mais sans aucun

fondement ni cohérence globale. Nous avons essayé de paramétriser le bruit pour générer au mieux les différents reliefs marins identifiés. Cependant les résultats étaient peu concluant, et il était notamment non envisageable d'avoir une cohérence « à grande échelle » des reliefs avec cette approche. Nous avons alors essayé d'utiliser un réseau de neurone afin de générer chacun des reliefs identifiés en s'appuyant sur les données de la terre. Cette approche pose plusieurs défis puisqu'il faut des dizaines de milliers d'images pour entraîner correctement un réseau de neurones de ce type et nous ne disposons que des données d'une seule terre. L'avantage de cette méthode et de paramétriser la génération et de permettre de générer les zones voisines de manière cohérente en récupérant les données présentes à une certaine profondeur du réseau.

Une autre étape importante de la génération, que nous n'avons pas traité est la création d'une carte des types pertinente. Étant donné un monde comportant des continents et des océans dont le profil des fonds marins est inconnu, nous subdivisons les zones sous-marines à générer en tuile de 250km de coté. Nous sommes capables de générer ces tuiles en y faisant apparaître un relief caractéristique. Il faut cependant créer une « carte des types », c'est à dire attribuer à chaque tuile un type pertinent (avec une cohérence globale). Par exemple l'ensemble des zones « dorsales » se doivent être connexes et délimiter des plaques. Nous n'avons pas développé cet algorithme mais nous pensons que cela peut être fait en utilisant un SinGAN[1], qui peut proposer des variations d'une image donnée en entrée, en ne s'entraînant que sur celle-ci et en conservant la même logique à petite et grande échelle.

Une fois ces deux étapes effectuées, on dispose d'une carte quadrillée en tuiles de 250 km de coté dont on connaît les types et d'algorithmes permettant de générer le profil d'altitude de chacune de ces tuiles séparément et en accord avec le type. Il reste alors deux étapes : créer un profil d'altitude avec une résolution de l'ordre du mètre et obtenir un recollement satisfaisant entre les tuiles. Nous ne disposons pas de relevé d'altitude suffisamment précis pour obtenir la « texture » des fonds océaniques (qui est très diverse). Aussi nous avons générer cette texture en utilisant du bruit de Perlin et en appliquant un algorithme d'érosion hydraulique. Nous avons essayé différentes stratégies pour le recollement entre les tuiles, avec de l'interpolation bilinéaire ou en adaptant les entrées du réseau générant les images. Il est toutefois difficile de ne faire apparaître aucun artefact.

3.2 Explication des étapes de la génération procédurale

Les étapes pour générer les fonds océaniques d'une planète inconnue sont donc :

- Quadriller notre carte d'altitude (comportant uniquement les continents) en tuiles.
- Établir le type de toutes les tuiles continentales grâce à notre algorithme de classification.
- Établir le type de toutes les tuiles de toutes les tuiles océaniques en complétant l'image, avec comme unique modèle de répartition la Terre.
- Générer le profil « grossier » (précision de nos données) de chacune des tuiles.
- Ajouter une texture plus détaillée sur la tuile.

Ces grandes étapes ont guidées notre recherche et nous avons étudier chacune d'entre elles. Nous n'avons cependant pas de résultat complet pour chacune de ces étapes, en particulier pour l'algorithme affectant le type à toutes les tuiles.

4 Classification hiérarchique

4.1 Les données réelles

Les données terrestres utilisées proviennent de *La Grille GEBCO (GEBCO_2022 Grid)*[2], elles sont téléchargeables sous la forme de fichiers tif qui peuvent être facilement lus en python. La grille GEBCO fournit une couverture mondiale de données d'élévation, en mètres, sur une grille d'un intervalle de 15 secondes d'arc. Elle se compose de 43200 lignes \times 86400 colonnes, ce qui donne 3 732 480 000 points de données.

Le chargement en python permet de la visualiser sous la forme d'une image en niveau de gris (c'est-à-dire un tableau en deux dimensions).

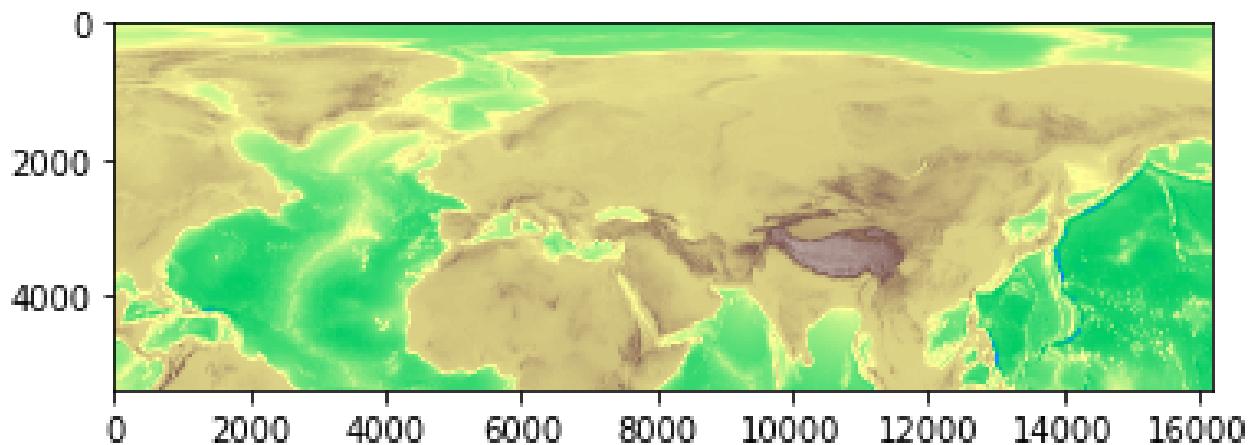


FIGURE 5 – Exemple des données fournies par GEBCO chargé sur Python

Nous avons décidé de ne pas prendre en compte les différences de distance dues à la nature sphérique de la terre (les points trop au nord/sud où l'erreur aurait été trop importante ont été ignorés lors de l'analyse de ces données), la terre est donc représentée par une image en deux dimensions, on considère que les points sont espacés d'environ 500m. Chaque point contient une valeur entière, son élévation en mètres.

Les analyses ont été effectuées sur une version à plus faible résolution (2km au lieu de 500m) et dans laquelle les bords supérieurs et inférieurs (proches des pôles) ont été rognés. On a donc un tableau composé de 15200 lignes et 43200 colonnes, soit 656 640 000 points (environ 2,6 GO de mémoire). Le tableau est représenté en python par une matrice 2D numpy.

La bibliothèque numpy permet de réaliser des opérations sur des grandes quantité de données très rapidement, car les calculs sont réalisés directement en C. Par exemple, pour multiplier tous les éléments d'un tableau par une constante, on pourrait parcourir le tableau avec une boucle for en python et faire la multiplication, mais on peut aussi utiliser une fonction numpy qui va faire ce parcours et ces opérations en C (ce qui est environ 100 fois plus rapide), avant de renvoyer les résultats en python. On peut donc réaliser des calculs complexes rapidement tout en gardant les avantages du langage très haut niveau qu'est Python. L'inconvénient de cette approche est qu'elle requiert d'effectuer toutes nos opérations simultanément, ce qui peut engendrer une complexité en mémoire élevée, surtout dans notre situation.

Afin d'éviter une complexité en mémoire trop importante, les opérations effectuées sur le tableau comme le calcul des indicateurs ou la classification de points (voir plus loin) seront effectuées en divisant le tableau avant de réassembler les résultats. Étant donné que la plupart de ces calculs ont un résultat par point, mais utilisent à chaque fois les points environnant, les divisions du tableau sont prises avec une bordure qui est ensuite rognée au niveau des résultats. Par exemple : pour appliquer une fonction f sur un tableau 1000×1000 en le divisant en 4 parties, le tableau sera divisé en 4 parties de taille 300×300 (en se superposant aux autres tuiles), f sera appliquée à chacune de ces parties et le résultat sera rogné à 250×250 pour chaque tuile puis refusionné, on évite ainsi les effets au bord.

L'objectif principal de l'étude des données est de créer une segmentation de notre grille (i.e. une carte des types), on veut la diviser en plusieurs régions, chaque point étant dans une région. Ou plus formellement : à partir d'une matrice A de taille $n \times m$ telle que $A[x, y]$ est la hauteur du point (x, y) , on cherche à obtenir une seconde matrice B de même dimensions $n \times m$ telle que $B[x, y]$ contient l'étiquette de la classe (une valeur entre 1 et $\mathbf{Nb}_{\text{classe}}$) dans laquelle se trouve le point (x, y) .

4.2 Calculs d'indicateurs

Afin de pouvoir créer notre classification, on doit d'abord calculer d'autres indicateurs que l'altitude. En effet, la classe dans laquelle se trouve un point ne peut dépendre que de son altitude, les indicateurs que nous allons calculer seront basés sur son agencement avec son voisinage et sur les motifs ainsi formés. Comment peut-on donc caractériser le voisinage d'un point ? La réponse la plus simple est de stocker tous les points de ce voisinage, on peut transformer notre grille $n \times m$ en une grille $n \times m \times (2r+1) \times (2r+1)$ avec chaque case contenant maintenant une petite grille centrée sur le point ; numpy offre même les fonctions nécessaires pour créer une telle grille à partir de la grille $n \times m$ en manipulant les adresses mémoires (pointeurs en C) des éléments du tableau (et donc sans aucun calcul). Il reste cependant à interpréter ce voisinage et à en tirer les mesures nécessaires pour la classification.

Afin de pouvoir interpréter notre voisinage, la méthode sélectionnée a été de passer par une paramétrisation quadratique [3]. Plus précisément, on effectue un développement du second ordre au voisinage du point central, c'est-à-dire qu'on cherche une fonction $q(x, y) = ax^2 + by^2 + cxy + dx + ey + f$ telle que $q(x, y) = \text{voisinage}(x, y)$, avec les coordonnées (x, y) centrées en 0.

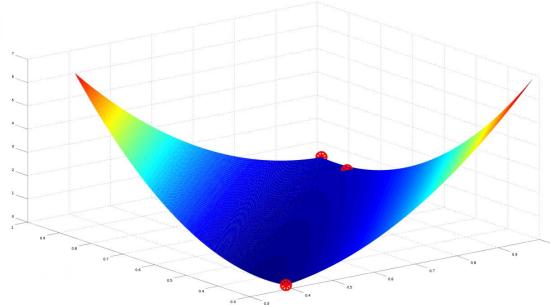


FIGURE 6 – $q(x, y) = ax^2 + by^2 + cxy + dx + ey + f$

En effet, une fois le modèle représenté par nos 6 coefficients, l'interprétation sera beaucoup plus facile. On peut par exemple calculer explicitement le gradient, déterminer la convexité,

ou encore utiliser les coefficients pour classifier la forme quadratique (elliptique, parabolique ou hyperbolique) et en déduire des informations sur les motifs du voisinage. Figure des formes quadratiques Reste donc à déterminer ces coefficients, on connaît la valeur de la fonction en $N = (2r + 1) \times (2r + 1)$ points, ce qui nous donne des équations linéaires en les coefficients à résoudre. Une méthode robuste et fonctionnant peu importe la taille du voisinage est de sommer nos expressions sur tout le voisinage. Par exemple : $\forall i \in [1, N]$, $q(x_i, y_i) = z_i$ et donc : $x_i \times q(x_i, y_i) = x_i \times z_i$ puis : $\sum_i x_i \times q(x_i, y_i) = \sum_i x_i \times z_i$. On obtient alors les équations suivantes :

$$\begin{pmatrix} \sum x_i^4 w_i & \sum x_i^2 y_i^2 w_i & \sum x_i^3 y_i w_i & \sum x_i^3 w_i & \sum x_i^2 w_i & \sum x_i w_i \\ \sum x_i^2 y_i^2 w_i & \sum y_i^4 w_i & \sum x_i y_i^3 w_i & \sum x_i y_i^2 w_i & \sum y_i^3 w_i & \sum y_i^2 w_i \\ \sum x_i^3 y_i w_i & \sum x_i y_i^3 w_i & \sum x_i^2 y_i^2 w_i & \sum x_i^2 y_i w_i & \sum x_i y_i^2 w_i & \sum x_i y_i w_i \\ \sum x_i^3 w_i & \sum x_i y_i^2 w_i & \sum x_i^2 y_i w_i & \sum x_i^2 w_i & \sum x_i y_i w_i & \sum x_i w_i \\ \sum x_i^2 y_i w_i & \sum y_i^3 w_i & \sum x_i y_i^2 w_i & \sum x_i y_i w_i & \sum y_i^2 w_i & \sum y_i w_i \\ \sum x_i^2 w_i & \sum y_i^2 w_i & \sum x_i y_i w_i & \sum x_i w_i & \sum y_i w_i & \sum w_i \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix} = \begin{pmatrix} \sum z_i \cdot w_i x_i^2 \\ \sum z_i \cdot w_i y_i^2 \\ \sum z_i \cdot w_i x_i y_i \\ \sum z_i \cdot w_i x_i \\ \sum z_i \cdot w_i y_i \\ \sum z_i \cdot w_i \end{pmatrix}$$

On a également ajouté le terme w_i , le poids du point i , qu'on prendra inversement proportionnel à sa distance au centre ; on peut ainsi donner plus d'importance aux points proches du centre dans l'estimation.

On a maintenant un système linéaire à résoudre de la forme : $AX = Y$ avec A indépendant des valeurs du voisinage (A ne dépend que des x_i et y_i et donc de la taille r du voisinage). On peut donc calculer A puis A^{-1} pour R fixé avant de l'appliquer sur l'ensemble des voisinages pour connaître les coefficients a, b, c, d, e, f en chaque point de la grille.

Quelques indicateurs :

Ordre 0 : altitude (coefficient f)

Ordre 1 : gradient (coefficient e et d)

- la pente : $\arctan(\sqrt{d^2 + e^2})$ (angle de pente)
- Exposition (orientation du terrain) : $\arctan(\frac{e}{d})$ fait référence à la direction vers laquelle un relief est incliné par rapport à un point cardinal, une zone avec des pentes variées dans toutes les directions peut indiquer une organisation plus complexe et irrégulière.

Ordre 2 : convexité (coefficients a, b, c) (cf. code ou papier source)

On peut également utiliser les coefficients pour identifier le relief local :

On utilise les indicateurs de convexité et la pente (cf. code pour les détails) pour classifier le point dans une de ces 6 catégories.

On arrive alors à un problème : le choix du paramètre r . En effet, les résultats de notre classification et les indicateurs en générale peuvent changer drastiquement selon la taille de notre fenêtre d'observation.

Cette dépendance de la classification (ainsi que de tous les autres indicateurs) par rapport à r semble poser un problème, mais elle apporte en réalité un indicateur très utile. En effet, la classification des reliefs locaux n'est en elle-même pas très utile pour ce que l'on souhaite obtenir : une segmentation de notre carte en région à grande échelle. Même si cette classification était fiable et indépendante du relief, elle serait inexploitable à l'échelle de la terre entière, on

ne pourrait pas en tirer une segmentation en régions bien délimitées et compréhensibles. En revanche, on peut changer notre problème de dépendance en r en une solution : en en faisant l'indicateur.

On va calculer la classification pour r allant de 1 à 16, et on va ensuite calculer l'**entropie** de cette classification : $H = \sum_{c \in \text{Classes}} N_c \times \log(N_c)$ avec N_c le nombre de valeurs de r (entre 1 et 16) où le point a été classifié dans la catégorie c et la convention $0 \times \log(0) = 0$. Une telle fonction sera minimale, si le point est toujours classifié dans la même catégorie. De cette entropie, on peut tirer un indicateur : **la rugosité**. Un terrain lisse aura une faible entropie car il n'aura pas ou peu de reliefs à petite échelle, et à l'inverse, un terrain rugueux aura beaucoup de reliefs à petite échelle et donc une entropie élevée.

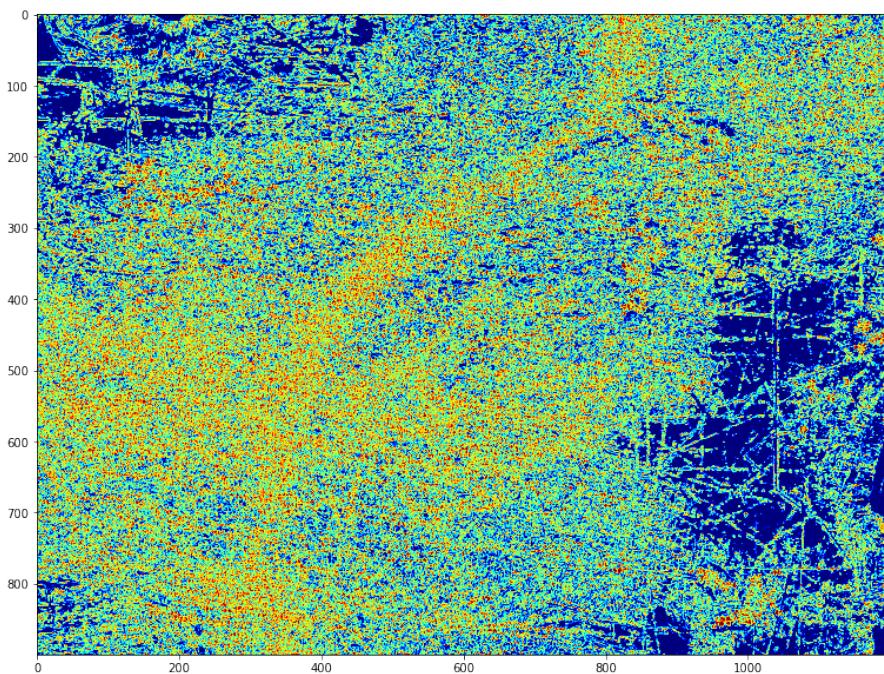


FIGURE 7 – Classification sur l'atlantique Nord

Ci-dessus la classification des reliefs locaux pour une carte d'altitude d'environ $4000\text{km} \times 4000\text{km}$ en Atlantique Nord, bien que quelques éléments soient visibles (comme la dorsale océanique), on observe bien que la carte est difficile à délimiter en régions ou à interpréter directement.

Finalement, on choisit trois indicateurs :

- **la pente** (moyenne de la pente pour $r = 1, \dots, 15$)
- **la rugosité** (entropie de la classification pour $r = 1, \dots, 15$)
- **l'organisation** : opposé de l'écart-type de l'exposition (orientation du terrain/direction du gradient). Un terrain sera donc considéré comme organisé si son orientation est indépendante de l'échelle d'observation, i.e si sa pente descend (ou monte) clairement.

Sur les figures ci-dessous on peut observer que la pente est très importante sur les grands reliefs, principalement les pentes continentales. La rugosité est plus élevée au niveau des fonds-marins. L'organisation est très élevée autour des continents, ce qui correspond en réalité au glacis continental : des dépôts de sédiments au pied des pentes continentales. Les calculs ont

été réalisés en une dizaine d'heures, sur une machine avec un processeur Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz et 16 GO de RAM.

Remarque : Le choix de ces indicateurs découlent de l'étude de travaux existants sur la classification des fonds marins[4]. Nous avons cependant divergé de ces travaux en décidant d'implémenter nous-même les calculs de ces indicateurs au lieu d'utiliser un logiciel spécialisé; et ce afin d'avoir un meilleur contrôle sur le processus (et une meilleure compréhension) et afin d'éviter tout problème de licence et de compatibilité.

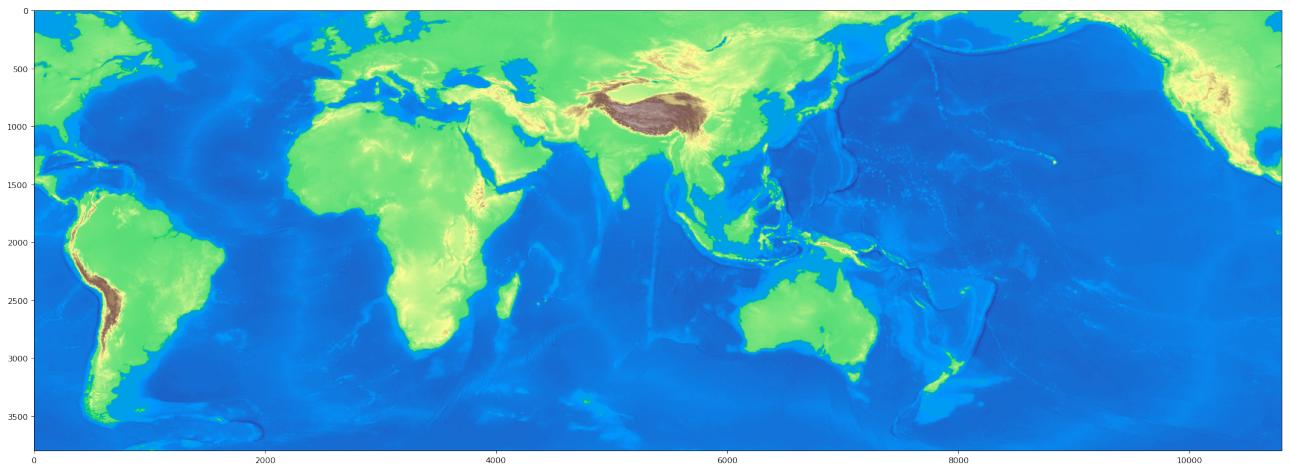


FIGURE 8 – L'élévation

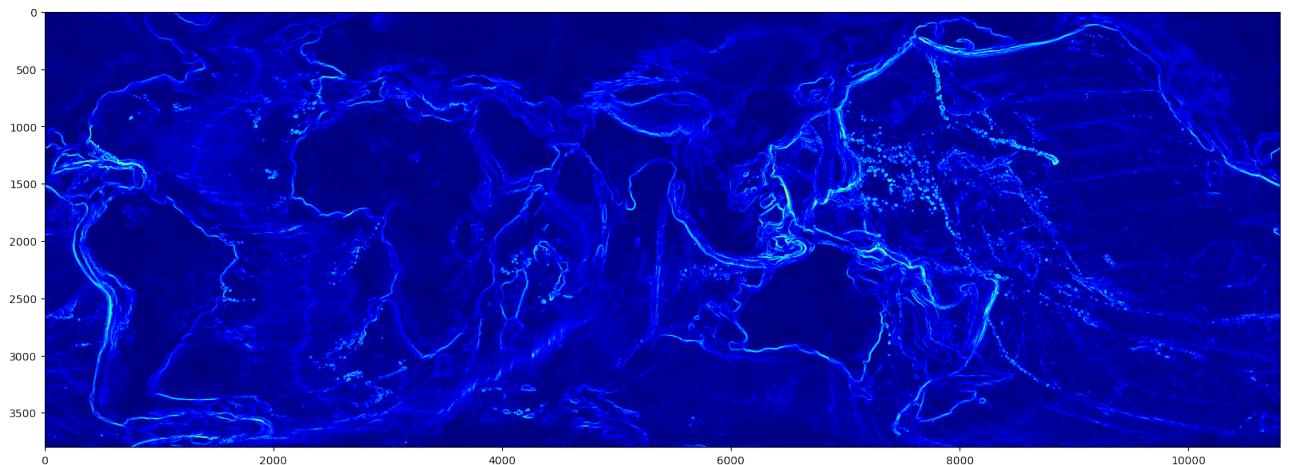


FIGURE 9 – La pente

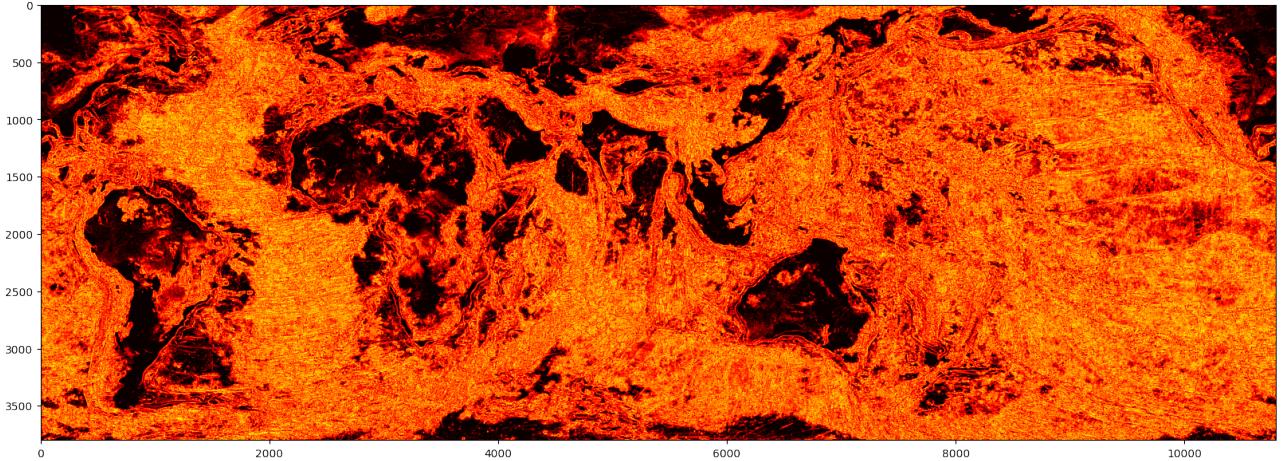


FIGURE 10 – la rugosité

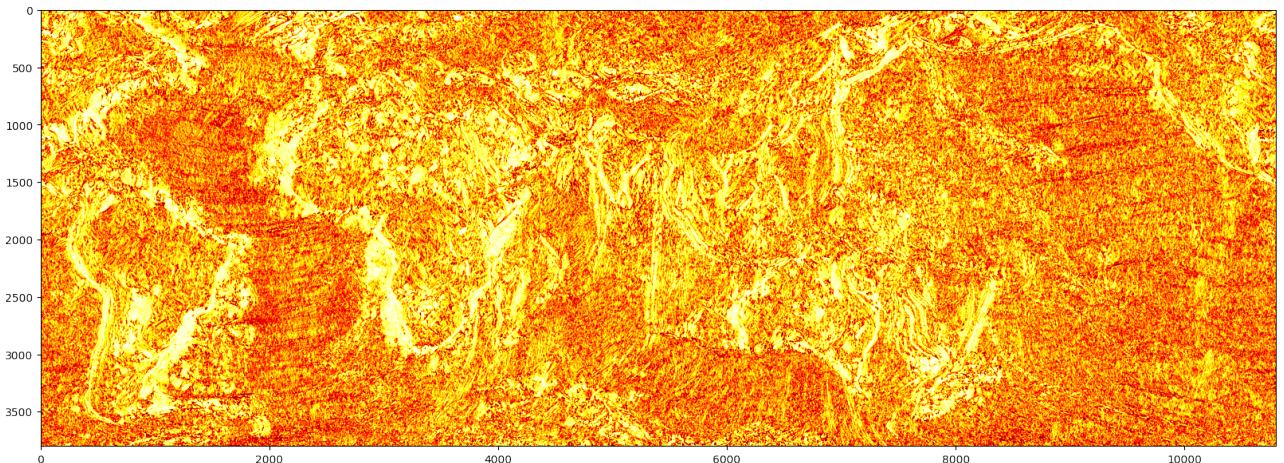


FIGURE 11 – L'organisation

Remarque : Le point d'arrêt est à $r = 15$, soit environ 60km , i.e une fenêtre d'observation de $124\text{km} \times 124\text{km}$, une échelle maximale suffisante pour analyser tous les motifs pertinents à classifier d'après notre étude des reliefs océaniques. L'analyse des reliefs locaux détecte en majorité des terrains plats à cette échelle. Il est également difficile de réaliser les calculs nécessaires pour des valeurs plus importantes de r (la complexité temporelle de la paramétrisation est un $O(T \times r^2)$ avec T la taille de la carte. Cette valeur maximale de r correspond aux travaux existants sur lesquels nous nous sommes appuyés.

4.3 Classification supervisée et création du jeu de données

Notre classification s'appuiera sur trois indicateurs principaux (pente, rugosité, organisation), avec un indicateur secondaire (l'altitude) (voir ci-dessus pour plus de détails sur ces indicateurs). Au lieu d'utiliser un algorithme de classification non supervisé comme K-means directement sur les données, le passage est fait par un arbre de décision codé à la main (approche inspirée de travaux existants[4]). En effet, on peut ainsi utiliser l'interprétation de ces paramètres pour en déduire des régions bien définies et interprétables.

On décide de définir 4 segments de pente (plat, pentu, très pentu, escarpé). 3 segments de rugosité (lisse, rugueux, très rugueux) et 2 segments d'organisation (organisé, très organisé). On peut donc diviser la carte en $4 \times 3 \times 2 = 24$ segments. On définit les délimitations des différents indicateurs à l'aide d'un clustering en une dimension.

L'observation des cartes nous donne plusieurs informations : les fonds marins sont généralement plus rugueux et désorganisés, les grandes pentes (en particulier la pente continentale) sont organisées mais ne sont pas toujours complètement lisses.

Utilisons ces informations pour diviser notre terrain en 6 régions : Terrain lisse, Terrain organisé, Terrain pentu, Terrain escarpé, Terrain très rugueux, Terrain très rugueux. L'objectif est d'avoir des régions spatialement cohérentes. Cette cohérence spatiale est amélioré par l'application d'un filtre de majorité.

On peut ensuite utiliser l'élévation comme variable auxiliaire pour créer une classification plus simple à interpréter.

- Un terrain lisse ou organisé à élévation élevée sera considéré comme faisant partie du plateau continental.
- Mais un terrain rugueux se trouve normalement au milieu de la mer, et donc un terrain rugueux à élévation trop élevée sera considéré comme faisant partie de la dorsale océanique. En effet, les reliefs de la dorsale océanique sont très rugueux et pas assez pentus pour pouvoir être identifiés ; mais cette méthode permet de contourner le problème et de délimiter la dorsale océanique.
- Les terrains organisés à basse élévation sont considérés comme faisant partie du glacis continental. En effet, cette zone formée par un dépôt de sédiments provenant du continent aura une structure bien organisée (une pente faible, mais décroissant de manière très régulière).
- Les terrains lisses à faible élévation sont considérés comme faisant partie de la plaine abyssale.
- Les terrains rugueux ou très rugueux à faible élévation sont classifiés comme respectivement : des fonds marins rugueux, ou des fonds marins très rugueux.
- Les terrains escarpés et pentus restent classifiés comme tel.
- Tous les points d'altitude > 0 sont classifiés comme de la terre.

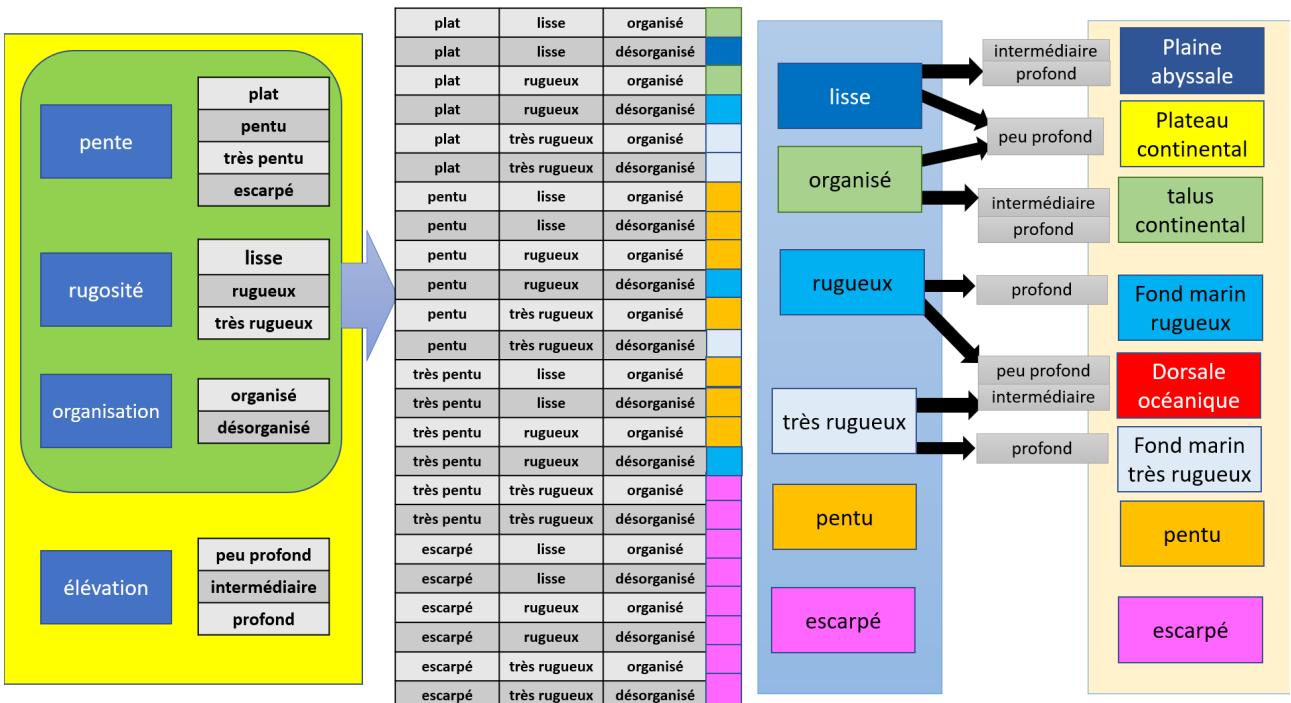


FIGURE 12 – L'organisation

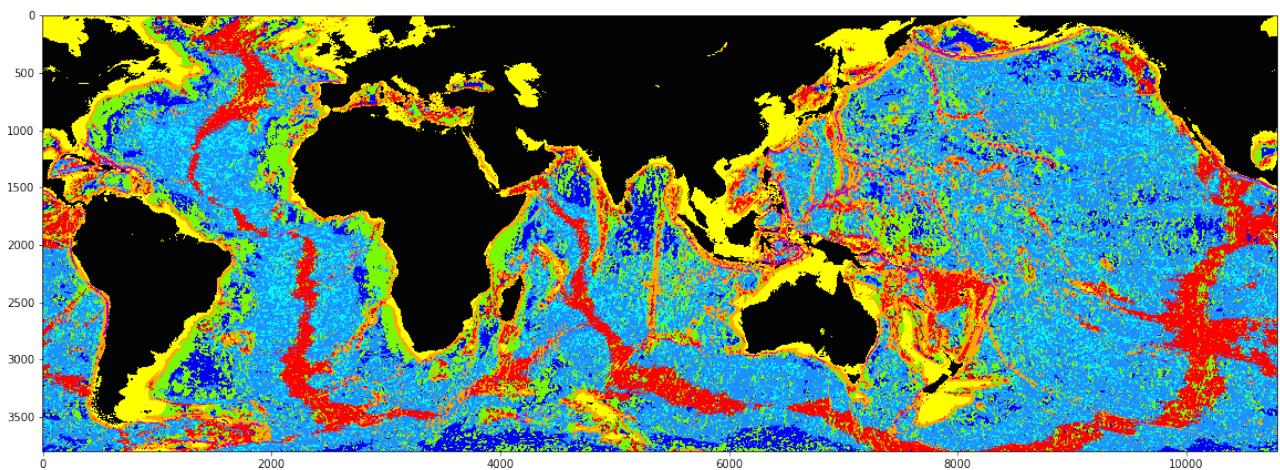


FIGURE 13 – La classification

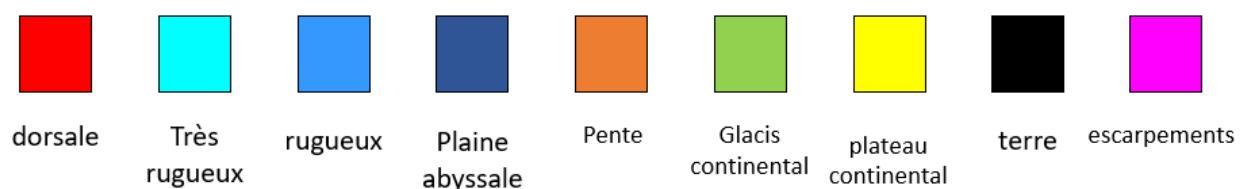


FIGURE 14 – légende de la classification

On observe que la carte de classification forme des structures reconnaissables et plutôt

consolidées spatialement. Cependant, il reste de nombreux points isolés, ainsi que des erreurs assez visibles au vu de l'interprétation des classes comme les points de glacis continental au milieu de l'océan, ou les zones de dorsales isolées sur les côtes.

On peut utiliser les interprétations des classes pour affiner la classification en imposant quelques règles. Le plateau continental doit être adjacent à la terre. Les zones violent ces contraintes seront reclassifiées selon leurs voisins.

Les zones pentues ou escarpées non adjacentes à la terre ou au talus continental (ou à une distance trop importante de ces derniers) seront reclassifiées dans une nouvelle catégorie : relief sous-marin.

Le reste des zones pentues seront considérées comme des pentes continentales.

Le glacis continental doit être adjacent à la terre, à la pente continentale/aux escarpements ou au plateau continental, où être connecté à ces derniers, par l'intermédiaire de plaine abyssale. Le glacis violent cette contrainte sera reclassifié comme plaine abyssale.

La dorsale doit être adjacente au sol océanique, les zones de dorsale en violation de cette contrainte seront reclassifiées en pente continentale ou en plateau continental selon leur pente.

Les parties connexes de la dorsale océanique devraient être très larges. La classification comportant du bruit, on définit le voisinage à une distance plus importante avant de calculer les composantes connexes et d'éliminer les zones de dorsale isolées.

On va également consolider les plaines abyssales car on sait qu'il s'agit en réalité de zones très larges et bien délimitées. On peut aussi incorporer aux dorsales une partie des reliefs océaniques qui leurs sont adjacents.

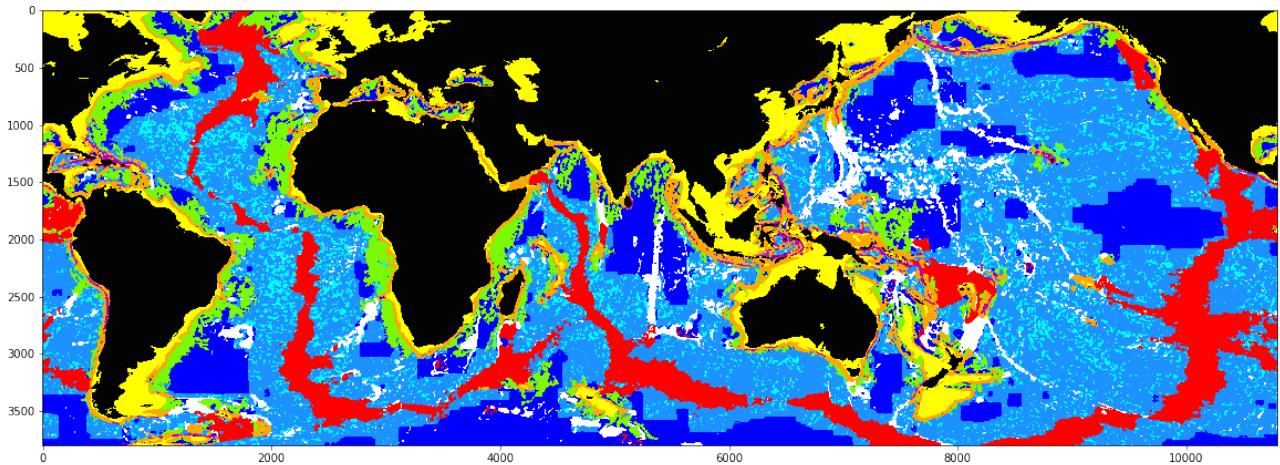


FIGURE 15 – La classification finale

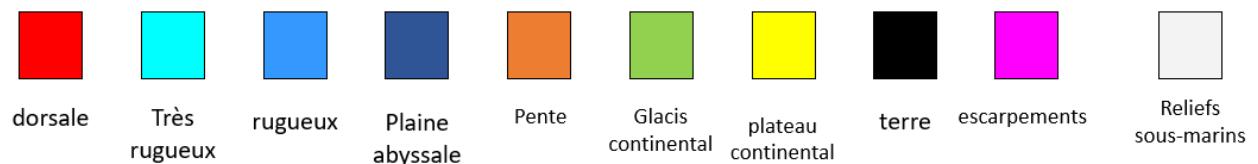


FIGURE 16 – légende de la classification finale

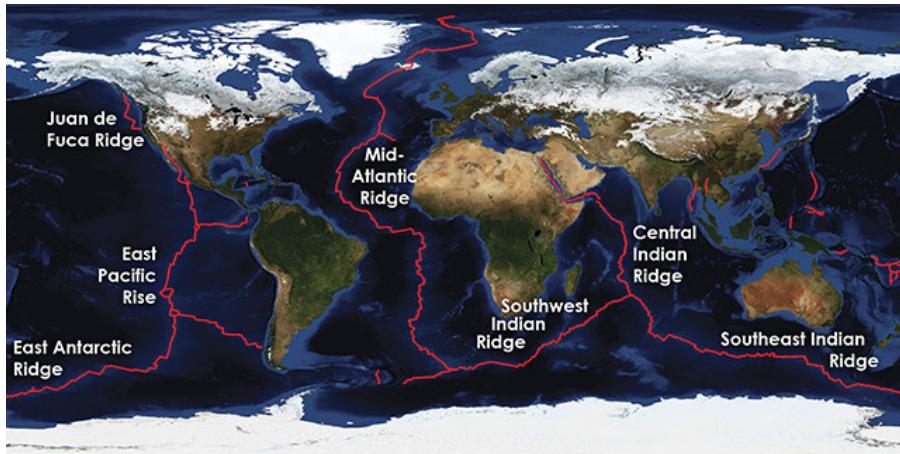


FIGURE 17 – Carte réelle des dorsales océaniques (pour comparaison)

Credit : K. Cantner, AGI

A cette étape, sont en disposition des régions plutôt bien délimitées spatialement, même si certaines classes restent intriquées entre elles comme les fonds marins rugueux et très rugueux. On peut aussi observer les pentes continentales : au niveau des pentes plus escarpées, il n'y a pas de glacis continental et on passe directement au fond-marin, appelés les marges de type pacifique (ou marges actives), par opposition aux pentes plus douces suivies d'un glacis continental : les marges de type atlantique (ou marges passives).

Cette segmentation du terrain en différents types sert à créer un jeu de données : un ensemble de profils d'altitude d'environ $250\text{km} \times 250\text{km}$, regroupés dans des classes.

Définissons d'abord les différentes classes du jeu de données, on va se baser sur la carte des types, et déterminer la classe d'une tuile en fonction des proportions des types qu'elle contient. On ne peut pas juste transposer les types aux classes en prenant le type majoritaire de chaque tuile car certains types, comme les pentes continentales, seraient effacés.

Les classes sont :

- plateau continental
- côte (terre + plateau continental)
- marge passive (plateau continental + glacis continental + pente sans escarpement)
- marge active (plateau continental + pente avec escarpement + fond marin sans glacis continental)
- glacis continental (glacis continental avec éventuellement un peu de plaine abyssale)
- dorsale océanique
- fond marin rugueux (fond marin rugueux+très rugueux à majorité rugueux)
- fond marin très rugueux (fond marin rugueux + très rugueux à majorité très rugueux)
- relief dans fond marin rugueux (relief sous-marin + fond marin rugueux)
- relief dans plaine abyssale (relief sous-marin + plaine abyssale)

On extrait entre 5000 et 10000 échantillons par classe (deux échantillons ne sont pas obligatoirement spatialement disjoints), deux échantillons peuvent partager une partie de leur surface. Pour chaque classe, on tire au hasard 10000 couples échantillons/rotation (0° , 90° , 180° , 270°) et on sauvegarde l'image après avoir appliqué la rotation. Les images sont sauvegardées au format png en niveau de gris 16 bits (donc des valeurs entières pouvant aller de 0 à 65535).

5 Génération des différents types par GAN

5.1 Motivations

Nous cherchons à générer différents types de terrain, et l'approche reposant sur du bruit de Perlin avec des paramètres ajustés pour chaque type n'est pas pleinement satisfaisante : on ne peut espérer aucune cohérence globale du terrain, et la paramétrisabilité est limité. En revanche l'approche du GAN semble prometteuse : étant donné un ensemble d'image de terrain d'un certain type, un GAN devrait pouvoir générer une image répondant à la même logique. La reproduction pourrait être alors beaucoup plus fidèle.

5.2 Principes

Les réseaux de neurones sont des modèles mathématiques inspirés du fonctionnement du cerveau humain. Ils sont constitués de couches chacune contenant de nombreux neurones artificiels, contenant chacun une donnée, interconnectés par des liaisons pondérées. Ces données sont traitées en effectuant des opérations mathématiques couche par couche.

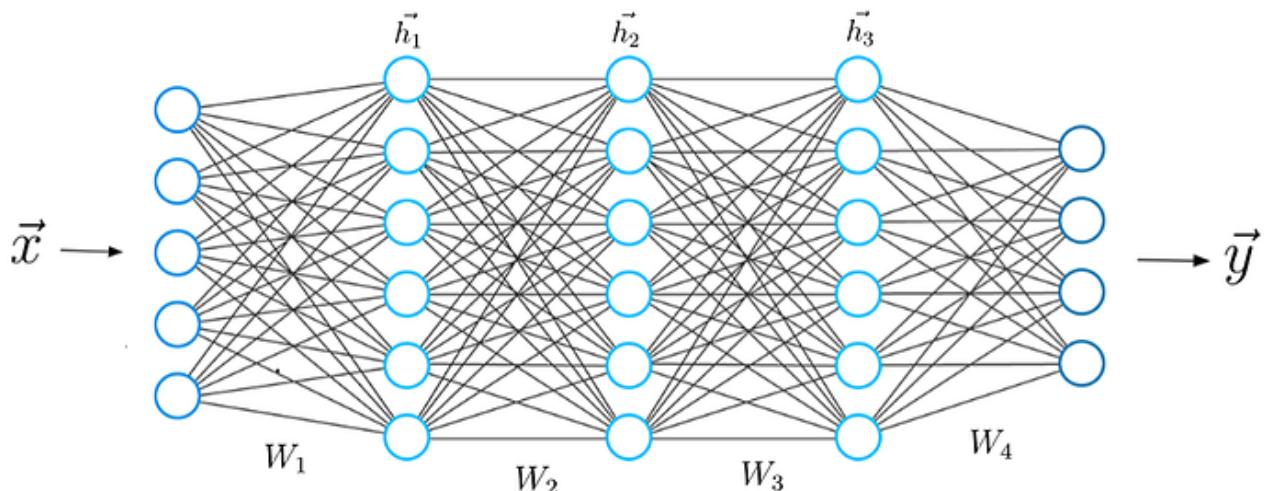


FIGURE 18 – Un réseau neuronal

Afin d'entraîner un réseau de neurones, on utilise une fonction de perte qui mesure l'écart entre les prédictions et les valeurs réelles associées aux exemples d'entraînement qu'on doit être minimisée afin d'améliorer les performances de ce réseau.

Les GAN utilisent deux réseaux de neurones, un appelé le générateur et l'autre le discriminateur. Le premier transforme des données aléatoires en données synthétiques, tandis que le second évalue leur qualité en les comparant à des données réelles. Tout en s'entraînant sur une large base de données constituée des données réelles.

La fonction de perte particulière qu'on utilise généralement pour les GAN est appelée la fonction de perte adverse qui est divisée en deux parties : la première mesure l'écart entre les prédictions du discriminateur et les vraies étiquettes des données, et la seconde mesure l'écart entre les prédictions du générateur et les étiquettes générées par le discriminateur dans

l'objectif qu'il attribue des étiquettes proches de 1 aux vraies données et des étiquettes proches de 0 aux données générées. En même temps, le générateur essaie de tromper le discriminateur en produisant des données qui obtiennent des étiquettes proches de 1. L'objectif global est de minimiser cette fonction de perte pour que le générateur produise progressivement des données réalistes.

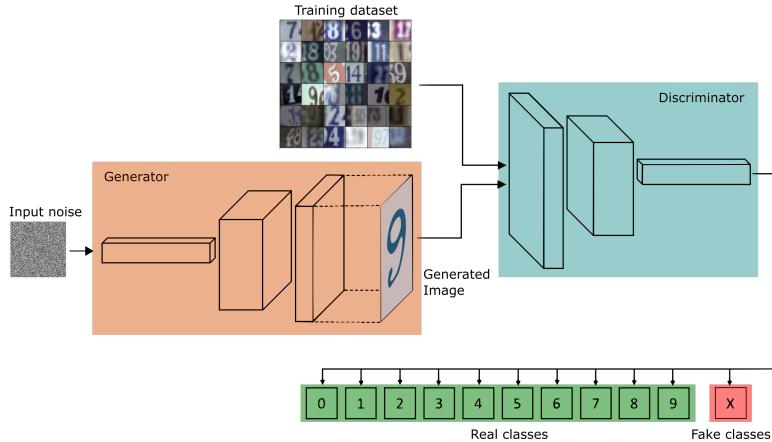


FIGURE 19 – Un GAN qui génère les chiffres de 0 à 9

Dans notre cas la description mathématique de notre réseau de neurones est une fonction G qui construit une variable aléatoire X qui suit une loi de relief Type $\in \{\text{Montagne, Plaine,...etc}\}$: $X = G^{-1}(\mathcal{U}([0, 1]))$.

5.3 Quelques exemples

Dans l'exemple plus haut, on a introduit le GAN générateur des chiffres de 0 à 9, il sera intéressant de détailler son fonctionnement pour comprendre les GAN en pratique.

Pour commencer, le générateur est un réseau de neurones qui prend un vecteur de bruit aléatoire comme entrée et produit une image de chiffre (par exemple, un chiffre manuscrit) en sortie. Le discriminateur est également un réseau de neurones, mais cette fois il prend une image (vraie ou synthétique) en entrée et produit une valeur unique qui représente la probabilité que l'image soit vraie (proche de 1) ou fausse (proche de 0).

Tout d'abord, un vecteur de bruit aléatoire est mis en entrée du générateur qui sera entraîné pour produire une image de chiffre synthétique à partir de ce vecteur.

Ensuite, le discriminateur est entraîné sur une large base de données de chiffres manuscrits et des images synthétiques produites par le générateur afin d'évaluer la probabilité que chaque image soit vraie ou synthétique. En conséquent il met à jour ses poids pour mieux distinguer les images réelles des images synthétiques et le générateur est de nouveau entraîné en utilisant la réponse du discriminateur à ses images synthétiques comme signal de rétroaction pour réajuster ses poids afin de produire des images de chiffres qui sont plus difficiles à distinguer des images réelles par le discriminateur.

Ce processus est réitéré jusqu'à ce que le générateur arrive à produire des images de chiffres synthétiques qui sont pratiquement indiscernables des images réelles.

Pour mesurer la performance du GAN, on utilise la fonction de perte adverse qui est une fonction spéciale utilisée pour l'apprentissage des GAN. Cette fonction est définie comme suit :

$$Loss = \log(D(x)) + \log(1 - D(G(z)))$$

où $D(x)$ représente la probabilité que l'image réelle x soit considérée comme vraie par le discriminateur, $G(z)$ représente l'image synthétique produite par le générateur à partir d'un vecteur de bruit z .

On a utilisé plusieurs couches dans ce GAN, en particulier :

1. Couche Dense (Fully Connected Layer) : Constituée de nombreux neurones interconnectés avec ceux de la couche précédente et de la suivante et utilisée pour apprendre des représentations complexes des données.
2. Flatten : une couche de pré-traitement qui sert à aplatisir les données d'entrée en un vecteur unidimensionnel
3. Couche de convolution(Convolutional Layer) : Elle consiste en une série de filtres qui permettent de détecter des motifs dans les données en entrée. Chaque filtre est déplacé sur l'image d'entrée pour produire une carte de caractéristiques en sortie.
4. BatchNormalization : Cette couche est utilisée pour normaliser les activations des neurones d'une couche précédente, en leur donnant une moyenne de 0 et un écart-type de 1. Cette normalisation permet d'accélérer l'entraînement et d'améliorer la stabilité du réseau.
5. LeakyReLU : une fonction d'activation qui introduit une pente négative pour les valeurs négatives, ce qui permet d'éviter la disparition du gradient ainsi que permettre à l'information d'être transmise même lorsque les activations sont faibles.
6. Couche de Pooling (Pooling Layer) : Elle permet de réduire la dimensionnalité des données en entrée en appliquant une opération de sous-échantillonnage, souvent celle appelée max-pooling, qui prend le maximum de chaque région de l'image et ne garde que cette valeur.
7. Couche de suréchantillonage (Upsampling Layer) : A l'inverse d'une couche de pooling, elle va augmenter la dimensionnalité, en remplaçant un point de l'entrée par une région de points ayant tous la même valeur dans la sortie.
8. Conv2DTranspose : Elle effectue l'opération inverse de la couche de convolution en agrandissant l'image en entrée. L'opération d'agrandissement de l'entrée repose sur l'application de filtres de convolution et est donc plus complexe qu'un simple suréchantillonage. On peut se la représenter comme combinant un suréchantillonage et une convolution.

Un autre exemple utilisé dans la génération des terrains est le pix2pix qui est un type de GAN qui a aussi pour objectif de réaliser une tâche de génération d'images mais au contraire du GAN précédent le générateur de pix2pix prend une image en entrée au lieu d'un vecteur unidimensionnel, à partir de laquelle il crée une seconde image. Le GAN est entraîné sur des paires d'images et apprend les liens entre les images d'entrée et les images qu'il doit créer.

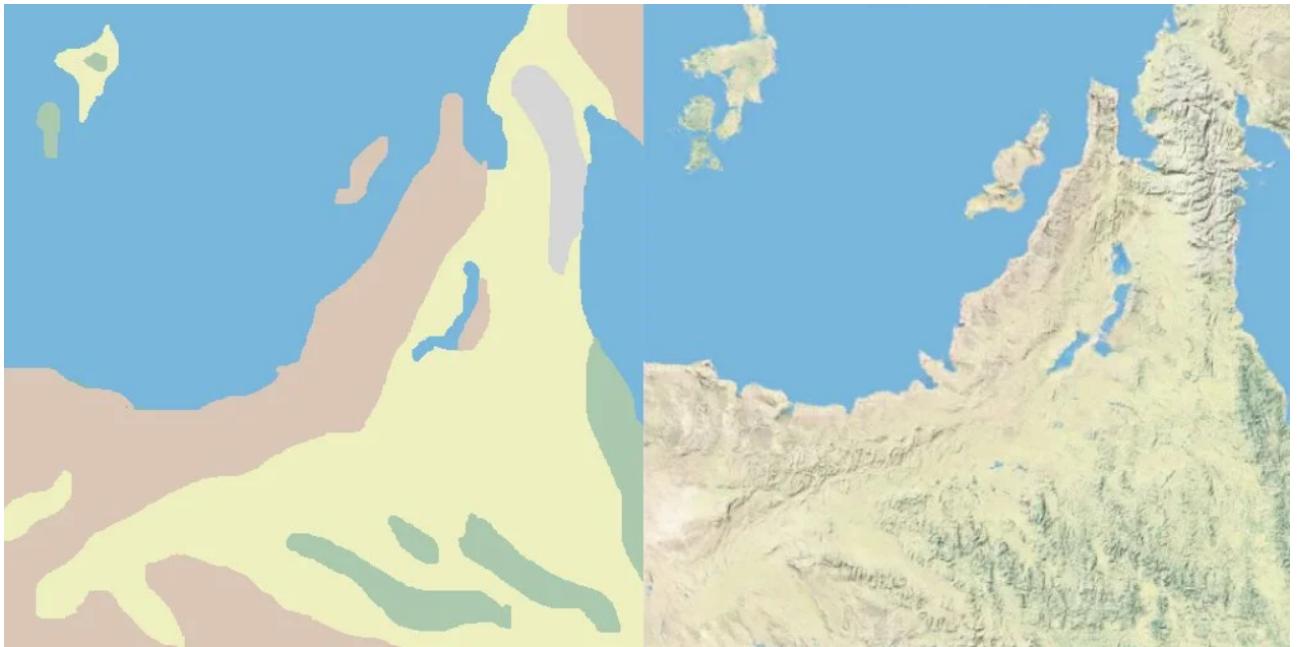


FIGURE 20 – pix2pix générant du terrain à partir d'une carte de segmentation

Dans l'exemple ci-dessus, la carte de segmentation du terrain est utilisée pour guider le générateur afin qu'il crée des images qui correspondent à la classe spécifiée dans la carte. Par exemple, dans les images ci-dessus, si la carte indique qu'une zone est une montagne, le générateur doit produire une image de montagne à cet endroit.

5.4 La convolution

La convolution est une opération mathématique implémentée à l'aide d'un noyau qui est appliqué à une image pour produire une nouvelle image. Pratiquement il s'agit de sélectionner une partie de l'image d'origine qui sera transformée par le noyau (appelée la « fenêtre de convolution »), ensuite les valeurs de la fenêtre de convolution sont multipliées par les valeurs correspondantes du filtre et sommées pour obtenir une seule valeur qui correspondra aux pixels de l'image de sortie. Une couche de convolution a généralement plusieurs filtres et chaque filtre correspond à une image 2D en sortie. Par exemple : une couche de convolution ayant une fenêtre de taille 3×3 , 128 filtres et une entrée de taille $(32 \times 32 \times 64)$ aura une sortie de taille $32 \times 32 \times 128$ chacun des 128 tableaux 2D étant le résultat de l'application du filtre correspondant sur l'image d'entrée (ou ici la somme des résultats de l'opération illustrée ci-dessus sur chacune des 64 images d'entrée).

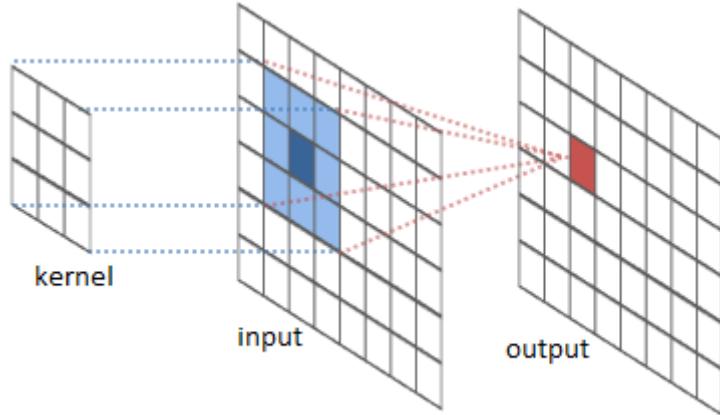


FIGURE 21 – Processus de convolution

Il est intéressant de noter qu'une couche de convolution ne requiert pas d'entrée à taille fixée, contrairement à, par exemple, une couche dense. Elle transforme une entrée de taille $n \times n \times m$ en une sortie de taille $n \times n \times N$ avec N son nombre de filtres. Cette particularité d'une couche de convolution est partagée par les couches avec lesquelles elle est souvent associée. Une couche de pooling transforme une entrée de taille $n \times n \times m$ en une sortie de taille $(n/2) \times (n/2) \times m$ en remplaçant 4 points de l'entrée par leur moyenne (ou parfois leur maximum selon le type de pooling) dans la sortie. A l'inverse, une couche de suréchantillonage va rempl

5.5 La perte de Wasserstein

L'un des problèmes que nous avons rencontré est que le discriminateur s'est entraîné plus rapidement que le générateur et cette domination complète du discriminateur a empêché l'entraînement de progresser car le générateur n'avait rien à apprendre (problème de la disparition des gradients). L'idée était alors de permettre un apprentissage correct, même quand le discriminateur domine le générateur. Au lieu de borner la sortie du générateur entre 0 et 1, et d'utiliser une fonction logarithmique pour la perte, on va simplement laisser une activation linéaire en sortie du générateur. Notre nouvelle perte, appelée perte de Wasserstein [5], sera tout simplement la valeur de cette sortie (sur les faux échantillons et son opposé sur les vrais échantillons). Elle peut donc être positive ou négative et n'est a priori pas bornée. On se prévaut contre une divergence vers l'infini en ajoutant un terme proportionnel (avec un faible coefficient) au carré de la sortie à la perte. L'avantage par rapport aux autres fonctions de perte est qu'on évite les problèmes de disparition des gradients dans le cas d'une domination du discriminateur et qu'on a donc pas à maintenir l'équilibre. Au contraire, si le discriminateur domine le générateur, une étape d'entraînement du générateur n'en sera que plus efficace. On peut donc même gagner en performances en entraînant le discriminateur plus souvent (une étape d'entraînement du discriminateur étant plus rapide qu'une étape d'entraînement du générateur).

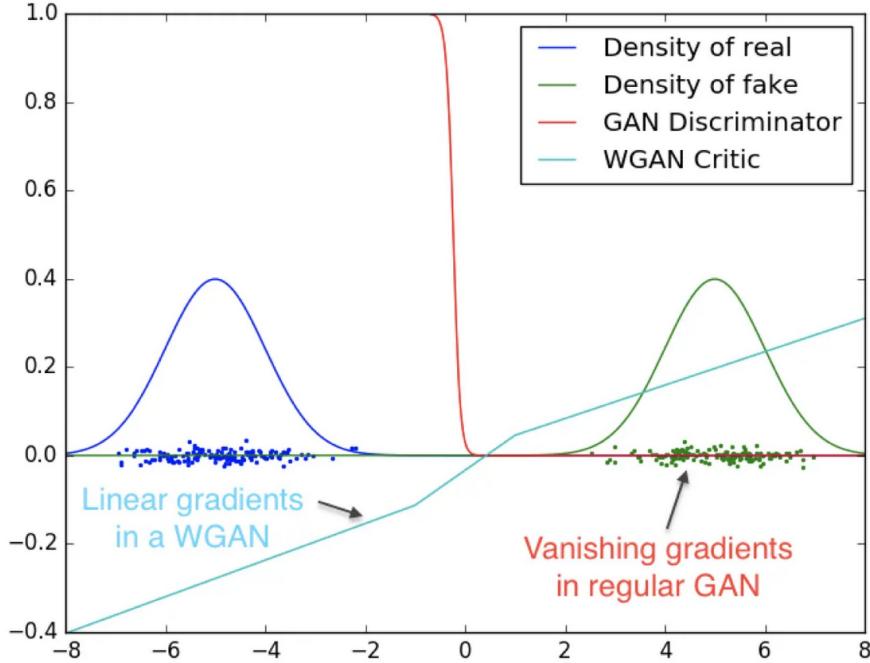


FIGURE 22 – Discriminateur optimaux pour la perte classique et la perte de Wasserstein dans le cas où le réseau essaie de différencier deux distributions Gaussiennes

Cependant, afin que l’entraînement puisse converger de manière optimale avec notre nouvelle fonction de perte, il est nécessaire de s’assurer que les gradient du discriminateur restent bornés (éviter le problème de l’explosion des gradients). On va donc ajouter un terme appelé pénalité du gradient [6], qui va correspondre à la différence entre 1 et la norme du gradient, cette mesure est réalisée sur un échantillon de type $\varepsilon * \text{vrai} + (1 - \varepsilon) * \text{faux}$ avec ε tiré au hasard dans $[0, 1]$.

Formule finale de la perte (pour un bruit aléatoire z et une image réelle X) :

$$D\text{Loss} = D(G(z)) - D(X) + \lambda \|(\nabla D)(\varepsilon G(z) + (1 - \varepsilon)X) - 1\|_2 + \mu D(X)^2$$

$$G\text{Loss} = -D(G(z))$$

5.6 Architecture et implémentation

On utilise la bibliothèque tensorflow pour l’implémentation. Cette bibliothèque permet de définir un graphe d’opérations et de variables en python, graphe qui sera ensuite exécuté. Il est ainsi facilement possible de remonter un tel graphe pour calculer des gradients, ainsi que de précompiler certaines opérations pour accélérer l’exécution.

Au niveau de la lecture du jeu de données : un ensemble de fichiers png avec un seul canal à 16 bits (entiers entre 0 et 65535), répartis dans des sous-dossiers selon leurs classes. On définit un pipeline d’entrée : les images sont lues et normalisées à moyenne 0 et variance 1, la sortie est une image de taille $128 \times 128 \times 1$, une étiquette correspondant à la catégorie, ainsi que l’ancienne moyenne et variance. Étant donné que le jeu de donnée est trop volumineux pour être chargé en mémoire, Tensorflow exécutera le pipeline en parallèle de la boucle d’entraînement

(en préparant le groupe d'images de l'itération $i + 1$ pendant l'exécution de l'itération i afin d'éviter que la lecture des fichiers sur le disque devienne bloquante pendant l'entraînement).

On choisit l'architecture suivante :

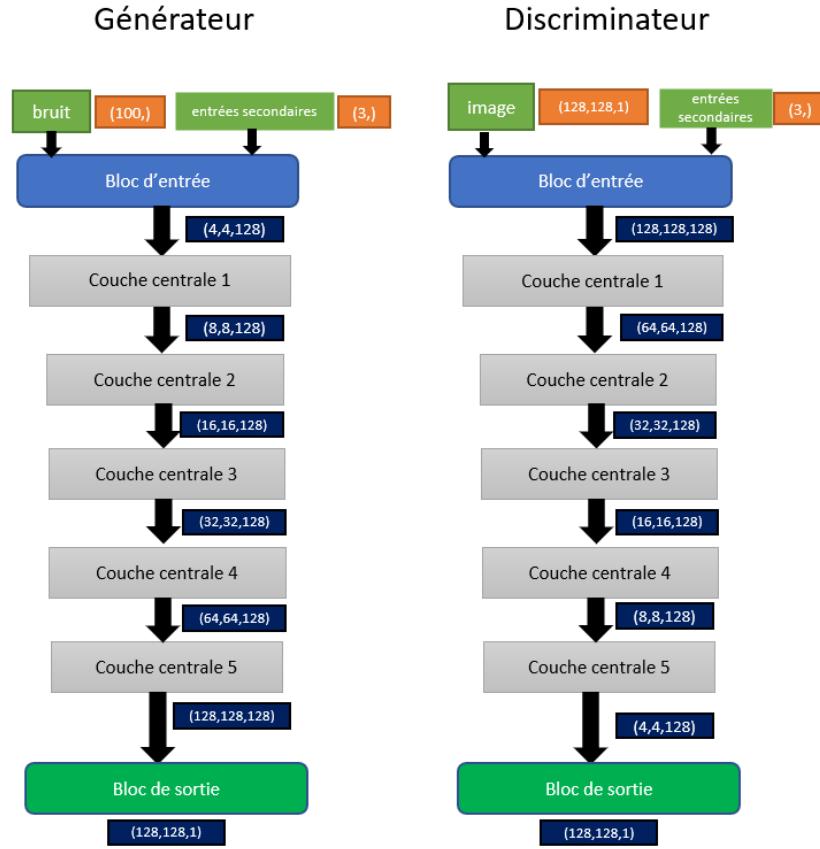


FIGURE 23 – Architecture générale du réseau

On veut conditionner les images générées par leur catégorie (ainsi que leur moyenne et variances réelles), ce qui est pourquoi on va incorporer cette catégorie (et moyenne/variance) aux entrées du générateur et du discriminateur. Le discriminateur apprendra alors à prendre la catégorie en compte, ce qui poussera le générateur à faire de même. Image bloc d'entrée du générateur et du discriminateur.

Entrée générateur

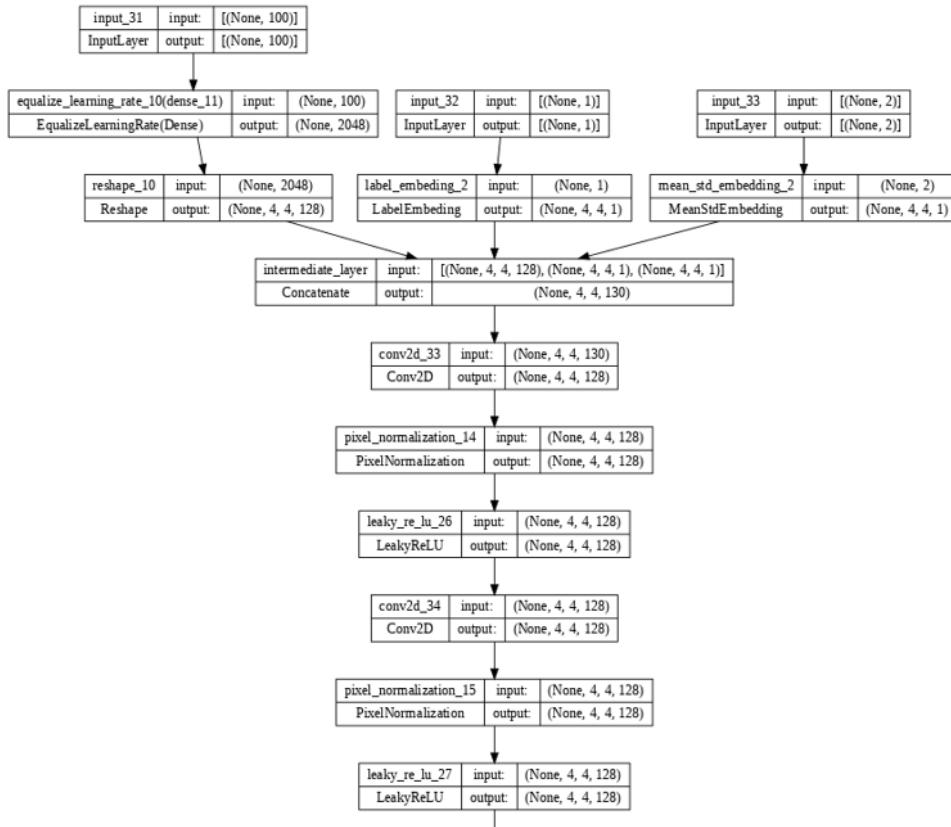


FIGURE 24 – Entrée du générateur

Entrée du discriminateur

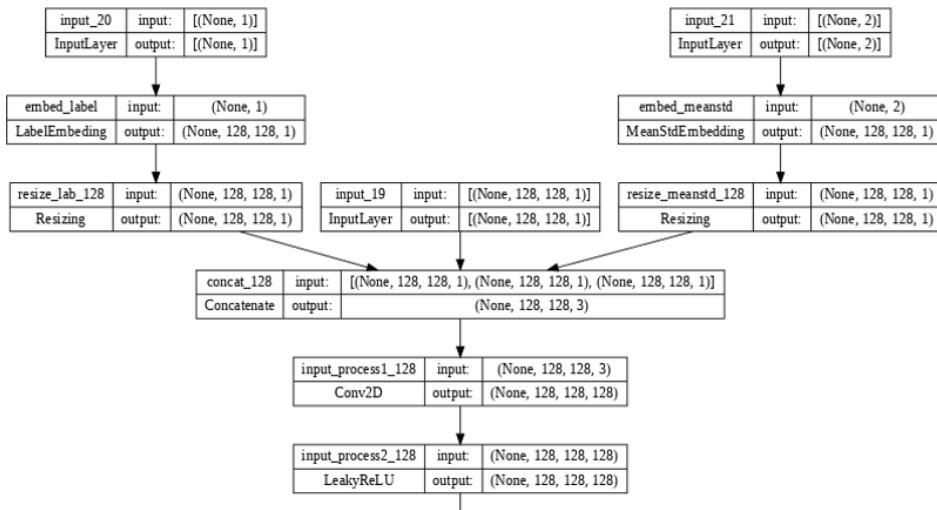
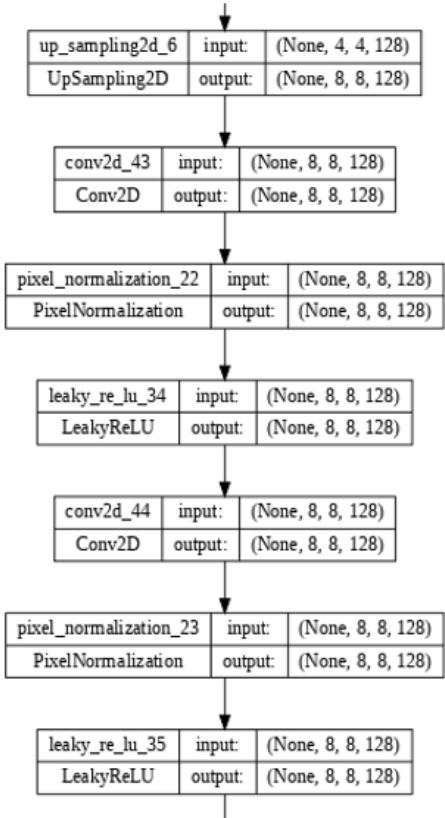


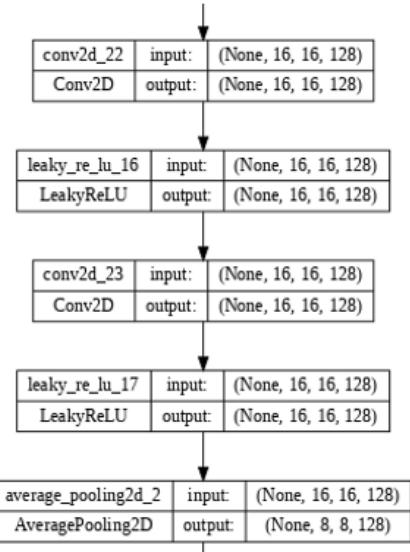
FIGURE 25 – Entrée du discriminateur

Bloc central du générateur



(a) Couche principale générateur

Bloc central du discriminateur



(b) Couche principale discriminateur

FIGURE 26 – Les couches principales du GAN

On a choisi d'utiliser un suréchantillonage suivi de convolutions classiques à la place de convolution transposées afin de limiter la présence d'artefacts damiers qui posaient problème auparavant. En effet, une convolution transposée soit, comme mentionné plus haut, une manière de combiner efficacement convolution et suréchantillonnage ; mais, dans notre cas, un suréchantillonnage suivi de convolutions, bien qu'un peu plus difficile à entraîner permettait d'éviter des artefacts au niveau de la sortie.

On va également améliorer la stabilité et la vitesse de l'entraînement avec une croissance progressive du réseau [7]. On optimisera d'abord le générateur et le discriminateur sur des images de taille 4×4 , puis on rajoutera un bloc à chacun, ce qui aura pour effet de changer la taille de la sortie du générateur (et de l'entrée discriminateur) à 8×8 , tout en continuant à profiter de l'entraînement existant des premières couches. On passera ensuite à 16×16 puis 32×32 ,... jusqu'à la taille originale de nos images d'entraînement. Toutes les couches restent entraînables pendant toute la durée de l'entraînement.

Afin d'éviter un choc dans l'entraînement, on met en place une phase d'apparition progressive des nouvelles couches, en entraînant d'abord un réseau de type $\alpha \times \text{nouveau} + (1-\alpha) \times \text{ancien}$ avec α augmentant progressivement de 0 à 1 au cours de cette phase. On passe ensuite à la phase principale d'entraînement pour ce niveau de profondeur (équivalent à $\alpha=1$) avant d'introduire la couche suivante, et ainsi de suite.

L'avantage est que l'entraînement des premières couches sera beaucoup plus rapide et bien qu'elles continuent à être entraînées après l'introduction des nouvelles couches, elles seront déjà optimisées et requêteront donc moins d'effort au niveau de l'entraînement, ce qui permettra donc un entraînement plus rapide.

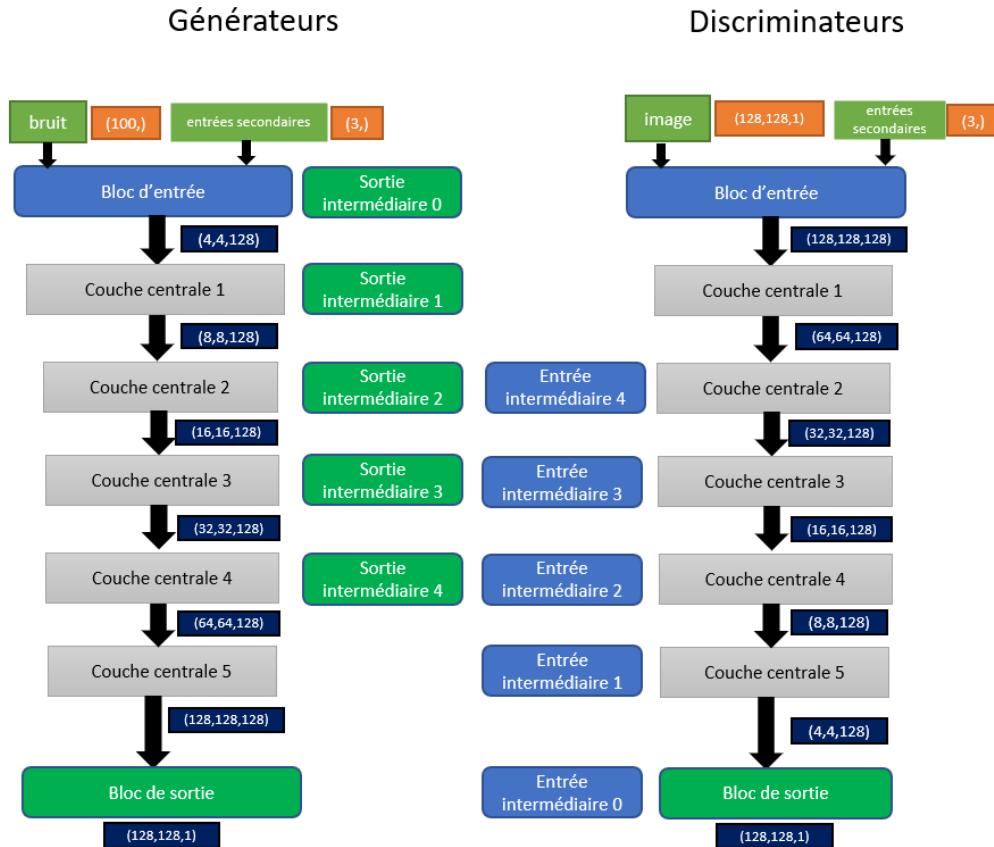
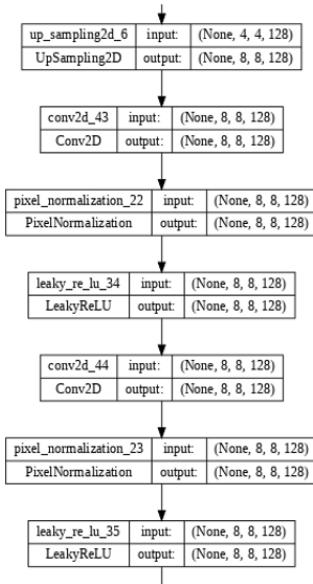


FIGURE 27 – Croissance progressive du réseau

La perte de Wasserstein réduit l'effondrement des modes, i.e. la tendance du générateur à ne générer qu'une seule image, mais, dans notre cas, elle n'est pas suffisante pour éviter entièrement le phénomène, ce qui est pourquoi on ajoute dans le discriminateur une couche calculant la variance pixel par pixel sur plusieurs images, le discriminateur apprendra à rejeter les groupes d'images de variance faible si le générateur commence à générer des images trop similaires entre elles.

Bloc central du générateur



Bloc sortie générateur

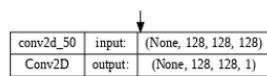


FIGURE 28 – Blocs de sortie du discriminateur et générateur

On définit donc nos modèles de générateurs et de discriminateurs dans tensorflow et on précompile les opérations de la boucle d’entraînement. Ce processus est répété à chaque augmentation du niveau de profondeur, en ajoutant un niveau aux modèles de la profondeur précédente. Les taux d’apprentissages, et les durées d’entraînement pour les différentes profondeurs sont disponibles dans le tableau ci-dessous.

Profondeur	Taille de la sortie	Durée de la phase d'apparition progressive	Durée de la phase principale	Taille de batch	Taux d'apprentissage (générateur et discriminateur)
0	4x4		5 epochs (< 1h)	20	5×10^{-4}
1	8x8	5 epochs (< 1h)	5 epochs (< 1h)	20	5×10^{-4}
2	16x16	10 epochs (1h)	10 epochs (1h)	20	5×10^{-4}
3	32x32	15 epochs (2h)	15 epochs (2h)	20	5×10^{-4}
4	64x64	20 epochs (13h)	30 epochs (15h)	20	5×10^{-4} à $\sim 1 \times 10^{-4}$ (décroissance avec le temps)
5	128x128	20 epochs (20h)	50 epochs (44h)	20	1×10^{-4} à $\sim 1 \times 10^{-5}$ (décroissance avec le temps)

Un epoch correspond à un entraînement sur la totalité du dataset (i.e. 11 000 images divisées en batchs de 20, donc 550 entraînements). La durée réelle d'un epoch augmente avec la taille du réseau.

FIGURE 29 – Paramètre de l'entraînement

L'entraînement a été réalisé sur une carte graphique (Nvidia K80/T4), en environ 4 jours.

5.7 Génération du profil d'altitude

Quelques images générées (abp : plaine abyssale, csh : plateau continental, rs/vrs : fonds marins rugueux/très rugueux, s-rs/s-abp : relief sous-marin dans fond marin rugueux/plaine abyssale)

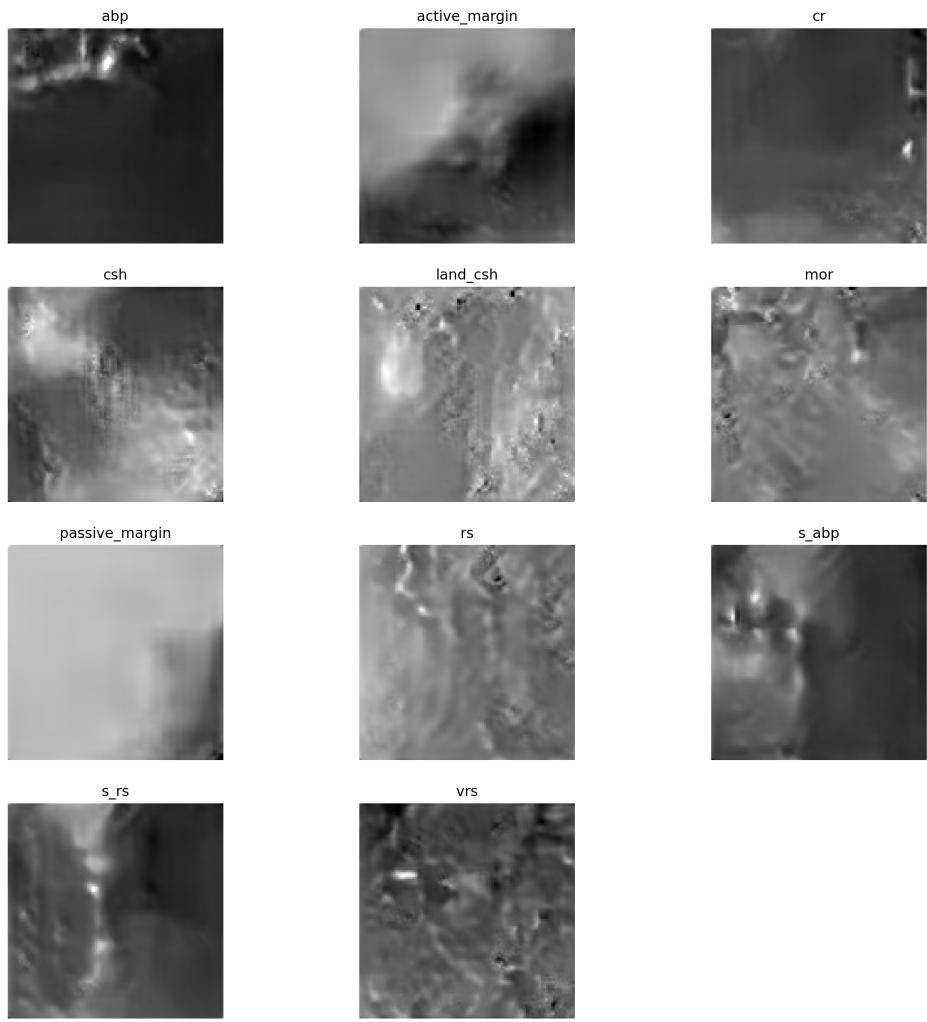


FIGURE 30 – Images finales

Observons les générations du GAN à différents niveaux de croissance progressive (les sorties intermédiaires du générateur) avec la même entrée.

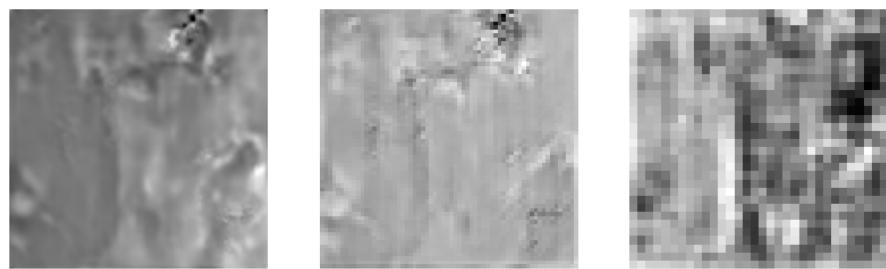


FIGURE 31 – fonds marins rugueux sur la sortie finale et les deux dernières sorties intermédiaires

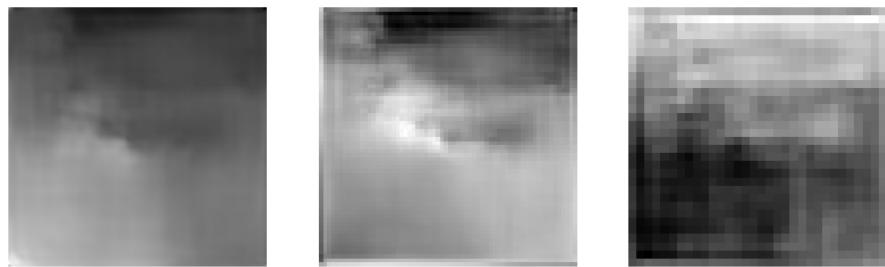


FIGURE 32 – Pente continentale sur la sortie finale et les deux dernières sorties intermédiaires

On peut observer que sur la seconde image, la pente est inversée sur la troisième sortie intermédiaire. En effet, quand le générateur ajoute une couche, il doit simplement produire une image plausible de pente continentale, pas forcément la même image que celle du bloc précédent. Le but est que l'entraînement du bloc précédent soit utile aux blocs suivant, ce qui reste le cas, même si sa sortie y est inversée.

Comparons maintenant notre GAN à des échantillons réels.

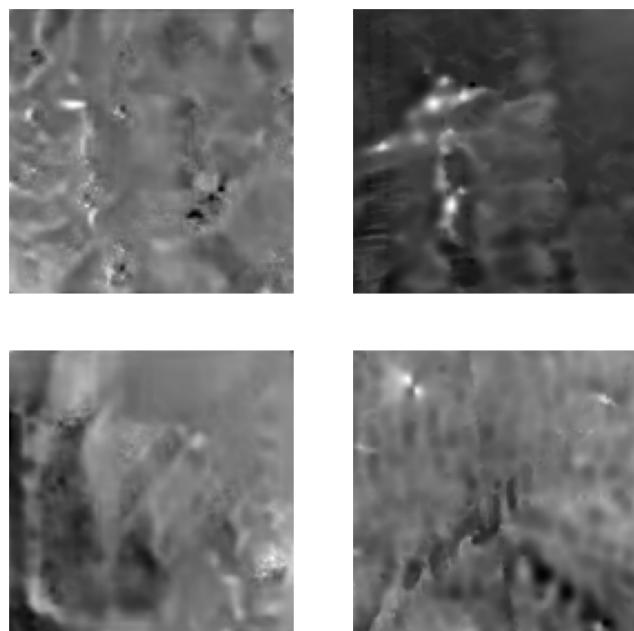
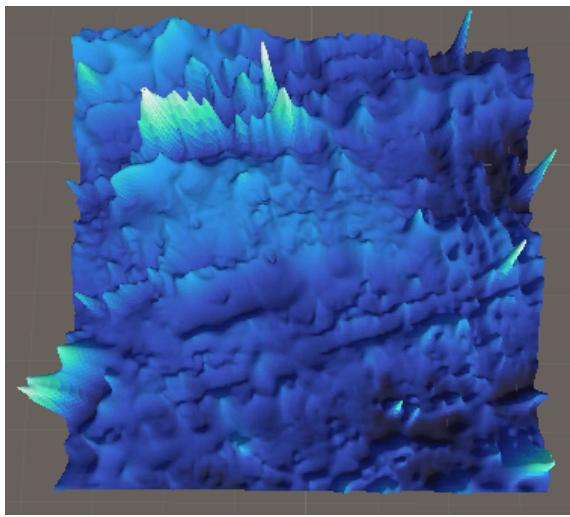
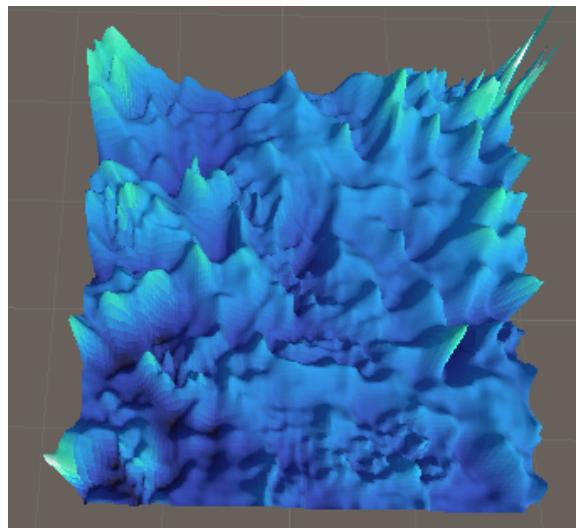


FIGURE 33 – comparaison échantillons réels (à gauche) et générés (à droite) sur sol rugueux



(a) Visualisation 3D d'un échantillon réel



(b) Visualisation 3D d'un échantillon synthétique

FIGURE 34 – Visualisation 3D de la comparaison entre échantillons

5.8 Génération d'un large profil d'altitude

Nous sommes capables de générer des profils d'altitude de taille 250 par 250 km. Les sorties du GAN sont en variance 1 et moyenne 0, mais, étant donné que la variance et la moyenne réelle sont des variables auxiliaires de la génération, on peut remettre nos échantillons à l'échelle sans occasionner trop de problèmes (la moyenne et la variance qu'on utilisera pour les remettre à l'échelle seront forcément adaptées à l'échantillon, étant donné qu'elles ont conditionnées sa création).

Le problème devient alors la création de profils de terrains plus grands, comment s'assurer de la continuité au niveau du raccordement d'échantillons. L'une des solutions est alors de se servir du GAN et de son architecture spécifique [8]. Au lieu d'assembler les sorties du générateurs, on assemble les sorties des couches intermédiaires avant de redonner l'assemblage en entrée au reste du générateur. En effet, à l'exception du bloc d'entrée, le générateur est entièrement convolutionnel, ce qui veut dire qu'il n'a pas de taille d'entrée fixe. Le réseau qui générera une image de taille 128×128 avec une entrée de taille 8×8 générera de la même manière une image de taille 512×512 avec une entrée de taille 32×32 . Mais étant donné qu'une couche convolution applique une série de filtres sur l'entrée, elle prend en compte le voisinage de chaque point de l'entrée pour la création de la sortie. Ce qui résulte naturellement en une sortie globale plutôt continue. Cependant, ces couches ont quand même été entraînées sur des entrées d'une certaine taille et ont appris à prendre en compte les phénomènes au bord. On évite donc de prendre des couches ayant une taille d'entrée trop petite, mais d'un autre côté, des couches à taille d'entrée importante seraient trop proches de la sortie, ce qui augmenterait les discontinuités. On choisira finalement un point de coupure entre les couches 2 et 3, soit une taille d'entrée de 16×16 .

Pour récapituler, notre algorithme de génération final prendra en paramètre : un tableau de types, un tableau de moyennes et un tableau de variances. Les cases sont remplies de gauche à droite et de haut en bas. Pour chaque case, on générera un certains nombres d'échantillons et on prendra le plus adapté aux cases déjà placées (soit la case à gauche et la case au-dessus). Ensuite,

on lancera la deuxième moitié du générateur sur la grille ainsi créée. Pour remettre les altitudes à l'échelle, on interpole les moyennes et les variances avant de faire une multiplication/addition point par point. On appliquera également un filtre médian à la sortie afin d'effacer les artefacts restants. Cette approche à la mise à l'échelle limite les discontinuités, mais elle génère des artefacts dus à l'interpolation, ce qui est pourquoi notre génération ne permet que de créer des terrains avec une variance très limitée dans l'altitude moyenne. De plus, notre méthode d'assemblage ne fonctionnera pas avec des types très organisés comme la pente continentale.

Le Pseudo-Code

Inputs :

- type: array (size , size) is terrain type for each tile
- means: array (size , size) contains mean elevation for each tile
- stds: array (size , size) contains standard deviation of elevation in

Initialize full_intermediary_input array (size , size , 128)

For i=0 to size :

 For j=0 to size :

 If (i , j) is the first position :

 Use GAN model and types[i , j] to generate an image and store it
 in selected_img

 Else :

 Generate a list of potential images using GAN model
 and types[i , j]

 Compute distances between potential images and adjacent samples

 Select the potential image with minimum distance and store it
 in selected_img

 Copy the selected_img to the current position of
 full_intermediary_input

Use GAN model to generate an image from full_intermediary_input and store
it in result

Resize means and stds arrays to result size

Compute final_result by multiplying result and stds , adding means

Output: final_result is a 2D array (final_size , final_size) where final_size
depends on input size (i.e number of tiles) and the tile overlap parameter

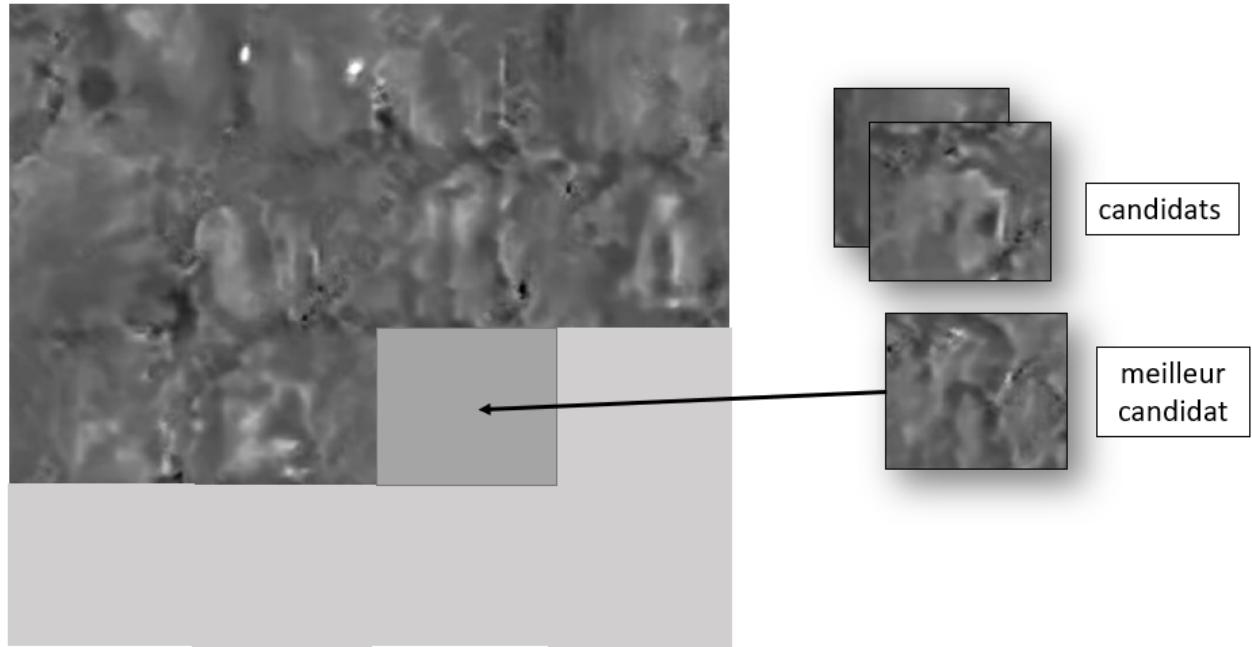
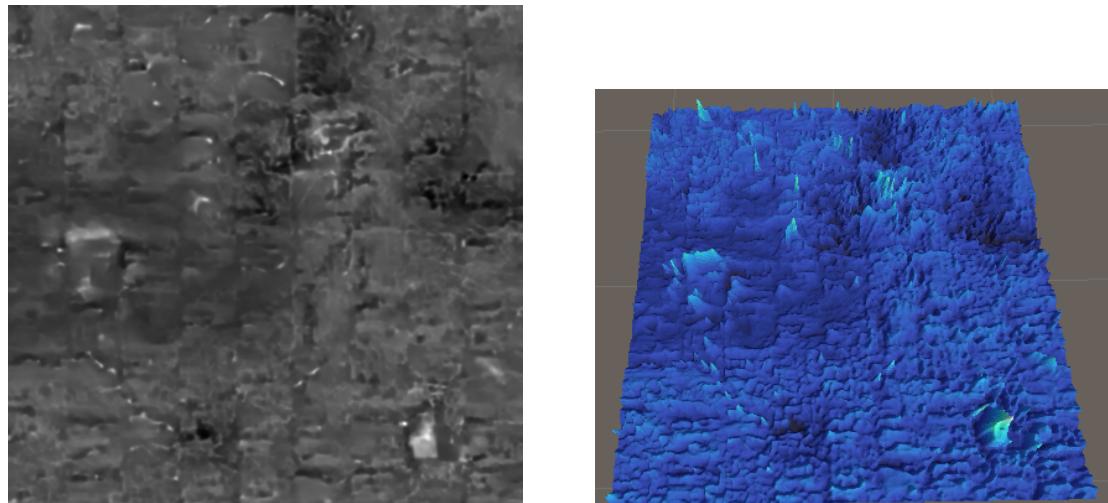


FIGURE 35 – Processus de génération



(a) Un terrain généré (fond marin avec quelques zones de reliefs)

(b) Vue 3D de haut du terrain généré

FIGURE 36 – Terrain généré en 2D et 3D (vue de haut)

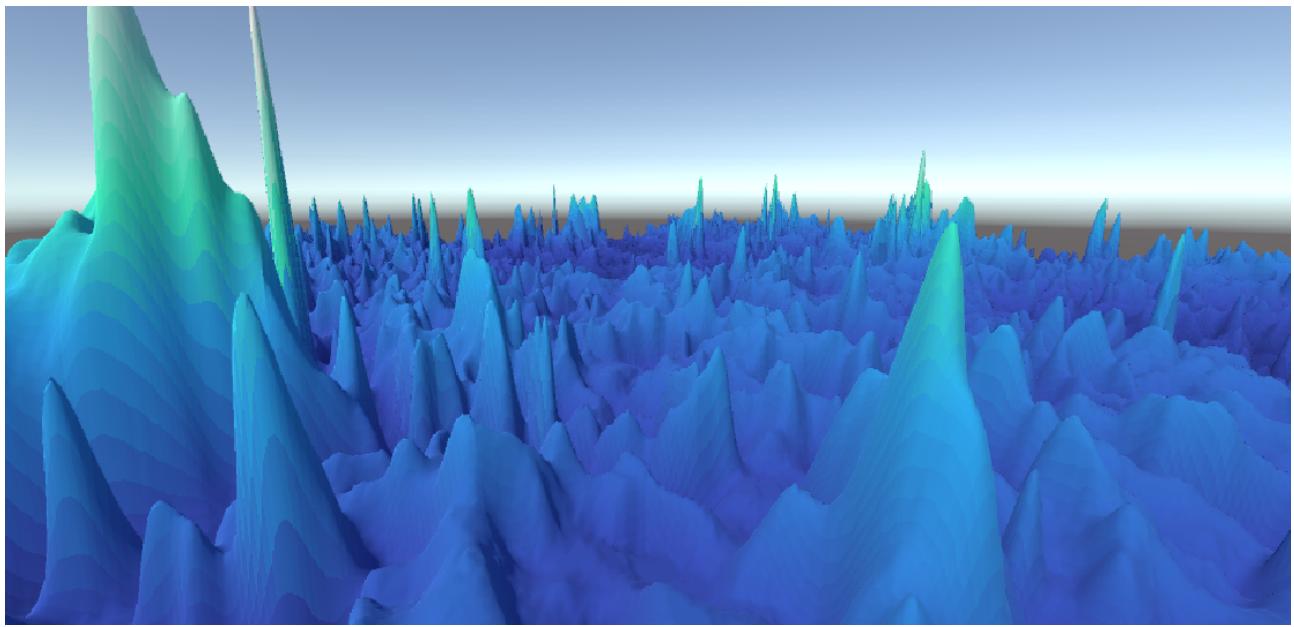


FIGURE 37 – Vue 3D immersive du terrain généré

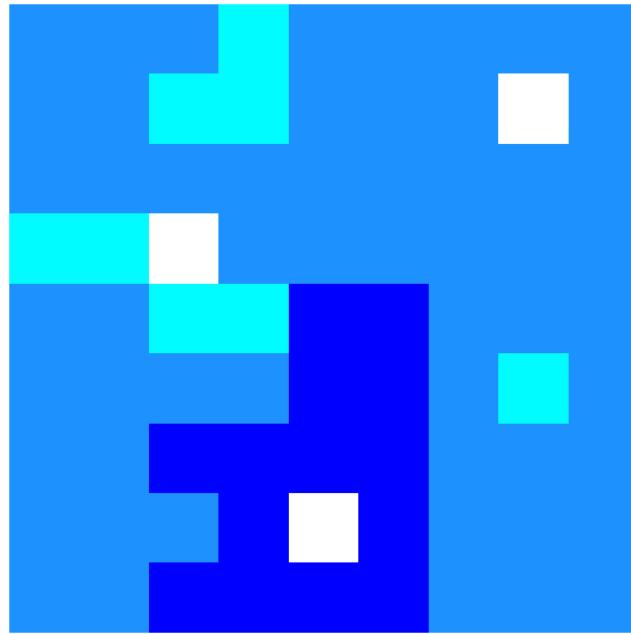


FIGURE 38 – Carte des types utilisée pour le terrain généré (bleu=rugueux, cyan=très rugueux, bleu foncé=plaine abyssale, blanc=reliefs)

6 Génération du profil local

6.1 L'utilisation d'un bruit pour le profil local

Pour créer une texture satisfaisante pour notre profil marin généré, il est essentiel de prendre en compte la variabilité naturelle des reliefs sous-marins. Malheureusement, nos données

ne nous permettent pas d'avoir accès à toutes les informations nécessaires pour créer une texture réaliste. Afin de contourner ce problème, nous utilisons deux mécanismes pour améliorer notre rendu graphique.

Le premier mécanisme consiste en la génération d'un bruit aléatoire que nous superposons à notre profil initial. Le bruit permet d'ajouter des variations locales à notre relief en simulant l'effet de petites irrégularités qui peuvent exister dans les fonds marins. Nous utilisons pour cela un algorithme de bruit fractal et/ou un bruit de Perlin qui nous permet de générer un bruit cohérent sur l'ensemble de notre grille. Nous pouvons ajuster les paramètres de l'algorithme pour obtenir un bruit plus ou moins granuleux, ou pour contrôler la fréquence des variations. Le bruit ajouté peut ensuite être combiné avec notre profil initial pour obtenir un nouveau profil plus complexe.

Le deuxième mécanisme consiste à appliquer un outil d'érosion qui permet d'obtenir un relief plus réaliste en modifiant la forme des montagnes et des vallées. Nous utilisons pour cela l'algorithme d'érosion hydraulique que nous avons développé, qui simule l'action de l'eau sur le terrain. L'application de cet outil permet de créer des crevasses, des ravins, et des falaises qui donnent un aspect plus naturel à notre relief sous-marin. En utilisant une combinaison de ces deux mécanismes, nous pouvons obtenir une texture satisfaisante pour notre profil marin généré.

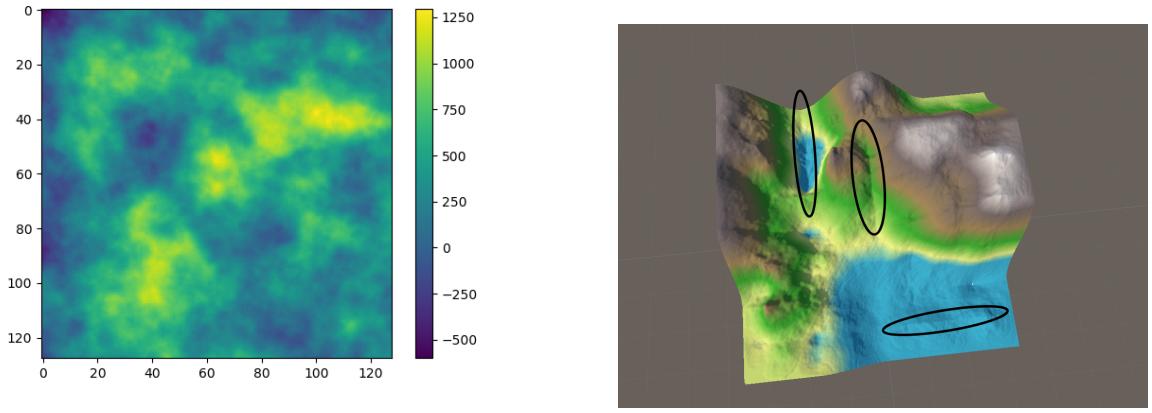
Pour générer notre bruit on s'est d'abord appuyé sur un bruit fractal[9] qui est une méthode très utilisée pour générer des textures naturelles, car elle permet de créer des détails à différentes échelles de manière réaliste. La superposition de plusieurs couches de bruit à des fréquences différentes crée des variations de hauteur plus ou moins fines, semblables à celles que l'on peut observer dans la nature.

Rigoureusement : étant donné une fonction pseudo aléatoire f définie sur \mathbb{C} , le bruit en un point de coordonnées (x, y) est défini par $B(x, y) = \sum_{k=0}^n \alpha^k f(\omega^k(x + iy))$ avec $\alpha < 1$ et $\omega \in \mathbb{C}$ des paramètres.

En pratique, la fonction f est définie sur \mathbb{Z}^2 et ses valeurs en des points à coordonnées non entière sont obtenues par interpolation à partir des quatre points à coordonnées entière les plus proche par une fonction lisse $s(x)$ (s pour smooth) vérifiant $s(0) = 0$, $s(1) = 1$ et éventuellement des propriétés supplémentaires au niveau des dérivées pour limiter la présence d'artefacts. Le bruit est ainsi qualifié de fractal car il est la superposition de plusieurs couches d'un même bruit (la fonction f) qui est atténué par un facteur α et zoomé d'un facteur ω à chaque itération (ce qui inclus une rotation). Ainsi, il y a $n + 1$ fréquences superposées : $1, |\alpha|, \dots, |\alpha|^n$ d'un même bruit.

L'interpolation joue également un rôle important dans la qualité du bruit généré. Une interpolation trop simple peut donner des motifs artificiels ou des bordures visibles entre les différentes couches de bruit, alors qu'une interpolation plus complexe peut donner un bruit plus réaliste, nous avons choisi la fonction $s(x) = -2x^3 + 3x^2$ reconnue par sa simplicité et popularité.

Il est effectivement important de choisir les paramètres de manière judicieuse pour obtenir une texture réaliste. Par exemple, pour simuler des montagnes, on peut choisir un facteur d'atténuation α relativement élevé pour obtenir des structures plus irrégulières et abruptes. À l'inverse, pour simuler des plateaux, on peut choisir un α plus faible pour obtenir des structures



(a) Bruit généré par la méthode fractale

(b) Des artefacts sur le terrain générée

FIGURE 39 – Terrain généré par bruit fractal et les artefacts observés

plus régulières et moins accidentées. De même, la valeur de ω permet de contrôler l'échelle de la texture, tandis que le nombre de fréquences superposées permet de contrôler le niveau de détail.

En fin de compte, le choix des paramètres dépendra du type de relief que l'on souhaite simuler et des résultats visuels que l'on souhaite obtenir. Il peut donc être nécessaire de procéder à plusieurs essais pour trouver les paramètres optimaux.

Cependant l'interpolation bilinéaire fait apparaître des artefacts : des effets indésirables qui font apparaître la grille de génération du terrain. C'est pourquoi on a pensé à l'érosion hydraulique pour permettre un passage plus naturel entre les zones.

Nous avons également examiné une méthode alternative, le bruit de Perlin, qui est bien documenté et a de nombreuses variantes et améliorations. Nous avons pris en compte les versions de 1985 et 2002 de Perlin, ainsi que les améliorations proposées en 2008 notamment le Simplex Noise. Cette technique permet de produire une grande variété de textures avec des propriétés variées, telles que des montagnes, des vallées, des plaines, et bien plus encore.

Cette méthode utilise des fonctions de bruit pseudo-aléatoires pour générer des textures naturelles. Le bruit de Perlin[10][11] est souvent utilisé dans les domaines de la génération de terrain, de l'animation, des effets spéciaux, et du graphisme en général.

Le bruit de Perlin , comme le bruit fractal, repose sur la combinaison de plusieurs octaves de bruit. Chaque octave est définie par une fonction de bruit pseudo-aléatoire générée à une fréquence plus élevée que l'octave précédent. Les octaves inférieures sont des versions plus « grossières » du bruit, tandis que les octaves supérieurs sont plus « détaillées ».

La méthode de combinaison des octaves utilise une interpolation bilinéaire pour obtenir une texture cohérente et sans artefacts. Cette méthode permet de générer des textures continues et réalistes.

6.2 Algorithme de l'érosion hydraulique

L'érosion est un processus naturel d'altération des reliefs terrestres et sous-marins. Dans le monde marin, l'érosion est omniprésente et constitue un élément fondamental de la géomorphologie sous-marine. C'est pourquoi, afin de rendre nos reliefs sous-marins plus réalistes, nous avons décidé de mettre en place un algorithme d'érosion similaire à celui observé dans la nature.

Notre algorithme d'érosion sous-marine, inspiré les articles[12] et [13], est basé sur l'érosion hydraulique provoquée par une ou plusieurs sources d'eau. En utilisant cette méthode, nous pouvons créer des textures plus réalistes à petite échelle, tout en restant fidèles aux phénomènes naturels observés. De plus, à une plus grande échelle, notre algorithme nous permet de représenter les phénomènes naturels d'érosion, tels que ceux causés par les courants marins.

Cette technique nous permet d'ajouter de la complexité et de la variété à nos reliefs sous-marins, en recréant de manière précise les processus naturels d'érosion. En utilisant cet algorithme, nous pouvons donc générer des reliefs sous-marins plus réalistes et fidèles à la nature, renforçant ainsi l'expérience visuelle et immersive pour les utilisateurs de nos modèles numériques.

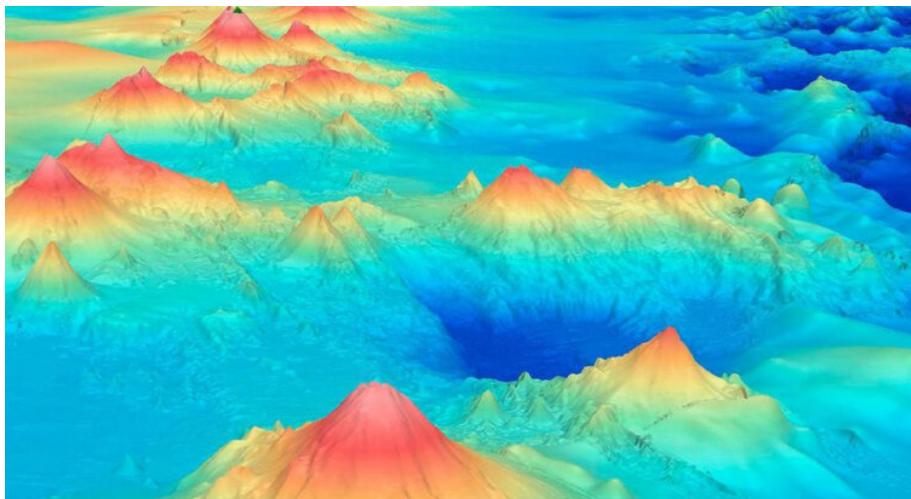


FIGURE 40 – Image bathymétrique sur laquelle apparaît l'effet de l'érosion

Source .© NOAA OFFICE OF OCEAN EXPLORATION AND RESEARCH - M.KONTENTE

Notre approche pour simuler l'érosion sous-marine consiste à considérer que le terrain est soumis à l'action des gouttes transportées par les courants marins. Nous avons attribué à chaque goutte une certaine capacité de charge en sédiments, qui est libérée lorsque la goutte entre en contact avec le relief sous-marin. Au fur et à mesure de son déplacement, la goutte dépose les sédiments qu'elle transporte, jusqu'à ce qu'elle perde toute son inertie ou atteigne la fin de sa durée de vie.

Notre algorithme prend ainsi en compte l'influence de multiples facteurs tels que la vitesse et la direction des courants marins, la densité et la granulométrie des sédiments transportés, ainsi que la topographie sous-marine.

L'approche basée sur les gouttes est particulièrement efficace car elle permet de prendre en compte les interactions entre les sédiments et les courants marins, ainsi que les effets de seuil. En effet, les gouttes de sédiments ne se déposent pas de manière uniforme, mais plutôt en des endroits où la pente est plus faible ou où le courant est plus lent. Par conséquent, notre

algorithme permet de simuler de manière précise l'impact des courants marins sur l'érosion sous-marine.

Pour réaliser notre simulation d'érosion sous-marine, nous avons développé un algorithme capable de calculer les déplacements, les vitesses et les inerties de toutes les gouttes dans chaque cellule élémentaire. Cette approche permet de prendre en compte l'effet cumulatif des gouttes au fil du temps, en supposant qu'une densité définie en gouttes d'eau est présente dans chaque cellule élémentaire.

Le modèle proposé d'érosion hydraulique est décomposé en cinq étapes. Avant d'entrer dans le modèle, nous décrivons d'abord les quantités et les notations nécessaires pour le processus de simulation (cf. Figure 2) :

- la hauteur du terrain b
- la hauteur de l'eau d (qui représente la quantité d'eau pour un modèle incompressible)
- la quantité de sédiments en suspension s
- le flux de sortie $f = (f_L, f_R, f_T, f_B)$
- le vecteur de vitesse $\vec{v} = (u, v)$

Ces valeurs sont stockées pour chaque cellule de la grille 2D et sont mises à jour pendant la simulation.

Les cinq étapes de notre simulation sont :

1. La hauteur d'eau dans chaque cellule augmente en raison des courants marins.
2. L'écoulement est simulé avec le modèle d'eau peu profonde. Ensuite, le champ de vitesse et la surface de l'eau sont mis à jour.
3. Le processus d'érosion-déposition est évalué avec le champ de vitesse.
4. Les sédiments en suspension sont transportés par le champ de vitesse.
5. L'eau diminue en raison de l'évaporation.

Les équations qui expliquent ces processus se trouvent dans l'article [14].

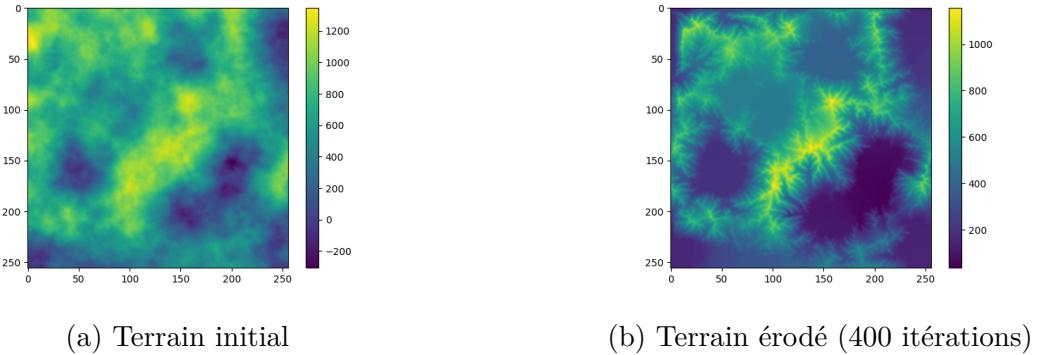
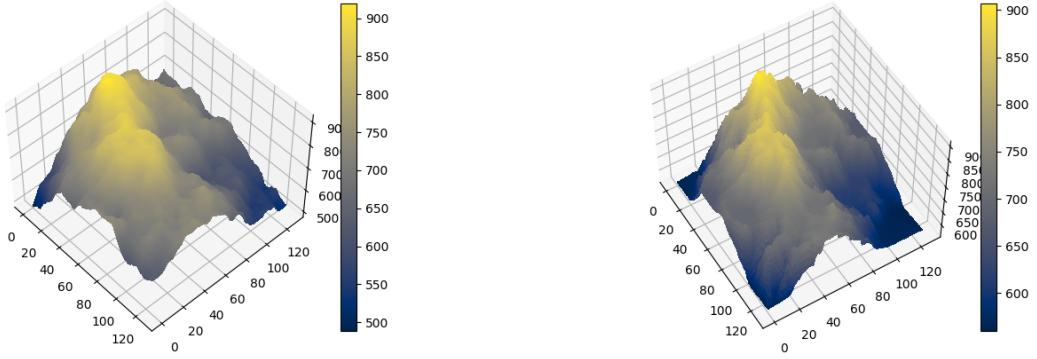


FIGURE 41 – Algorithme d'érosion appliqué à échelle globale



(a) Terrain initial

(b) Terrain érodé (200 itérations)

FIGURE 42 – Algorithme d'érosion appliqué à échelle locale

Enfin, l'algorithme applique un lissage par convolutions avec un noyau gaussien, ce qui permet de réduire les artefacts et de créer des rendus réalistes de reliefs érodés sans altérer l'aspect général du relief. Ces fonctionnalités permettent de rendre compte des effets de l'érosion à différentes échelles, en créant des détails fins et des aspérités sur le relief sous-marin.

L'outil d'érosion nous permet de générer des reliefs plus réalistes à grande et à petite échelle.

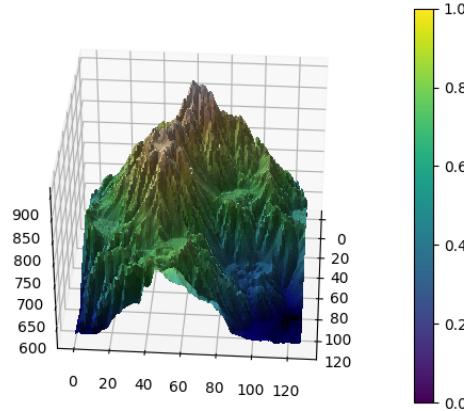


FIGURE 43 – Rendu 3D de l'algorithme d'érosion sur Python

La complexité temporelle de l'algorithme est de l'ordre de $O(n^3 \log(n))$ avec $n = \max(\text{longueur}, \text{largeur})$ de la zone générée. Cela signifie que l'érosion d'un terrain de taille 128×128 prend environ 3,4 secondes, tandis que celle d'un terrain de taille 512×512 prend environ 32 secondes.

Bien que plusieurs tentatives d'optimisation aient été entreprises pour réduire la complexité temporelle de l'algorithme, elles n'ont pas réussi à réduire significativement les temps d'exécution. C'est pourquoi nous avons ajouté d'autres outils d'érosion plus rapides et efficaces pour générer des reliefs sous-marins plus réalistes.

En particulier : nous avons développé une érosion par sources d'eau ponctuelles, qui permet de modéliser l'effet érosif de gouttes d'eau individuelles transportées par les courants marins, l'outil polyligne qui permet de modéliser les rivières sous-marines en utilisant une fonction linéaire par morceaux, et l'érosion locale par courants marins qui permet de simuler l'effet cumulatif de l'érosion sur de petites zones du fond marin.

Grâce à ces différents outils, nous pouvons maintenant générer des reliefs sous-marins plus réalistes et détaillés, tout en réduisant les temps d'exécution par rapport à l'algorithme d'érosion hydraulique initial puisque les modifications deviennent locales et non globales.

7 Présentation du résultats

7.1 Utilisation de Unity pour le rendu graphique

Le processus de génération de profils d'altitude a été effectué en utilisant un algorithme qui a créé des représentations en 2D et en 3D des données en utilisant la bibliothèque matplotlib de Python. Toutefois, pour une évaluation plus précise et subjective des résultats générés, il était nécessaire d'améliorer l'affichage des résultats. Ainsi, pour obtenir une visualisation plus réaliste de la génération d'altitude, nous avons décidé d'utiliser le moteur de jeu Unity et le code en C#.

Dans le processus, le profil d'altitude est d'abord créé en Python et est ensuite chargé dans Unity sous la forme d'un tableau 2D de valeurs flottantes. Ce tableau est alors utilisé pour générer un maillage 3D en combinant des points et des polygones, ainsi qu'une palette de couleurs pour indiquer les variations d'altitude. Ces deux éléments sont ensuite affichés en 3D par Unity pour donner une visualisation plus réaliste et précise des données d'altitude générées.

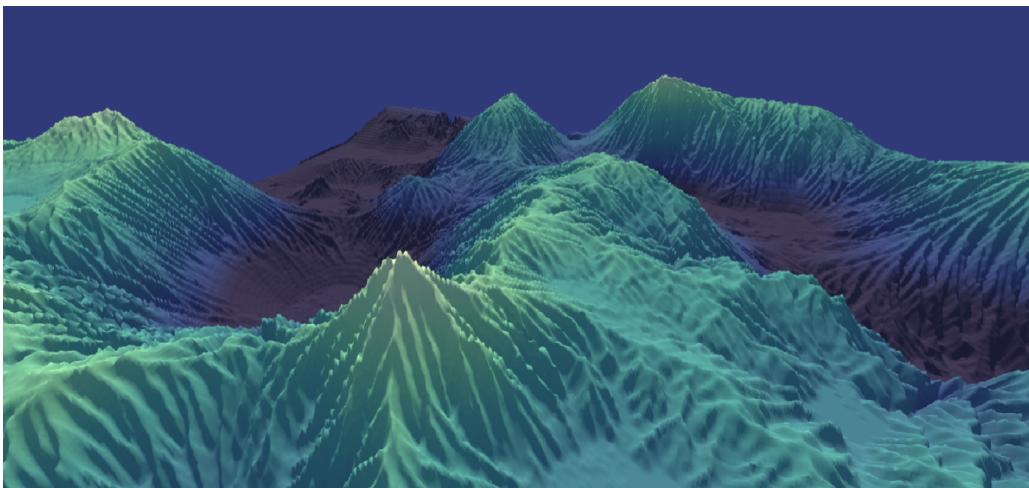


FIGURE 44 – Rendu 3D d'un terrain sous-marin avec Unity

Après avoir réalisé le rendu 3D du terrain en utilisant Unity, nous avons créé un observateur en C# qui peut se déplacer sur le terrain et observer l'environnement à la première personne. Cette fonctionnalité a permis une évaluation subjective plus précise du réalisme de la génération du terrain en permettant de visualiser le terrain de plus près.

Il pourrait aussi être intéressant de générer le terrain en temps réel en utilisant la position de l'observateur. Cette approche permettrait de créer un environnement interactif où l'utilisateur peut naviguer et interagir avec le terrain de manière dynamique, mais elle nécessite une expertise et des connaissances plus approfondies dans l'utilisation de Unity.

8 Conclusion

Nous avons suivi notre stratégie de génération initiale, et avons obtenu des algorithmes satisfaisant pour la classification des reliefs sous-marins existant (classification hiérarchique), la génération des différents types de relief par GAN et la génération d'une texture en combinant un bruit et un algorithme d'érosion hydraulique. Nous avons parfois du tester plusieurs méthodes et certaines se sont révélées infructueuses, comme les tentatives de classification par clustering. Pour compléter notre génération à l'échelle planétaire, il manque un algorithme d'affectation des différents types, et nous pensons qu'une approche par réseau de neurone aurait également pu être fructueuse et permettre de générer une carte des types en prenant éventuellement les reliefs terrestres en paramètre. De plus, notre algorithme de génération est capable de générer des fonds marins à partir d'une carte des types, mais il n'est pas capable de générer correctement un terrain avec une grande variation d'altitude, et ne peut pas donc actuellement générer de terrains contenant les fonds marins et les continents.

9 Répartition du travail dans le groupe

Pendant la réalisation du projet, nous avons tous travaillé ensemble pour la recherche, le traitement et l'analyse de toutes les parties du sujet. Cependant, pour l'implémentation pratique, nous avons décidé de répartir les tâches en fonction des préférences et des compétences de chacun. Nous avons donc attribué les différentes parties du projet de la manière suivante :

Félix Houdouin a travaillé sur plusieurs parties du projet, notamment l'implémentation de l'algorithme de la classification hiérarchique, la recherche sur les reliefs marins et la possibilité de les caractériser par classification hiérarchique, le travail sur la génération des textures à l'aide de l'algorithme de Perlin et ses variantes, ainsi que l'étude des artefacts et des problèmes de recollement.

Mohamed Amine Bouguezzoul a travaillé sur la génération du terrain par la méthode des fractals, en fournissant également la documentation et l'implémentation de la physique de l'érosion. Il a également travaillé sur la génération des différents types de terrains et l'algorithme d'affectation des caractéristiques des zones et la réduction des artefacts au niveau des frontières.

Nicolas Welti a travaillé sur le rendu 3D dans Unity, l'implémentation en Python de l'algorithme de génération du terrain, l'obtention et la lecture de données terrestres et l'algorithme de classification hiérarchique. Ainsi que sur le GAN, son architecture, son entraînement et son utilisation pour la génération d'un terrain plus large.

Bien que nous ayons tous travaillé sur l'ensemble du projet, la répartition des tâches nous a permis de nous concentrer sur nos domaines de prédilection et d'optimiser notre temps et nos ressources pour parvenir à un résultat final efficace et satisfaisant. Cependant la quantité du travail commun est aussi comparable à celle du travail individuel, en particulier : la recherche

des ressources, la formulation mathématique des problèmes envisagés, la mise en forme des codes et l'engagement à intervenir activement lors des réunions avec notre tuteur. il est important de noter que nous étions conscients que la collaboration et la communication restent essentielles pour garantir la cohérence globale du projet et s'assurer que toutes les parties s'intègrent bien les unes avec les autres.

10 Liens Github des codes

Tous les codes utilisés pour la réalisation de ce PSC peuvent être trouvé sur le lien suivant :
<https://github.com/nico5655/OceanPSC>

Bibliographie

- [1] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. Singan : Learning a generative model from a single natural image. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.
- [2] Géoportail — [geoportail.gouv.fr/donnees/carte-mondiale-fonds-marins](https://www.geoportail.gouv.fr/donnees/carte-mondiale-fonds-marins). [Accessed 08-Mar-2023].
- [3] Joseph Wood. *The geomorphological characterisation of digital elevation models*. University of Leicester (United Kingdom), 1996.
- [4] MAV Gorini. Physiographic classification of the ocean floor : a multi-scale geomorphometric approach. In *Proceeding of Geomorphometry Conference, edited by : Purves, R., Gruber, S., Straumann, R., and Hengl, T., University of Zurich, Zurich, Switzerland*, pages 98–105, 2009.
- [5] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.
- [6] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.
- [7] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2018.
- [8] Anna Frühstück, Ibraheem Alhashim, and Peter Wonka. TileGAN. *ACM Transactions on Graphics*, 38(4) :1–11, jul 2019. doi : 10.1145/3306346.3322993. URL <https://doi.org/10.1145%2F3306346.3322993>.
- [9] Ares Lagae, Sylvain Lefebvre, Rob Cook, Tony DeRose, George Drettakis, D.S. Ebert, J.P. Lewis, Ken Perlin, and Matthias Zwicker. State of the art in procedural noise functions. In Helwig Hauser and Erik Reinhard, editors, *EG 2010 - State of the Art Reports*. Eurographics, Eurographics Association, May 2010. URL <http://www-sop.inria.fr/reves/Basilic/2010/LLCDDELPZ10>.
- [10] Travis Archer. Procedurally generating terrain. In *44th annual midwest instruction and computing symposium, Duluth*, pages 378–393, 2011.
- [11] Aymeric Augustin and Fabrice Neyret. Flow-noise en temps réel. Rapport de stage d’option, ecole polytechnique - Evasion, laboratoire GRAVIR, juin 2006. URL <http://morpho.inrialpes.fr/Publications/2006/AN06>.
- [12] Hans Theobald Beyer. Implementation of a method for hydraulic erosion. *Munich, Germany : Technische Universität München*, 2015.
- [13] Jacob Olsen. Realtime procedural terrain generation. 2004.
- [14] Xing Mei, Philippe Decaudin, and Bao-Gang Hu. Fast hydraulic erosion simulation and visualization on gpu. In *15th Pacific Conference on Computer Graphics and Applications (PG’07)*, pages 47–56. IEEE, 2007.