
Université Paris 8 - Vincennes à Saint-Denis

**Licence Informatique**

# Rapport de projet : Générateur de bruit de Perlin

Yanis Makhloufi

Date de soutenance : le 19/01/2022

Tuteur – Université : Jean-Pascal Palus

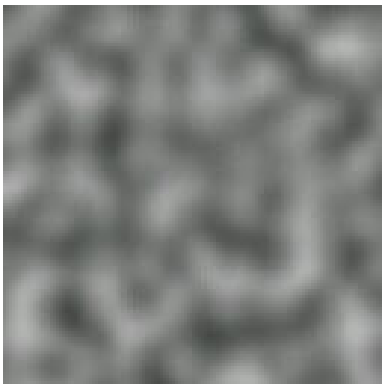
# Résumé

En brève, le projet est de réalisé un générateur de bruit de perlin, en Python. Le bruit de perlin étant une image représentant en clair des points noirs, sur fond blanc, il nous fallait donc élaborer un programme qui va générer une image, avec nos perlin. Dans le premier chapitre, nous expliquerons les packages que nous avons importés pour la rèaliser la suite du code. Puis, nous détaillons dans les deux chapitres suivants les trois différentes fonctions, les boucles, les variables et les fonctions directement importés et expliquerons leur fonctionnement. Enfin, en conclusion nous donnerons le résultat obtenu à l'issus du code.

# Table des matières

# Introduction

Le Générateur de bruit de Perlin, est un générateur qui va générer une image (plus précisément dans notre cas un graphique) avec des points noirs, que l'on appelle un bruit de perlin sur fond blanc. La particularité de ce générateur est qu'il génère une image avec des positions totalement aléatoire pour chaque perlin. Nous l'avons réalisé à l'aide de fonctions, variables, boucles ainsi que de fonctions directement importés de certains packages. Le générateur devrait donc nous générer une image de ce style :



# Chapitre 1

# Les imports utilisés

```
import numpy
import matplotlib.pyplot as afficheur
```

**Code source 1.1** – Imports

Pour réalisé ce générateur, nous avons eu besoin d'utiliser des imports, deux en particulier : le package *numpy*, ainsi que le package *matplotlib.pyplot*.

## 1.1 Le package *numpy*

Nous allons donc pour créer le générateur, utilisé le package *NumPy* qui va nous servir à utiliser des fonctions de calculs pré-défini dans ce package, dans des tableaux

## 1.2 le package *matplotlib.pyplot*

Nous allons aussi utilisés le package *matplotlib.pypot* qui lui va nous permettre d'afficher cette image : en réalité elle sert a réaliser des graphiques à partir de certaines informations, étant donné que le bruit de Perlin n'est pas réellement une image, mais plus une graphique, nous allons l'utiliser pour au final obtenir une image tel quel.

# Chapitre 2

# Les fonctions

Pour commencer, nous allons expliquer le rôle de chacune des fonctions présente dans le code, commençons par la fonction grille :

## 2.1 La fonction grille

```
def grille(gx, gy, x, y):
    grille_x = x - gx
    grille_y = y - gy
    return (
        grille_x * bruit[gx, gy, 0] + grille_y * bruit[gx, gy, 1]
    )
```

**Code source 2.1** – Fonction grille

La fonction grille va donc initialiser une grille avec pour variables x et y, étant donné que nous sommes sur un résultat qui sera un graphique, on défini notre abscisse et notre ordonnée. Elle va part la suite, multiplié chaque variable par le bruit avec une position qui est défini plus tard par une fonction du package numpy.

## 2.2 La fonction diffusion

```
def diffusion(d0, d1, w):
    return d0 + w * (d1 - d0)
```

**Code source 2.1** – Fonction diffusion

La fonction diffusion elle, va gérer la diffusion du bruit sur l'ensemble du graphique, par un calcul : en effet, la fonction va prendre la variable d0 pour abscisse et la variable d1 pour ordonnée, et elle va additionner la

variable d0 par une variable quelconque w, et faire la multiplication de la soustraction des deux variables d0 et d1. Cette opération sera répéter autant de fois qu'il y aura de perlin dans le graphique.

## 2.3 La fonction perlin

```
def perlin(h, j):
    x0 = int(h)
    y0 = int(j)
    x1 = x0 + 1
    y1 = y0 + 1
    lx = h - x0
    ly = j - y0
    a0 = grille(x0, y0, h, j)
    a1 = grille(x1, y0, h, j)
    ax0 = diffusion(a0, a1, lx)
    a0 = grille(x0, y1, h, j)
    a1 = grille(x1, y1, h, j)
    ax1 = diffusion(a0, a1, lx)

    return diffusion(ax0, ax1, ly)
```

**Code source 2.1** – Fonction perlin

La fonction perlin, va relier plusieurs points entre eux avec les axes x et y, afin de pouvoir former dans la grille (appel de la fonction grille), une dimension dans laquelle on va y diffuser nos pixels à l'aide de la fonction diffusion.

# Chapitre 3

# Boucles et variables

Ensuite, nous allons expliquer toute la partie restante du code, c'est-à-dire les boucles, les variables ainsi que les fonctions qui affiche le graphique etc...

## 3.1 Boucle 1

```
for a in range(p + 1):
    for b in range(p + 1):
        x = numpy.random.normal()
        y = numpy.random.normal()


        normal = (x ** 2 + y ** 2) ** 0.5
        x /= normal
        y /= normal

        bruit[a, b, 0] = x
        bruit[a, b, 1] = y
```

**Code source 3.1** – 1ère boucle

Dans cette boucle, pour chaque a et b, une valeur aléatoire sera généré pour x et pour y (utilisation d'une fonction du package NumPy pour tirer aléatoirement un nombre). Nous allons ensuite normaliser x et y pour avoir 1 seule et même valeur, ensuite re-diviser en deux pour avoir x avec 0 et y avec 1. (la variable bruit sera expliqué plus tard)

## 3.2 Boucle 2

```
for x in range(xperlin):
    for y in range(xperlin):
        bruit_perlin[x, y] = perlin(x / intensite_perlin,
        y / intensite_perlin)
```

**Code source 3.1** – 2ème boucle

Ici, nous avons une boucle qui va, pour chaque x et y, va générer et placer le perlin dans le graphique en appliquant notre fonction perlin() (détaillé précédemment) et va diviser nos variables x et y par la variable "intensite-perlin" [1]

## 3.3 Variables

```
p = 20
bruit = numpy.zeros(shape=(p + 1, p + 1, 2))
intensite_perlin = 12
xperlin = (intensite_perlin * p)
bruit_perlin = numpy.zeros(shape=(xperlin, xperlin))
```

**Code source 3.1** – Variables

Différentes variables sont initialisés bien avant, afin de réaliser nos fonctions et ainsi, pouvoir les faire fonctionner :

- La variable p correspond au nombre de pixel, c'est à dire la taille de l'image,
- La variable bruit correspond à la génération du bruit de perlin (sa valeur est initialisé),
- La variable intensite-perlin est le facteur qui gère l'intensité du flou sur le graphique,
- La variable xperlin est le résultat de la multiplication du facteur qui gère l'intensité du flou par le nombre de pixels sur le graphique,
- La variable bruit-perlin est le perlin, qui est généré selon les valeurs choisis (dans ce cas la aléatoirement).

## 3.4 Autres

```
bruit_de_perlin, axes = afficheur.subplots()
axes.imshow(bruit_perlin, cmap="binary")
afficheur.show()
```

1. qui gère l'intensité du flou sur le graphique

**Code source 3.1** – Différentes fonctions d'affichage du graphique

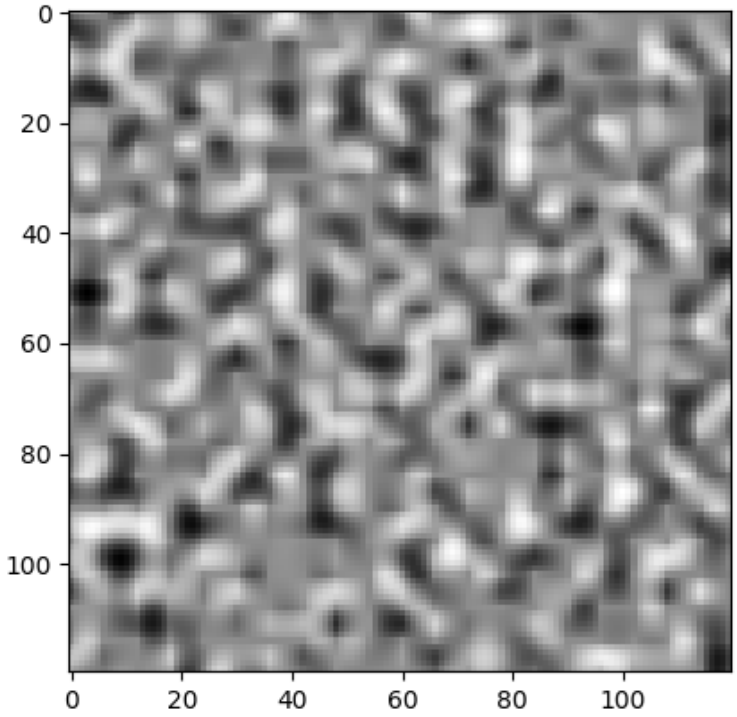
Les dernières fonctions du code, ci-dessus sont des fonctions qui vont gérer exclusivement l'affichage de notre graphique avec le package *matplotlib.pyplot*, la première va initialiser le graphique ainsi que les coordonnées avec "afficheur.subplots() [2]. La deuxième ligne va déclarer le format de notre image sous le format binaire avec 1 comme noir et 0 comme blanc. Et pour finir, la dernière ligne, va lancer l'affichage de la génération de notre image finale. **À noter** que les fonctions imshow, show et subplots sont des fonctions prédéfinis dans le package *matplotlib.pyplot*.

2. le mot afficheur n'est pas compris dans le package : nous avons déclaré lors de l'import du package que nous allons l'utiliser sous le nom de afficheur.

# Chapitre 4

# Conclusion

Pour conclure, voici ci-dessous le résultat obtenu à la fin lorsqu'on lance le programme :




Les axes à gauche et en bas sont les axes d'abscisse et d'ordonnée, étant donné que nous sommes en réalité sur un graphique, ils s'affiche. Cette affichage étant géré par un package, est assez complet puis ce qu'il propose des fonctionnalités comme celle de pouvoir directement enregistrer l'image obtenu sur sa machine, en cliquant sur la disquette.