Images et génération procédurale

Traces de recherche dans le cadre du TIPE

Gabriel Dahan

23 mai 2024

Table des matières

1 Introduction

Souvent, lorsque l'on a besoin de générer de grands environnements naturels lors de la création d'un film (Avatar pour citer le dernier) ou d'un jeu (Minecraft, évidemment), il faut avoir recours à des algorithmes qui font le travail attendu à notre place.

Il existe de nombreuses méthodes utilisées pour cela, dont les *heightmaps* (ou champs de hauteur) et les *normalmaps* (ou). Ce sont des images composées de différents niveaux de gris correspondant chacun à une hauteur dans l'espace : l'objectif de ces cartes est alors de minimiser tout d'abord le coût en stockage, mais aussi et surtout de permettre une génération *pseudo-aléatoire* plus simple de terrains (puisqu'on n'a besoin de s'intéresser qu'à deux dimensions au lieu de trois).

2 Heightmaps

2.1 Définition

Simplement, une *heightmap* est une image en noir et blanc, dont le niveau de gris de chaque pixel correspond à une hauteur dans l'espace.

Supposons qu'on dispose d'une *heightmap* H de taille $w \times h$. On définit l'application h_H par l'application qui à chaque pixel de $P_{w,h} = [\![0, w]\!] \times [\![0, h]\!]$ associe une valeur de $\Lambda = [\![0, 255]\!]$ (0 correspondant au noir et 255 au blanc) :

$$\begin{aligned} h_H : (x, y) &\mapsto h(x, y) \\ P_{w,h} &\to \Lambda \end{aligned}$$

L'objectif étant d'obtenir un rendu 3D donnant à chaque pixel une hauteur dans l'espace suivant son niveau de gris, on définit la transformation suivante :

$$\begin{aligned} \Gamma_H : (x, y) &\mapsto (x, y, h_H(x, y)) \\ P_{w,h} &\to P_{w,h} \times \Lambda \end{aligned}$$

2.2 Algorithme Naïf

On propose alors un script Python simple qui récupère itérativement chaque pixel d'une *heightmap* donnée et lui applique cette transformation.

```
im = Image.open('heightmap.jpg') \
    .convert('L') # L -> 8-bit pixels, grayscale

plan = np.array(
    [(x, y)
        for x in range(im.size[0])
        for y in range(im.size[1])]
)

pixels = im.load()
def gamma(x: int, y: int) -> tuple[int, int, int]:
    return x, y, pixels[x, y]

space = [gamma(*t) for t in plan]
```

On dispose alors d'une partie finie de $\mathbb{N}^3$ qu'on aimerait représenter graphiquement. Pour cela on utilise la librairie matplotlib qui permet un affichage dynamique de ces données. Cependant, la taille des images utilisées peut être très grande, et cette librairie n'est pas faite pour ce type de rendu, ce qui le rend très lent.

3 OCaml et son support de la librairie Raylib

Pour ce projet, il me faut un langage permissif, où tout n'est pas déjà fait et prêt à l'emploi, pour laisser place à la créativité. Par préférence pour les langages fonctionnels, je me tourne alors vers OCaml, dans lequel j'utiliserai la librairie Raylib pour toutes les représentations graphiques.

3.1 Initialisation du projet et Raylib

On crée un projet dune, un système de build OCaml favorisant l'utilisation de librairies.

```
eval $(opam env)
dune init proj mapper
cd ./mapper
```

On installe alors la librairie Raylib à l'aide d'opam,

```
opam install raylib
```

avant d'ajouter la librairie aux dépendances du projet *(./mapper/bin/dune)* :

```
(executable
    (public_name mapper)
    (name main)
    (libraries lib raylib)
)
```

3.2 Adaptation de programmes écrits en C

4 Application avec le bruit de Perlin

Dans le cadre de la génération procédurale, on aimerait qu'à chaque lancement du programme principal, le terrain généré soit différent. Pour cela, il faut générer une image monochrome, toujours différente, dans laquelle tout point admet des voisins dont la "luminosité" n'est pas trop éloignée de la leur. Ainsi la variation de gris reste continue.

```
let random_gradient = ...
```

5 Bibliographie

— Génevaux J.D., Galin E., Guérin E., Peytavie A., Benes B. : Génération procédurale de rivières et de terrains. https://liris.cnrs.fr/Documents/Liris-6433.pdf

— Peytavie A. : Génération procédurale de monde. https://theses.hal.science/tel-00841373/document

— Raylib documentation : https://ocaml.org/p/raylib/latest/doc/Raylib/MaterialMapIndex/index.html

— Raylib github : https://github.com/raysan5/raylib/

— Raylib-ocaml github *(OCaml bidings for Raylib)* : https://github.com/tjammer/raylib-ocaml/

— Nasa Visible Earth : https://visibleearth.nasa.gov/collection/1484/blue-marble?page=2