[HAI809I - Projet] Compte Rendu N°2

Yahnis Saint-Val (yahnis.saint-val@etu.umontpellier.fr)

Léa Serrano (lea.serrano@etu.umontpellier.fr)

7 mars 2023

Génération procédurale de cartes d'environnement

1 Qu'est-ce qu'une image RGB-D ?

Une image RGB-D est une image RGB à laquelle on a ajouté une dimension de profondeur (Depth).

Cela permet de traiter une image 2D comme si elle était en 3D ce qui peut être utile pour la détection d'objets dans une scène par exemple.

Avec une image RGB et une image de profondeur, on peut réaliser plusieurs choses :

— La segmentation de l'image : Grâce à ces deux images, on va pouvoir obtenir une image des contours de l'image puis on obtiendra une image de la segmentation de l'image.

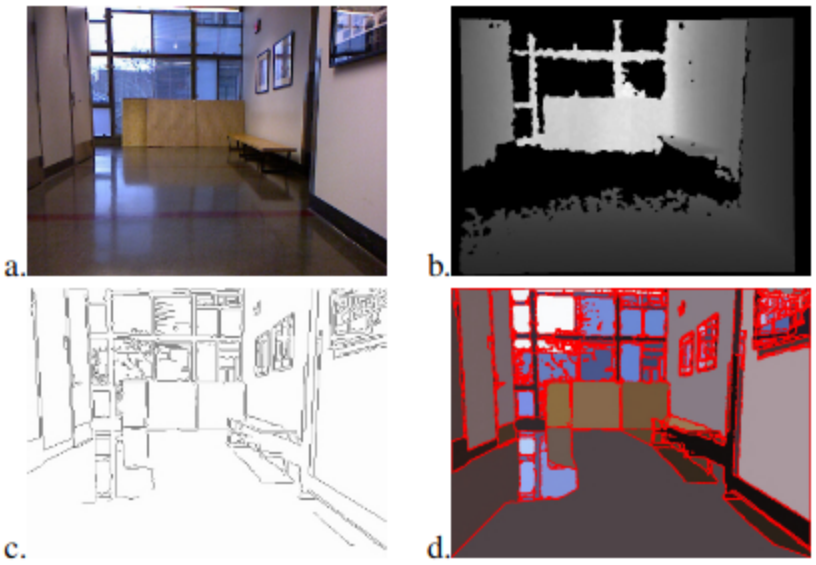


FIGURE 1 – a : image de base, b : image de la profondeur, c : image de l'extraction des contours, d : image segmentée

— L'ajout de profondeur dans une image 2D : Cela consiste à donner un effet 3D à une image 2D afin de pour avoir un effet de profondeur sur l'image 2D, cela permettra d'obtenir une image 2.5D.

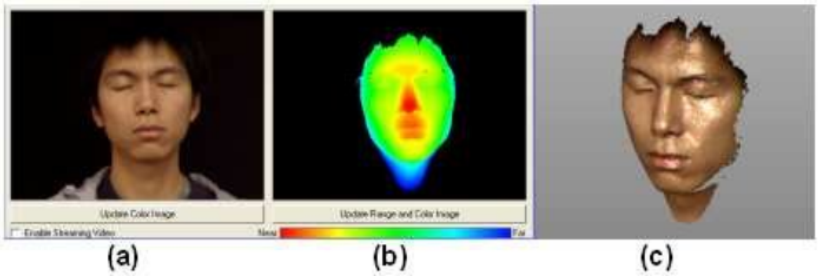


FIGURE 2 – a : image de base, b : image de la profondeur, c : image 2.5D

Une image RGB-D permet aussi de représenter un environnement de manière plus précise qu'avec une simple photographie satellite.
On peut en plus obtenir une information sur la hauteur et donc le relief du terrain :

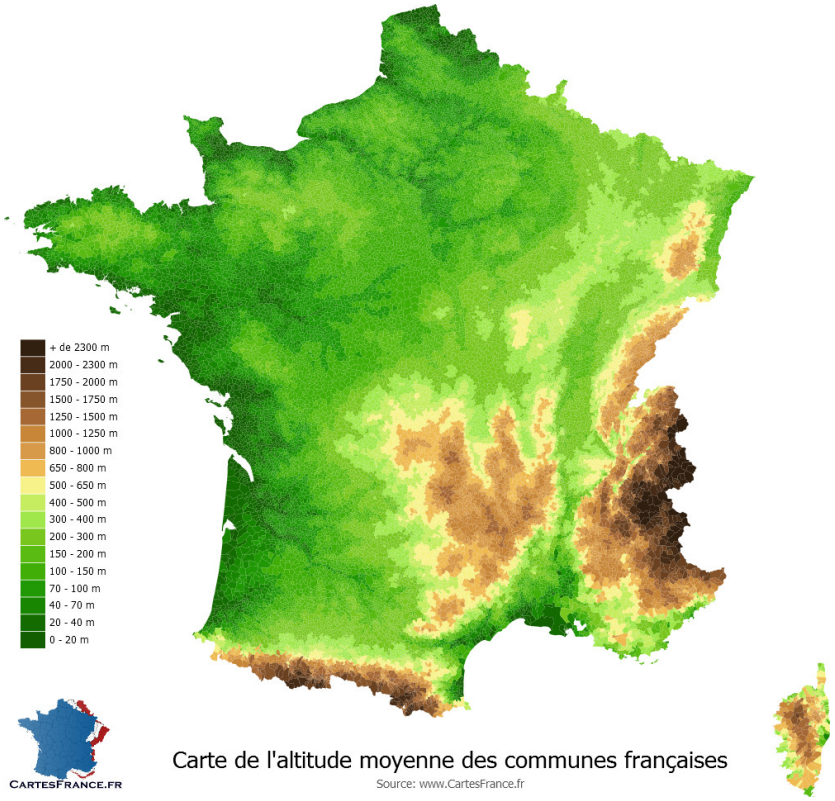


FIGURE 3 – Carte d'altitude de la France

Notre projet consiste à utiliser des images RGB-D, générées procéduralement par du bruit et une suite de traitements d'image, afin d'obtenir in-fine des cartes d'environnements fictifs, idéalement détaillées avec différentes niveaux de résolutions, pour permettre d'obtenir des villes, des régions, des pays...

2 Projets similaires trouvés sur le web

Map Maker permet de générer des cartes d'altitudes en utilisant différentes techniques connues, telles que :

- Le "Value Noise", qui consiste à placer des points à des positions aléatoires, de leur assigner des valeurs aléatoires, et ensuite interpoler leurs valeurs pour calculer les pixels de la carte.
- Le "Perlin Noise", similaire au value noise, mais avec des vecteurs à n-dimensions.
- Le "Simplex Noise", similaire au perlin noise.
- Le "Worley Noise", qui consiste à placer des points aléatoires, et ensuite à calculer pour chaque pixel la distance au point le plus proche.
- Différents algorithmes d'érosion, afin de simuler l'effet du vent et de la pluie sur l'environnement.

World-Generator génère une "carte du monde" à partir de cartes d'altitude et de température.
La combinaison altitude-température permet de déterminer approximativement un biome (mer, désert, forêt, plaine, ...), et ensuite de générer une image colorée.

Fantasy-Map-Generator permet de créer des cartes du monde dans un style fantaisie.
Les cartes sont très détaillées, et l'application permet d'afficher les différentes informations telles que la température, le relief, des textures, les précipitations, la démographie...
Il y a même des villes et villages, qui peuvent être visualisés via une application externe (City Generator)

3 Setup du projet

Cette semaine, nous avons mis en place les principaux composants de notre projet, et structuré les différents dossiers de l'application :

- Un dossier "Library" qui contient tout le code "externe" au projet, c'est à dire qui ne lui est pas spécifique, ou que nous n'avons pas écrit nous même.
  On y retrouve pour le moment la "librairie" permettant de manipuler des images (que nous utilisons en TP), ainsi qu'un .hpp permettant de générer du Perlin Noise (Perlin Noise C++)

- Un dossier "App" qui contient le code de l'application. On y retrouve pour le moment :

  - ProjectManager.hpp, un singleton permettant de créer/charger un projet de création d'environnement. Chaque "projet" est stocké dans un dossier éponyme, dans lequel on retrouvera toutes les données relatives à celui-ci (cartes, textures, ...)

  - MapManager.hpp, un singleton permettant de créer/mémoriser/sauvegarder les cartes qui seront générées. Les cartes sont stockées dans un dictionnaire, associant un nom à chaque carte. Elles peuvent êtres sauvegardées dans le projet actuellement ouvert dans le ProjectManager.

4 Utilisation de PerlinNoise.hpp

Pour utiliser cette librairie, il faut d'abord l'include dans notre code, puis déclarer qu'on utilise l'espace de nom "siv" :

```
#include "../Library/PerlinNoise.hpp"
using namespace siv;
```

Ensuite, on déclare une instance de la classe "PerlinNoise", avec une seed :

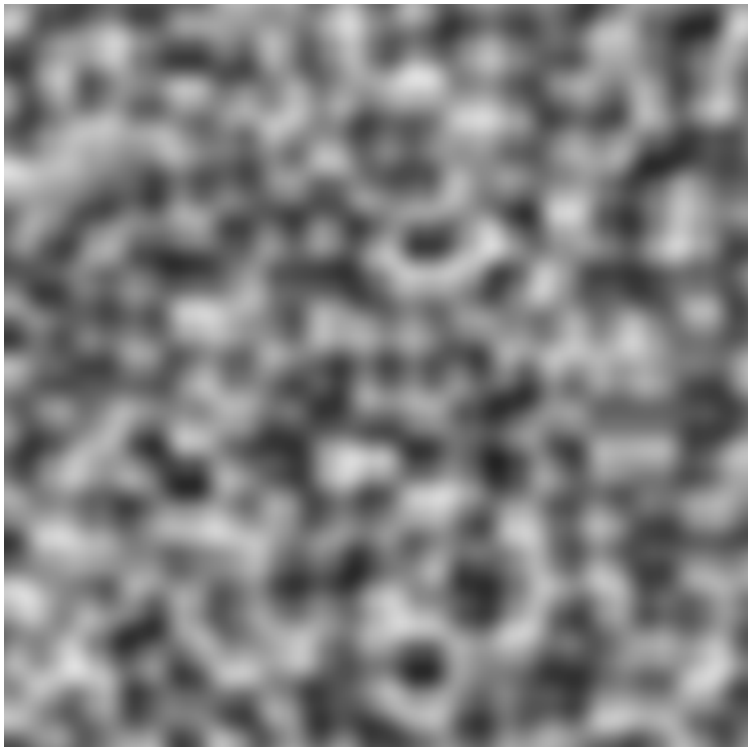
```
uint32_t seed() {return (rand() / RAND_MAX) * pow(2,32); }
const PerlinNoise perlin{seed()};
```

La seed est un nombre compris entre 0 et 2$\hat{}$32-1, et permet de randomiser la génération à chaque lancement. Nous avons donc créer une méthode "seed()" premettant de récupérer une seed aléatoire.

Il suffit ensuite d'appeler une des méthodes "octave2D_*()", en lui donnant les coordonnées normalisées du pixel à générer, ainsi que le nombre d'"octaves", qui correspond à la précision du bruit généré :

```
double pixel = perlin.octave2D_01(x,y,octaves);
```

On obtient bien une image de Perlin Noise :



(a)

FIGURE 4

5 Recherches sur le WorleyNoise

Nous avons essayé d'inclure le code de WorleyNoise que nous avions trouvé la semaine dernière (Worley Noise C++). Ce code utilise une ancienne version de la librairie TGUI mais si on retire toutes les méthodes de TGUI, le code fonctionne.

Le souci est que lorsqu'on veut fusionner notre code actuel et ce code, on doit réussir à combiner les makefiles.
Au niveau de la ligne du makefile qui génère l'exécutable, le code du WorleyNoise que nous avons trouvé nécessite que l'on écrive les choses d'une certaine façon, et si on change le makefile de notre code pour correspondre à celui du WorleyNoise, cela fait que notre makefile ne fonctionne plus.

Nous avons donc décidé de ne pas utiliser ce code, et nous allons coder nous même l'algorithme de WorleyNoise.
Voici le principe de cet algorithme :

- On génère un ensemble de point de contrôles aléatoires
- Pour chaque point dans l'image de sortie, on calcule sa distance aux points de contrôles
- Chaque point de l'image de sortie aura pour valeur, la différence entre le point de contrôle le plus proche et le deuxième point de contrôle le plus proche

6 Objectifs pour cette semaine

Cette semaine, nous planifions de compléter la classe "MapManager" et "ProjectManager", puis de commencer à écrire le code pour générer les cartes d'altitude :

Nous créerons une classe "HeightMap", regroupant les différentes méthodes de génération et de traitement relatifs aux cartes d'altitude.
Nous souhaitons être capables de générer des cartes d'altitudes suffisament détaillées, de facon paramétrable, afin de pouvoir les utiliser pour la suite du projet.

Ensuite, si nous avons terminé, nous commencerons à implémenter l'algorithme de Worley Noise, qui nous pourrait nous permettre entre autres de générer des fleuves.