Procedural generation of an alien planet

By Haitham Al Saidi

What is procedural generation?

Procedural generation is an **algorithmic method of generating content** using a mixture of human generated assets and **pseudo-randomness**.

It can be used to generate terrain, textures, level structures and much more.

Examples of use:

- Minecraft (and other sandbox games)
- Binding of Isaac (and other roguelikes)
- Dungeons and Dragons (and other tabletop games)
- Lord of the Rings (and other films, LOTR uses MASSIVE to generate fighting armies of thousands of soldiers)

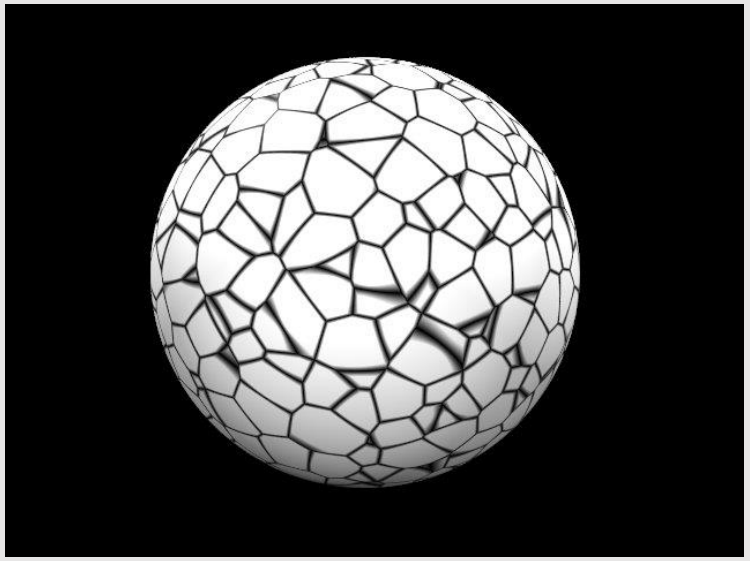


Fig 1.1: Procedural texture using Voronoi tessellation.

Fig 1.2: Fractal trees. The images of the trees was generated from using a L-system.

Why use procedural generation?

Increased gameplay variety

More replayability

Smaller file sizes

Larger amounts of content

Lower budget requirements

Fig 1.3: A procedurally generated planet in Spore. This is one of over **500,000 visitable planets** in the Spore galaxy. This was one of my absolute favorite games as a child, and I still occasionally play it today.

Drawbacks of procedural generation

Reduced quality control

More taxing on hardware

Potentially more repetitive worlds

Difficult to script pre-set game events

Limited gameplay types

Fig 1.4: A Minecraft player with (at the time) a PC that could run almost any AAA game on maximum graphics at 60 FPS. Here he is shown lagging significantly (13-40 fps) due to the taxing nature of procedurally generated content.

Aims and Objectives

Aim: To create a video game with Unity that utilizes procedural generation and Perlin noise to achieve pseudo-random endless terrain generation.

Objectives:

- Learn how to use **C#, Visual Studio** and **Unity** effectively and efficiently
- Create a **chunk** of terrain using a **triangle mesh**
- Use **Perlin noise** to create **variation** in topography
- Implement **infinite, repeated and pseudo-randomly generated** chunks
- Ensure **acceptable performance** (min 60-144) by utilizing techniques such as **multithreading** and **variable levels of detail**
- Improve **visual detail** by implementing features such as colour and texture **shaders**, as well as **flat shading** lighting

Toolset and Methodology

Unity: Game engine

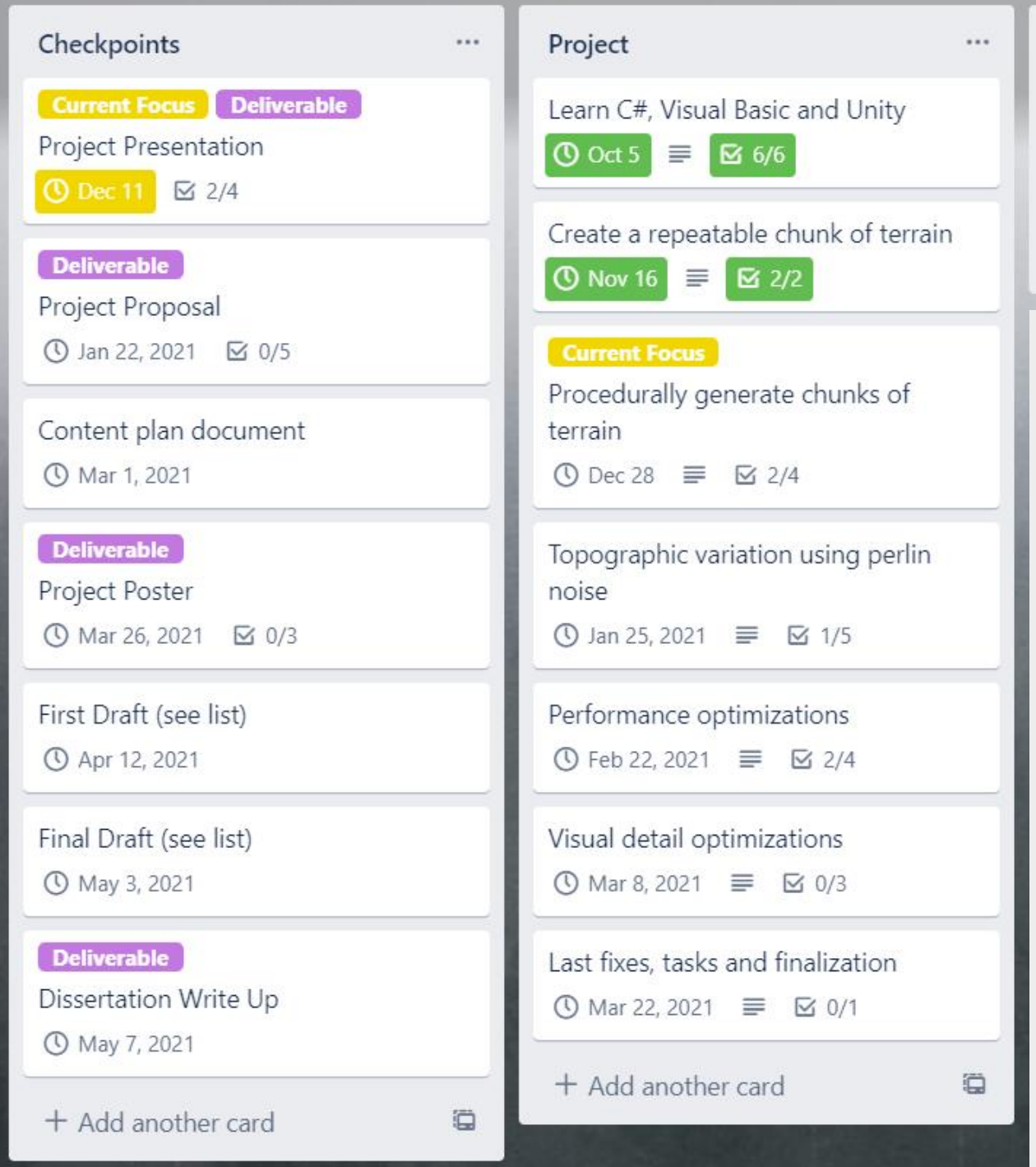
C#: Scripting language for use with Unity

Visual Studio: IDE for C# with build in Unity project support

GitHub: Space to store project and track changes

Trello: Project and time management tool

Kanban development model. Mix of soft and hard deadlines. Add tasks as I go along as I see fit (e.g further optimizations, unforeseen issues, timing miscalculations)



Plan, timeline and progress

I use **Trello** as a **project planning** tool, a **time management** tool and a **progress tracking** tool.

Major milestones in progress are denoted as **checkpoints** on the list in the left. The **project** list keeps track of my **project objectives**.

Each **card** contains a **checklist** with smaller **sub-tasks** to complete.

Cards are in a **general order**, however I can come back to them later if I wish.

Additional sub tasks can be added later if I deem it necessary or if there is **something unforeseen**.

Image References

- Voronoi Tessellation (Fig 1.1): https://en.wikipedia.org/wiki/Procedural_generation#/media/File:Blender3D_VoronoiCrackle.jpg
- Fractal Trees (Fig 1.2): https://en.wikipedia.org/wiki/Procedural_generation#/media/File:Dragon_trees.jpg
- Spore Image (Fig 1.3): https://scientificgamer.com/the-procedural-generation/
- Minecraft Image (Fig 1.4): https://www.reddit.com/r/Minecraft/comments/ayif6w/low_fps_on_good_computer/