Abstract
Procedural approaches to terrain modelling have garnered attention in recent years for its applicability to video game world design. However, these approaches lack user control, particularly in flexibly adapting to a designer’s desires to produce the effect of a certain world or setting. In this project, we aim to execute an artistic vision of a post-apocalyptic world by applying procedural methods to terrain and texture generation, and integrating the results. We will use previous work on subdivision, fractal noise generator, and Voronoi diagram methods to achieve real-time terrain generation. We will also implement solid and cellular texturing approaches for texture generation. In addition to the procedurally generated post-apocalyptic world itself, the main original contribution of the project will be a framework that expresses the potential artistic needs of the game’s designer within a user interface.

Description

1. Procedural terrain generation

1a. Base height map
A height-map is a grid of coordinates (x, y), each mapping to an elevation value. Although there are many methods to generating height-maps, our selection will be limited to those with real-time (less than one second) potential. As such, we start with exploring relatively simple methods such as midpoint displacement, to more complex approaches in the realm of fractal noise generators.

The final height map will be a combination of the above which most expresses the game’s vision of post-apocalyptic terrain. Specifically, the main setting of a lighthouse amidst decaying, man-made floating islands ultimately requires a more geometric influence, rather than organic, although the two could both be included for contrasting effects.

1b. Additional terrain features and optimizations
To add the desired geometricity to the islands, we will use a novel method of combining the ridge lines and flat shapes generated by overlaid Voronoi diagrams with a perturbation filter, as described by Olsen [5].

The game’s island setting lends itself to creative approaches to the challenge of real-time terrain generation. We will segment the terrain generated into sizes large enough for each island and distribute them on the map, as the additional time spent generating features below water would be wasted.
Although forms of erosion (hydraulic, thermal) are usually simulated to give the terrain a more realistic, weathered look, we determined that this would be unnecessary for a landscape

2. Texture generation

2a. Procedural textures
There are two main approaches to procedural texturing: solid and cellular. We will implement versions of both using Perlin noise and Voronoi-like tiling, respectively.

2b. Material aging for artistic effect
There are many types of aging [10] — roughly categorized as mechanical, chemical, and biological. They interact in complex ways on a real-world surface. For this project, we will be concentrating on the mechanical category, as the algorithms for the others are difficult to adapt to the Unity graphics system. Direct deformation of mesh simulating dents will be explored, in addition to bumpmaps for “wrinkled” terrain, and simulation of lichen using a form of fractal noise.

We will also apply unusual textures (whether procedurally generated or supplied by default) such as rusted metal, glass, and steel to terrain, rather than soil, grass, or other natural components. For water, we will apply a colormap simulating the flow of the chemical particles of corrosion. This will create a post-industrial, post-apocalyptic feel.

3. User framework and interface
The envisioned interface will allow the user to combine procedural scripts (implemented in a common interface as described in the framework) with the option to add layers of their choosing, including terrain, texture, vegetation, etc. The user will be able to manipulate the exact scripts for each layer, the scripts’ exposed parameters, and their composition. Due to time constraints, additional features for the interface are desired but perhaps not achievable, and so we aspire to be as general in our approach as possible to encourage future development.

Roadmap

Checkpoint 0 (2/12):

Goals
- From reading surveys of procedural generation methods for both terrain and texture, determine ones that will be implemented.
- Learn the basics of Unity, setting up C# development environment.
- Add lighthouse to test mesh knowledge
- Add basic sky box.

Resources: Unity tutorials [1][2], research papers [9]

Checkpoint 1 (2/26): Height-map generation, prototyping
Goals
Base height map (2D) generated with
1. Subdivision approach, using midpoint displacement approximation
2. Fractal noise generator approach, using exponentially distributed noise.
3. Voronoi diagram with noise perturbation
Addition of basic water map.
Reading more about Unity graphs.

**Glue:** Extrusion of 2D height-map into 3D using triangles

**Resources:** Olsen [5], Parberry [7]

**Testing:** Verifying 2D heightmap output against expected outcome.

Checkpoint 2 (3/19): Terrain generation, prototyping

**Goals**
Manipulate combinations of the above for the final height map.
Adding the circular-shape island mask
Adding the “global mask”, which will add large stretches of water between islands, and is based on location of each island from story center (lighthouse)
Addition of A* map, or navigation mesh, for island

**Glue**
Placement of lighthouse in appropriate position.

**Resources:** [3], Olsen [5]

**Testing:** Successfully rendered scene, using 512x512 vertex grid

Checkpoint 3 (4/2): Procedural material prototyping

**Goal**
Solid texturing approach using noise function (e.g. Perlin) [8]
Cellular texturing approach using Voronoi diagrams [4]
Colormap to simulate various materials, such as flesh, metal, and wood

**Glue**
Make simple object on which textures can be placed, such as sheets, bricks, and boxes.
Apply textures to terrain

**Testing:** textures should hold well under minute changes in terrain.

Checkpoint 4 (4/16): Physical aging effects

**Goals** (some ideas, will probably implement 2)
Dents: deformations in mesh, applied directly to terrain or object
Simulate lichen growth using miniature height-map and colormap.
Using noise-based approach for bumpmaps to simulate wrinkling
Apply corrosion effects to water using colormap
Peeling effects using extrusion

**Glue:** Applying applicable transformations to procedural textures above and built-in textures such as metal, glass, and steel.
References: Mérillou [14]
Testing: Island and object surfaces looks more interesting

Checkpoint 5 (4/30): Framework
A Unity plugin with the following features:
- Options to add and delete the layer being worked on
- Option to add, delete, and replace own procedural scripts to each layer
- Options for various combination methods of procedural algorithms for each layer
- Ability to specify types of terrain, with two default modes (basic and island)

Checkpoint 7 (5/4): Evaluations / final report
- **Game evaluation:** demo testing, speed evaluation / optimizations
- **Distribution:** testing with steam console, porting to different platforms
- Writing final report

Stretch goals:
- **Game**
  - Basic user interface: basic menu, saving.
  - Basic story elements, such as prologue text and story setup
  - Furnishing inside of lighthouse
- **Graphics**
  - Adding procedural fractal clouds, transportation device, and weather effects
  - Adding more sophisticated water animations (splashing, rippling, etc)
  - Global light source to simulate time passing
- **Sound:** Quiet soundtrack. water sound effects.
- **Plugin**
  - Additional modes such as mountain, plains, and hills
  - Ability to apply type of terrain to areas in global map

Deliverables
1. **Playable game:** Developed with Unity and Xamarin Studio
   With real-time, procedural terrain, focusing on island and post-apocalyptic features, and procedural aged textures as applied to static objects and surfaces of islands

2. **Unity plugin** for the user interface component

3. **Final report:**
   With detailed documentation of each procedural mechanism used, its integration into the game and implementation; as well as any difficulties encountered, and evaluation of performance to fulfill real-time component.

4. **Github repo** with code documentation
Resources & References