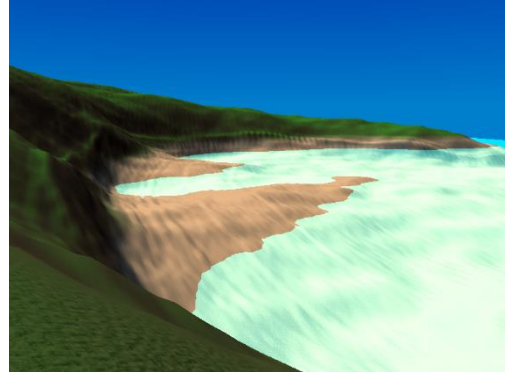


# Generating a Procedural Island

AG1102A Coursework Report

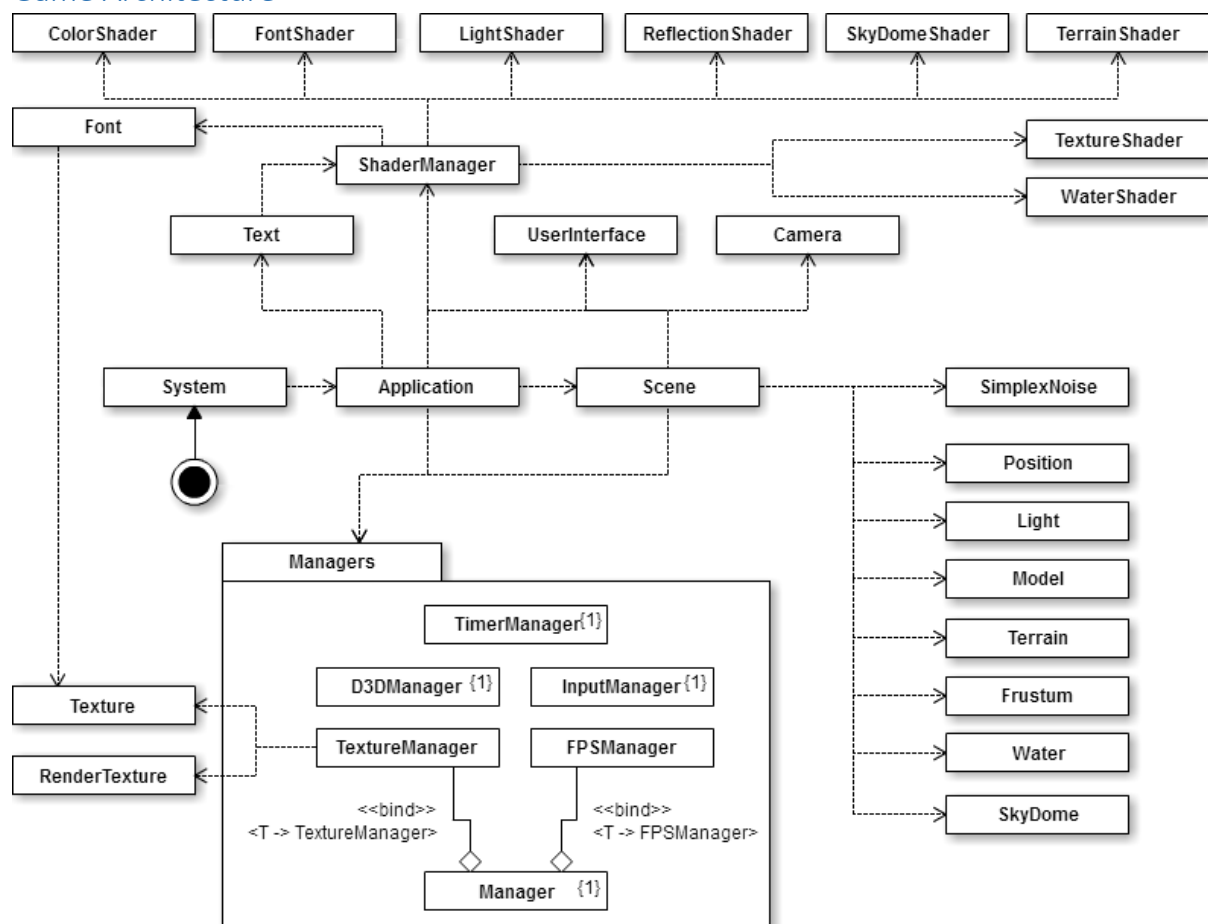
## Game Feature Outline

The game features an island scene where the player can walk around and explore. The island is shaped to have a central bay with a beach inside. The island itself is a combination of a hand drawn height map and simplex noise to allow for some randomised displacement. The island is also procedurally textured and coloured. Below a certain height, the texture will be sandy or rocky if the slope of the terrain is a steep incline. Above that height, the texture is grassy, with a dirt texture applied on a steep incline. The water has a scrolling texture on it and oscillates its height giving the simulation of movement.



Finally a skybox is drawn in the scene to give a general gradient backdrop.

## Game Architecture



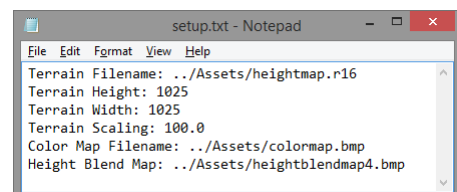
The game closely follows Rastertek's Terrain tutorial with a few adjustments. First of all is the separation of manager oriented classes. These I reviewed as only ever needing to be instantiated once, and therefore could be made singletons. This ensures only one manager is ever being utilized in the program. This was done for a few classes by putting the functionality for singletons directly

into each class, however FPSManager and TextureManager took advantage of the Manager template class I had created as a helper class to inherit from to avoid repeated code where possible.

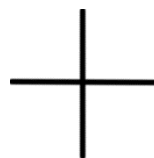
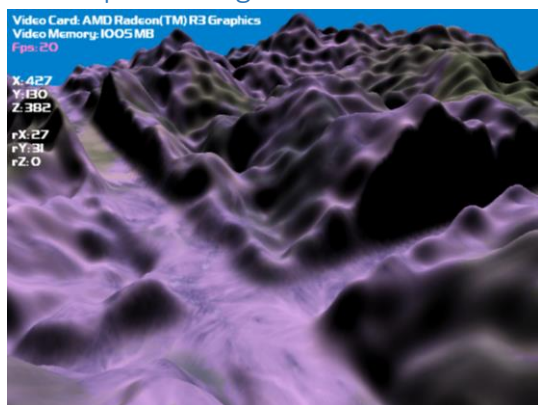
The game starts at the System class to initialise the program, and the generic game functionality is started in the Application class. The Scene class is where most of the composition of the game features takes place, and the Scene has access to most classes within the architecture.

A ShaderManager and TextureManager class is introduced in order to produce less overhead on managing the various textures and shaders within the program.

Assets for the game is held in a separate Asset folder and includes a setup.txt file. This can be used to quickly change different values without having to recompile the source code for every small change. It was ultimately underutilised in my program, because this particular setup file is related to variables specifically relating to the terrain. A better implementation would be a singleton which read the setup file and could be accessed from any class, allowing all classes to have variables stored within the setup file.

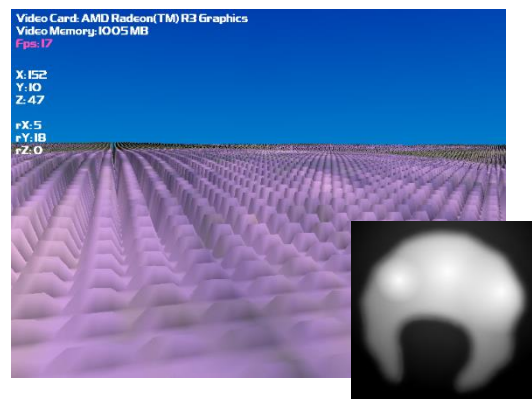
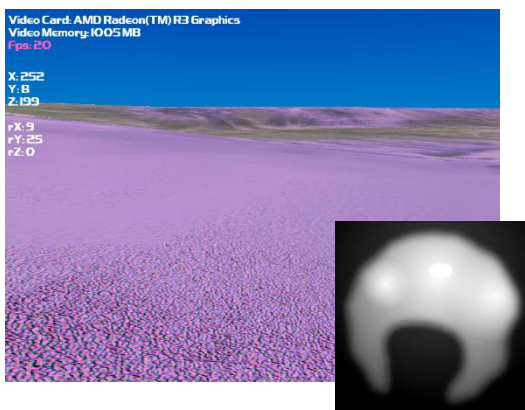


## Development Log

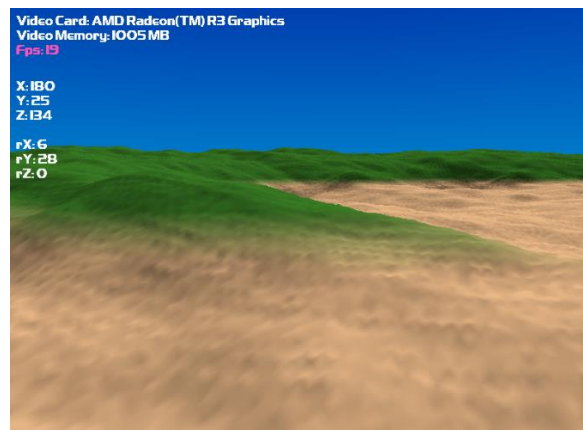
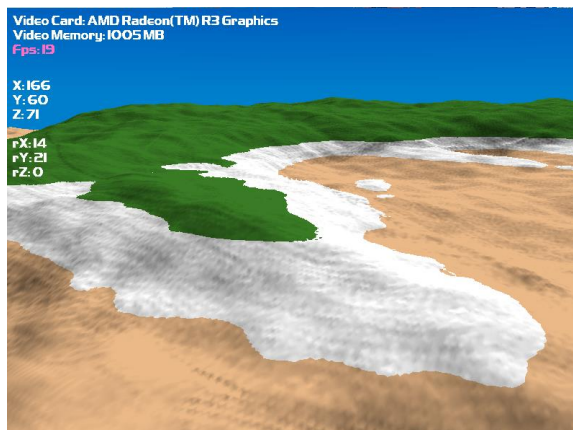


In my initial implementation I wanted to have a square piece of land that was divided up into several bodies of water. While this could have been done by varying the values of the simplex noise till the desired effect was achieved, I decided to blend the simplex noise with an image that I had drawn in order to specify areas that I did not want affected by simplex noise. In this first test I checked that was achievable using a simple cross as a heightmap, specifying not to use noise within black areas.

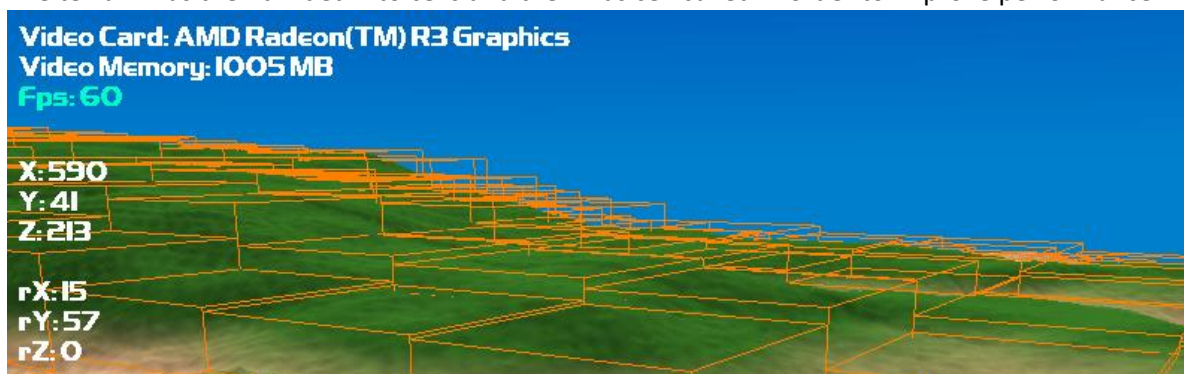
I extended upon this by deciding that I wanted to have an island where the overall height of the terrain was determined by a bitmap made in GIMP. This however resulted in a strange formation of vertices upon the terrain, a problem that was resolved by simply taking the same image in Paint, drawing a white circle (that you can see in the centre of the island) and saving the file which corrected the issue. The white circle however created a plateau which didn't look appealing, so this was later corrected to black line drawn in the top left of the image which remains in the final build.



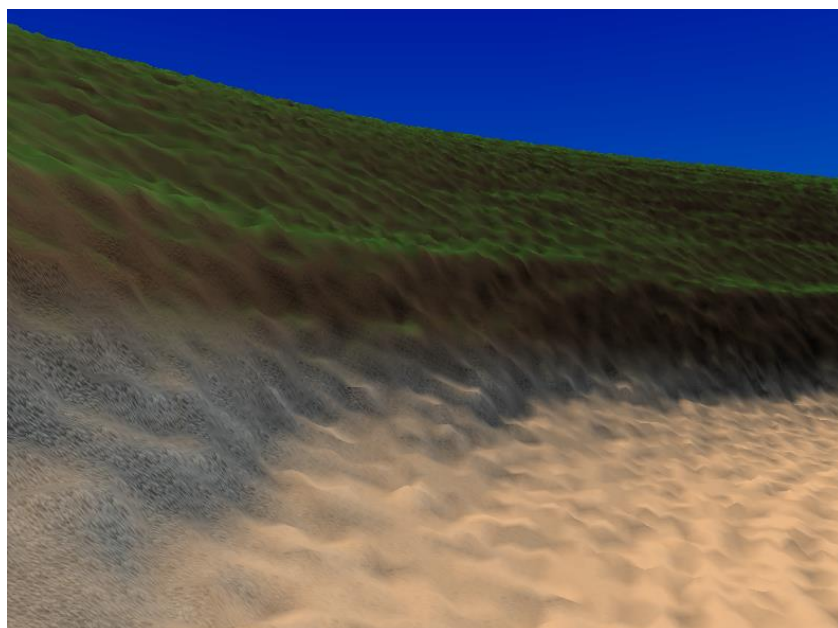
I then went on to create a blending effect for the terrain's color which was done dynamically by passing the height to the terrain's pixel shader.



The terrain was then divided into cells and then was cell culled in order to improve performance.

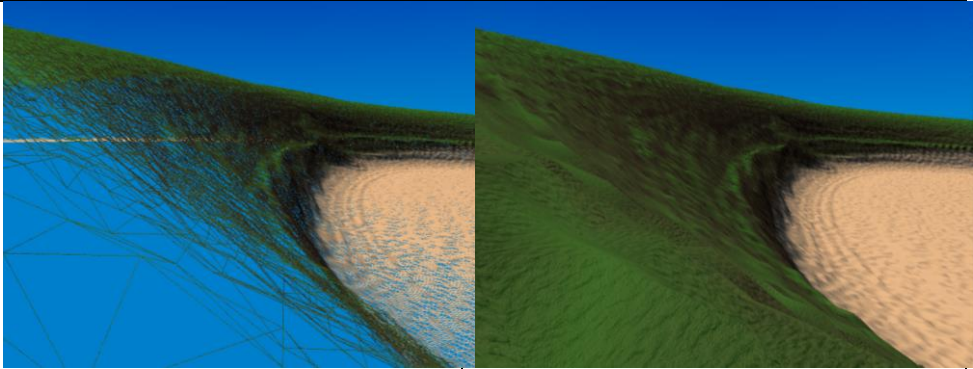
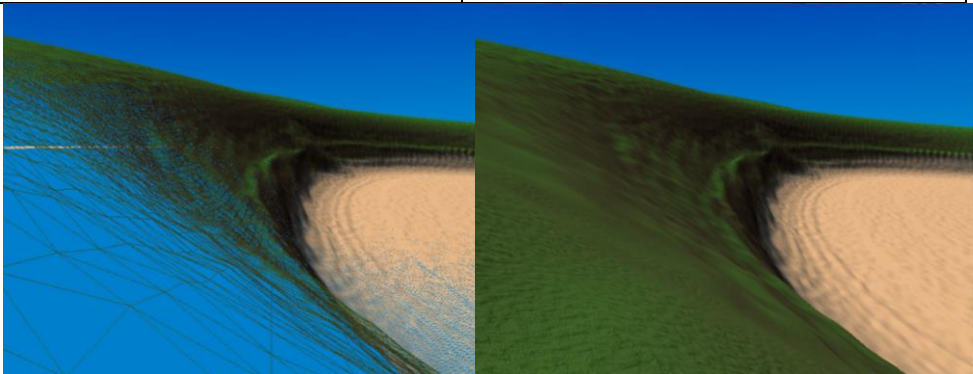
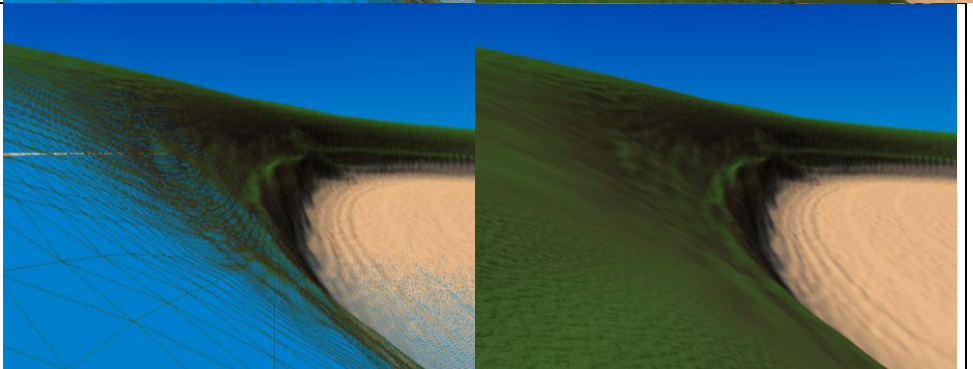
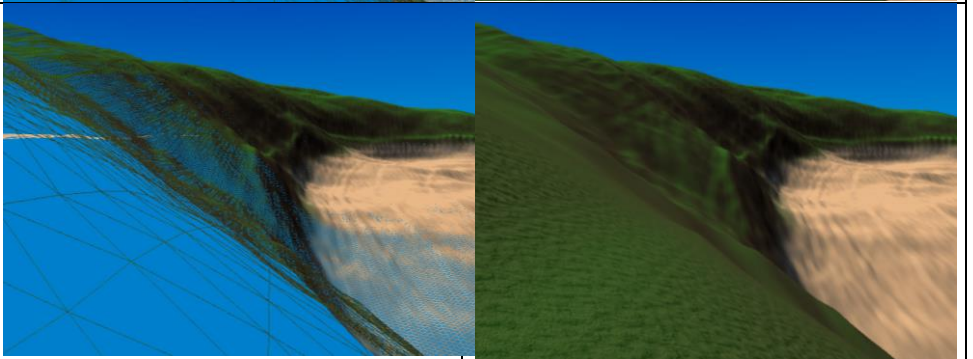


And then the slope angle is determined in the terrain's pixel shader to specify where the terrain's color and texture should change.

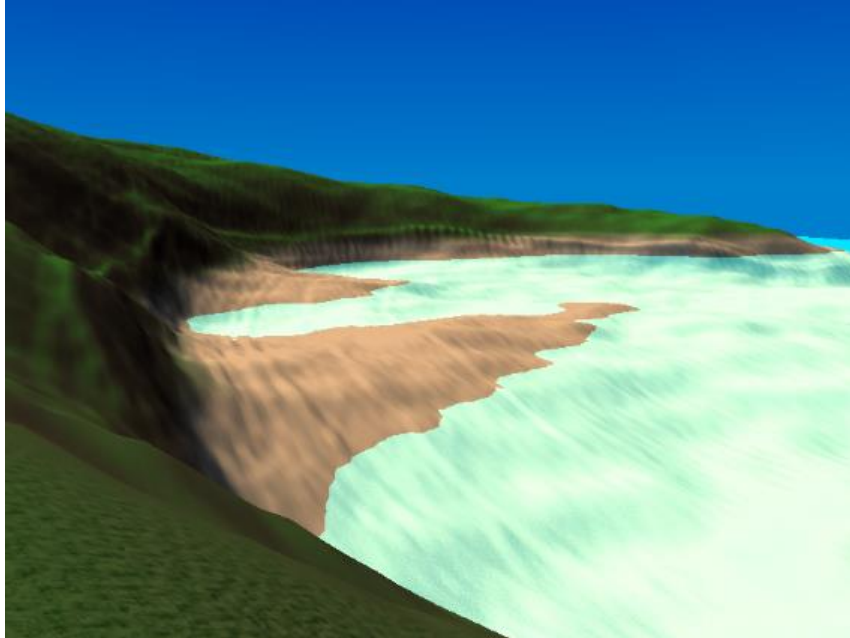




I then introduced smoothing to the terrain, as the bitmap image that I had generated was causing the terrain to be too bumpy. The only drawback to this is the algorithm significantly increases the launch time of the game.

No Smoothing 18 sec Launch Time	
One Smoothing 45 sec Launch Time	
Two Smoothing 1 min 15 sec Launch Time	
Two Smoothing With Simplex Noise Amplitude: 500	

Water was the last element that was introduced to the game. The water that is utilised has alpha transparency and scrolling texture, as well as using a sin function to change its height. It was originally going to also implement a reflection of the terrain by rendering the terrain first to the texture, and although this functionality is still within the program's code, its implementation is broken and could not be included in the final build.



### Final Review & Evaluation

I am very happy with how the final result of the terrain has turned out, but am disappointed with running out of time to finish the water texture and the reflection that was associated with it. I spent too much time concerning myself about the smoothness of the terrain itself instead of moving on to other features, and my poor estimation of time remaining got the best of me. I will be continuing to work on this in the future in order to firstly get a working reflection, but also to scatter trees and other foliage about the terrain using simplex noise in order to determine the location and frequency of objects to be generated above a certain height value.

### Game Instructions

Launch the GameEXE shortcut to run.

Arrow keys to move, F1 to show UI, F2 to show wireframe, F3 to show terrain clusters and F4 in order to disable height locking. When height locking is disabled press A to go up and Z to go down.

### Bibliography

Sebastien Rombauts (2015) A Perlin Simplex Noise C++ Implementation (1D, 2D, 3D, 4D)