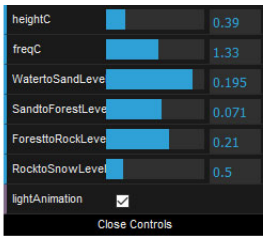
Procedural Planet Creation - TNM084

Jonathan Bosson

January 2016



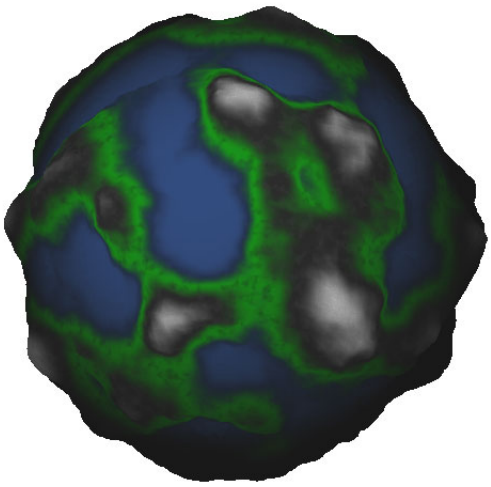


Figure 1: Planet generated through Procedural Methods

1 Introduction

Using procedural methods is still an unorthodox method to render an image. Still it is a very powerful tool to generate random looking environments with a lot of detail. It's also something that quite easily can be done in real time and simple to make minor changes very quickly. This report will go through the methods used to create an arbitrary planet after the user's desired look.

2 Method

This has been an individual project developed in JavaScript, WebGL as well as Requirejs as module loader for GLSL. A simplex noise function has been used created by Ian McEwan, Ashima Arts.

3 Building the Planet

So how does one start making a planet? Most of the times the solution is simpler than one would think. WebGL comes with a handy package to deal with 3D graphics in the form of scene-, camera- and 3D-objects. Once the scene graph has been set up a sphere can simply be added. The goal here is to displace the vertices in the vertex shader to create mountains and valleys on the planet. To do this with high detail it is important to create the sphere with a high amount of segments. This is nothing to be afraid of, the GPU is more than capable to handle the amount of vertex points required for the desired detail. To simulate water level a secondary smaller sphere is created inside the planet. This *'water planet'* will intersect the main planet to achieve an water level. It will also have its own shaders in order to animate waves.

Once the displacement is done and the form of the planet is finished we need to give it appropriate colors representing different types of environments. This is done in the fragment shader. Colors of water, sand, forest, mountain and snow has been implemented representing five different biome's on the planet.

A simple GUI has been added in order to let the user interact with the planet as well as change the look of it. To make this possible the functions that generates the planet relies on a few variables the user can choose. The user can choose a constant determining the height and frequency of the planet's mountains as well as at what height the different biomes should intersect.

4 Displacement in vertex shaders

Simplex noise(1) has been used to generate a random look to both the form and colors of the planet. In the vertexshader a scalar value E is created representing the elevation a vertex point is to be displaced. A and F in the equation below is the altitude and frequency, two variables the user can change to receive desired results. A higher frequency results in more frequent mountain tops and the

higher altitude the higher said mountains will reach. The P in the equation is the point in the vertex points location in the 3D space.

$$E = A * snoise(F * P) \tag{1}$$

In order to generate smoother changes in the noise it is customary to sum up several results of noise that cooperate with each other. The trick to this is to let each noise function be a child to the first *'parent noise'*. For each time another noise function is added the altitude needs to be halfed and the frequency doubled. Doing this means we let each noise function build on the last one. This results in a smooth interpolation in the noise with a lot of detail.

In the *water planet* has time been used as a variable to a small bump-like displacement. This produces an animation of waves on the ocean, which changes with time.

5 Determining color in fragment shaders

On the planet there's five different biomes, all of them with their own set of color mixes. The snow biome is mostly white but has some graininess to it to better represent real snow. The rock and forest biome both have graininess and patches of the same color in another hue. This graininess is generated through the same simplex noise as described above but on the color channels. The patches through mixing two similar colors with a clamp value that varies.

Once all biome colors have been evaluated we need to determine where each biome starts and finish. This is done through the input variables from the user. By determining at what height two biomes should intersect an interpolation between two colors is made with the help of GL function smoothstep and mix(2).

In both the *water planet* and parent planet's fragment shader has a blinn phong shading model been implemented to calculate the local illumination on the object.

6 Results

The code can be downloaded or viewed at github here[1] or live tested live on Firefox here.[2]

[1] https://github.com/jonathanbosson/procPlanets
[2] http://jonathanbosson.github.io/procPlanets/planet/

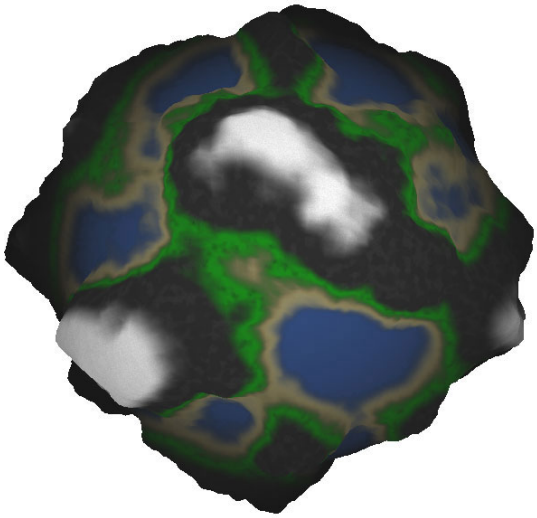


Figure 2: Earth like planet

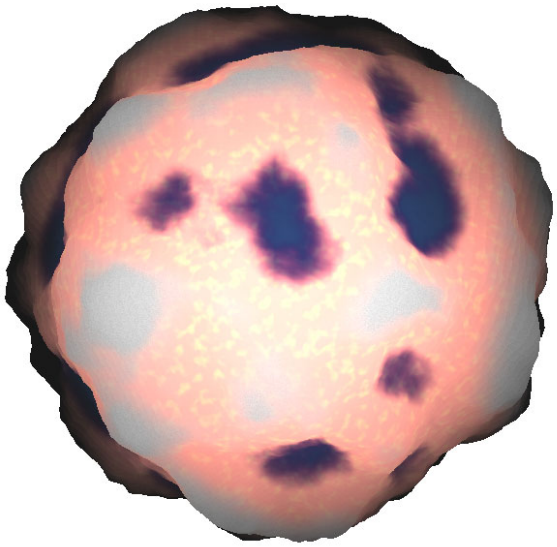


Figure 3: Surrealistic magma-planet

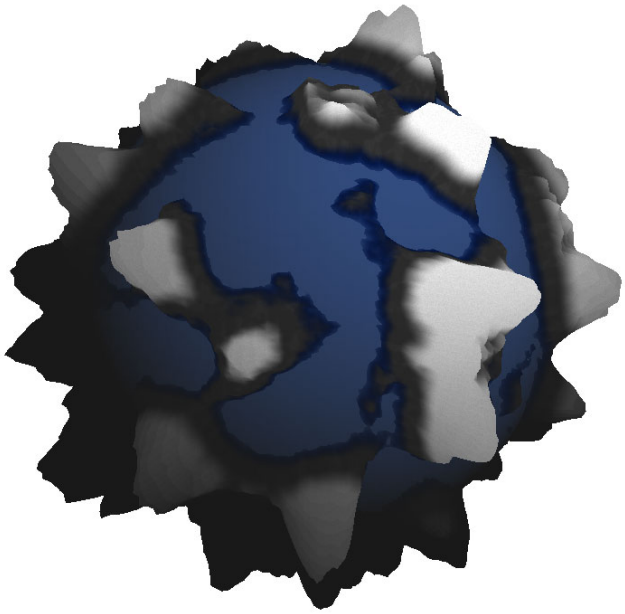


Figure 4: Waterplanet with spiky mountains

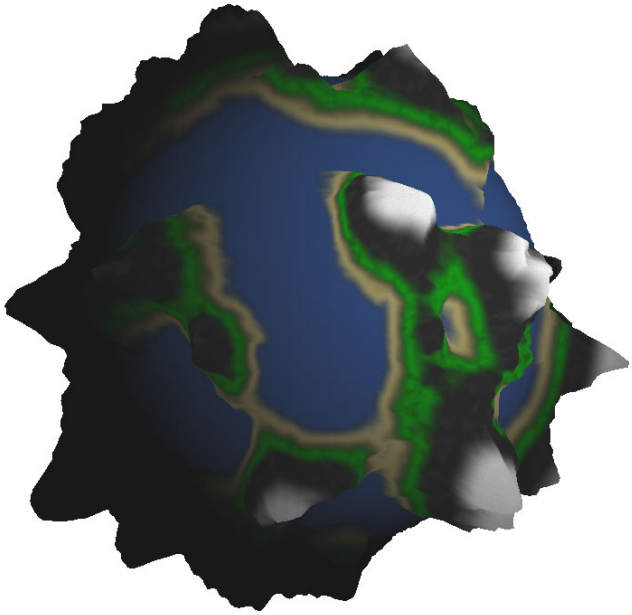


Figure 5: Earth like planet with light animation

References

[1] Stefan Gustavsson. *Simplex Noise Demystified*. 2005-03-22. `http://webstaff.itn.liu.se/~stegu/TNM084-2015/simplexnoise.pdf`

[2] *Reference for the built-in functions of the OpenGL ES Shading Language*. `http://www.shaderific.com/glsl-functions/`