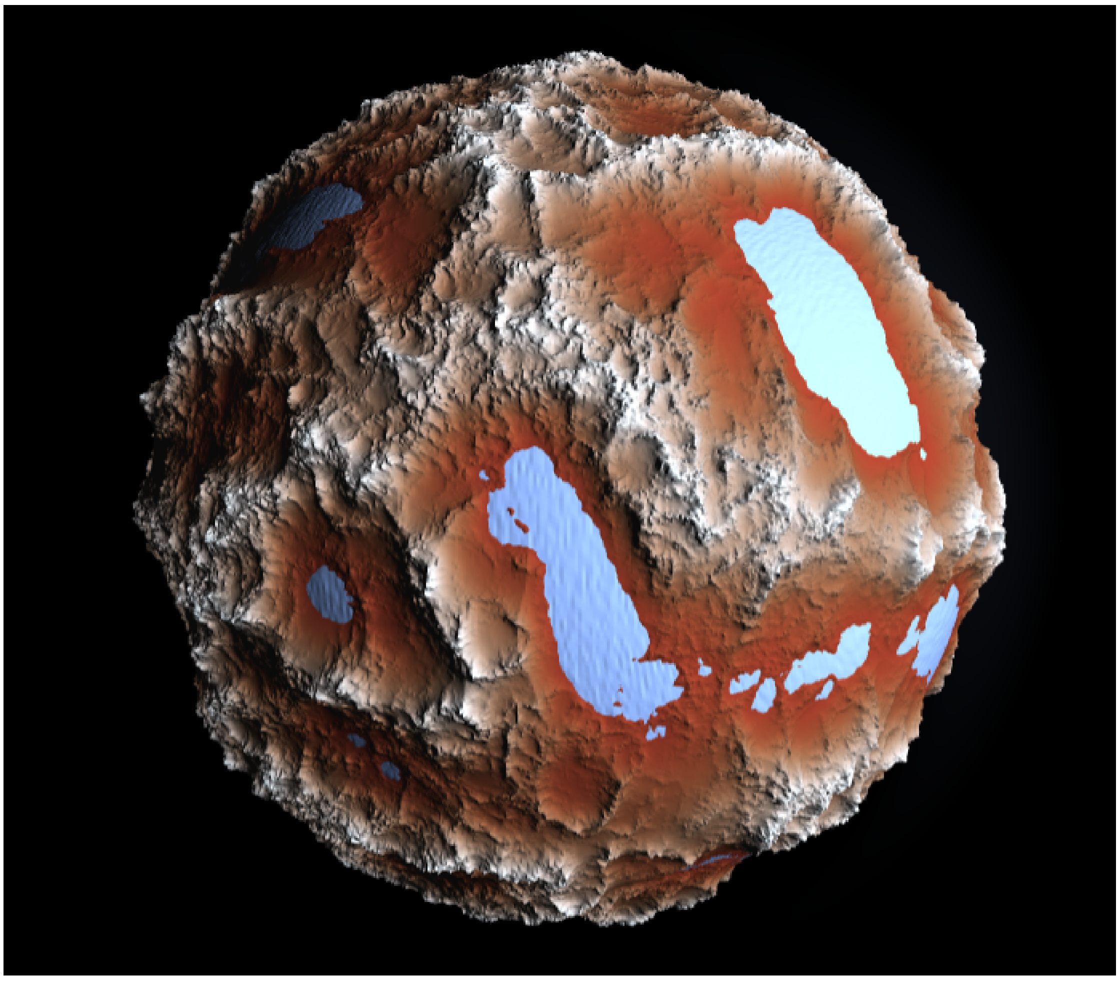
Procedural Planet Generation

Procedural images

TNM084



Sofia Sproge, sofsp932

MT4

January 4, 2022

Abstract

This paper is about how computer graphics can be used in order to create large and highly complex 3D models. In order to get a deeper understanding of the subject, a planet with procedural terrain and animated procedural ocean was created. The algorithm for creating the terrain uses Fractal Brownian motion and turbulence with Perlin noise. The result is a highly detailed mountain terrain. The ocean is also created with Perlin noise and then rendered over the terrain in order to fill the lower points with water. The result shows that procedural graphics can render highly detailed 3D objects quickly and that it is also easy to vary the results with just a few variable changes in the algorithm.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

In computer graphics there is always a challenge to create large areas such as open worlds in computer games or scenery in movies. The 3D models and materials created for these kinds of projects are often very time consuming when created manually. There is also a limitation on how much memory these kind of projects can use. Textures needs to be stored as images and often in different resolution which forces developers to create less varying content. Procedural graphics on the other hand, creates both 3D objects and materials with algorithms. This enables the possibility to create highly detailed and varying work, faster and with more control. The algorithms is used in real time, which means that if it is used to create the materials, less memory is needed. There are countless things that can be created with procedural graphics and it is because of noise. Noise is a tool for creating randomization within these algorithms and it can be controlled in different ways to yield satisfying results. This project is about how to create algorithms with noise.

1.2 Aim of the project

The aim of this project is to get a deeper understanding of how procedural graphics can be used to create large landscapes and how these methods can be tailored in order to create a planet with mountain terrain and oceans in WebGL.

Chapter 2

Method

In order to create the different parts of the planet, already existing polygon meshes were modified to satisfying results and in both cases a hemisphere polygon mesh was used. This is done by displacing the vertices of every triangle along its normal 2.1 in the vertex shader and then exchange the normal with a new one that is calculated from the resulting position in the fragment shader. The displacement factor is calculated with different methods that uses a noise function.

$$newPosition = position + normal * displacement \tag{2.1}$$

2.1 Perlin Noise

A noise function is a function that returns a value between 0 and 1 where the values can be completely random which is called white noise or it can have a pattern[1]. When the noise is visualized, the chosen pattern can be seen and this will determine the characteristics of the generated material or 3D object. In this project, Perlin Noise was used for all procedural graphics and this is because the values are pseudo random and therefor result in smooth noise that is useful for creating natural terrain and waves[2].

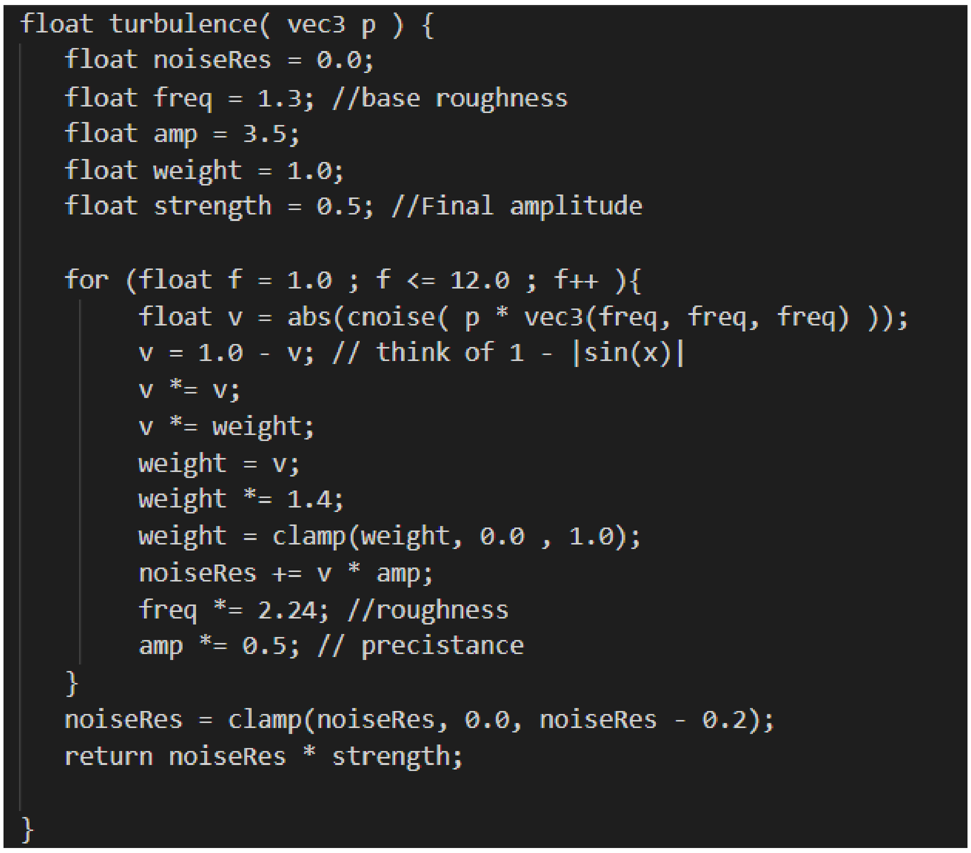
2.2 Mountain Terrain

The mountain terrain was created with Fractal Brownian Motion and turbulence was added in order to make the peaks of the mountains look sharp. The number of segments on the polygon mesh was kept high in order to be able to create small details.

2.2.1 Fractal Brownian Motion

Regular Perlin Noise has a small amount of variation in frequencies and amplitude which could be used to create a terrain in itself but not enough in order to create a mountain chain. Fractal

Brownian Motion (FBM) uses multiple iterations in order to sum multiple octaves of noise in order to create more irregularities in the resulting noise but still self similar. The amplitude of the noise is also decreased at every iteration in order to create even more detail [3]. This function was then optimised with a varying frequency (freq), weight and amplitude (amp), see figure 2.1. A higher frequency value creates more mountain chains because it directly correlates to the frequency of the Perlin noise. The frequency can also be changed with each iteration with a gain in order to vary the Perlin noise. The weight value is dependent on the previous iteration and can therefore create more details in the ridges and ignore the lover values in order to create valleys that are smooth. Amplitude decides how sharp the mountain peaks should be.



```
float turbulence( vec3 p ) {
   float noiseRes = 0.0;
   float freq = 1.3; //base roughness
   float amp = 3.5;
   float weight = 1.0;
   float strength = 0.5; //Final amplitude

   for (float f = 1.0 ; f <= 12.0 ; f++ ){
        float v = abs(cnoise( p * vec3(freq, freq, freq) ));
        v = 1.0 - v; // think of 1 - |sin(x)|
        v *= v;
        v *= weight;
        weight = v;
        weight *= 1.4;
        weight = clamp(weight, 0.0 , 1.0);
        noiseRes += v * amp;
        freq *= 2.24; //roughness
        amp *= 0.5; // precistance
   }
   noiseRes = clamp(noiseRes, 0.0, noiseRes - 0.2);
   return noiseRes * strength;

}
```

Figure 2.1: The FBM and turbulence function that is used to calculate the displacement value for the mountain terrain. freq changes the base roughness of the mountains. amp changes the amplitude of the value over each iteration and creates details. weight makes the noise in the ridges more detailed than in the flat valleys.

2.2.2 Turbulance

Turbulence is essentially FBM but it is used to make the peaks look sharp and the valleys look smooth. This is done by taking the absolute value of the Perlin noise (see figure 2.2) and then subtract it from the offset value (see figure 2.3) in order to invert it [3]. The rest of the algorithm is identical to FBM, see figure 2.1.

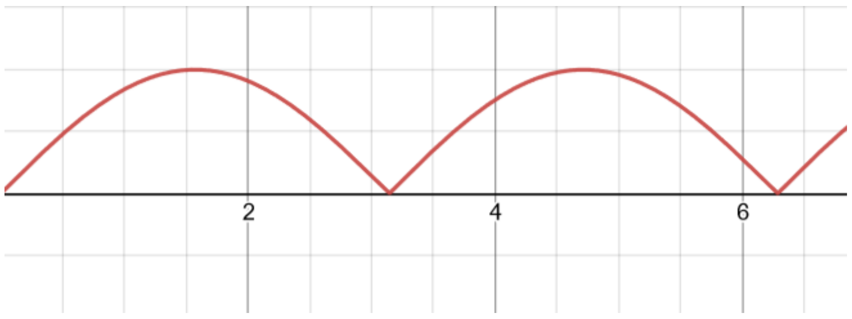


Figure 2.2: The absolute value of Perlin noise visualized with a sinus function

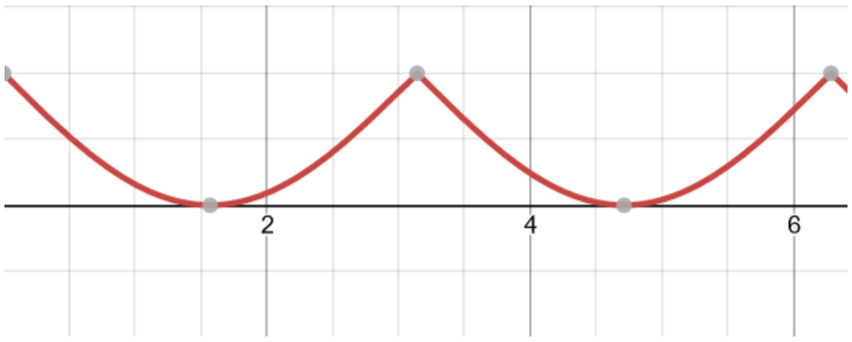


Figure 2.3: The absolute value of Perlin noise subtracted from the offset (in the code, 1), visualized with a sinus function

2.2.3 Color

The final color was calculated in two parts, diffuse and ambient. The diffuse part uses the amount of light incoming from the light source (light contribution) in order to calculate what should be in shadow or not. The factor is calculated with the dot product between the direction towards the light and the new normal from the displaced position, see 2.2 and 2.3.

$$lightDirection = normalize(lightPosition - displacedPosition); \tag{2.2}$$

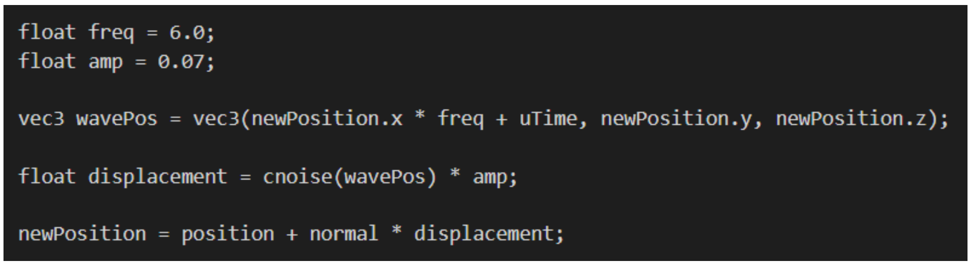
$$lightContribution = dot(lightDirection, newNormal); \tag{2.3}$$

The color of the terrain was made with the mix function. It takes two colors and a weight that specifies at which point the colors should be interpolated[4]. This was done with four different colors, where they are mixed multiple times in order to create an interpolation between all colors. The weight is dependent on the displacement value which enables the control over the colors between the highest and lowest point on the mesh. The mixed color value was then multiplied with the light contribution in order to yield the diffuse part.

The ambient part was a small amount of light contribution to all values in order to avoid the shadows being completely black and it was added to the diffuse part to get the resulting color.

2.3 Oceans

The oceans were made by rendering a hemisphere with a little bigger radius than the terrain and as a result, filling the lower valleys with water. The water was then animated with a time variable that is updated at every frame. The wave like pattern was created by Perlin noise based on the position of the vertex where the x coordinate was multiplied with a chosen frequency (freq) and the time variable added to the product. The amplitude (amp) of the resulting noise was then lowered and finally the displacement variable was obtained, see figure 2.4.



```
float freq = 6.0;
float amp = 0.07;

vec3 wavePos = vec3(newPosition.x * freq + uTime, newPosition.y, newPosition.z);

float displacement = cnoise(wavePos) * amp;

newPosition = position + normal * displacement;
```

Figure 2.4: The formula that creates the animated wave pattern

The color of the ocean was made the same way as the terrain color 2.2.3, except that it only has one color instead of multiple. This means that it does not use the mix function but instead uses the original chosen color.

Chapter 3

Results

In this section, the resulting planet is presented where it either contains terrain, ocean or both.

3.1 Terrain

The difference in the resulting terrain when changing the number of iterations (octaves) can be seen in the figures bellow. Figure 3.1 is the result from using one iteration, figure 3.2 is from using three iterations and finally, figure 3.3 is from using 12 iterations. The hemispherical polygon mesh has 200 x 200 segments and the values for amp, freq, weight can be seen in Table 3.1.

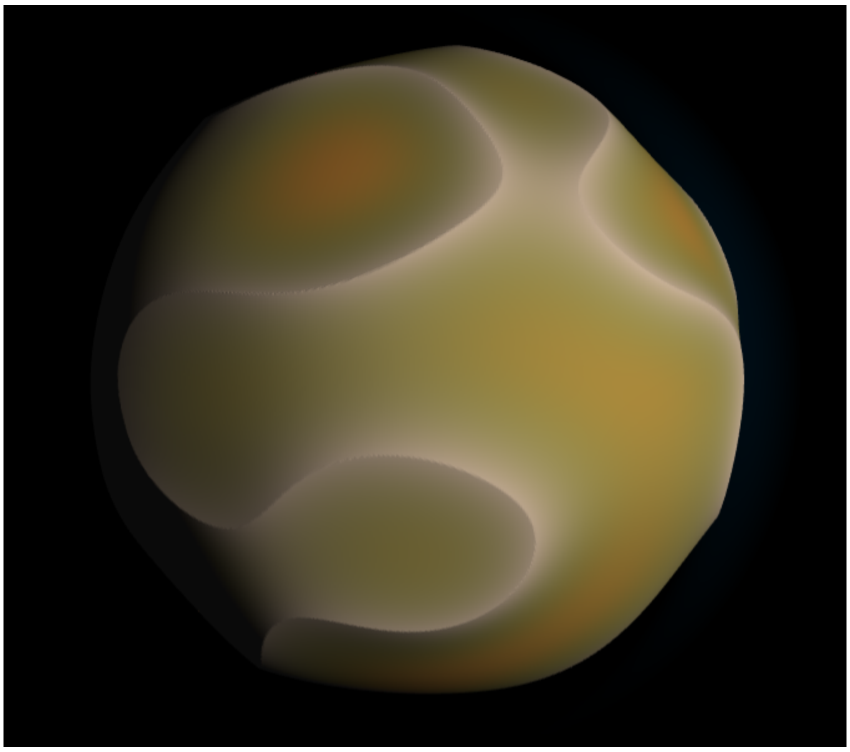


Figure 3.1: The result from using one iteration (octave) to create the mountain terrain

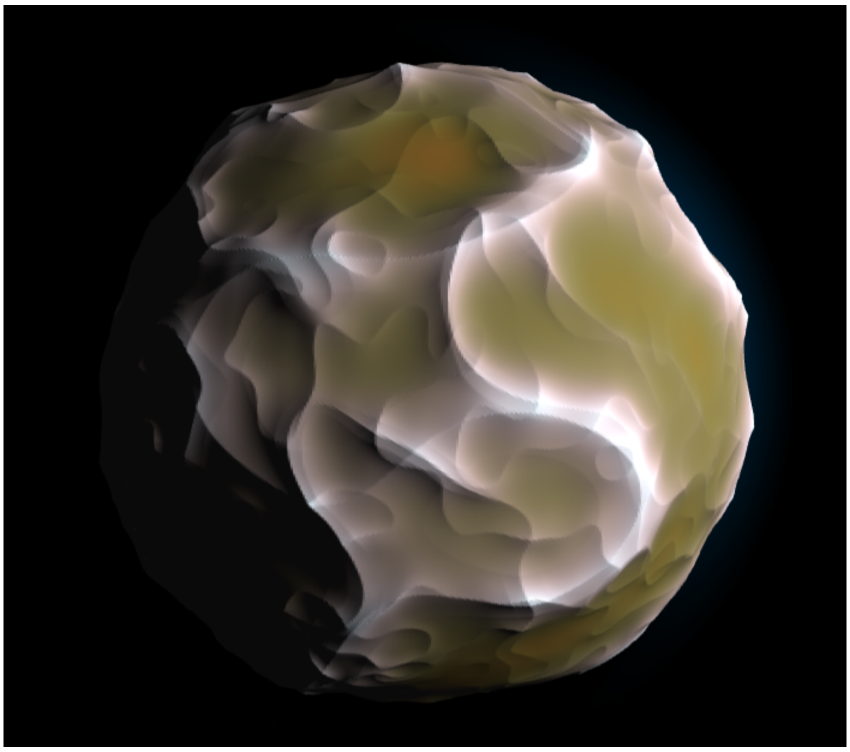


Figure 3.2: The result from using three iterations (octaves) to create the mountain terrain

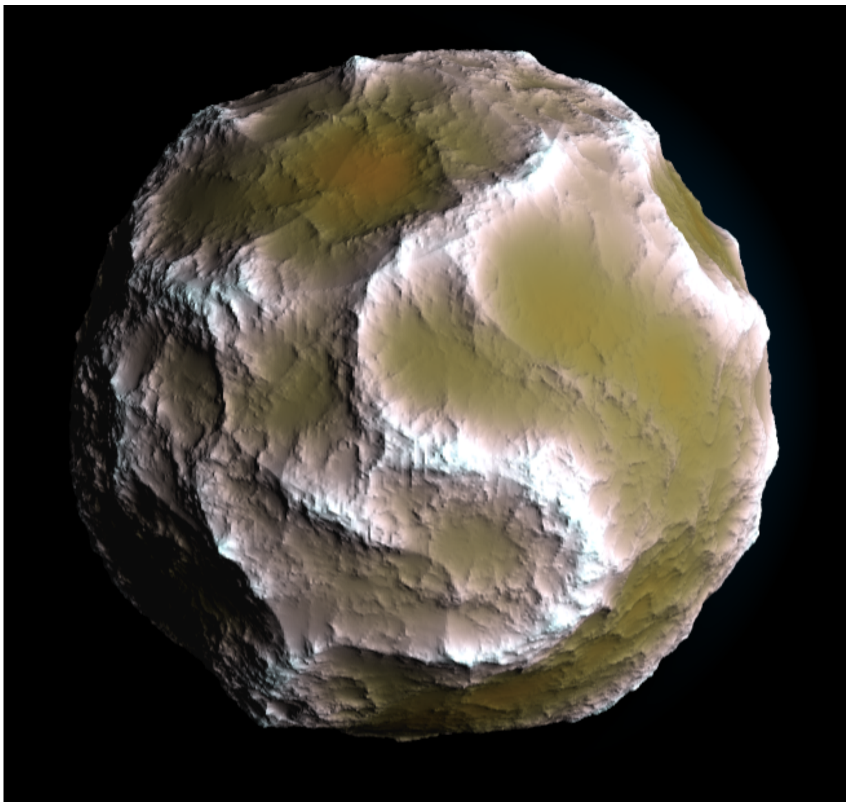


Figure 3.3: The result from using 12 iterations (octaves) to create the mountain terrain

Table 3.1: Values used to create Figure 3.1, 3.2, 3.3

freq	amp	weight	strength	freq gain	weight gain	amp gain
1.3	3.5	1.0	0.5	2.24	1.4	0.5

3.2 Ocean

A rendered result for the ocean can be seen in figure 3.4. The hemispherical polygon mesh has 100 x 100 segments and the Perlin noise has a frequency of 0.6 and an amplitude of 0.07.

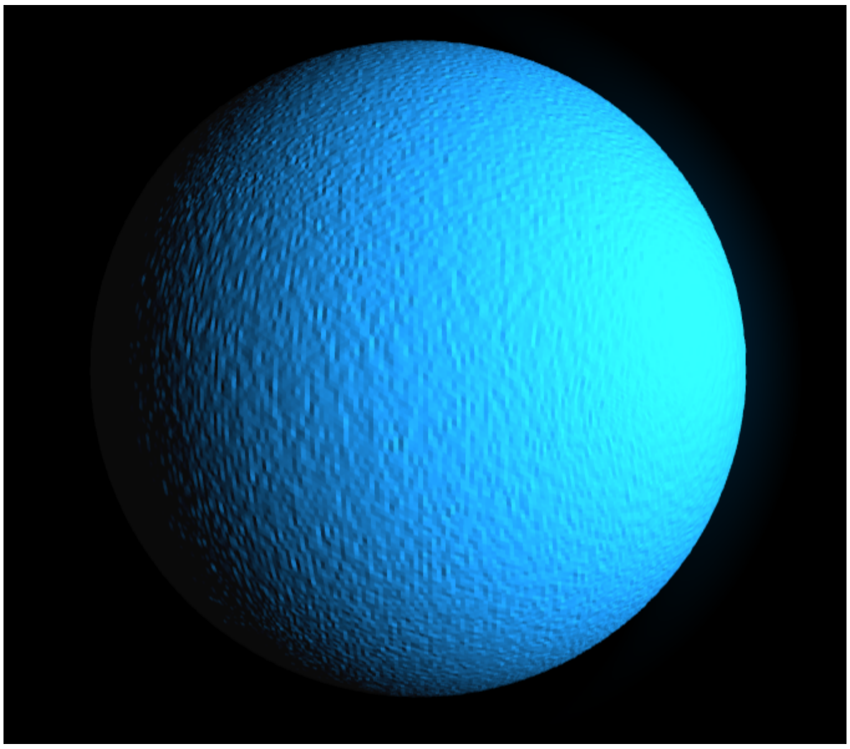


Figure 3.4: The resulting ocean using an frequency at 0.6, amplitude 0.07 and 100 x 100 segments.

Another result from using the ocean algorithm can be seen in figure 3.5. The frequency used is 6.0 and amplitude is 0.03. This hemispherical polygon mesh uses 150 x 150 segments.

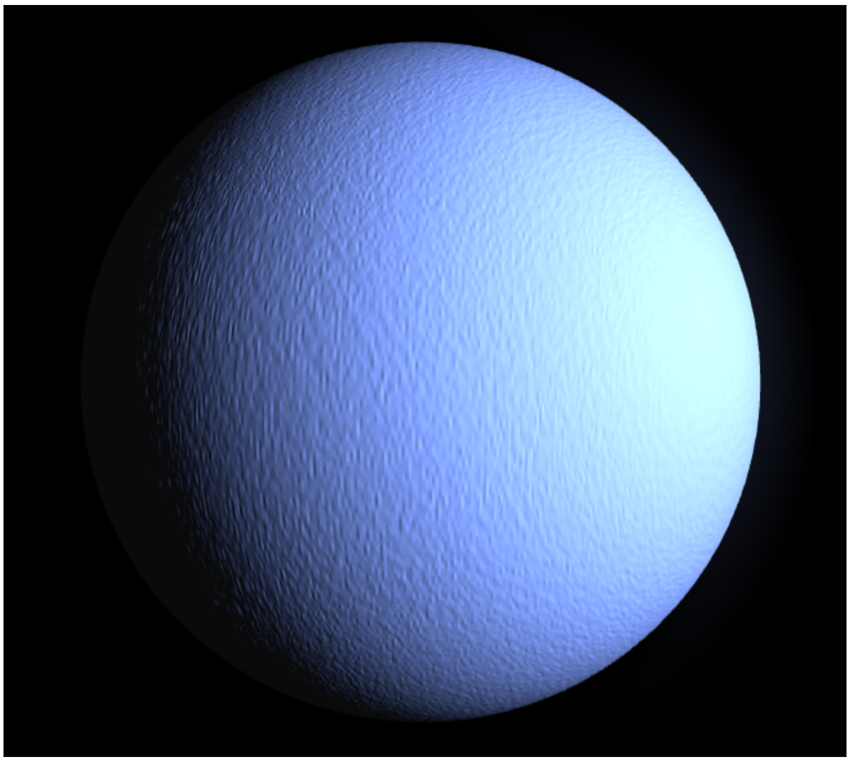


Figure 3.5: Resulting ocean with frequency at 6.0, amplitude 0.03 and 150 x 150 segments.

3.3 Terrain and Ocean

The results in figure 3.3 and 3.4 put together yield the result in figure 3.6. The same setup to create the planet can be seen in figure 3.7 but here, the terrains polygon mesh is instead 100 x 100 segments.

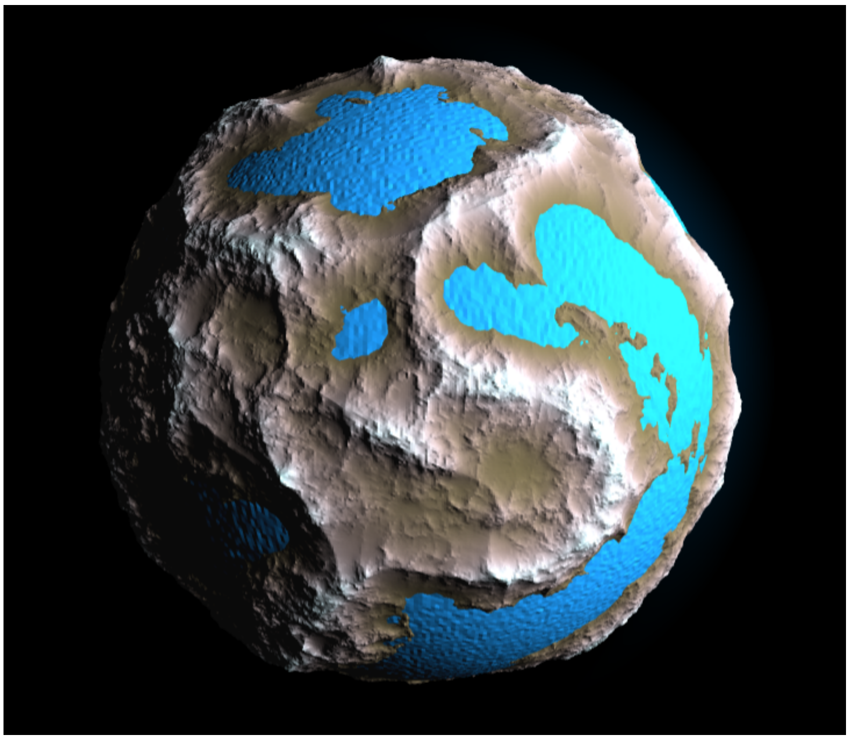


Figure 3.6: The result from putting the objects in figure 3.3 and 3.4 together.

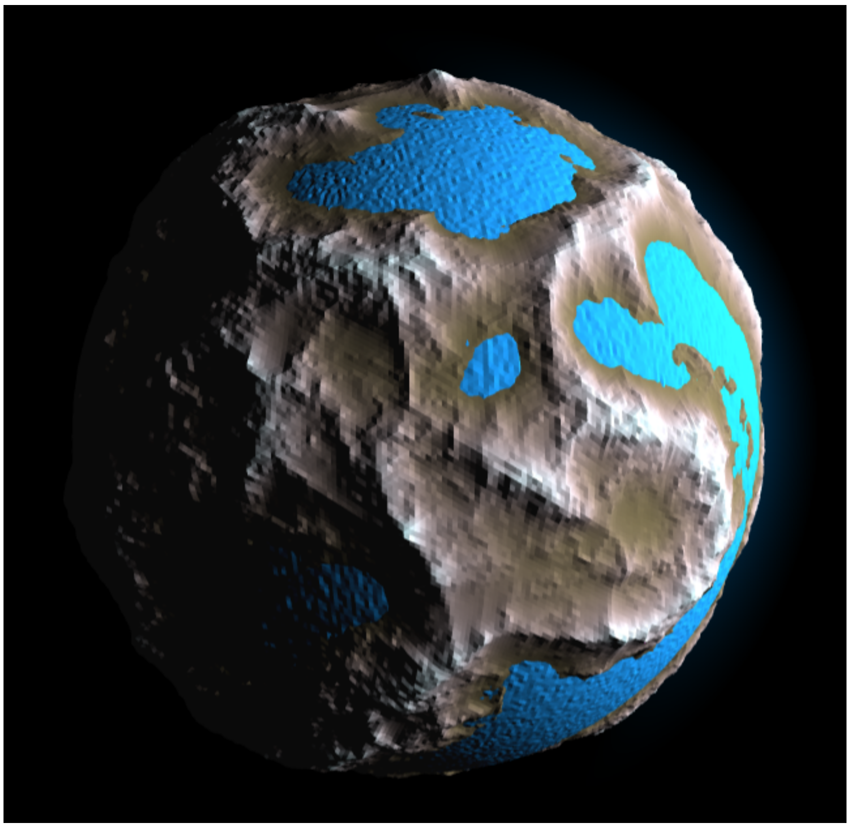


Figure 3.7: The same ocean and terrain setup as in figure 3.6 but with a terrain polygon mesh with 100 x 100 segments

Another rendered planet can be seen in figure 3.8 and the values used is in table 3.2. The terrains hemispherical polygon mesh has 250 x 250 segments. The oceans hemispherical polygon meshes radius has been lowered compared to the result in figure 3.6. The ocean rendered by itself can be seen in figure 3.5.

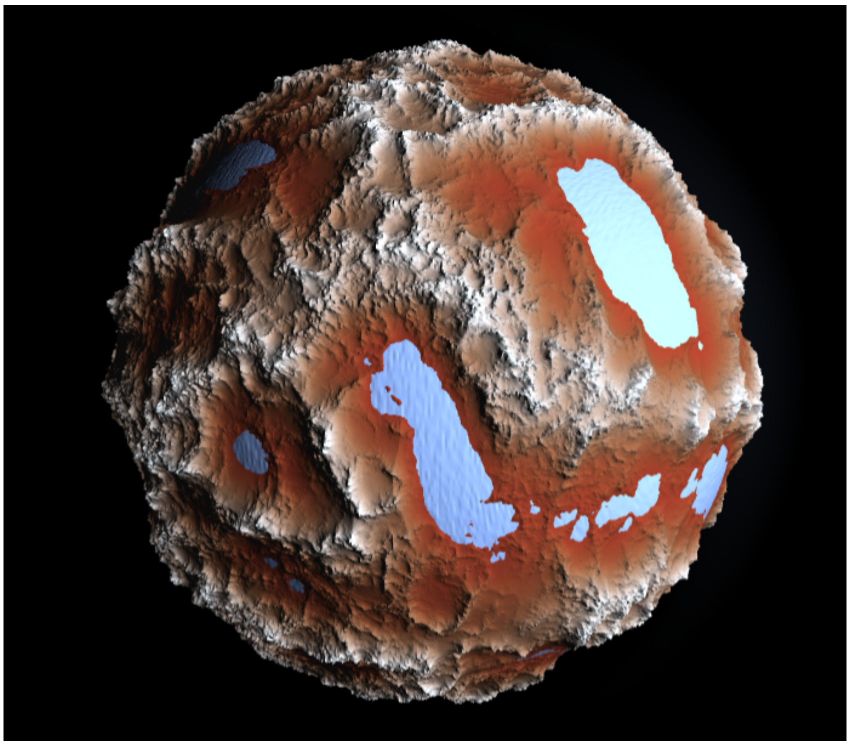


Figure 3.8: A generated planet where the values can be seen in table 3.2

Table 3.2: Values used to create the planet in figure 3.8

freq	amp	weight	strength	freq gain	weight gain	amp gain	Water freq	Water amp
2.1	2.4	1.0	0.5	2.28	1.3	0.6	6.0	0.03

Chapter 4

Analysis and Discussion

In figure 3.1 the result is plain Perlin noise and mountains have begun to form but they lack detail. In figure 3.2, three octaves of Perlin noise has been used and it is apparent that more details in the form of ridges has been created. When the octaves are around 12 such as the result in figure 3.3, the terrain starts to look realistic. The number of segments in the polygon mesh is also crucial when the goal is to create a lot of detail and realism, which can be seen when comparing figure 3.6 and 3.7. Figure 3.8 has the highest amount of segments in both the terrain and ocean which results in it looking far more sharp but the backside is that it is very heavy to render. For a still image, it might not be a problem but because the ocean is animated and the view of the planet can be rotated, it can cause the rendering to freeze.

Different types of terrain can be created by changing the frequency, weight and amplitude. The result in figure 3.8 has a higher frequency compared to the result in figure 3.6 and the difference it has lead to is that it has more mountain chains. The change in weight and amplitude is mostly made in order to fine tune the new displaced value. A higher weight value also makes the terrain look more eroded.

Comparing the ocean in figure 3.5 with 3.4, the first one has a higher number of segments and a lower amplitude. The water looks much smoother which would mean that the waves are more calm and a little more realistic but otherwise there is not much difference. Because the ocean is rendered far away and the waves therefore become infinitely small, it is possible to render a lower number of segments in order to save rendering time.

To improve this project, the coloring for the terrain could also be made procedural. Right now, the different levels are in only one color that is interpolated between each other but it would be more realistic with some color differences within each level. This could be done with noise and does not necessarily needs to be Perlin noise.

Chapter 5

Conclusion

The resulting planet in both figure 3.6 and 3.8 shows that combining Perlin noise with Fractal Brownian motion and turbulence is very effective in creating mountain terrain of an originally, hemispherical polygon mesh. Increased octaves results in more natural detail in the terrain and a higher frequency adds more mountain chains. A higher number of segments in the polygon mesh enables the possibility to create more realistic rendering but at the cost of time complexity. The ocean can be rendered with simple Perlin noise and still look realistic even if the number of segments are kept low.

The resulting code has the ability to create highly complex 3D models with fast rendering time. It is easy to change the variables in the algorithms if a new planet is wanted but with a slight variation. In conclusion, this shows that procedural graphics is a very versatile tool in computer graphics where an object is to be varying each time or large objects are to be created.

Bibliography

[1] Harold Serrano. Noise in computer graphics- a brief introduction. *Harold Serrano*, 2015-07-15.

[2] Khan Academy. Perlin noise. *Khan Academy*.

[3] Patricio Gonzalez Vivo and Jen Lowe. Fractal brownian motion. *The Book of Shaders*, 2015.

[4] Khronos Group. Mix, 2011-2014.