Procedural Planet Generation

CS 450 Final Project Report

Written by

Nicholas Ryan Olson
olsonn2@oregonstate.edu
olsonn2@engr.orst.edu

December 7, 2021
CS 461 - Intro to Computer Graphics
Oregon State University
Fall 2021

Table of Contents

1. Project Proposal

1.1 Project Inspiration

My inspiration for this project proposal is an ongoing series of videos by Sebastian Lague on Procedural Planet Generation in Unity [1]. A screenshot example of one such planet can be found in Figure 1. This project includes generating procedural terrain, lighting ground and ocean differently, creating shaders to simulate atmospheric refraction, creating biomes, and simulating ocean depth, so far. He does talk through his methodology, but he can skip a lot of boilerplate in Unity, and I would be re-implementing any functionality from scratch.

Figure 1: Screenshot of the project inspiration by Sebastian Lague [1].

1.2 Reducing Scope

This inspiration project is obviously way more work than any of the 100 point assignments in the course so far. My proposal, then, is to start with creating the procedural terrain. If this is all I accomplish in the week, then I'll call it finished, but if I have more time, I'll try to accomplish stretch goals (in order):

- Recreating parametric atmosphere shaders
- Adding an ocean that fills above terrain below a certain depth and is colored blue
- Lighting the different surfaces differently (ocean vs land, etc.)
- Showing pin-point stars as a background for the planet
- Multiple planets placed around in space
- Allowing you to hover over a planet, creating a selection cursor around it
- Allowing you to click another planet to smoothly fly there to orbit instead

I list these all out because I'm not sure how much exactly you'd consider to be worth 100 points, so feel free to add any number of these to the MVP requirements or to give any other feedback!

2. Project Retrospective

I had significantly less time this week to work on the project than I was hoping for, but I still ended up with something I'm proud of. In the end I met the MVP of procedurally generated terrain, although it was far harder than I anticipated. Screenshots of two randomly generated planets that my program created can be found in Figure 2 and Figure 3. Additional features will not all be pictured in this report, but will all be shown in the video accompanying this report [2].

Figure 2: First example planet generated by my program, w/ front lighting.

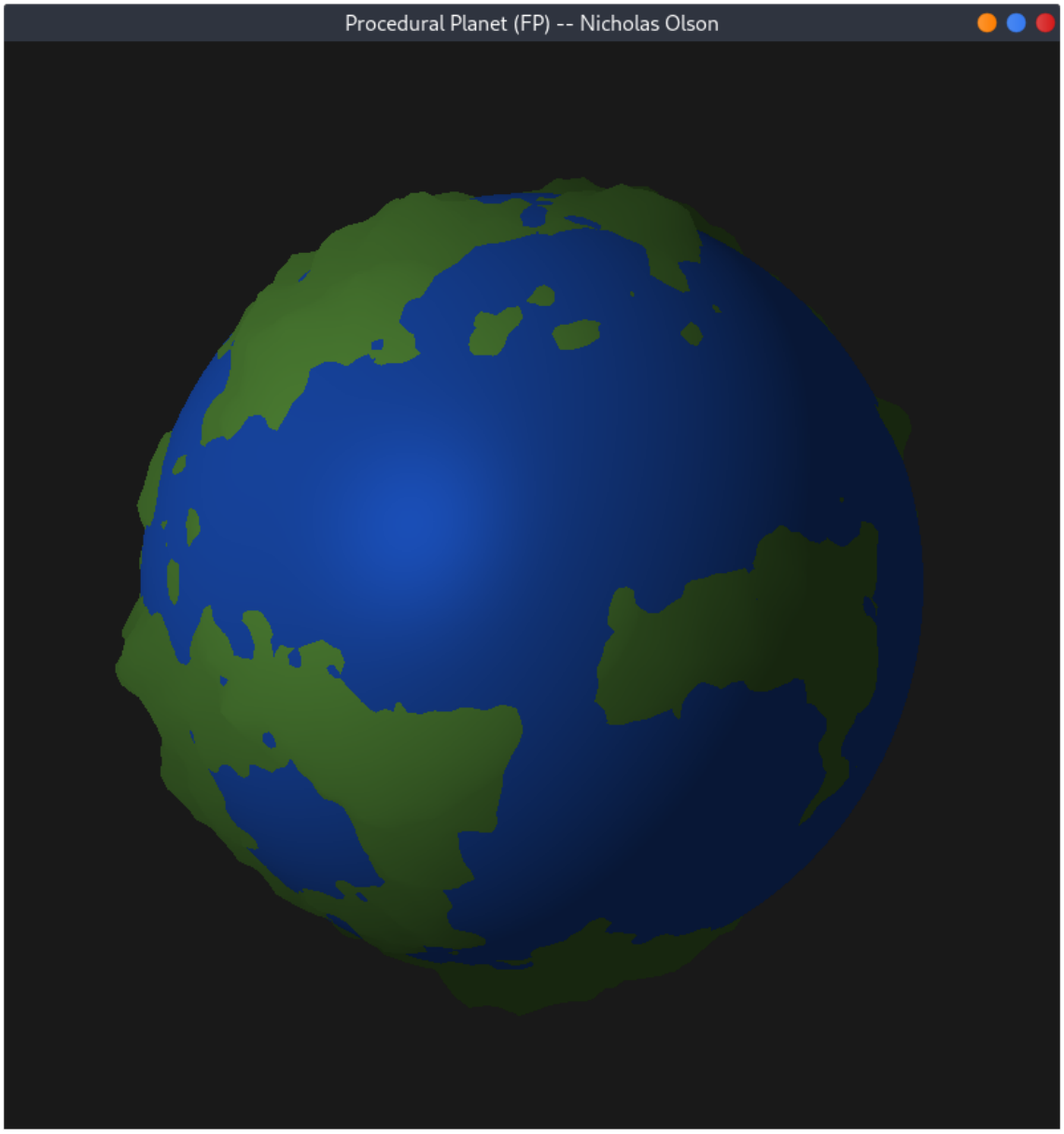

Figure 3: Second example planet generated by my program, w/ scene lighting.

2.1 Project Process and Differences from Proposal

In order to replicate some aspects of Sebastian Lague's Unity project in OpenGL, I started by finding and pulling in a C++ module by Song Ho Ahn to render an Icosphere (subdivision of a icosahedron, which gives near-regularly-sized triangle faces that don't stretch toward the poles) [3] and a C++ noise library "fastnoiselib" by GitHub user Auburn to generally simplex2 noise patterns [4]. I originally planned on using the texture coordinates index into a 2D texture filled with these noise values to offset the geometry, but quickly realized that the texture coordinates of the icosphere lead to many, many seams in the geometry. I thought I might try to use 3D textures to use the position of the vertices as the index in the shader, but I had a really hard time

getting it to work, so shifted to trying to use the X and Y coordinates of the vertex to index into the 2D texture. I was able to get some distortions happening (the first of which I rendered is shown in Figure 4) and I was pretty excited.

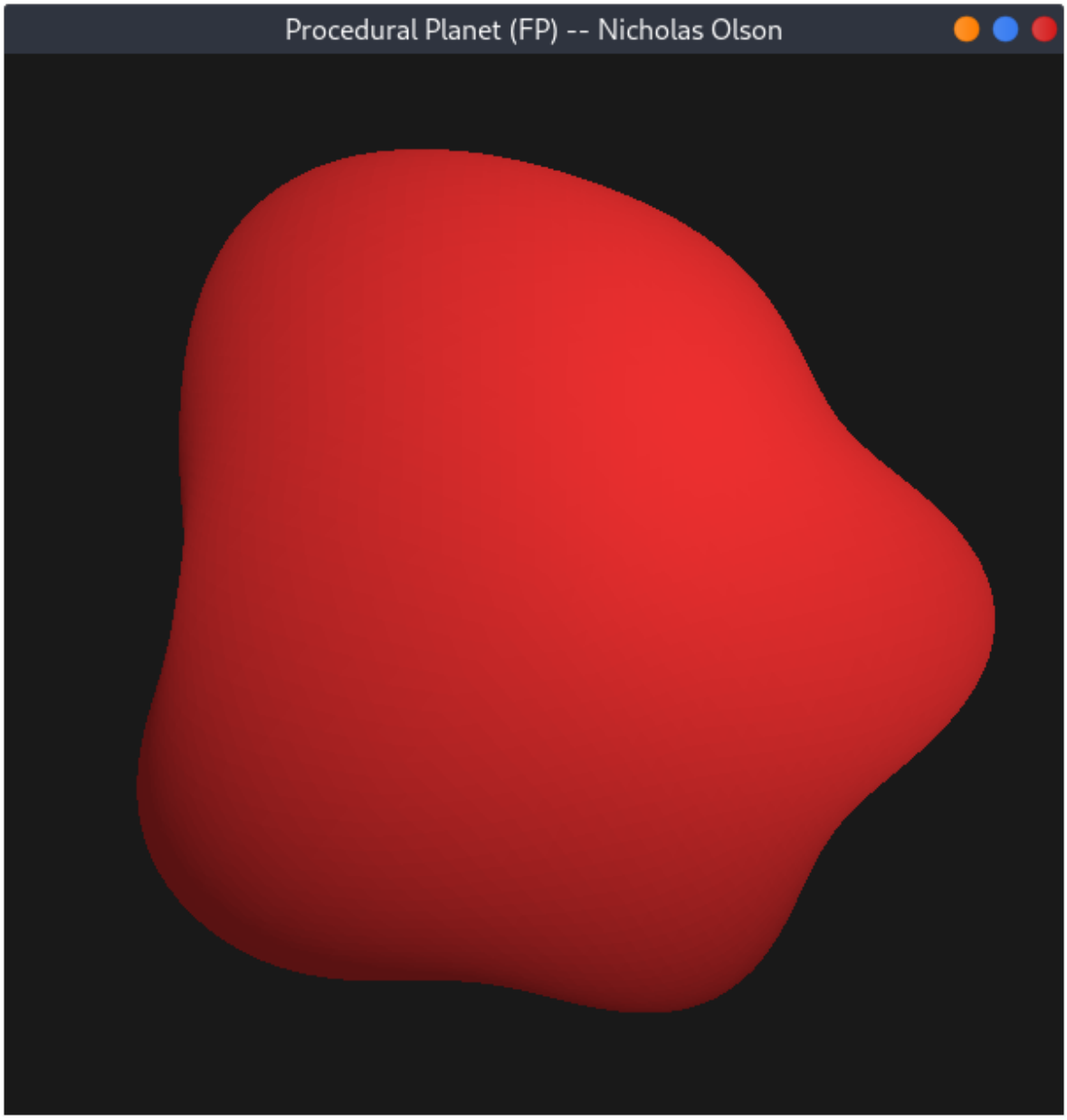

Figure 4: First successful shader-drawn, noise-distorted geometry of the project.

From this point, I combined some noise patterns to try to approximate terrain, and then tried to color all terrain under a given height blue, to make it more obvious what I was looking at. This made me realize how weird, stretchy, and symmetric my planets were with my 2D texture method, though. My favorite example of this, which looks like a smiling face, can be seen in the following Figure 5.

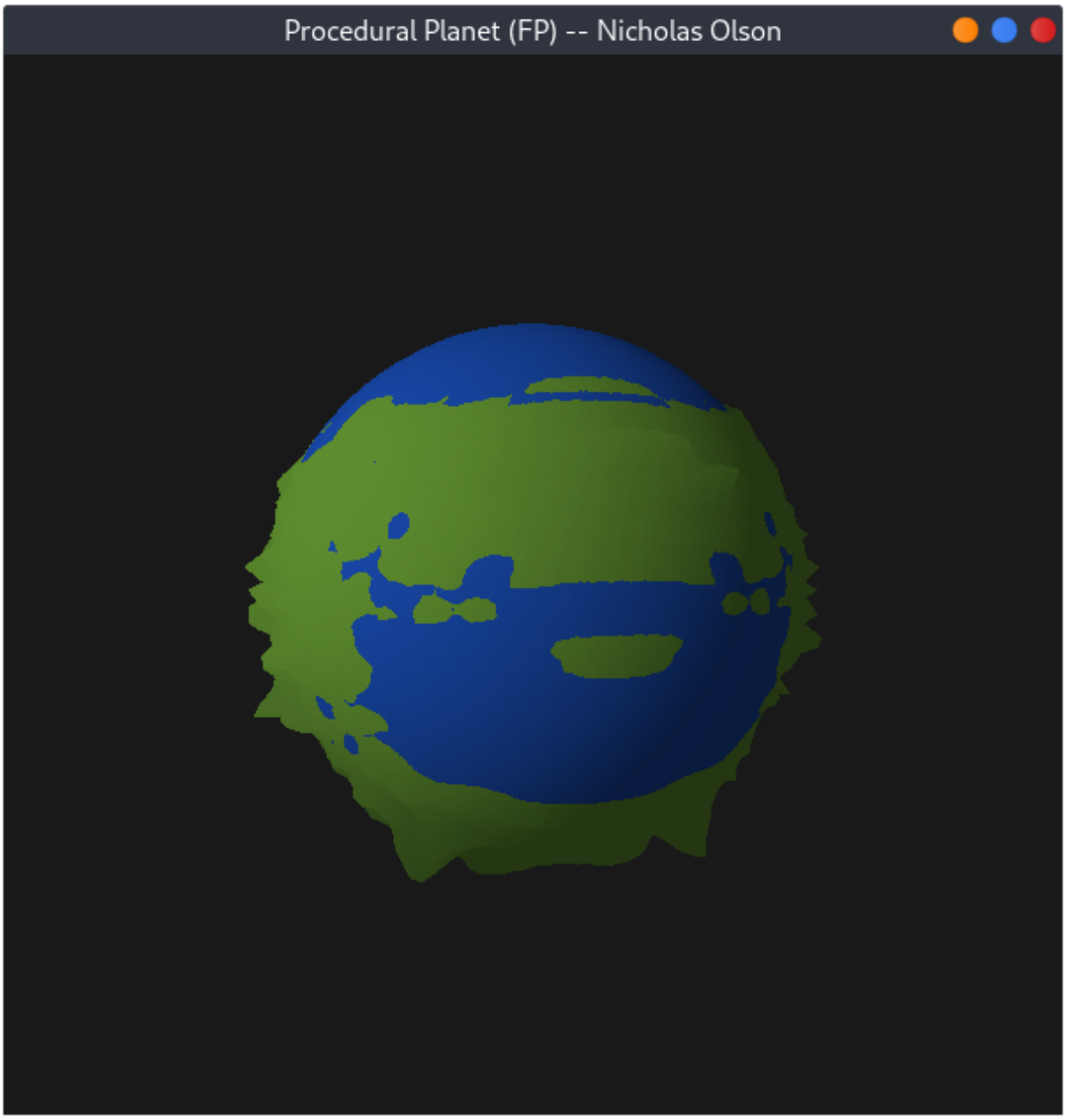

Figure 5: Symmetric, poorly done terrain that happens to look like a happy creature.

At this point I spent many, many hours trying to get 3D textures to work, and almost gave up entirely, but eventually got them working. It's hard to tell what change finally did it, because I had been having the noise coming through as zeroes in the vertex and fragment shaders, tried a bunch of different things, saw the fragment shader was still zero (black), tried a bunch more things, then eventually realized that some change down the line had properly sent the noise into the vertex shader, but it was then getting lost in translation. My best guess is that it was originally a mipmapping issue. After fixing this second issue, I finally had 3D textures and displacements working, without weird symmetry or stretching.

After resolving the 3D texture issue, I refocused, and with the little time I had left I added a second sphere at an “ocean height” relative to the average value of the noise for the planet terrain, added more layers of noise and filtering to the generation, added a second lighting mode, and enabled the user to adjust the height of the ocean and the resolution of the icosphere, as well as generate a new planer with keybinds. With that, I cleaned up the codebase a little and started on this report!

All in all, I did just about meet the MVP I set out in the proposal, and additionally my stretch goals of the ocean surface and the different lighting between the land and water (ocean is more shiny than land). Otherwise, I did just about what I set out in the proposal.

2.2 Project Takeaways

First and foremost I learned the trials and tribulations of seemingly easy switches like from 2D to 3D texturing. Otherwise, the biggest takeaway for me is how much of the work an established game engine like Unity does for a developer. Sebastian, in the videos that inspired this project, writes the fun shading code, but almost none of the nitty gritty details that I had to get into to make this all work. Additionally, he was able to have parameters that he could adjust live with input boxes and sliders in the Unity UI, which I technically *could* have coded in my development process, but would have taken more than the time I had to finish. This was tedious and frustrating at times, but also an incredibly fun process. I’ve greatly enjoyed this class!

2.3 Project Keybinds

The project uses several keybinds to enable further functionality than just rotating and zooming around the scene with the mouse. These follow (* implies case-sensitivity):

- Space starts/stops the planet from rotating about the Y-Axis
- Enter generates a new planet in place
- L switches between a light at the eye and a fixed light in space
- O and P switch between orthographic (default) and perspective projection, respectively
- W and S increment and decrement the resolution of the icosphere
- M switches the icosphere in and out of smooth shading mode
- D and A increase and decrease the height of the ocean on the planet in steps
- ?* shows and hides the keybind help string
- R resets the program to initial state
- Q quits the program

2.4 Project Demonstration Video

My demonstration video can be found at this link [2].

References

[1] Sebastian Lague, “Procedural Planet Generation Playlist,” *YouTube*, 16-Aug-2018. [Online]. Available: https://www.youtube.com/playlist?list=PLFt_AvWsXl0cONs3T0By4puYy6GM22ko8. [Accessed: 08-Nov-2021].

[2] N. Olson, “Procedural Planet - Nicholas Olson - Computer Graphics Final Project,” *YouTube*, 07-Dec-2021. [Online]. Available: https://youtu.be/fXphVQqbEFI. [Accessed: 07-Dec-2021].

[3] S. H. Ahn, “OpenGL Sphere,” songho.ca. [Online]. Available: http://www.songho.ca/opengl/gl_sphere.html. [Accessed: 07-Dec-2021].

[4] Auburn, “fastnoiselite: Fast portable noise library - C# C++ C Java(script) HLSL,” GitHub. [Online]. Available: https://github.com/Auburn/FastNoiseLite. [Accessed: 07-Dec-2021].