John Barnyk

Mr. Richards

Graduation Project

25 February 2022

Log Check #2

For this Log Check, while I did put in a couple of hours less, I remained focused on my goal. This time around I focused on the brunt of render scripts. The actual progress may seem considerably lesser than the previous log check, this time I did a lot more learning than doing. I spent much of this log check on understanding, not just reading, the theory behind raymarching.

The first major issue I hit was the site I was learning from. I have been using the website inspirnathan.com, which is a blog run by an independent developer on a variety of topics. The issue arose when the site became no longer accessible. I initially panicked, as this is the best tool I have been able to find for learning glsl to date. To adjust to this, I at first tried some other sites or tutorials, but none were anywhere near the grade of the blog. Then, I remembered a website called archive.org, which is a site dedicated to archiving the internet, for this exact reason. I checked and the site did have backups of the blog, so now I just read the tutorials through archive.org.

Another issue I hit was related to the programing language. I had to learn GLSL, which is a programming language that utilizes the graphics processor of the computer instead of the random-access memory, like python. Learning this language was slightly annoying as GLSL is closer to C-style language than I am used to. I followed the tutorial and gained enough of a grasp of the language and proper usage of it.

The next two bumps I hit were in the code. The first of these was the matter of rendering skewed cubes, that is cubes that do not line up with the coordinate plane. Where the equation for an aligned cube is simply the equation for a sphere, split into x, y, and z, with the largest being used, a skewed cube could not use such streamlining. To remedy this solution, one could do some shenanigans involving a large amount of sine and cosine math, I realized that, for my application, it would not be necessary, as all cubes will be aligned. In the future, should I desire skewed cubes, I have most of the math completed, I would just need to implement it.

The other code-related issue that arose had to do with the world generation. It is extremely inefficient to load a 16^3 block of cubes at once when one knows the player will not see them. The game will likely not even run on the computer, as the computer would have to take the load of basically wasting computational power on thousands of cubes that will not be seen. That is 8 vertices, 12 edges, and 6 faces for each cube, each of which has several additional values that are stored in either the memory (RAM) or the graphics memory (VRAM). Most computers have very limited VRAM, as many do not have graphics cards, which are used to more efficiently calculate and store graphics tasks. Without these, these jobs are put on to the already strained RAM. While I could just run the program on my home computer, which is considerably stronger than the computer I use for school, the better solution is to optimize how the game world is loaded.

To solve this problem, I generated meshes. Meshes are simply a combination of vertices, edges, and faces which, when combined, create a 3d model. I made a script that checks where blocks should be in the world and then makes/fixes an optimized mesh accordingly.

While I hit several different problems in different fields in this log check, I believe I overcame them all in stride. In the next log check, I will focus on implementing this renderer into the actual game space.

Word Count: 662

Mentor’s Signature: \_\_\_\_\_\_\_\_\_\_\_\_

Mentor’s Comment: