What do you think happens inside your computer or console when you play your favorite next-gen video game? Most people don’t even wonder what goes on under the hood of all of these games. Today, I will be showing you my attempt at making a 3d rendering engine for 3d models and explaining of how it works.

I would like to start off by saying that the way I do it is by all means not the end all be all way to render 3d objects. To start, there are many different API to render 3d graphics. An api, in the case of 3d rendering, is a set of code that takes code that developers write, and translates it into a language most computer’s graphics proessors can understand. I am personally using an API called OpenGL for my rendering. OpenGL is a graphics API that is different from most rendering API’s because it is open source, and compatible with nearly every computer made in the past 10 years.

Also, I am most defiantly not using the most efficient way offered in OpenGL to render objects. As this was my first time making a rendering program, I only used the method that looked the best for using small amounts of processing power.

With that out of the way, Let’s get on with the program.

Scene 2:

As you can see here, a red cube is currently being rendered in the program. This is just a simple demonstration of objects turning with this render engine. As I go over to the next object note the camera I am currently controlling. This is actually not a camera. In fact, it’s not even moving. The entire world is shifting in the opposite direction that the camera is moving.

As you can see, this green sphere is chopped up into triangles. This is to demonstrate how OpenGL goes about rendering objects. The rendering mechanic of OpenGL that I am currently using is based entirely out of triangles. You might expect objects to be rendered using rectangles or other polygons rather than using triangles, but triangles are actually really efficient for computer memory. For every surface, the computer only needs to store 9 data points, the x, y, and z coordinates of each point. This may not seem like it would save much data, but think about this example. Rectangles use 12 data points of memory per surface, and when average objects use up to 30,000 surfaces, rendering with triangles saves up to 90,000 data points over rendering using rectangles.

The Java code for this project is over 4,000 lines, and almost all of it is essential to the program not crashing horribly. However, most of it is just a lot of complex math to change a 3d world into a rectangle that can be displayed on a screen. As this would take way too much time to explain, and as I’m not a mathematician, I will just be going over what everything does. This explanation does require a small understanding of how java works though. All that is really necessary to know is that java divides code up into classes and methods. A method is a set of code that can be executed more than once when referenced by name. Classes can be either static or dynamic. Static classes are classes where methods are only ever executed from a single instance of that class. Dynamic classes are classes that can be instanced, and can have different information stored in them depending on the information given by the static code that instances them.

Let’s start where the program starts, in the launch class. There is not much that is special about this class, it is just where the main loop runs, updating everything in the program 60 times a second.

Proceeding to what is essentially the heart of the program; we will now look at the Scene Loader class. The Scene Loader class itself is static, but it deals with instances of a dynamic class, Scene Bases. Each scene base has a update and input loop, but these are only running in one Scene Base at a time. The Scene Loader keeps track of all the scenes registered when the program starts, and determines by a request method what loops should be running at what times. The Scene Loader also is where the main rendering happens. The Scene Bases contain instances of another dynamic class, Render Objects. The Scene Loader first detects which Scene Base is selected and then takes the render objects from the Scene Bases, and renders the object mesh from in the render objects and sends all the accompanying data such as the location transformation and material, to the shaders. Open GL uses it’s own programming language called GL SL for shaders, which is heavily based off the coding language C. All the shaders do is essentially take all the data and determine the offset, color, and rotation of each pixel, using calculus.

Now that you know all this information about 3d rendering, the next time you play a 3d video game, think about the long sequence from 2d screen to virtual world.