**Jackson Buchmeyer**

**Assignment #2 Paper**

First, it is important to note that most of the graphics pipeline is performed in the GPU. The beginning of the graphics pipeline starts in the CPU. The first part of the pipeline involves setting up our data. This involves creating a link between the attribute variables in what’s called a vertex shader program and the GPU buffers that contain the data those variables utilize. These variables are what are called uniform variables because they remain constant throughout the rendering process.

A shader program is essentially a bunch of instructions that tell that computer how to handle graphics data within the pipeline. It’s with these shader programs that a programmer can influence the pipeline. The vertex shader program is responsible for generating coordinates for the vertices. This shader is called once per vertex. It takes this model and then positions in front of a virtual camera.

Moving on, the clipper gets rid of everything that is not in view of the virtual camera. It essentially determines what bounds of our screen are available. Then, the primitive assembler takes our vertices and forms the shapes that we are trying to make. For example, in our triangle JavaScript code, it creates triangles using those points that we fed the program. Following that, we enter the stage of the “viewport transform”. This maps the data from the coordinates created by the vertex shader program into a viewport defined in pixels. A viewport is a polygon viewing region.

Next, the process of rasterization begins. The GPU first figures out what pixels in the raster image are inside its boundaries. Then, it activates the next shader program, the fragment shader program. This is another part of the pipeline that is manipulatable by programmers, and it assigns colors to pixels that appear on the surface of whatever we’re trying to render. Again, this fragment shader is executed on every pixel that makes up the geometric primitive and outputs the color value for each pixel. A fragment can be described as a potential pixel. The fragment processor allows for the adding of shaders, lights, etc. to pixels.

Finally, the color of the pixel from the fragment shader and the color of the pixel assigned to the output draw buffer are combined to create the finalized output image.

**Sources:**

1. <https://runestone.academy/runestone/books/published/learnwebgl2/01_the_big_picture/3_3d_graphics_pipeline.html>
2. <https://www3.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html>