Concept:

For myself there was a lot in these two weeks. I feel like there were a bunch of different moving parts in terms of learning how WebGL operates. While also learning how rasterization works and what it means to create/use a shader.

One concept I thought was interesting was the z-Buffer solution to the painter's algorithm problem. It reminded me of how we coded our raytracer, where we only colored in the pixel of whatever object was the closest. I am sure they are not the same, since the z-Buffer is a bit different behind the curtains. However, I am curious if there are some ways to efficiently fix the "z-fighting" issue when using the z-Buffer method.

The next concept that caught my attention was the shaders pipeline. From my understanding all of the "hard work" is done in the vertex shader, and then the fragment shader is a lot less work in terms of, it is just coloring in the pixels with the correct color. With the fragment shader using data passed in from the vertex shader part. It is interesting to see how these two correlate with each other when understanding the pipeline. From my understanding, if your math (vertex shader) is wrong, then the fragment shader won't be entirely correct.

Experience:

I found it interesting that webGL was like a state machine. Meaning if you modify something, that modification will stay until you change it or do something else with it. Understanding this, can help a lot when using webGL and making API calls.

For example, I struggled with why my code wasn't working.

```
// 1. update the triangle geometry world matrix here (used to place the triangle into world s
// 2. reset view matrix to identity
// 3. set the projection matrix using values from online guide

var rotationMatrix = new Matrix4().makeRotationY(degrees);

triangleGeometry.worldMatrix.makeIdentity();

var translationMatrix = new Matrix4().makeTranslation(0,0,-7);
triangleGeometry.worldMatrix.multiply(translationMatrix).multiply(rotationMatrix);
projectionMatrix.makePerspective(45,(768/512),0.1,1000);

gl.bindBuffer(gl.ARRAY_BUFFER, triangleGeometry.positionBuffer);
gl.vertexAttribPointer(
    triangleGeometry.shaderProgram.vertexPositionAttribute,
    3,
    gl.FLOAT,
    gl.FALSE,
    0, // stride - used for mixing data in a single buffer, ignore
    0  // offset - used for mixing data in a single buffer, ignore
);
```

Once I re-read through the explanation of how webGL works, which was a lot of documentation and the first few pages of the homework document. It is because I wasn't using .bindBuffer(). Which from my understanding, basically stores information to use all the time. So when you buffer the triangleGeometry.positionBuffer, it is permanent of whatever that value is until you change it. So you could just buffer it once, and that is it. But that won't do anyone any good. Basically you have to be making a API call for any change you want to happen.