Final Project Report

Team Members
Carolyn Lai - 89089268
Wesley Tseng - 22428172

Goal
To implement stylized cel shading onto standard objects and scenery in WebGL

Report
For our final project, we utilized the base code from Programming Assignment 1 (*PA1*). The final project mimics the HTML interface of *PA1* and adds a few sliders in the HTML file, *cel_shading.html*. More specifically, we added a checkbox to show an outline, a dropdown box to enable cel shading, and sliders that modified the number of bands displayed in the cel shading. Within the HTML file, we removed all excess shaders except the diffuse shader and created two additional fragment and vertex shaders. The first set of shaders focused on producing the cel shading effect on the 3D model. We utilized the vertex shader from *PA1* to maintain the lighting and camera positions, but we modified the fragment shader by distinguishing the colors into a discrete set categorized by light intensity levels. On top of that, the fragment shader will detect the selected number of cel shading bands and adjust the discrete values accordingly. The second set of shaders focused on producing the outline of the 3D model. For the vertex shader, we rendered cel-shaded scene into a texture via a frame buffer and then ran the Sobel image filter on the texture to detect the edges of the objects in the scene. The fragment shader would take a pixel and calculate the magnitude of the difference in pixel intensity within a 3x3 border of of the original pixel. After taking the magnitude, the fragment shader would determine if the calculated value exceeded a certain threshold - if the value exceeded the threshold, then a darker line would be drawn. The outline color was then selectable based on a color picker and made togglable via JavaScript functions (inside *final.js*) that modified the values on the canvas.

Most of the shading work is performed inside the JavaScript file, *cel_shading_webgl.js*. Inside the JavaScript file, we implemented multiple vertex and fragment shaders to produce the cel shading effect as well as the outline for the cel shading model. At first, we thought that by simply creating another shader program and passing in the appropriate uniform variables, we would be able to produce the additional outline effect for the cel shaded model. Unfortunately this wasn't the case. What ended up happening was that whatever shader program was called first would be the only shader program used and any of the following programs called would not be acknowledged. Our original approach for generating the outlines of the scene was to calculate the position of the vertex as if we were producing the 3D model regularly, but then we would offset the vertex position by a little bit. In other words, this created an enlarged version of the original 3D model. As a result of only one program being used at a time, we could only either have the 3D cel shaded model or the enlarged black model. We thought utilizing *gl.cullFace(gl.FRONT)* would resolve the issue, since it essentially removes the front polygons of the model, it would correctly outline the model. We had no luck with this approach either.

As a result, we shifted our approach to utilizing the Sobel operator as a way to form the outlines of our 3D model. During this time, we discovered the solution to our problem of being unable to pass our shader outputs to other shader programs - create a *gl.FRAME_BUFFER* for the shader program. We used the frame buffer to capture the output of the first shader program and then we stored the output as a texture within the frame buffer. After that, we called the second shader program via *gl.useProgram()*, and passed the generated texture (*sampler2D*) as a parameter. We then applied the Sobel edge detection algorithm to the texture, which generated the outlines of the image and outputted the final image. This allows us to successfully create an outline effect for our 3D model while maintaining the cel shading effect.

<u>Goals That Were Missed</u>

- Implement and use the depth texture buffer for the Sobel operator
    - We ran the edge detection algorithm on the cel shaded scene, which generally has lots of hard edges due to the discretized color and therefore the edge detection algorithm creates extra outlines on the models. Given enough time, we wanted to instead use the depth values of the scene (stored as a texture) as our input to the edge detection algorithm, which would output a far nicer outline.
- Modify the outline program to enable various outline thicknesses
    - As a result of using the Sobel operator, our ability to change outline offset/thickness was rendered useless.
    - Our original approach would have allowed us to adjust the outline thickness by offsetting the vertices by a greater amount.
- Adding additional 3D models
- Adding various kinds of textures to 3D models

<u>Sources Referenced:</u>

- Computer Graphics Stack Exchange
- Arnaud Droxler’s GitHub Cel Shader example
- WebGL Fundamentals
- LearningWebGL.com
- https://gist.github.com/Hebali/6ebfc66106459aacee6a9fac029d0115
    - We modified this user’s approach to the Sobel operator