CMPT 361: WebGL Tutorial

We will go through the contents of this tutorial in class. **Please ensure you have a working JavaScript development environment (editor, debugger, and local server) ahead of time. See the previous tutorial for further information.**

Visual Studio Code Extensions

1. The GLSL Lint extension will provide GLSL syntax highlighting and validation, which will be useful for writing shader code.
2. An extension which highlights TODO comments such as Todo Tree may be helpful for following this tutorial.
3. Please also see the extensions in the previous tutorial document.

Project Setup

1. Extract the tutorial files into a new folder for your project.
2. Start a local web server from your project folder, either by using an editor extension or other methods such as the provided Python script.

Part 1: Initializing the Canvas and WebGL Context

1. Inside the **initializeContext** function in the **tutorial.js** file, get and store the canvas and the WebGL2 context.
2. Determine the ratio between physical and CSS pixels.
3. Set the height and width of the canvas using the ratio applied to the clientWidth and clientHeight.
4. Set the viewport size.
5. Set the clear color to white.
6. Set the line width to 1.0.
7. Call **initializeContext** at the very start of the **setup** function.

Part 2: Creating Vertex Buffers

1. Inside the **createBuffers** function, create a buffer for the vertex position data.
2. Bind the position buffer to **ARRAY_BUFFER**.
3. Set the buffer data to the provided **positions** array.
4. Repeat these steps for the vertex color data.
5. Add a call to **createBuffers** inside the **setup** function.

Part 3: Creating a Vertex Shader

1. Inside the **main.vert** file, define the inputs for the position and color vertex data.
2. Define the output for the vertex color.

3. Define a uniform mat4 variable to store the transformation matrix of the vertex shader.
4. Inside the **main** function, transform the vertex position by the transformation matrix and store the result in **gl_Position**.
5. Also pass the color to the output defined in the second step.

Part 4: Creating a Fragment Shader

1. In **main.frag**, define the color input to the fragment shader, which should match the output of the vertex shader.
2. Define the output of the fragment shader, which is also a vec4 color.
3. Inside the **main** function, assign the input to the output.

Part 5: Loading and Compiling Shaders from Files

1. Add a call to **loadShaders** using the **await** keyword inside the **setup** function in the **tutorial.js** file. This function is implemented for you and will read the shader source code from the local web server.
2. Inside the **compileShaders** function, create a vertex shader.
3. Specify the shader source code for the vertex shader.
4. Compile the shader.
5. Check the compile status of the shader and log any errors.
6. Repeat steps 2-5 for the fragment shader.
7. Create a shader program.
8. Attach the vertex and fragment shaders to the program.
9. Link the program.
10. Check the link status of the shader program and log any errors.
11. Add a call to **compileShaders** after the call to **loadShaders** inside the **setup** function.

Part 6: Setting Uniform Variables

1. Inside the **setUniformVariables** function, make the rendering context use the compiled shader program.
2. Get the location of the uniform variable for the transformation matrix in the vertex shader.
3. Set the data of the uniform variable to that of the 4x4 identity matrix.
4. Add a call to **setUniformVariables** after the call to **compileShaders** in the **setup** function.

Part 7: Creating Vertex Array Objects

1. Inside the **createVertexArrayObjects** function, create a new vertex array.
2. Bind the vertex array to the context so we can record the state in the following steps.
3. Get the shader location of the vertex attribute for the position.

4. Bind the position buffer to the ARRAY_BUFFER target again.
5. Specify the layout of the vertex attribute relative to the array.
6. Enable the position vertex attribute using the shader location from step 3.
7. Repeat steps 3-6 for the color vertex attribute.
8. Unbind the new VAO to prevent accidental modification.
9. Add a call to **createVertexArrayObjects** after **setUniformVariables** in **setup**.

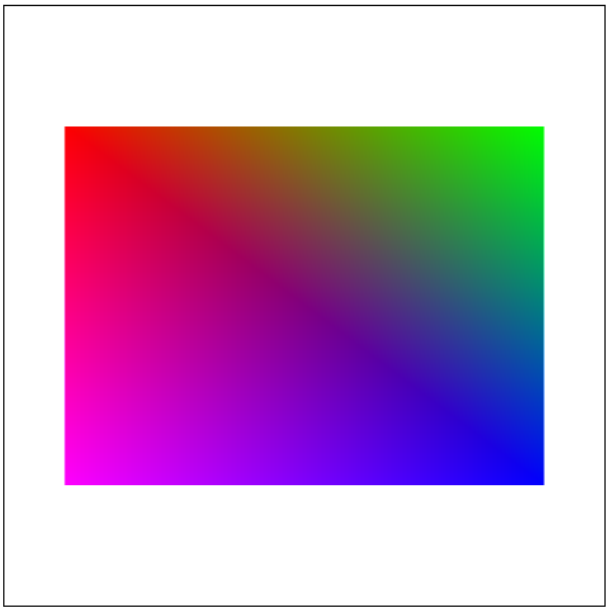
Part 8: Rendering

1. In the **render** function, clear the screen using **COLOR_BUFFER_BIT**.
2. Set the rendering state to use the shader program that was created earlier.
3. Bind the vertex array object created earlier.
4. Draw 6 vertices using the **TRIANGLES** mode.
5. To call the render function more than once, pass the function as a parameter to **requestAnimationFrame**. This will call it repeatedly.
6. Add a call to **render** after **createVertexArrayObjects** inside **setup**.

Final Result

WebGL and GLSL Tutorial

Canvas



Message Box

```
[msg]: WebGL initialized.
[msg]: Created buffers.
[msg]: Shader files loaded.
[msg]: Shader program compiled successfully.
[msg]: Set uniform variables.
[msg]: Created VAOs.
```

Additional Information

1. Event listeners were added to the canvas rather than the window (as in the previous tutorial) in order to only capture keyboard and mouse events relevant to the area of the WebGL viewport.
2. You will need to take into account the CSS styling and transformations between the scene and the viewport when processing the mouse position. You may find relevant information in Chapter 3 of the textbook.
3. This tutorial does not handle window resize events. For example, you may notice that stretching the window will not adjust the resolution of your scene. You will need to adjust the viewport and any other code which is dependent on the canvas size within a handler function for this event.
4. We are using WebGL2 which provides features such as vertex array objects and support for GLSL 3.00 ES.

Textbook Chapters

Textbook: Interactive Computer Graphics: A Top-Down Approach with WebGL

1. Chapter 3: Interaction and Animation, which also covers topics such as window events and position input.
2. Chapter 4: Geometric Objects and Transformations, which also discusses linear algebra, homogeneous coordinates, coordinate frames, row-major vs column-major representations, and how to draw a spinning cube in WebGL.
3. Chapter 5: Viewing, which discusses camera positioning and perspective projection matrices.

Additional Resources

1. WebGL 2.0 Quick Reference
2. WebGL 2.0 Specification
3. MDN WebGL Documentation
4. Khronos WebGL Overview
5. WebGL Wiki
6. OpenGL ES Shading Language Manual
7. Core Language (GLSL) OpenGL Wiki
    a. Vertex Shaders
    b. Fragment Shaders