CMPT 361 Graphics Programming Lab

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 JavaScript and WebGL tutorials for further information.**

Project Setup

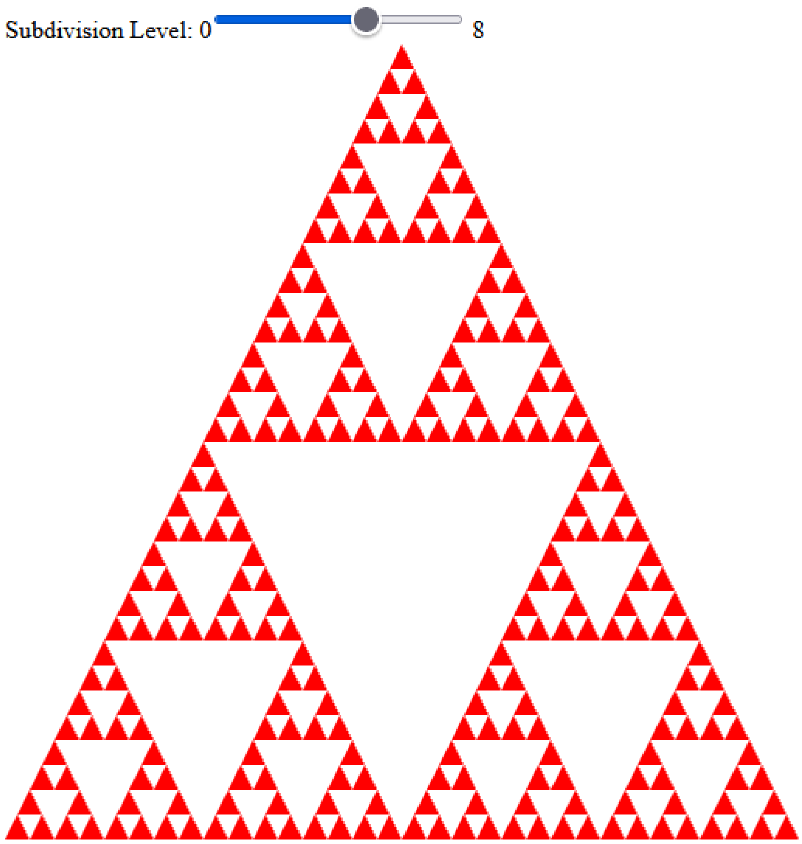
1. Extract the tutorial files into a new folder for your project.
2. Start a local web server from your project folder, either by using an editor extension or other methods such as the provided Python script.

Part 1: 2D Sierpinski Gasket

This example is based on the Gasket program in Chapter 2.8 of the textbook, which contains more detailed code explanations.

1. Inside the **init** function:
    - 1.1. Call the **initializeContext** function.
        - 1.1.1. This uses helper functions provided by the Angel library.
    - 1.2. Use the **initShaders** helper to load the GLSL shader sources from the **“vertex-shader”** and **“fragment-shader”** HTML script elements.
        - 1.2.1. Note: These shaders are written in GLSL 1.00.
    - 1.3. Use the shader program produced by the previous step.
    - 1.4. Create a new vertex buffer to store the data for the gasket.
    - 1.5. Bind the vertex buffer to **ARRAY_BUFFER**.
    - 1.6. Use the **bufferData** function to allocate memory which can contain up to $3^{max\ subdivisions\ +1}$ vertices. The usage can be set to **DYNAMIC_DRAW** since the vertex data changes.
        - 1.6.1. Note: Each subdivision divides each triangle into 3 new triangles.
    - 1.7. Set the vertex attribute pointer for the position vertex attribute.
    - 1.8. Enable the vertex position attribute.
    - 1.9. Add a call to **updateGasket** for the initial subdivision level of 0.
    - 1.10. Make the slider update the gasket.
2. Inside the **updateGasket** function:
    - 2.1. Clear the **points** array.
    - 2.2. Copy the updated points into the position vertex buffer using **bufferSubData**. Make sure to **flatten** the points.
3. Inside the **divideTriangle** function:
    - 3.1. Push a triangle to the list of points.
    - 3.2. Compute the points which bisect lines ab, ac, and bc, using the **mix** function.
    - 3.3. Decrement the subdivision count.
    - 3.4. Divide the triangle into 3 new triangles.
4. Inside the **render** function:
    - 4.1. Clear the screen.
    - 4.2. Draw the correct number of points.

4.3. Call requestAnimationFrame to update the next frame.
4.4. Optional: Enable face culling to understand the effect of the winding order.



Part 2: Drawing a Cube

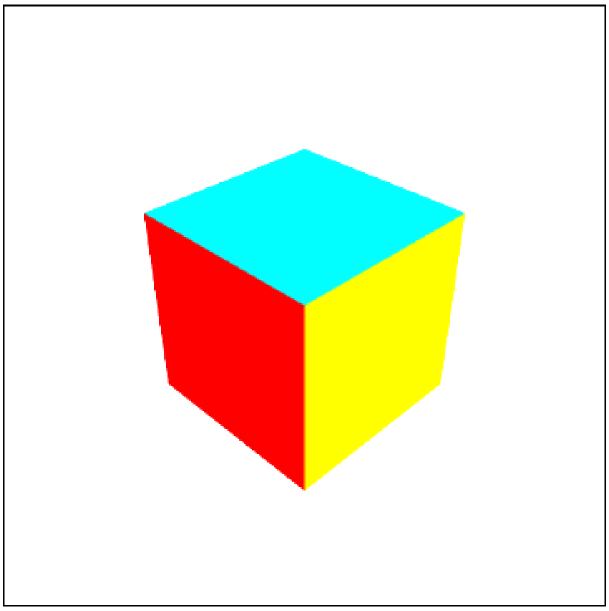
We will render a cube with a perspective projection and make it spin when the user clicks. This is based on the cube example in Chapter 4 of the textbook, which describes various geometric transformations.

1. Inside the **initializeContext** function:
    1.1. Enable depth testing.
2. Inside the **setup** function:
    2.1. Assign the initial values of **angle** and **angularSpeed**.
3. Inside the **setUniformVariables** function:
    3.1. Create a rotation matrix using the angle.
        3.1.1. You will need to use a function from the provided **MV.js** to generate the rotation matrix.
    3.2. Define a vec3 eye location for your camera.
    3.3. Define a vec3 target position which your camera will look at.
    3.4. Define a vec3 that specifies which direction is up.
    3.5. Create a view matrix using these 3 vectors.
        3.5.1. You may use the **lookAt** function.
    3.6. Calculate the aspect ratio using the canvas width and height.

    3.7. Create a projection matrix using the **perspective** function.
    3.8. Calculate the new transformation matrix by multiplying the projection, view, and rotation matrices in the correct order with the **mult** function.
    3.9. Update the value of the matrix in the shader.
4. Inside the **updateAngle** function:
    4.1. Initialize the value of the previousTimestamp variable the first time the function is called.
    4.2. Calculate the change in time in seconds.
    4.3. Update the angle using angularSpeed and the change in time.
        4.3.1. Note: You may have to make the angle wrap around from 360 to 0 in order to avoid floating point errors.
    4.4. Decrease angularSpeed using the change in time.
    4.5. Update the value of the previous timestamp.
5. Inside the **render** function:
    5.1. Clear both the color and depth buffers.
    5.2. Call **updateAngle** before drawing.
    5.3. Also call **setUniformVariables** right after **updateAngle**.
    5.4. Draw the correct number of vertices using the **TRIANGLES** mode.
6. Inside the **setEventListeners** function:
    6.1. Increase the rate of rotation when the user clicks.

Drawing a Cube

Canvas



Additional Information

1. The Angel library provided by the textbook authors, along with some useful linear algebra functions, are available to download from here.
    a. By default, the setupWebGL function creates a WebGL 1 context. You can enable WebGL 2 by adding “webgl2” to the start of the list on line 131 of webgl-utils.js.
2. Make sure that **requestAnimationFrame** queues the initial call to your rendering function to ensure the correct timestamp is passed in.
3. Shader uniform variables can be fixed-size arrays. For example:
    a. uniform mat4 matrices[4];
    b. The string used when getting the location of the second matrix would be “matrices[1]”.
    c. You may need an additional uniform variable to tell your shader how many array elements are actually filled in.
4. Please check the extensions and resources mentioned in the JavaScript and WebGL tutorials.

Textbook Chapters

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

1. Chapter 2: Graphics Programming, which covers the Sierpinski Gasket example in more detail.
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 positioning the camera and perspective projections.
4. Chapter 7: Discrete Techniques, which discusses buffers and texture mapping.