CSE 470 Homework #1: Kaleidoscope

Big Picture:

Create a 2D geometric object, called a prototype, which is centered at the origin and defined by solid-colored triangles. Your program will display the prototype and ten instances of it, located on a circle about the prototype. The prototype will rotate clockwise and the instances will rotate counterclockwise. As the objects rotate, they will change color based on function that uses their location as input. The result is what you might call a simple, simulated kaleidoscope.

Learning Objectives:

1. Understand and modify an existing webGL program.
2. Understand the structure and interactions between HTML, JS, webGL, GLSL.
3. Create a geometric object defined by triangles.
4. Learn how to create an animated graphics program.
5. Set color based on the dynamics of the program.
6. Learn to send data and other variables to the vertex shader.
7. Experiment with transformations in the vertex shader.
8. Create effective debugging code and comments.

Specifications:

1. Start with the simpleSquare demo program.
2. Create an interesting object. Points will be rewarded for creativity.
    a. The object must have at least 20 vertices. (Note: 2d coordinate locations may appear more than once in the vertex list.)
    b. The object must be centered at the origin and be defined in clip coordinates.
    c. The object must be defined by solid-colored triangles and each triangle must be a unique color. (This means that each vertex of a triangle is assigned the same color.)
    d. The vertex colors must be sent to the vertex shader.
3. Replicate the object at least ten times on a circle about the original (prototype) design.
    a. In the JS code, create a list of centers for the instances of the prototype. Send each of these centers to the vertex shader as a uniform variable (when drawing).
    b. In the vertex shader, translate the prototype to the instance position.
    c. It is okay for the objects to intersect a bit, but the should be distinguishable. Just make it look nice!
4. Rotate the instances about the rotating prototype.
    a. Create a rotation variable, as done in the rotatingSquare demo.
    b. The prototype should rotate clockwise.
    c. The instances should rotate counterclockwise about the prototype.
    d. Choose a rotation speed so that the objects are visible and no negative rendering artifacts appear.

    e. FYI: The tricky part here is figuring out how you will make the prototype behave differently than the instances.
5. Modify the colors of the vertices based on their location in clip coordinates.
    a. Create a color function in the vertex shader that modifies one color component based on the x-location and modifies another color component based on the y-location.
    b. Example: suppose the x-coordinate will determine how much of the original red component is retained. At x=0, 100% of the red is retained; at x=-1 and x=+1, 0% of the red is retained.
    c. Lots of different options here. (I will demonstrate linear, quadratic, and cosine functions in class.) However, you do need to implement something that is observable, understandable, and not just random colors.
    d. Include a comment next to the code that explains what you have implemented.

Tips:

1. Get started early.
2. If you get stuck, don’t be afraid to ask for help.
3. Make sure you understand simpleSquare, rotatingSquare, and review the class notes.
4. Make a plan of action before writing any code.
    a. Recommended order: geometry creation, create instances, rotate, color change.
5. Work on one task at a time and thoroughly test it before moving to the next task.
6. PRINT debugging code! Avoid the silent program. (Unfortunately printing to the “console” in the shaders is *not* possible. Tip: You can set a color if a certain condition exists.)
7. Make a paper sketch of your object and label the vertices before coding them.
    a. Note: You can build the object with parametric equations (e.g., a circle).

Submission Instructions:

1. Turn in your assignment on Canvas.
2. All assets you created must be in one zipped file. Do not include the Common folder. The references to the files in the Common folder *must* be as done in the class examples.
3. Name your zip file: LastnameFirstInitial_HW1.zip.
4. We will grade the last submission only.
5. Review the late submission policy in the Syllabus.