Lab 3 - OpenGL Primitives

You may work in pairs on this assignment. To receive credit, demonstrate your completed program to me by Feb 13th.

In this assignment, you'll be learning about OpenGL primitives: points, lines and triangles. Your assignment is to adapt the provided code to detect mouse clicks which will create points on the screen. The provided code allows for storage of just three points. Change the code to be able to store and display any number of points. Pressing the spacebar should cycle through the different OpenGL draw modes to better understand how primitives are formed. Pressing the c key should clear the points so you can begin clicking anew. Pressing the w key will toggle whether primitives are outlined or filled in (this will come in handy to better view the individual triangles created from a triangle strip or triangle fan). To achieve this behavior, follow the parts below.

Part 1 - Orthographic Projection

In the previous lab, we used a function to convert our coordinates before passing them to OpenGL. In this lab, we'll instead use a projection matrix to do the conversion in our vertex shader. In the provided code, the shaders have been broken out into vert.glsl and frag.glsl. Modify vert.glsl to have a uniform mat4 variable for your projection matrix. Use it to transform each vertex into normalized device coordinates. In your program's OpenGL initialization code, look up the uniform location using glGetUniformLocation (similar to how we've done so with glGetAttributeLocation). You'll need access to the location in your program's resize method, so store it in a private class variable. In your resize method, create a 4x4 orthographic matrix and bind it to your shader's uniform variable. If done correctly, your program should display up to three points after clicking on the window.

Part 2 - Point Storage

Modify how points are stored in your program to allow for any number of points to be created. You are welcome to allocate (and reallocate when necessary) your own memory, but I recommend using the Standard Template Library's vector class to handle memory allocation for you.

Part 3 - Draw Mode

Modify your program to cycle to the next draw mode every time spacebar is pressed. Each of the following draw modes should be cycled through: GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN. Experiment with different vertex positions to understand how the vertex ordering affects the construction of primitives.

Things To Notice

GL_TRIANGLES VS GL_TRIANGLE_STRIP and GL_TRIANGLE_FAN

Notice how easy it is to draw overlapping triangles with triangle strips and triangle fans. Working with triangle strips and fans can be a little trickier than just triangles, but the benefit of triangle strips and fans is their compact data format. Triangle strips and fans will have N triangles for every N+2 points, whereas individual triangles requires 3N points. With the speed of modern graphics processors today (and the extra benefit of element arrays, which we'll get to in later labs), the flexibility of specifying individual triangles is often a good trade off for the hit in performance. In certain cases, though, triangle strips or fans are well suited for the task and can be more performant. Triangle strips can be good for dynamically generated tail geometry (like a moving comet's tail, or when leaving a trail of where you swiped on the screen). Triangle fans are good for representing 2D convex geometry, which is often used in 2D rigid body solvers.

Orthographic projection

Notice how when resizing the window, nothing gets scaled. That's because we're specifying our coordinates in window coordinates (also called screen space) and our projection matrix is changing on resize. Now when we size the window smaller than before, we cut off some of the geometry we've drawn. What if we always wanted our geometry to fit the width of the window? Think about how you can use the aspect ratio (the ratio of the width to the height) of our window, to keep our geometry uniformly scaled, while still fitting to the window the best we can.

Recommended Reading

Wikipedia: Orthographic Projection
std::vector documentation
glm - OpenGL Mathematics Library
OpenGL Programming Guide, 8th Edition - Chapter 3, pages 85-90