Introduction to Computer Graphics

January 30, 2019 - Lecture 3

1 General

All commands to run the code shown today are in Linux. Because the professor uses Linux during class it is highly encouraged to learn Linux and to install a Linux machine if necessary - the recommended one is Ubuntu 16.04.

You can run the OpenGL examples (that these notes will briefly cover) on the ilab machines available to us.

2 OpenGL Programming

Today's class lecture was focused on OpenGL programming and how to follow the OpenGL pipeline.

Here is the link that thoroughly goes into detail about the OpenGL pipeline:

http://duriansoftware.com/joe/An-intro-to-modern-OpenGL.-Chapter-1:-The-Graphics-Pipeline.html

The OpenGL graphics pipeline basics:

- The host program fills OpenGL-managed memory buffers with arrays of vertices
- These vertices which are passed to the vertex shader are projected into screen space, assembled into triangles, and rasterized into pixel-sized fragments
- The fragments are assigned color values and drawn to the framebuffer - these are known as fragment shaders which decide what color each pixel will receive
- The process of rasterization in the GL pipeline rasterizes these vertices and shaders on the computer screen
- Blending operations to get the final OpenGL image on your screen.

There are various functions associated with OpenGL that come with various graphics libraries. One library is known as glew.h that should be installed for compiling OpenGL code.

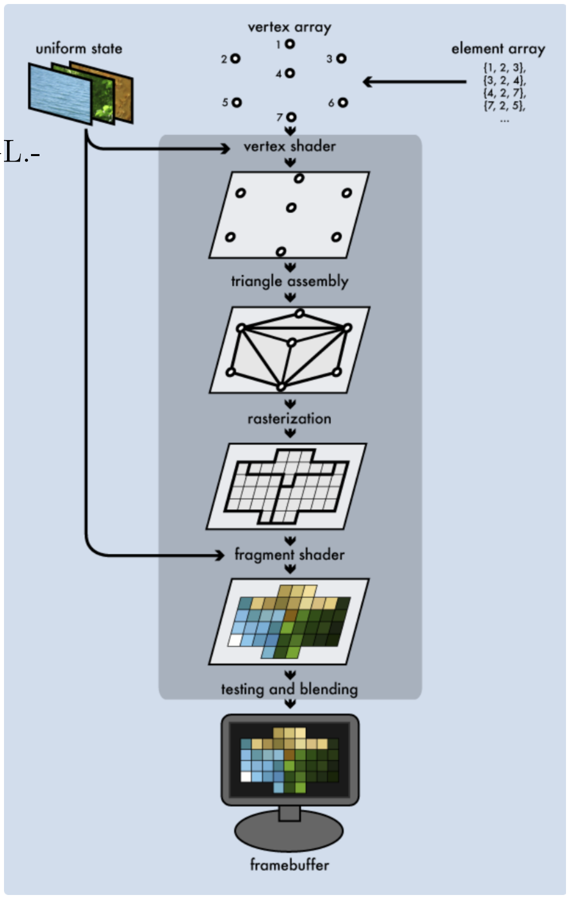

Figure 1: OpenGL pipeline

2.1 First OpenGL script

The most basic OpenGL script shown in class was opening a window that had a turquoise color hue to it.

A few key points about the code that were mentioned:

- To initialize glew.h start with the line of code glfwInit()
- By setting the Resizable flag to true, you are able to resize the OpenGL window
- The main loop in the code is constantly checking whether you want to close the window or not

C++ and OOP:

- Continue to read and familiarize yourself with C++ since most graphics code and code that the professor is sharing is written in C++
- The main obvious difference between C++ and java is that C++ has pointers. Pointers have control over explicit memory management but you also need to know how to deal with memory when coding in C++

OpenGL 3.0 is making more efficient use of the modern GPU. While this is a positive step forward, many simple operations, such as showing a simple window, are written in more than a few lines of code.

2.2 Second OpenGL script

Next we have a non trivial code example that shows two triangles in the opened window.

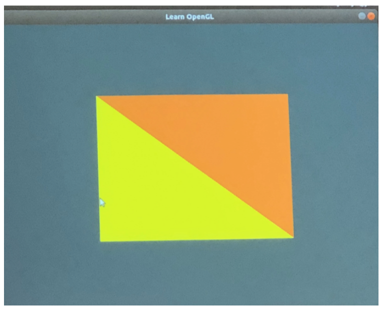

Figure 2: Two shaders shown as two triangles

Two main GL types:

- GLfloat is gl version of floating point
- GLuint is gl version of int

Often OpenGL programmers will set the window according to the width and length so that the image (for example the two triangle images) is resizable as the window grows or shrinks

Vertex shaders can be defined so that they understand the coordinates/position of a vertex. Fragment shaders receive the varying values output by the vertex shader and interpolated by the rasterizer as inputs. It outputs color and depth values that then get drawn into the framebuffer. We only have one triangle associated with each framebuffer but you could put more.

Once we specify the shader source, then we must execute the command glCompileShader which is written in the OpenGL shading language.

Debugging - Always put in error messages or break points for potential invalid or wrong code to make sure that your commands are working and doing what you want.

Two notes on Shader Programs:

- Fragment shaders are associated with whatever fragment you want - it can take a list of elements but the main property is that all elements with fragment shader have the same color ie. two fragment shaders give two different colors
- To create shader programs you attach each vertex shader and each fragment shader specified

Next, we need to create vertex array objects to bind them to two shader programs

Setting vertex attribute pointers - glVertexAttribPointer - define an array of generic vertex attribute data - and when going from vertex to vertex you need to skip three floats.

Lastly, make sure to clear the main window before the coding the main while loop (which, to recap, is constantly running and checking if you have exited the window).

https://www.khronos.org/opengl/ - website to go to for all the new OpenGL functions and information since these are the people that created it

2.3 Third OpenGL script

Here we go over a more complicated OpenGL script that shows one triangle with a gradient of colors. Here we are doing more work in the vertex shader and less work in the fragment shader. Each vertex of the triangle has a certain color and then those colors are blended together to create a gradient look.

Small hint: always try to minimize copy and pasting code so that you can minimize speed and maximize efficiency of the code - with C++ you can define a class and we can use the object oriented benefits of the programming language.

In this example we create a Shader object which is initialized in main.cpp:

- The file name is Shader.h
- the object initialization of a Shader object takes in two parameters:
- shader.vs - the vertex shader
- shader.frag - the fragment shader

In the constructor of Shader.h, we therefore pass a path to the vertex and fragment shader which initializes everything in the class. Shader.h is an abstraction for what a shader program needs to be - it takes together the parameters of the vertex and fragment shaders and makes it into a shader program and then deletes the vertex and fragment shaders since they are now linked into our program and no longer necessary.

In the main C++ file - the window initialization is the same as before and we just initialize a Shader object. The difference between the last example for vertex shader creation is that we have 6 elements (3 elements for the vertex position and 3 elements for the vertex color) that define each vertex position in the vertex array GLfloat vertices[]. You need to be careful when defining the vertex array since there are 6 elements per vertex instead of 3.

We take a look at the glVertexAttribPointer function. Now the stride in the GPU is 6 since each vertex is 6 elements long and not 3 anymore - so we skip 6 floating point numbers to get to the next position.

And lastly, the glEnableVertexAttribArray(0) function gives the specified vertex position (0 in this case).