Project 2 - Transformations

In this assignment, I understood how to load an obj file, save its data into a vertex buffer and display it on the window.

What you have implemented & How to use your implementation:

Step 1: Vertex buffers:

To display an obj file on the window, we have to load an obj file and save it into a vertex buffer. I use the cyTriMesh code which includes functions to load and access vertex data from the obj file. We then create a vertex buffer, save the vertex data into the buffer.

Please check my Mesh class to see how I load an obj file, create a buffer and render the obj file

Step 2: GLSL shaders:

The next step is having some shaders working with the vertices. I use the cyGL code to handle shaders. The cyGL provides GLSLProgram allows us to load a vertex shader and fragment shader. In the project, we just keep it as simple as possible. I just convert the vertices with MVP transformations (see below) and set the vertices with a single color. Also, we can use the F6 key to recompile the GLSL shaders, or the F7 to switch to another shader.

Please check my Effect class to see how I compile the shaders.

Step 3: Transformations:

The last step is to convert our obj file (teapot) coordinate into a projected window. This transformation is called MVP Transformations (model, view, and projection).

Please check my implementation of MVP in InitGL() function.

Model:

For convenience, I just set my model matrix at (0, 0, 0). The only translation I did is to center our teapot using its bounding box.

View:

View Matrix describes where is my camera and where does the camera look at. In my assignment, I put my camera at (0, -30, 50) and look at the (0, 0, 0), where is the teapot's position. My view matrix implementation is like

View.SetView(cy::Point3<float>(0, -30, 50), cy::Point3<float>(0, 0, 0), cy::Point3<float>(0, 1, 0))

Projection:

We can set up the projection matrix with parameters: field of view, aspect ratio, near clipping plane and far clipping plane. My implementation for Projection Matrix is

Projection.SetPerspective(1, 1.0f, 0.1f, 100.0f)

Finally, the MAP transformations would be like:

MVP = Projection * View * Model

The MVP transformations will be sent into vertex shader to transform all vertices. In my project, we can use left click to rotate the camera and right click to zoom in and zoom out the camera.

What you could not implement:

I didn't implement orthogonal transformation because it is optional.

Additional functionalities beyond project requirements:

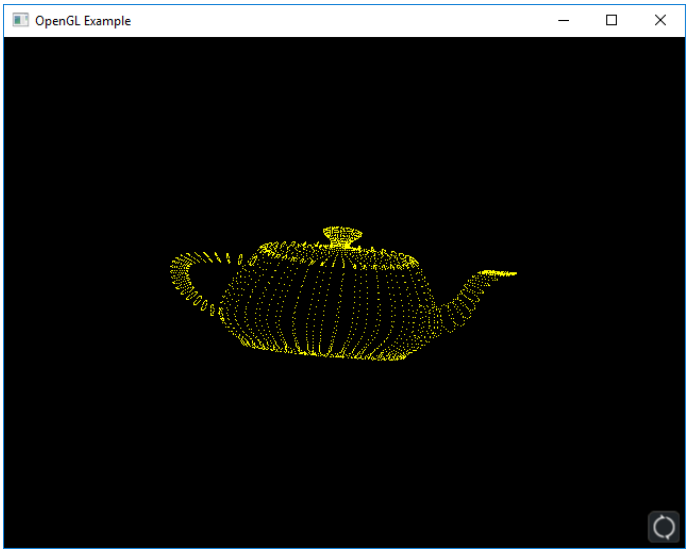
None

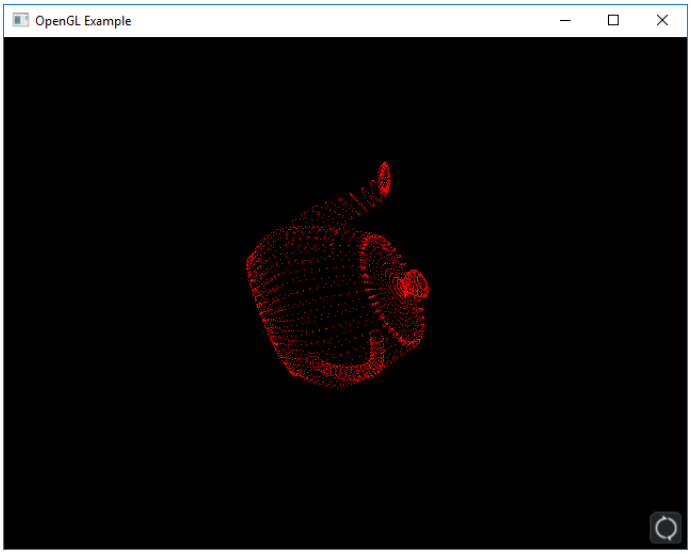
What operating system and compiler you used:

I work and compile my code in Visual Studio 2017 in Windows 10.

External libraries and additional requirements to compile your project:

I use cyCodeBase to finish my project. This CodeBase provides many functionalities such as point3D, matrix4, obj file handler and shader handler to help my work. Also, I use GLEW library for initializing OpenGL extension functions. These libraries can be found in my Externals folder.





Ref:

http://glew.sourceforge.net/

http://www.cemyuksel.com/cyCodeBase/index.html

http://www.opengl-tutorial.org/