

Mark Eilers

CS 499: Computer Science Capstone

3-2 Milestone Two: Enhancement One: Software Design and Engineering

July 19, 2024

### 3-2 Milestone Two: Enhancement One: Software Design and Engineering

For the first section of the ePortfolio, Software Design and Engineering, I have chosen to modify my 5-3 Texturing a Pyramid assignment from my CS 330 Computer Graphics and Visualization course at Southern New Hampshire University. In the original assignment, we were given a 2D rendering of a triangle that we were then asked to provide a simple texture to. For this enhancement, I have chosen to create the 3D shape using Visual Studio as my integrated development environment (IDE). To create the 2D and 3D rendering using this specific IDE, I used OpenGL, which is an application programming interface (API). To use OpenGL, a coding language is required. For this specific project, both original and enhanced, I have used C++ as my coding language.

I chose this assignment as my first artifact for several reasons. I greatly enjoyed my time in CS 330 Computer Graphics and Visualization course. Writing code that then generated a virtual space to showcase the generated object was very satisfying and rewarding. Additionally, we had several tutorials that helped hone in our skills for the final project. In the assignment I chose to enhance, it was early in the development process for that course. I wanted to apply my overall skills in a course I enjoyed bringing a simple assignment up to the standards of the final project. I determined this was the best project to start my enhancements because I know that OpenGL is still widely used as an API and is still supported by current hardware. Since many companies use OpenGL to show my enhancement within this API, I will have a great project to showcase my skills to companies and future employers. Additionally, I show in this enhancement my impressive amount of abilities using C++ programming language as well as the

OpenGL API. Overall to enhance this original artifact, I have transformed the original 2D image, with a simple carpet texture into a 3D pyramid with stone. Additionally, in the enhancement, I have incorporated both mouse and keyboard controls that allow the user to move around the virtual space, and platform for the pyramid to sit on top of. I have also included a lighting source to improve the overall look of the scene. For the remaining portion of this assignment, I will be referring to the screenshots at the end of this document. Each screen shot is labeled as a figure and will be referred to as such. Additionally, each screenshot provides a short description to identify what the image is showing.

To start my enhancement, I needed to ensure the proper libraries were incorporated into the project. As seen in **Figure 1**, several different libraries were included such as camera controls. This was done to make sure that everything the project needed to function correctly was built-in at the start of the project. Once I updated the necessary includes, I then updated the texture from a simple carpet to actual stone. To accomplish this, I search for a PNG image that I liked and included it into a folder titled resources. From there, I changed the location of the image from carpet to stone. This can be seen in **Figure 2**. Once I had updated the textures, I wanted to work on the camera controls. In **Figure 3** and **Figure 4**, I updated the code to include switch cases for the mouse and if statement for keyboard controls. In the enhanced artifact, the mouse and keyboard controls allow the user to move all around the virtual scene to see everything that is included. This provides and interactive component to the project that makes it more appealing to the user.

Once I had completed some of the more fundamental portions of the code, movement and texture, it was time to update the vertices for the pyramid. This is where some problem became apparent. I wanted to create a platform for the pyramid to sit on, so I started off by creating the

vertices for a “table” from there I then had to update the pyramid itself. The original pyramid was a 2D triangle. I needed to update this to a fully 3D shape as seen in **Figure 5**. The issue was finding vertices that sat in the middle of the table without either going through the table or floating above it. To accomplish this task, I started by building a square base to the pyramid that sat directly on the surface of the table. With this completed, I then had a starting point from the remaining side that I could then build from. Additionally, creating the different sides so that they were all filled in, and looked appropriate was a challenge. To overcome this challenge, I started by building one side of the pyramid. After several rounds of updating, testing, and changing, I arrived at a shape that I was happy with. From there I used some math concepts to calculate the vertices of the remaining shapes. Keeping in mind that the pyramid was in the center of the image, I could then use simple addition and subtraction to calculate the other sides of the shape. This involved knowing exactly where the center was, and using mathematics on the vertices to place the objects where I saw best fit.

After the pyramid was made, I realized that the lighting was off. The object was more enhanced than before, but I decided to then go back and change some of the coding to add more lighting. In **Figure 6** and **Figure 7** code was implemented to create a light source above and to the front of the pyramid. This gave the virtual image more depth as the front side of the object was illuminated by the light source while the back side of the object was darker. In the end, **Figure 8** and **Figure 9** show the pyramid before and after the enhancement. Before the enhancement, the pyramid was a 2D shape that had a carpeted texture. The object was floating in virtual space with poor lighting and no camera controls. The enhanced project created a 3D pyramid that is sitting on the surface. The pyramid is illuminated from the front to add depth. Additional camera and keyboard controls were added to incorporate an interactive component

into the design. This allows users to move around the scene to get a better understanding of the 3D space.

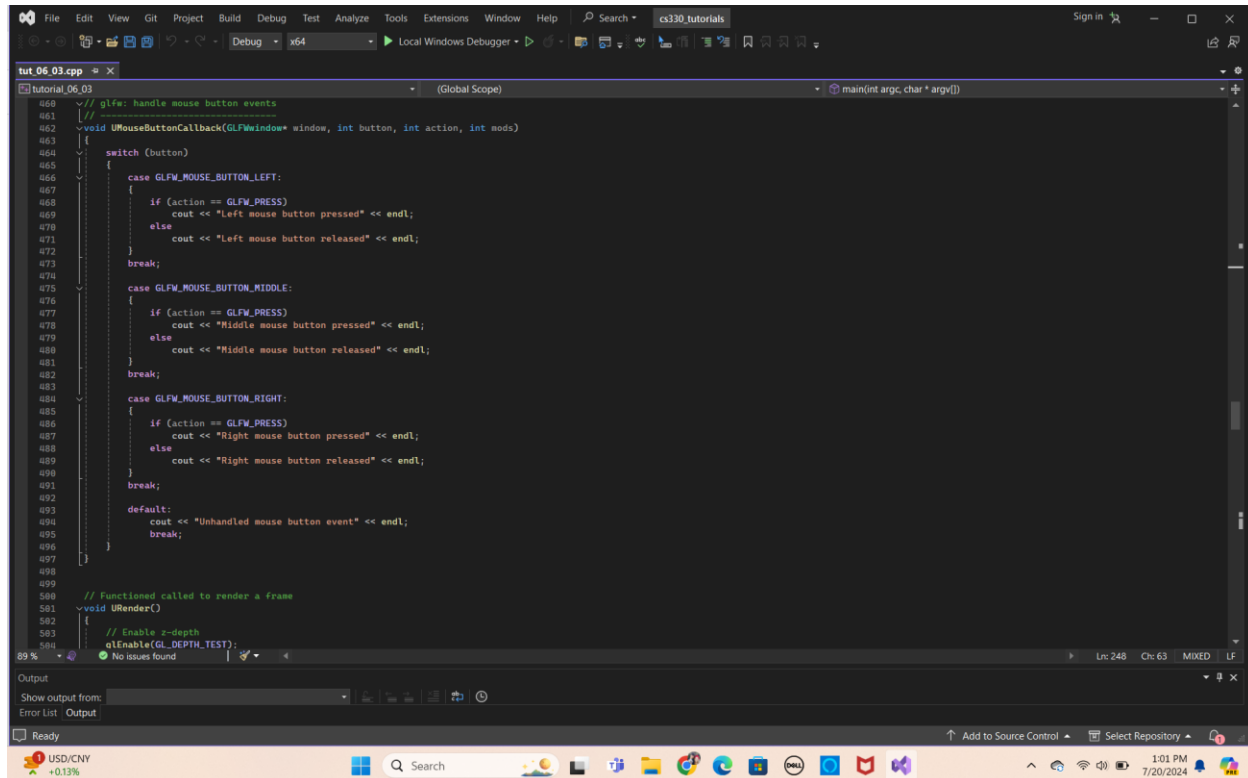
From the artifacts original 2D coding to the enhanced and improved 3D with mouse and keyboard controls, as well as additional lighting and improved texture, I have showcased my knowledge and skills outlined in the course outcomes. I have effectively designed and evaluated computing solutions that solve a given problem using algorithmic principles. I have also demonstrated an understanding for computer science practices and standards appropriate to the solutions. I have effectively managed the trade-offs involved in my design choices, which falls under the algorithm section of the course. This is due to my determining lighting calculations by calculating a “surface normal”, as well as calculating the vertices placements to make a functioning base and pyramid that are 3D as opposed to 2D. Additionally, the outcome of this project also falls under data structures. This is primarily due to the data that is stored. For example, vectors. The algorithms in this project will use vector data structure to solve which coordinate is needed to position the light in the correct area. Finally, through my ability to follow appropriate programming standards in C++, proper naming conventions and in-line comments, I have demonstrated my ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals. By the end of this project enhancement, I have code that is aligned to C++ standards, error free, and is easily readable due to the in-line comments I have provided.

```
1 // Mark Eilers
2 // CS 499 Computer Science Capstone
3 // 3-2 Milestone Two: Enhancement One: Software Design and Engineering
4 // July 18, 2024
5
6 #include <iostream> // cout, cerr
7 #include <cstdlib> // EXIT_FAILURE
8 #include <GL/glew.h> // GLW library
9 #include <GLFW/glfw3.h> // GLFW library
10 #define STB_IMAGE_IMPLEMENTATION
11 #include <stb_image.h> // Image loading Utility functions
12
13 // GLM Math Header inclusions
14 #include <glm/glm.hpp>
15 #include <glm/gtx/transform.hpp>
16 #include <glm/gtc/type_ptr.hpp>
17
18 #include <learnopengl/camera.h> // Camera class
19
20 using namespace std; // Standard namespace
21
22 // Shader program Macro/
23 #ifndef GLSL
24 #define GLSL(Version, Source) "Version " #Version " core\n" #Source
25 #endif
26
27 // Unnamed namespace
28 namespace
29 {
30     const char* const WINDOW_TITLE = "Texturing a 3D Pyramid"; // Macro for window title
31
32     // Variables for window width and height
33     const int WINDOW_WIDTH = 800;
34     const int WINDOW_HEIGHT = 600;
35
36     // Stores the GL data relative to a given mesh
37     struct GLMesh
38     {
39         GLuint vao; // Handle for the vertex array object
40         GLuint vbo; // Handle for the vertex buffer object
41         GLuint nVertices; // Number of indices of the mesh
42     };
43
44     // Main GLFW window
45     GLFWwindow* mWindow = nullptr;
```

**Figure 1:** Screenshot shows the update to the necessary libraries to make the enhancement. Most notably, the camera function was included that will allow the enhancement to use camera controls.

```
232 int main(int argc, char* argv[])
233 {
234     if (!UInitialize(argc, argv, gWindow))
235         return EXIT_FAILURE;
236
237     // Create the mesh
238     UCreateMesh(gMesh); // Calls the function to create the Vertex Buffer Object
239
240     // Create the shader programs
241     if (!UCreateShaderProgram(cubeVertexShaderSource, pyramidFragmentShaderSource, gPyramidProgramId))
242         return EXIT_FAILURE;
243
244     if (!UCreateShaderProgram(lampVertexShaderSource, lampFragmentShaderSource, gLampProgramId))
245         return EXIT_FAILURE;
246
247     // Load texture
248     const char* texFilename = "../resources/textures/stoned.png";
249     if (!UCreateTexture(texFilename, gTextureId))
250     {
251         cout << "Failed to load texture " << texFilename << endl;
252         return EXIT_FAILURE;
253     }
254
255     // tell OpenGL for each sampler to which texture unit it belongs to (only has to be done once)
256     gUseProgram(gPyramidProgramId);
257     // We set the texture as texture unit 0
258     glUniform1i(glGetUniformLocation(gPyramidProgramId, "uTexture"), 0);
259
260     // Sets the background color of the window to black (it will be implicitly used by glClear)
261     glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
262
263     // render loop
264     while (!glfwWindowShouldClose(gWindow))
265     {
266         // per-frame timing
267         float currentFrame = glfwGetTime();
268         gDeltaTime = currentFrame - gLastFrame;
269         gLastFrame = currentFrame;
270
271         // Input
272         UProcessInput(gWindow);
273
274         // Render this frame
275     }
```

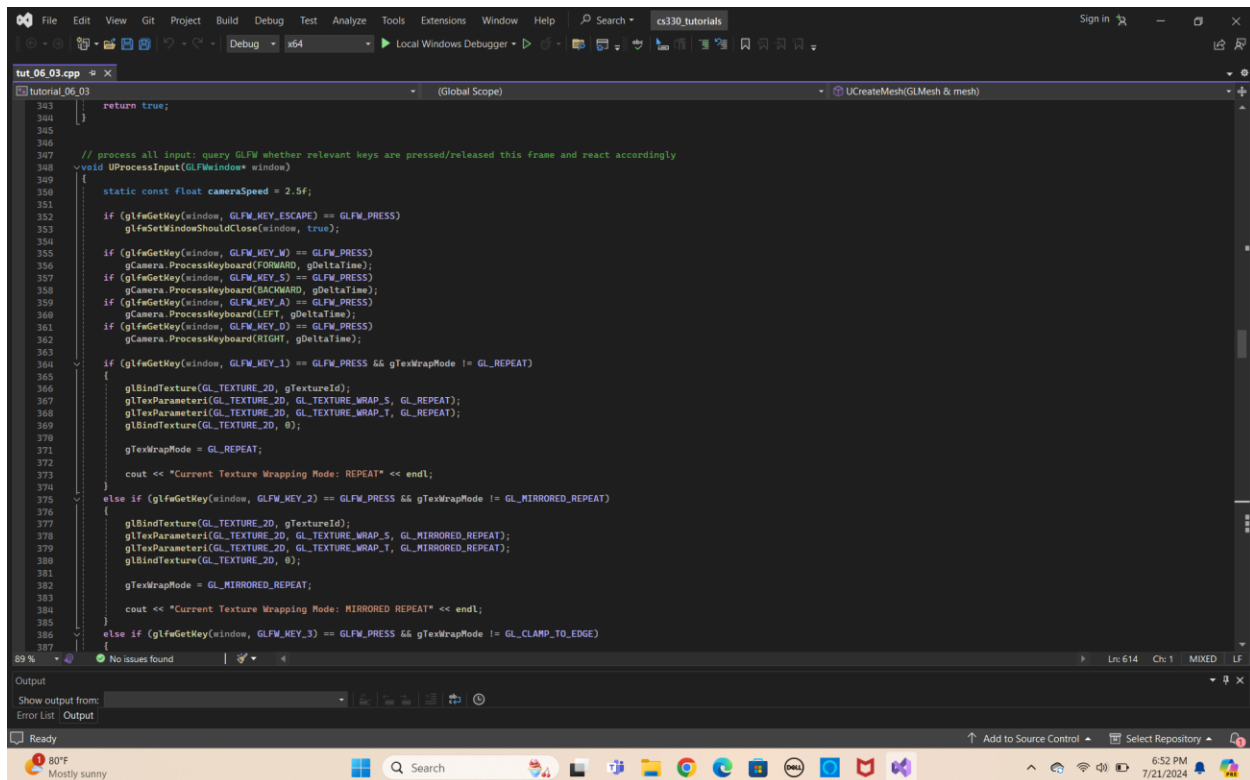
**Figure 2:** Screenshot shows the update on the texture to give the pyramid a more realistic texture. It was changed from a carpeted texture to a stone texture. This was done by finding a PNG image that was appropriate for the pyramid, including it in the resource folder and updating the code to access the stone image instead of the carpet.



The screenshot shows a C++ code editor with a dark theme. The code is for a GLFW application, specifically handling mouse button events. The code is organized into a switch statement that handles three mouse buttons: LEFT, MIDDLE, and RIGHT. Each button has two cases: GLFW\_PRESS and GLFW\_RELEASE. The code uses cout to print messages when a button is pressed or released. The code is located in a file named 'tut\_06\_03.cpp' and is being debugged using the 'Local Windows Debugger'. The code is as follows:

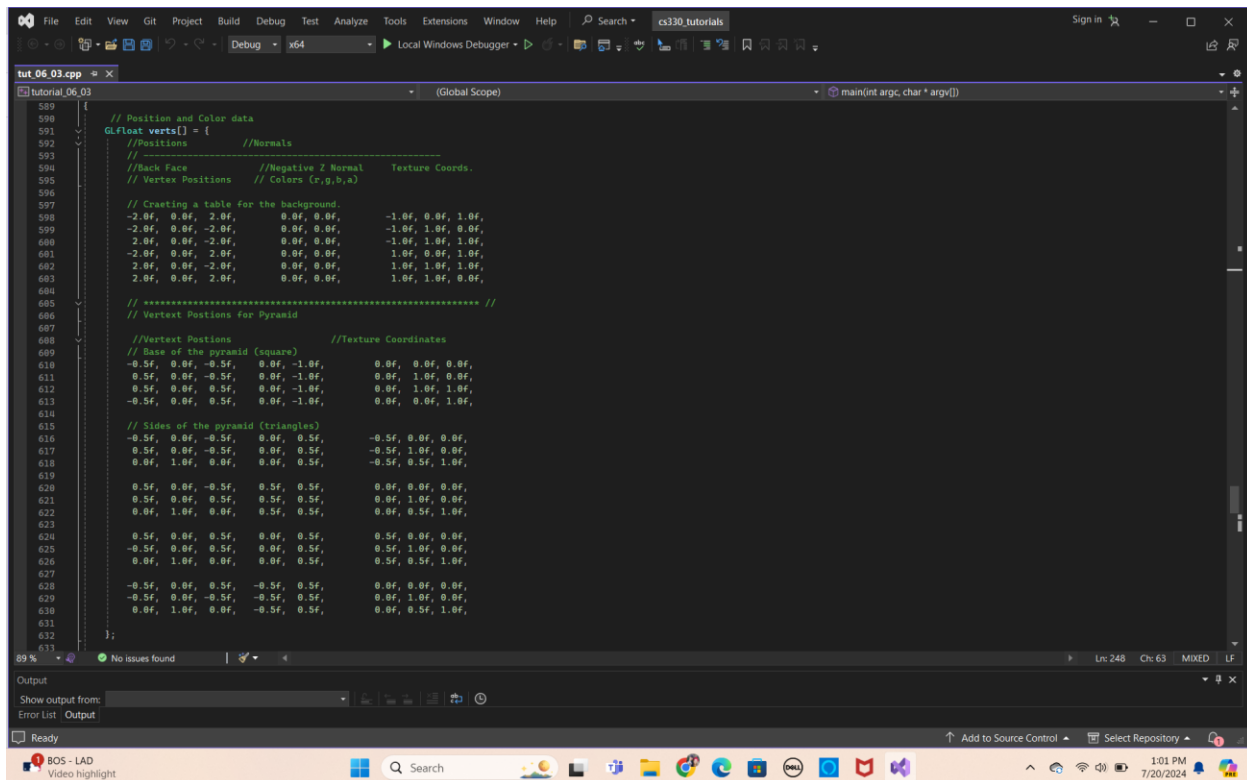
```
460 // glfw: handle mouse button events
461 //
462 void UMouseButtonCallback(GLFWwindow* window, int button, int action, int mods)
463 {
464     switch (button)
465     {
466         case GLFW_MOUSE_BUTTON_LEFT:
467         {
468             if (action == GLFW_PRESS)
469                 cout << "Left mouse button pressed" << endl;
470             else
471                 cout << "Left mouse button released" << endl;
472             break;
473         }
474         case GLFW_MOUSE_BUTTON_MIDDLE:
475         {
476             if (action == GLFW_PRESS)
477                 cout << "Middle mouse button pressed" << endl;
478             else
479                 cout << "Middle mouse button released" << endl;
480             break;
481         }
482         case GLFW_MOUSE_BUTTON_RIGHT:
483         {
484             if (action == GLFW_PRESS)
485                 cout << "Right mouse button pressed" << endl;
486             else
487                 cout << "Right mouse button released" << endl;
488             break;
489         }
490         default:
491             cout << "Unhandled mouse button event" << endl;
492             break;
493     }
494 }
495
496 // Function called to render a frame
497 void URender()
498 {
499     // Enable 2-depth
500     glEnable(GL_DEPTH_TEST);
501 }
```

**Figure 3:** Screenshot show the updated code for the mouse controls using switch controls. The code includes switch cases that utilize the mouse to allow the user to move up, down, right, left, and zoom into the scene.



```
343     return true;
344 }
345
346 // process all input: query GLFW whether relevant keys are pressed/released this frame and react accordingly
347 void UProcessInput(GLFWwindow* window)
348 {
349     static const float cameraSpeed = 2.5f;
350
351     if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
352         glfwSetWindowShouldClose(window, true);
353
354     if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
355         gCamera.ProcessKeyboard(FORWARD, gDeltaTime);
356     if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
357         gCamera.ProcessKeyboard(BACKWARD, gDeltaTime);
358     if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
359         gCamera.ProcessKeyboard(LEFT, gDeltaTime);
360     if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
361         gCamera.ProcessKeyboard(RIGHT, gDeltaTime);
362
363     if (glfwGetKey(window, GLFW_KEY_1) == GLFW_PRESS && gTexWrapMode != GL_REPEAT)
364     {
365         glBindTexture(GL_TEXTURE_2D, gTextureId);
366         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
367         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
368         glBindTexture(GL_TEXTURE_2D, 0);
369         gTexWrapMode = GL_REPEAT;
370     }
371     cout << "Current Texture Wrapping Mode: REPEAT" << endl;
372
373     else if (glfwGetKey(window, GLFW_KEY_2) == GLFW_PRESS && gTexWrapMode != GL_MIRRORED_REPEAT)
374     {
375         glBindTexture(GL_TEXTURE_2D, gTextureId);
376         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);
377         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);
378         glBindTexture(GL_TEXTURE_2D, 0);
379         gTexWrapMode = GL_MIRRORED_REPEAT;
380     }
381     cout << "Current Texture Wrapping Mode: MIRRORED REPEAT" << endl;
382
383     else if (glfwGetKey(window, GLFW_KEY_3) == GLFW_PRESS && gTexWrapMode != GL_CLAMP_TO_EDGE)
384     {
385         glBindTexture(GL_TEXTURE_2D, gTextureId);
386         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
387         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
388         glBindTexture(GL_TEXTURE_2D, 0);
389         gTexWrapMode = GL_CLAMP_TO_EDGE;
390     }
391     cout << "Current Texture Wrapping Mode: CLAMP TO EDGE" << endl;
392 }
```

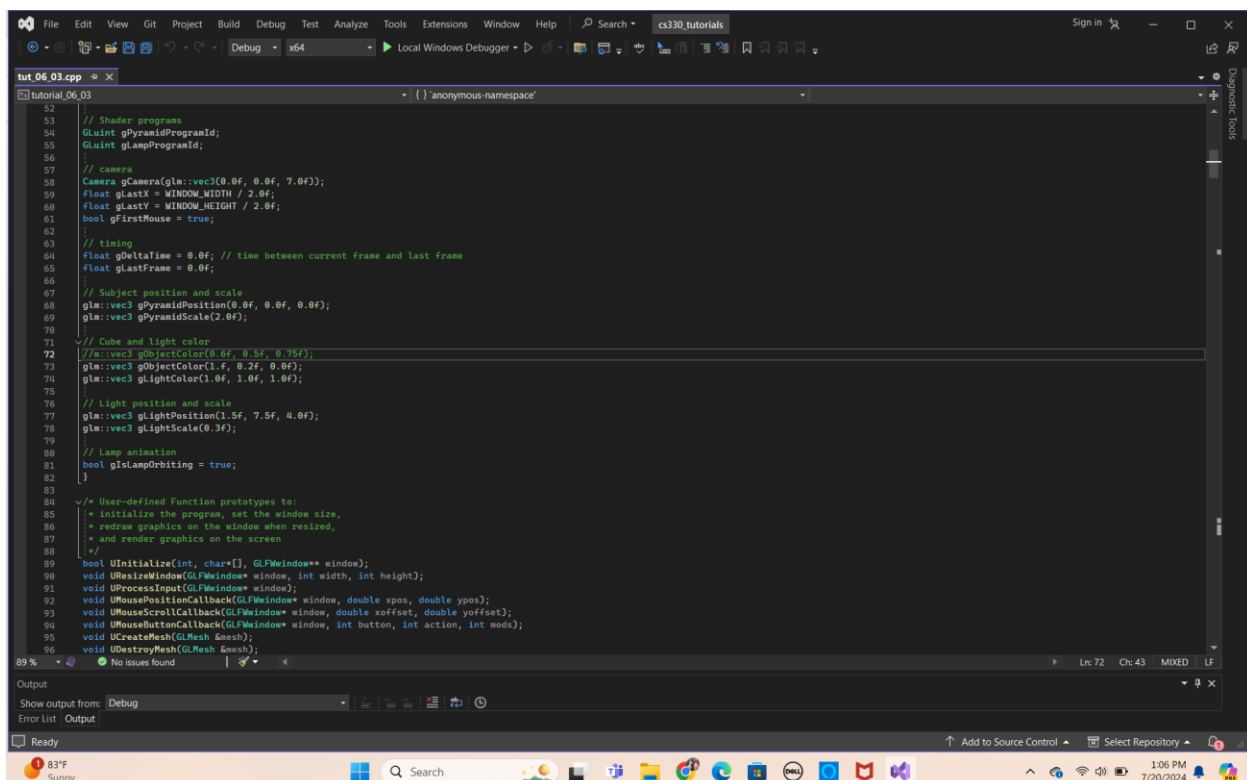
**Figure 4:** In this screenshot code was implemented for the keyboard commands. If statements were used and synced with the keyboard to execute a command when pressed. For example, if The W button was pressed on the keyboard, this would move the camera up. Likewise, A would move to the left, S would move down, and D would move the camera to right.



The screenshot shows the Visual Studio Code editor with the file `tut_06_03.cpp` open. The code is in C++ and defines a table of vertices for a pyramid. The vertices are organized into three sections: a table for the background, a section for the base of the pyramid (square), and a section for the sides of the pyramid (triangles). The vertices are defined as `GLfloat` arrays, each containing 12 values (3 for position, 3 for normal, and 6 for texture coordinates). The code is commented with `//` and `/*` to describe the data.

```
589 {
590     // Position and Color data
591     GLfloat verts[] = {
592         //Positions      //Normals
593         //Back Face      //Negative Z Normal      Texture Coords.
594         // Vertex Positions      // Colors (r,g,b,a)
595
596         // Creating a table for the background
597         -2.0f, 0.0f, 2.0f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
598         -2.0f, 0.0f, -2.0f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
599         2.0f, 0.0f, -2.0f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
600         -2.0f, 0.0f, 2.0f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
601         2.0f, 0.0f, -2.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
602         2.0f, 0.0f, 2.0f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
603
604         // *****
605         // Vertex Positions for Pyramid
606
607         //Vertex Positions      //Texture Coordinates
608         // Base of the pyramid (square)
609         -0.5f, 0.0f, -0.5f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
610         0.5f, 0.0f, -0.5f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
611         0.5f, 0.0f, 0.5f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
612         -0.5f, 0.0f, 0.5f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
613
614         // Sides of the pyramid (triangles)
615         -0.5f, 0.0f, -0.5f, 0.0f, 0.5f, -0.5f, 0.0f, 0.0f,
616         0.5f, 0.0f, -0.5f, 0.0f, 0.5f, -0.5f, 1.0f, 0.0f,
617         0.0f, 1.0f, 0.0f, 0.0f, 0.5f, -0.5f, 0.5f, 1.0f,
618
619         0.5f, 0.0f, -0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 0.0f,
620         0.5f, 0.0f, 0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f,
621         0.0f, 1.0f, 0.0f, 0.5f, 0.5f, 0.0f, 0.5f, 1.0f,
622
623         0.5f, 0.0f, 0.5f, 0.0f, 0.5f, 0.5f, 0.0f, 0.0f,
624         -0.5f, 0.0f, 0.5f, 0.0f, 0.5f, 0.5f, 1.0f, 0.0f,
625         0.0f, 1.0f, 0.0f, 0.0f, 0.5f, 0.5f, 0.5f, 1.0f,
626
627         -0.5f, 0.0f, 0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 0.0f,
628         -0.5f, 0.0f, -0.5f, -0.5f, 0.5f, 0.0f, 1.0f, 0.0f,
629         0.0f, 1.0f, 0.0f, -0.5f, 0.5f, 0.0f, 0.5f, 1.0f,
630
631     };
632 }
```

**Figure 5:** Shows the updated vertices for the pyramid. In the code, a table was made for the pyramid to sit on, and the different sides of the pyramid are made. This code turns the 2D pyramid from the original into a 3D structure in the enhancement.

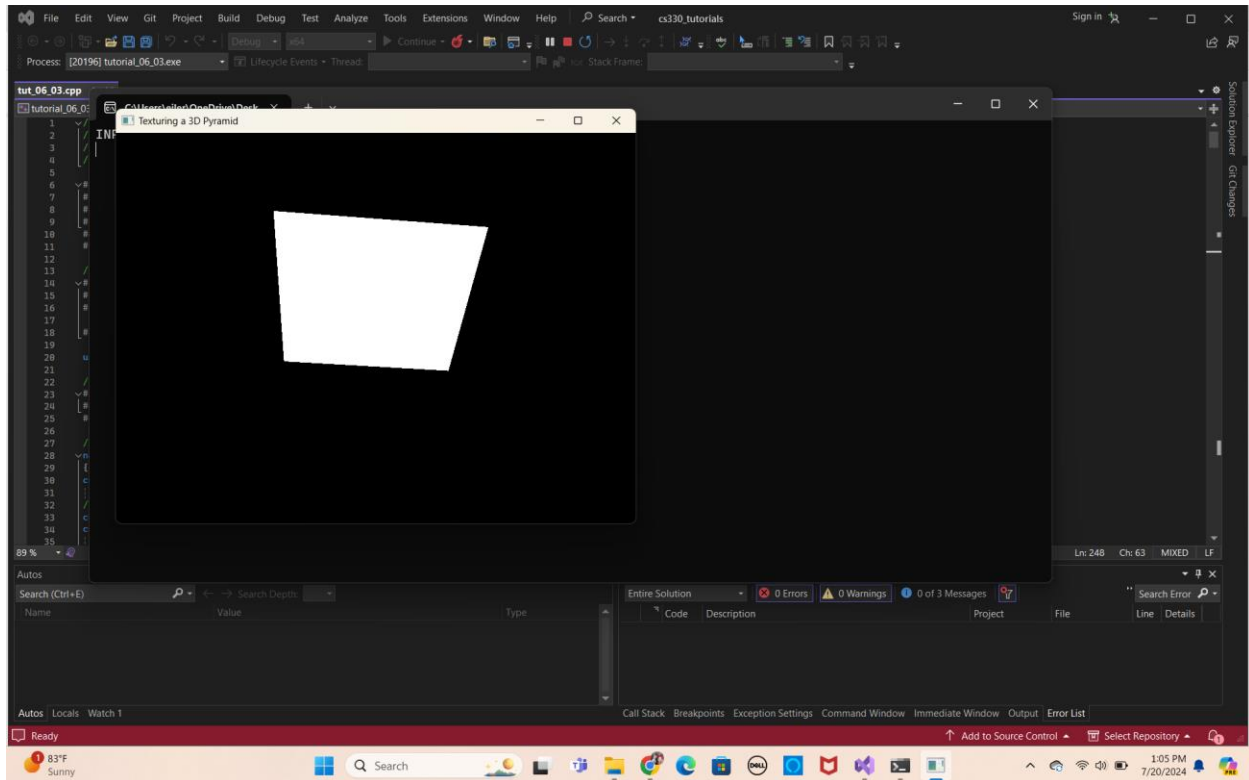


The screenshot shows the Visual Studio Code editor with the file `tut_06_03.cpp` open. The code is in C++ and defines the main logic for the pyramid. It includes shader programs, camera settings, timing, subject position and scale, cube and light color, light position and scale, lamp animation, and user-defined function prototypes. The code is commented with `//` and `/*` to describe the data.

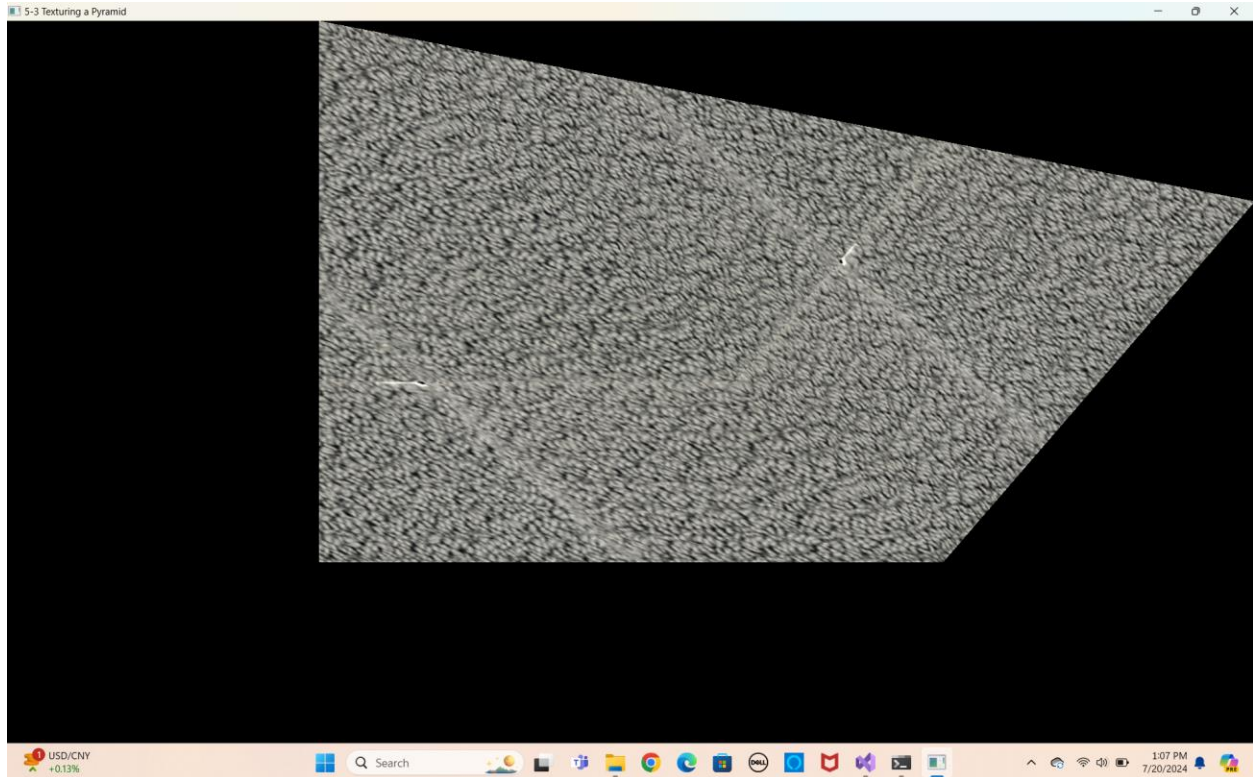
```
52 // Shader programs
53 GLuint gPyramidProgramId;
54 GLuint gLampProgramId;
55
56 // camera
57 Camera gCamera(gla::vec3(0.0f, 0.0f, 7.0f));
58 float glastX = WINDOW_WIDTH / 2.0f;
59 float glastY = WINDOW_HEIGHT / 2.0f;
60 bool gFirstMouse = true;
61
62 // timing
63 float gDeltaTime = 0.0f; // time between current frame and last frame
64 float glastFrame = 0.0f;
65
66 // Subject position and scale
67 gla::vec3 gPyramidPosition(0.0f, 0.0f, 0.0f);
68 gla::vec3 gPyramidScale(2.0f);
69
70 // Cube and light color
71 //m::vec3 gObjectColor(0.0f, 0.5f, 0.75f);
72 gla::vec3 gObjectColor(1.0f, 0.2f, 0.0f);
73 gla::vec3 gLightColor(1.0f, 1.0f, 1.0f);
74
75 // Light position and scale
76 gla::vec3 gLightPosition(1.5f, 7.5f, 4.0f);
77 gla::vec3 gLightScale(0.3f);
78
79 // Lamp animation
80 bool gIsLampOrbiting = true;
81
82
83
84 /* User-defined Function prototypes to:
85  * initialize the program, set the window size,
86  * redraw graphics on the window when resized,
87  * and render graphics on the screen
88  */
89 bool UInitialize(int, char[], GLFWwindow** window);
90 void UResizeWindow(GLFWwindow* window, int width, int height);
91 void UProcessInput(GLFWwindow* window);
92 void UMousePositionCallback(GLFWwindow* window, double xpos, double ypos);
93 void UMouseScrollCallback(GLFWwindow* window, double xoffset, double yoffset);
94 void UMouseButtonCallback(GLFWwindow* window, int button, int action, int mods);
95 void UCreateMesh(GLMesh &mesh);
96 void UDestroyMesh(GLMesh &mesh);
```



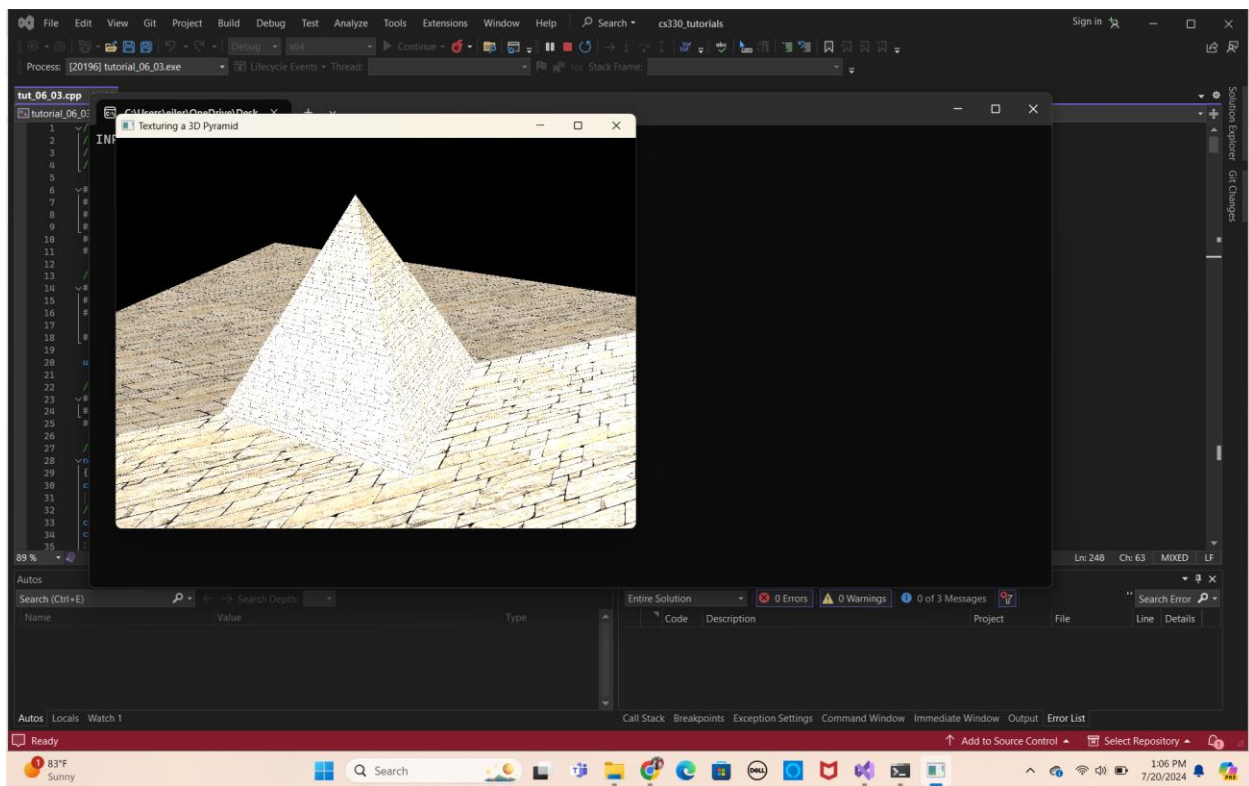
**Figure 6:** Screenshot shows the updated code to include the lamp lighting to enhance the overall scene with a light from above. Calculations were done to determine surface normal lighting for the pyramid and incorporated into the code. The image of the light will be seen in the next screenshot.



**Figure 7:** Shows the code in action. The image incorporates a lighting source to enhance the lighting in the overall scene. This light source is above the pyramid and to the front. This way the lighting provides some depth as the front of the pyramid is illuminated, while the back of the object is in shadow.



**Figure 8:** The screen shot shows the original assignment before the inclusion of the enhancements. In this image, a 2D pyramid is formed with a carpeted texture included. This image does not include any lighting sources, or 3D shapes. The overall image is improved in figure 8.



**Figure 9:** Screenshot shows the updated code in action. A 3D pyramid is shown with an enhancement to the texture, lighting and overall scene. The enhancement also allows the individual to move around to see above, below and around the object using keyboard and mouse controls.