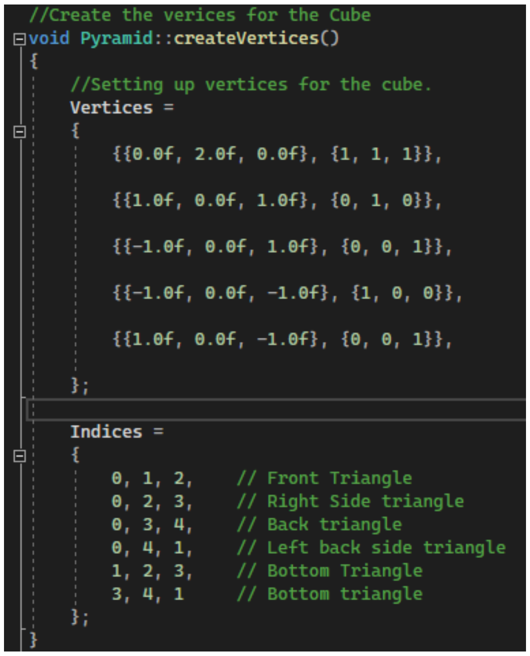
Joey Paschke
CS 480
11/13/2022

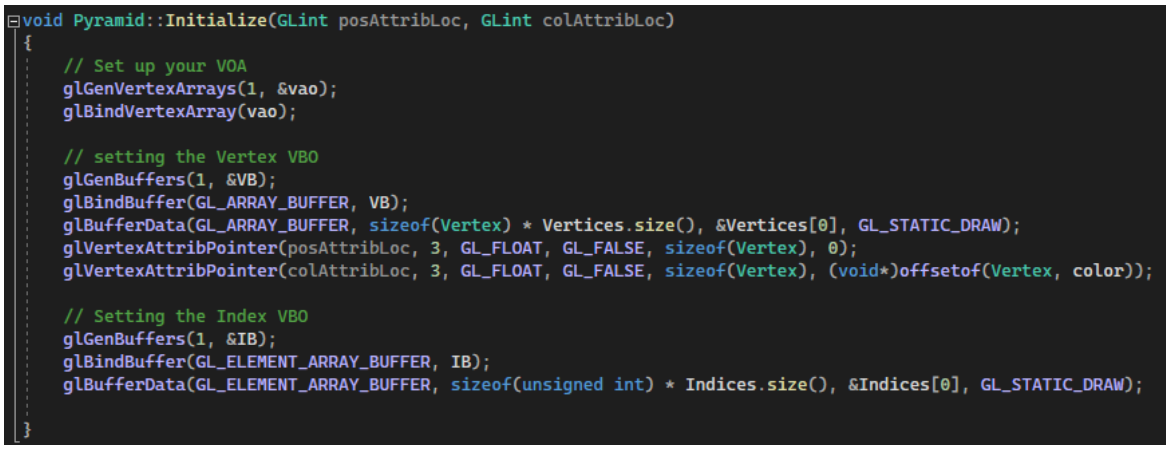
Project #3 Documentation

1. Details of pyramid class:



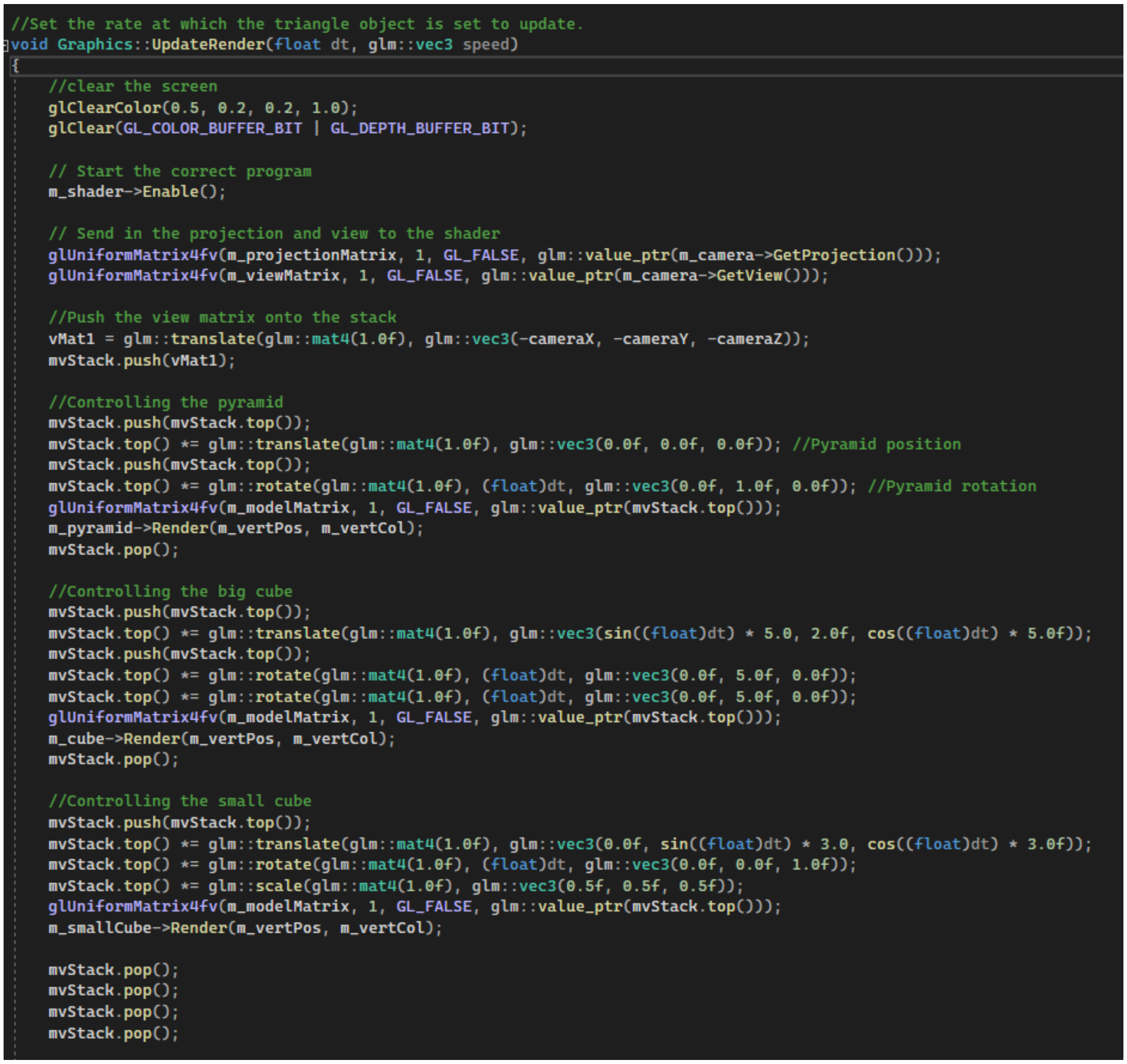
```
//Create the verices for the Cube
void Pyramid::createVertices()
{
    //Setting up vertices for the cube.
    Vertices =
    {
        {{0.0f, 2.0f, 0.0f}, {1, 1, 1}},

        {{1.0f, 0.0f, 1.0f}, {0, 1, 0}},

        {{-1.0f, 0.0f, 1.0f}, {0, 0, 1}},

        {{-1.0f, 0.0f, -1.0f}, {1, 0, 0}},

        {{1.0f, 0.0f, -1.0f}, {0, 0, 1}},

    };

    Indices =
    {
        0, 1, 2,    // Front Triangle
        0, 2, 3,    // Right Side triangle
        0, 3, 4,    // Back triangle
        0, 4, 1,    // Left back side triangle
        1, 2, 3,    // Bottom Triangle
        3, 4, 1     // Bottom triangle
    };
}
```

Above is how I declared the vertices and indices for the pyramid object. I drew a pyramid onto an XYZ plot and traced each vector we needed to make the pyramid faces. Similarly, with the indices, I drew out the pyramid and visually analyzed it to find its indices.



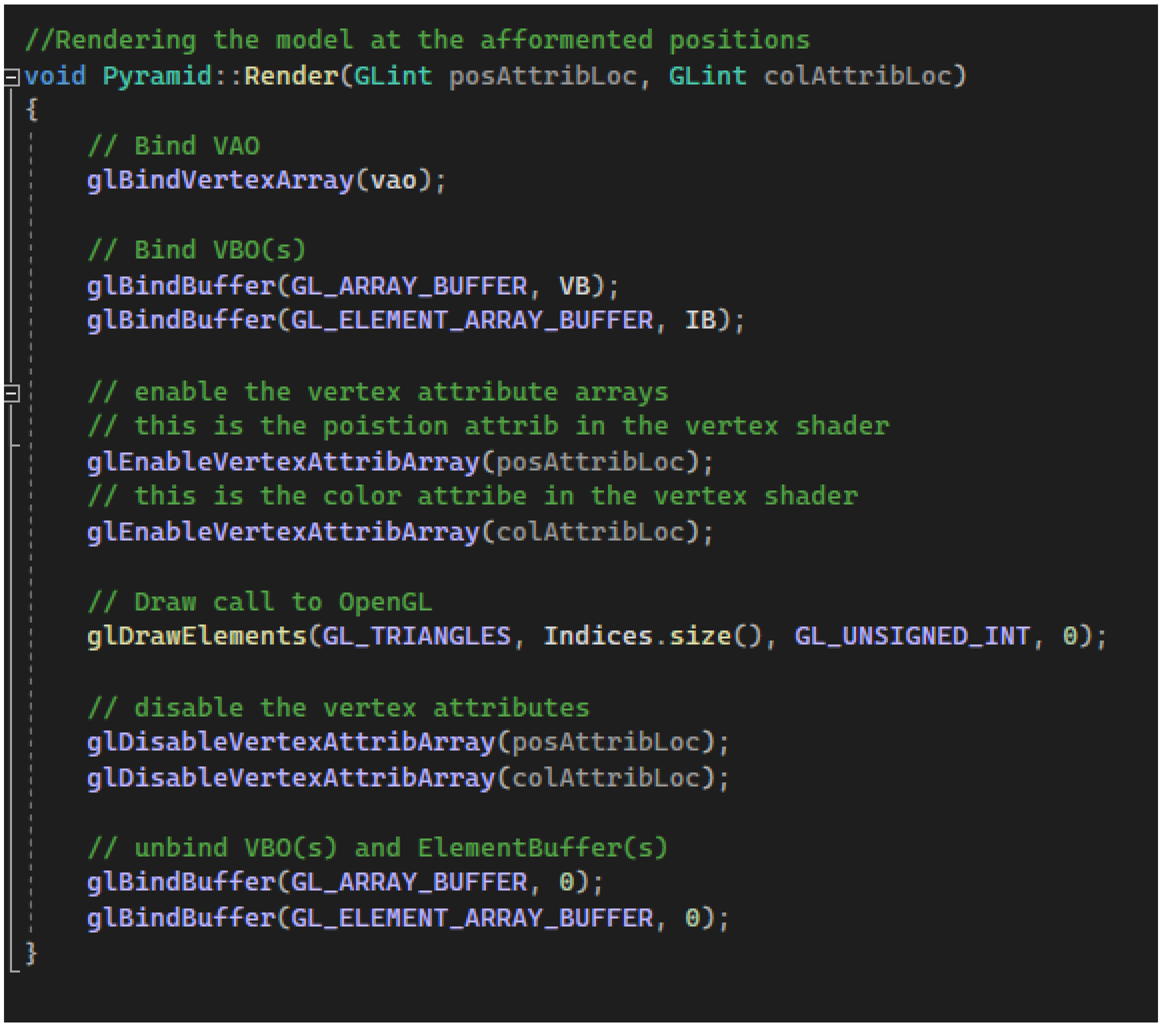
```
void Pyramid::Initialize(GLint posAttribLoc, GLint colAttribLoc)
{
    // Set up your VOA
    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);

    // setting the Vertex VBO
    glGenBuffers(1, &VB);
    glBindBuffer(GL_ARRAY_BUFFER, VB);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertex) * Vertices.size(), &Vertices[0], GL_STATIC_DRAW);
    glVertexAttribPointer(posAttribLoc, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), 0);
    glVertexAttribPointer(colAttribLoc, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, color));

    // Setting the Index VBO
    glGenBuffers(1, &IB);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IB);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(unsigned int) * Indices.size(), &Indices[0], GL_STATIC_DRAW);

}
```

Similar to the cube class, I have an initialized function of the pyramid class that handles setting up the VAOs and VBOs of the object. Additionally, I tried to make this function as flexible as possible so that it could be used with any object/vertices.



```
//Set the rate at which the triangle object is set to update.
void Graphics::UpdateRender(float dt, glm::vec3 speed)
{
    //clear the screen
    glClearColor(0.5, 0.2, 0.2, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Start the correct program
    m_shader->Enable();

    // Send in the projection and view to the shader
    glUniformMatrix4fv(m_projectionMatrix, 1, GL_FALSE, glm::value_ptr(m_camera->GetProjection()));
    glUniformMatrix4fv(m_viewMatrix, 1, GL_FALSE, glm::value_ptr(m_camera->GetView()));

    //Push the view matrix onto the stack
    vMat1 = glm::translate(glm::mat4(1.0f), glm::vec3(-cameraX, -cameraY, -cameraZ));
    mvStack.push(vMat1);

    //Controlling the pyramid
    mvStack.push(mvStack.top());
    mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f, 0.0f)); //Pyramid position
    mvStack.push(mvStack.top());
    mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)dt, glm::vec3(0.0f, 1.0f, 0.0f)); //Pyramid rotation
    glUniformMatrix4fv(m_modelMatrix, 1, GL_FALSE, glm::value_ptr(mvStack.top()));
    m_pyramid->Render(m_vertPos, m_vertCol);
    mvStack.pop();

    //Controlling the big cube
    mvStack.push(mvStack.top());
    mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(sin((float)dt) * 5.0, 2.0f, cos((float)dt) * 5.0f));
    mvStack.push(mvStack.top());
    mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)dt, glm::vec3(0.0f, 5.0f, 0.0f));
    mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)dt, glm::vec3(0.0f, 5.0f, 0.0f));
    glUniformMatrix4fv(m_modelMatrix, 1, GL_FALSE, glm::value_ptr(mvStack.top()));
    m_cube->Render(m_vertPos, m_vertCol);
    mvStack.pop();

    //Controlling the small cube
    mvStack.push(mvStack.top());
    mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, sin((float)dt) * 3.0, cos((float)dt) * 3.0f));
    mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)dt, glm::vec3(0.0f, 0.0f, 1.0f));
    mvStack.top() *= glm::scale(glm::mat4(1.0f), glm::vec3(0.5f, 0.5f, 0.5f));
    glUniformMatrix4fv(m_modelMatrix, 1, GL_FALSE, glm::value_ptr(mvStack.top()));
    m_smallCube->Render(m_vertPos, m_vertCol);

    mvStack.pop();
    mvStack.pop();
    mvStack.pop();
    mvStack.pop();
```

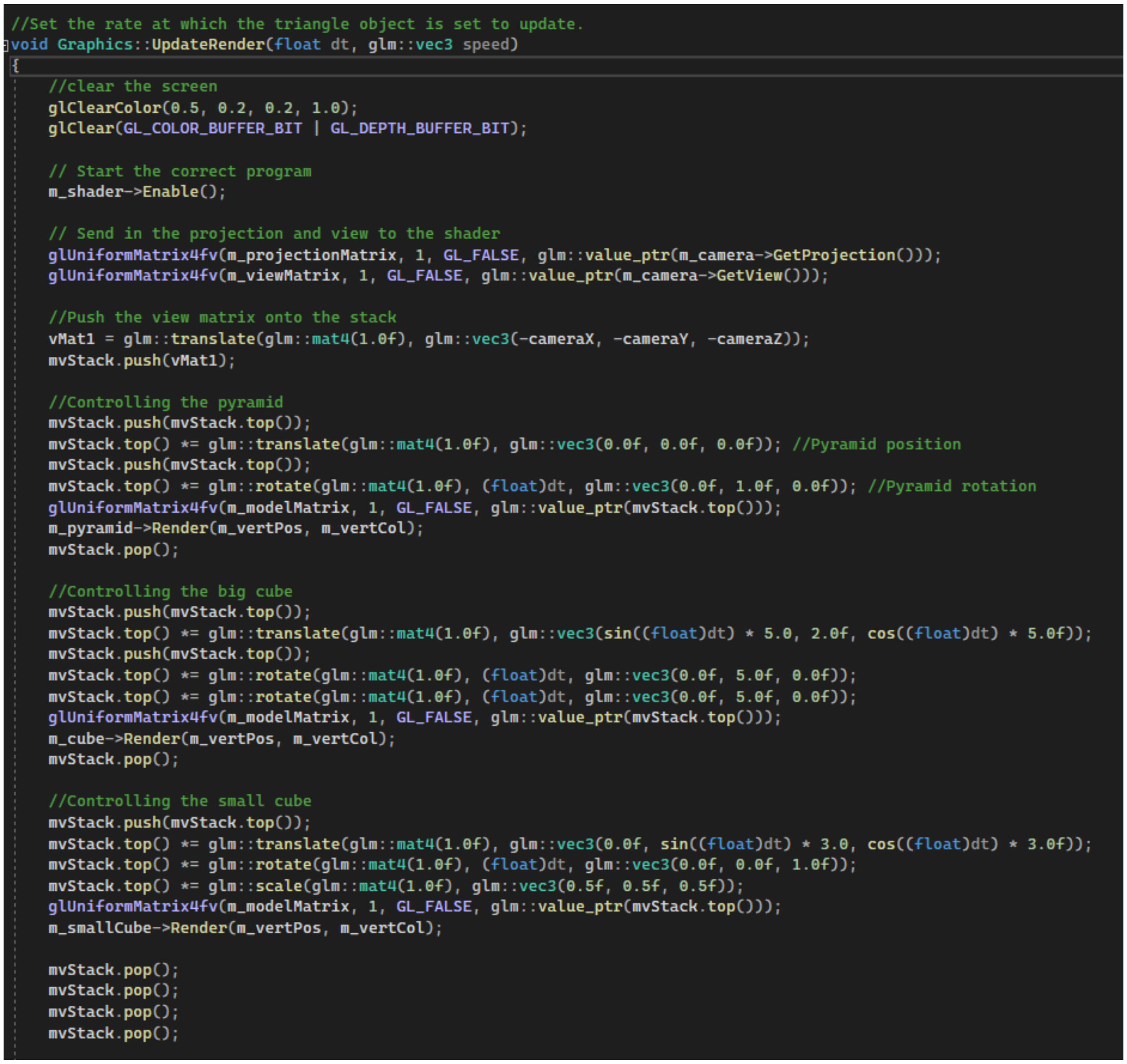
For my implementation of the model matrix of the pyramid I actually didn't use any of the class models. Instead, I created an empty matrix in the graphics.cpp and would push that into the stack, move it around how it needs to, and then render that specific object at the point that the original matrix is. This way we don't have to worry about a matrix for each model.



```
//Rendering the model at the afformented positions
void Pyramid::Render(GLint posAttribLoc, GLint colAttribLoc)
{
    // Bind VAO
    glBindVertexArray(vao);

    // Bind VBO(s)
    glBindBuffer(GL_ARRAY_BUFFER, VB);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IB);

    // enable the vertex attribute arrays
    // this is the poistion attrib in the vertex shader
    glEnableVertexAttribArray(posAttribLoc);
    // this is the color attribe in the vertex shader
    glEnableVertexAttribArray(colAttribLoc);

    // Draw call to OpenGL
    glDrawElements(GL_TRIANGLES, Indices.size(), GL_UNSIGNED_INT, 0);

    // disable the vertex attributes
    glDisableVertexAttribArray(posAttribLoc);
    glDisableVertexAttribArray(colAttribLoc);

    // unbind VBO(s) and ElementBuffer(s)
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
}
```

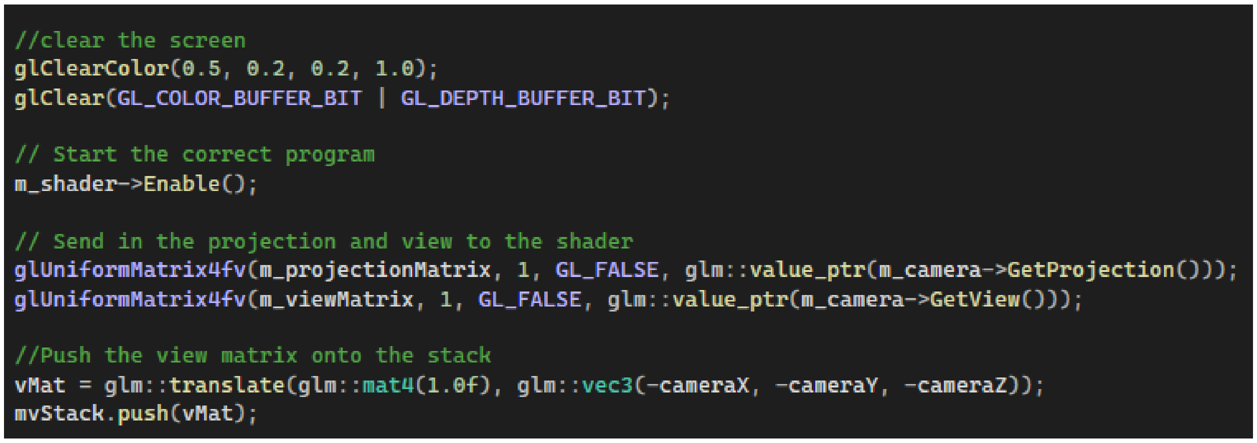
This shown above is how I handle the rendering of the pyramid object. Because of the way that I made my vertices and indices, I’m using drawElements to draw the pyramid. Additionally, this is where all of the VAOs and VBOs are taken care of.

2. Solar system details:



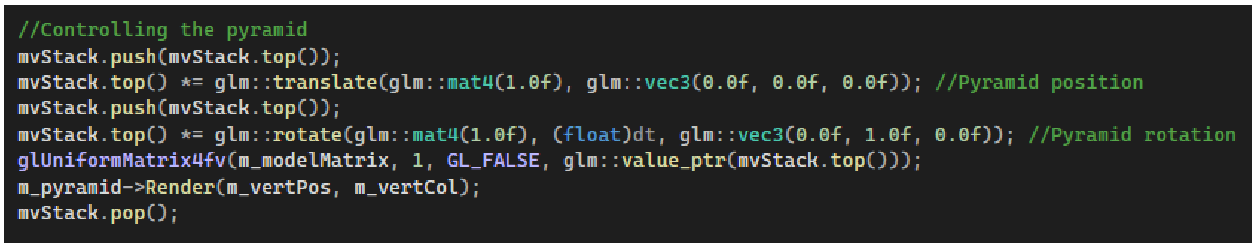
```
//Set the rate at which the triangle object is set to update.
void Graphics::UpdateRender(float dt, glm::vec3 speed)
{
    //clear the screen
    glClearColor(0.5, 0.2, 0.2, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Start the correct program
    m_shader->Enable();

    // Send in the projection and view to the shader
    glUniformMatrix4fv(m_projectionMatrix, 1, GL_FALSE, glm::value_ptr(m_camera->GetProjection()));
    glUniformMatrix4fv(m_viewMatrix, 1, GL_FALSE, glm::value_ptr(m_camera->GetView()));

    //Push the view matrix onto the stack
    vMat1 = glm::translate(glm::mat4(1.0f), glm::vec3(-cameraX, -cameraY, -cameraZ));
    mvStack.push(vMat1);

    //Controlling the pyramid
    mvStack.push(mvStack.top());
    mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f, 0.0f)); //Pyramid position
    mvStack.push(mvStack.top());
    mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)dt, glm::vec3(0.0f, 1.0f, 0.0f)); //Pyramid rotation
    glUniformMatrix4fv(m_modelMatrix, 1, GL_FALSE, glm::value_ptr(mvStack.top()));
    m_pyramid->Render(m_vertPos, m_vertCol);
    mvStack.pop();

    //Controlling the big cube
    mvStack.push(mvStack.top());
    mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(sin((float)dt) * 5.0, 2.0f, cos((float)dt) * 5.0f));
    mvStack.push(mvStack.top());
    mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)dt, glm::vec3(0.0f, 5.0f, 0.0f));
    mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)dt, glm::vec3(0.0f, 5.0f, 0.0f));
    glUniformMatrix4fv(m_modelMatrix, 1, GL_FALSE, glm::value_ptr(mvStack.top()));
    m_cube->Render(m_vertPos, m_vertCol);
    mvStack.pop();

    //Controlling the small cube
    mvStack.push(mvStack.top());
    mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, sin((float)dt) * 3.0, cos((float)dt) * 3.0f));
    mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)dt, glm::vec3(0.0f, 0.0f, 1.0f));
    mvStack.top() *= glm::scale(glm::mat4(1.0f), glm::vec3(0.5f, 0.5f, 0.5f));
    glUniformMatrix4fv(m_modelMatrix, 1, GL_FALSE, glm::value_ptr(mvStack.top()));
    m_smallCube->Render(m_vertPos, m_vertCol);

    mvStack.pop();
    mvStack.pop();
    mvStack.pop();
    mvStack.pop();
```

This here is the main driver code to make all of the objects move in relation to each other. In the header file of the graphics class, I added a stack variable that will be used to order the movements of the objects. Above you can see that I combined the Update and render functions of the graphics class.



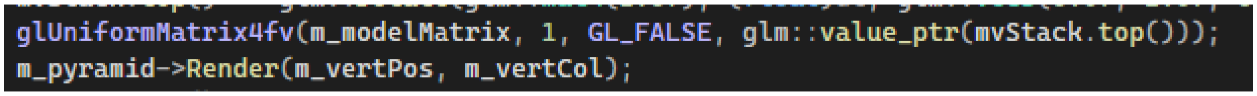
```
//clear the screen
glClearColor(0.5, 0.2, 0.2, 1.0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// Start the correct program
m_shader->Enable();

// Send in the projection and view to the shader
glUniformMatrix4fv(m_projectionMatrix, 1, GL_FALSE, glm::value_ptr(m_camera->GetProjection()));
glUniformMatrix4fv(m_viewMatrix, 1, GL_FALSE, glm::value_ptr(m_camera->GetView()));

//Push the view matrix onto the stack
vMat = glm::translate(glm::mat4(1.0f), glm::vec3(-cameraX, -cameraY, -cameraZ));
mvStack.push(vMat);
```

Starting off here I made sure to keep all the initialization that was previously in the render function of the graphics class. This includes clearing the screen, enabling shaders, and setting up the view and projection matrices of the camera. At the bottom, I added the declaration of the view matrix that I use to move around and place objects. Lastly, I push this view matrix to the stack.



```
//Controlling the pyramid
mvStack.push(mvStack.top());
mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f, 0.0f)); //Pyramid position
mvStack.push(mvStack.top());
mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)dt, glm::vec3(0.0f, 1.0f, 0.0f)); //Pyramid rotation
glUniformMatrix4fv(m_modelMatrix, 1, GL_FALSE, glm::value_ptr(mvStack.top()));
m_pyramid->Render(m_vertPos, m_vertCol);
mvStack.pop();
```

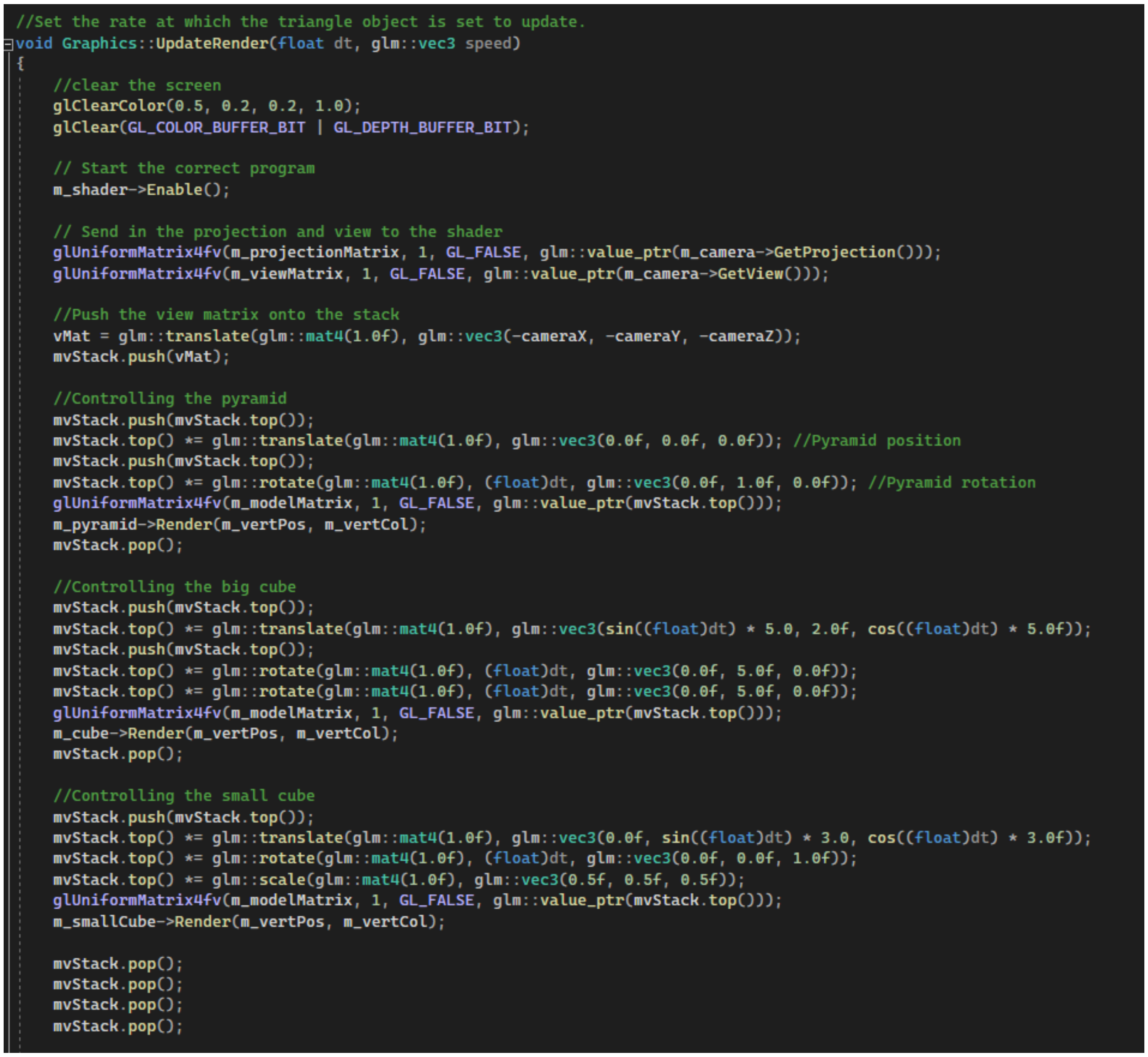
Now because this is in the updateRender function which is called in the engine class, this runs numerous times. So at each step of movement, this code needs to be run. The code above shows the translations and rotations that I do to the view matrix. Once we have translated it we keep the translation in the stack for the next object to reference. That way the cube will move based on the pyramid.



```
glUniformMatrix4fv(m_modelMatrix, 1, GL_FALSE, glm::value_ptr(mvStack.top()));
m_pyramid->Render(m_vertPos, m_vertCol);
```

This glUniformMatrix is helping me store the position of the pyramid which should be at the top of the stack to a variable. That way we can use this later to position the other objects. Once the position is stored we render the object at the position. This is then done with all 3 objects many times a second which creates a smooth animation.

2. **Details of change in the graphics/engine pipeline that enables your solar system animation.**



```
//Set the rate at which the triangle object is set to update.
void Graphics::UpdateRender(float dt, glm::vec3 speed)
{
    //clear the screen
    glClearColor(0.5, 0.2, 0.2, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Start the correct program
    m_shader->Enable();

    // Send in the projection and view to the shader
    glUniformMatrix4fv(m_projectionMatrix, 1, GL_FALSE, glm::value_ptr(m_camera->GetProjection()));
    glUniformMatrix4fv(m_viewMatrix, 1, GL_FALSE, glm::value_ptr(m_camera->GetView()));

    //Push the view matrix onto the stack
    vMat = glm::translate(glm::mat4(1.0f), glm::vec3(-cameraX, -cameraY, -cameraZ));
    mvStack.push(vMat);

    //Controlling the pyramid
    mvStack.push(mvStack.top());
    mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f, 0.0f)); //Pyramid position
    mvStack.push(mvStack.top());
    mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)dt, glm::vec3(0.0f, 1.0f, 0.0f)); //Pyramid rotation
    glUniformMatrix4fv(m_modelMatrix, 1, GL_FALSE, glm::value_ptr(mvStack.top()));
    m_pyramid->Render(m_vertPos, m_vertCol);
    mvStack.pop();

    //Controlling the big cube
    mvStack.push(mvStack.top());
    mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(sin((float)dt) * 5.0, 2.0f, cos((float)dt) * 5.0f));
    mvStack.push(mvStack.top());
    mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)dt, glm::vec3(0.0f, 5.0f, 0.0f));
    mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)dt, glm::vec3(0.0f, 5.0f, 0.0f));
    glUniformMatrix4fv(m_modelMatrix, 1, GL_FALSE, glm::value_ptr(mvStack.top()));
    m_cube->Render(m_vertPos, m_vertCol);
    mvStack.pop();

    //Controlling the small cube
    mvStack.push(mvStack.top());
    mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, sin((float)dt) * 3.0, cos((float)dt) * 3.0f));
    mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)dt, glm::vec3(0.0f, 0.0f, 1.0f));
    mvStack.top() *= glm::scale(glm::mat4(1.0f), glm::vec3(0.5f, 0.5f, 0.5f));
    glUniformMatrix4fv(m_modelMatrix, 1, GL_FALSE, glm::value_ptr(mvStack.top()));
    m_smallCube->Render(m_vertPos, m_vertCol);

    mvStack.pop();
    mvStack.pop();
    mvStack.pop();
    mvStack.pop();
```

As I mentioned above, I combined the Update and Render functions of the graphics class in order to streamline the process. Once they were combined, I created a stack and empty view matrix that I would be able to pop in and out of the stack and move where I needed it. Once I moved the view matrix where I needed it, I would render the object at that point. With the help of the stack being able to base positions off of other objects, it allowed me to seamlessly animate the solar system.