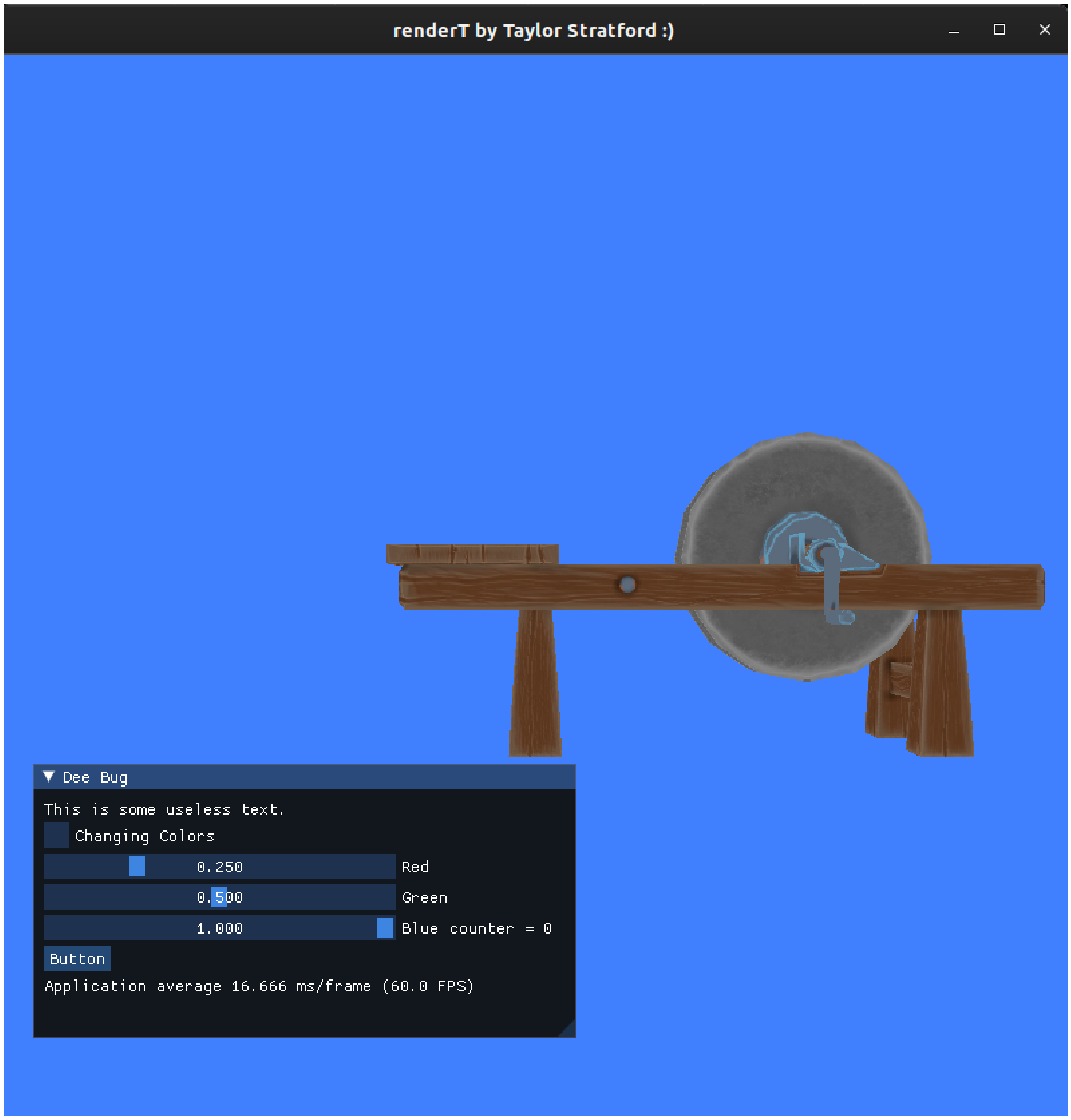
renderT

By Taylor Stratford



This is some useless text.

Changing Colors

0.250 Red

0.500 Green

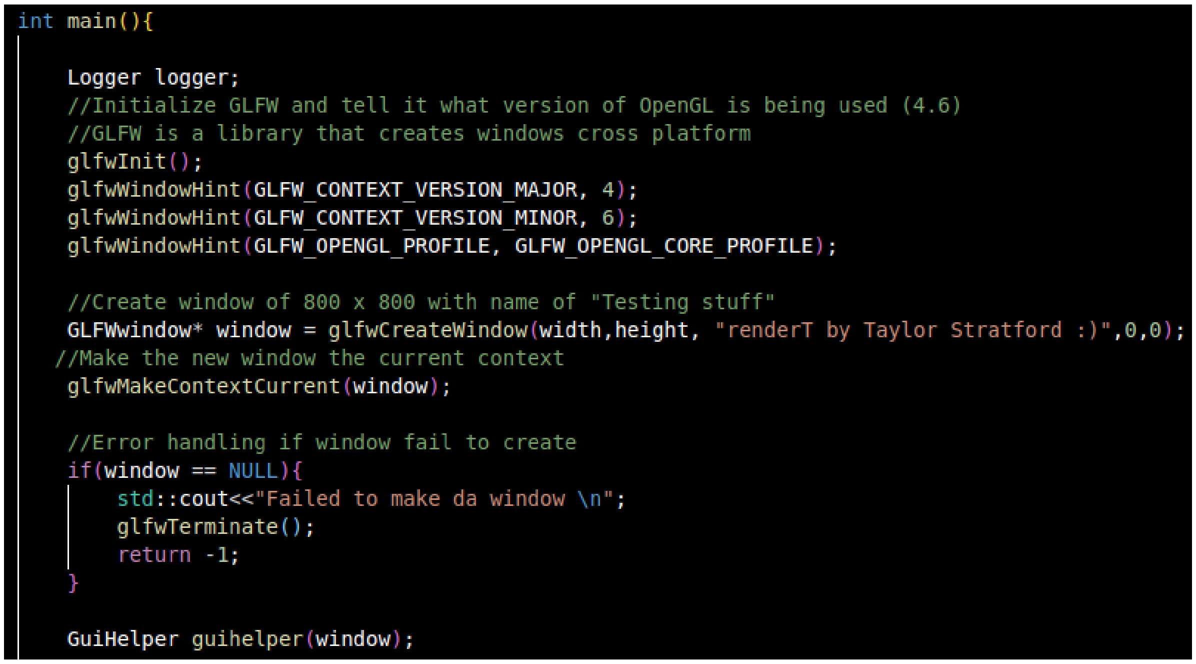
1.000 Blue counter = 0

Button

Application average 16.666 ms/frame (60.0 FPS)

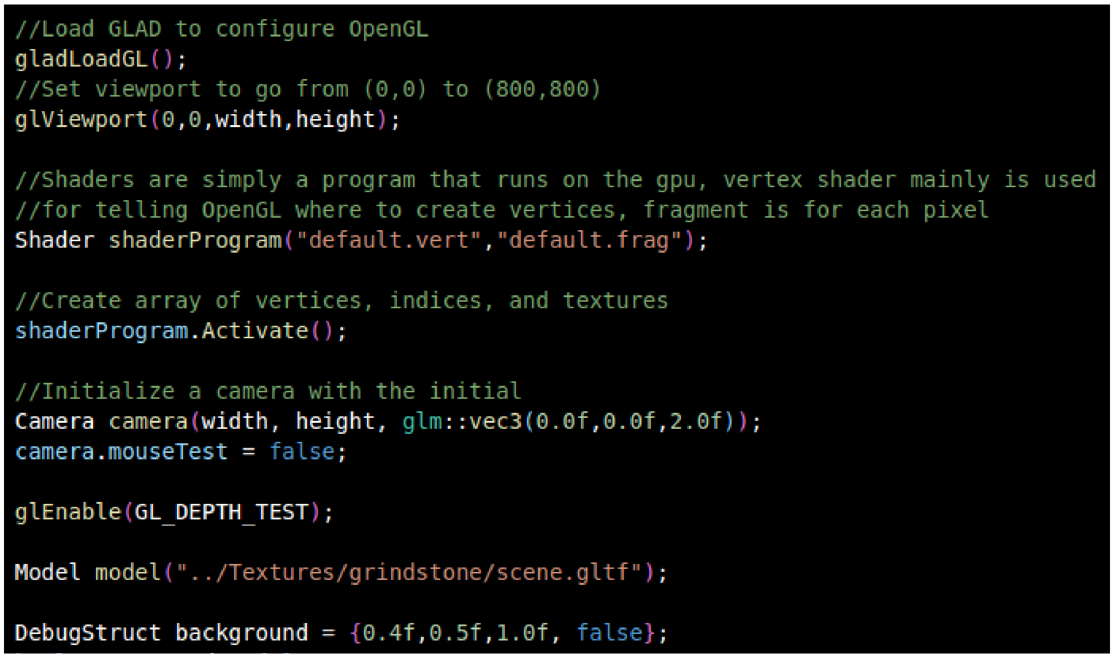
Walkthrough

Starting at the top, an instance of my logger class is initialized and glfw (cross platform windowing library) is configured for OpenGl 4.6 and then a glfw window is created an nullchecked, then an instance of GuiHelper (Helper class i created to interface with ImGui) is created and set to the window we just created



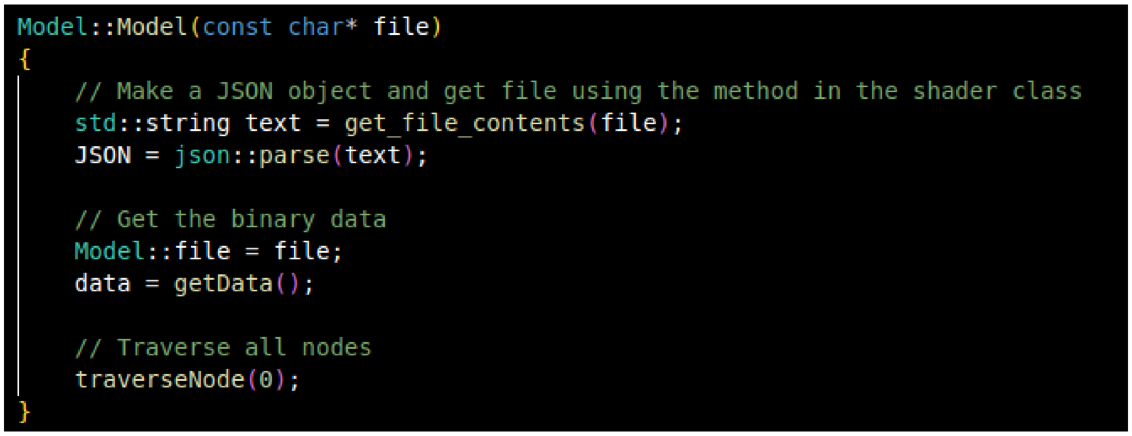
```cpp
int main(){

    Logger logger;
    //Initialize GLFW and tell it what version of OpenGL is being used (4.6)
    //GLFW is a library that creates windows cross platform
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 6);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

    //Create window of 800 x 800 with name of "Testing stuff"
    GLFWwindow* window = glfwCreateWindow(width,height, "renderT by Taylor Stratford :)",0,0);
   //Make the new window the current context
    glfwMakeContextCurrent(window);

    //Error handling if window fail to create
    if(window == NULL){
        std::cout<<"Failed to make da window \n";
        glfwTerminate();
        return -1;
    }

    GuiHelper guihelper(window);
```

GLAD (OpenGL loader) is loaded and viewport of 800x800 (defined above main function) is set. An instance of helper Shader class is created and the vertex and fragment shaders I made are loaded (These shaders are fairly basic) and then activated. The shader constructor reads the contents of each file, converts them into a c string, then creates and compiles each shader, attaches them to the ID member of the shaderprogram then links that ID, validates it then deletes the shader. Activate calls "glUseProgram" which sets that shader to be used. An instance of Camera and model are also created and more detail on how those classes work will be explained below. An instance of DebugStruct is created which sets the background color and ImGui has hooks into this object



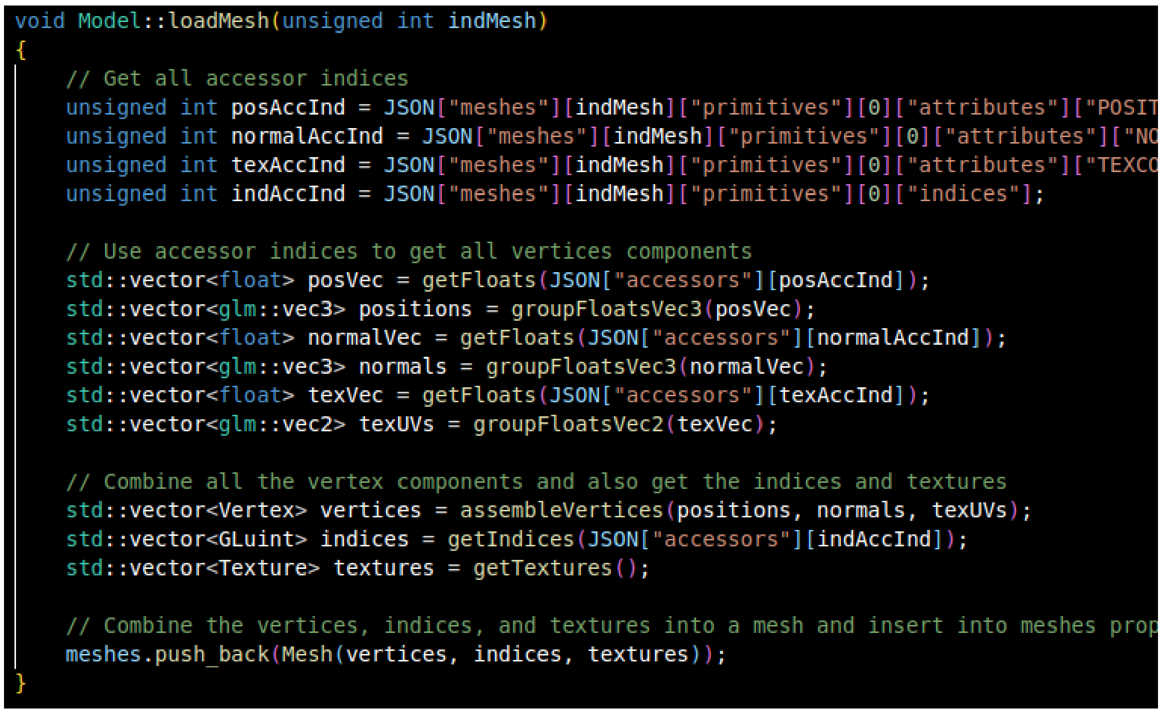
```cpp
//Load GLAD to configure OpenGL
gladLoadGL();
//Set viewport to go from (0,0) to (800,800)
glViewport(0,0,width,height);

//Shaders are simply a program that runs on the gpu, vertex shader mainly is used
//for telling OpenGL where to create vertices, fragment is for each pixel
Shader shaderProgram("default.vert","default.frag");

//Create array of vertices, indices, and textures
shaderProgram.Activate();

//Initialize a camera with the initial
Camera camera(width, height, glm::vec3(0.0f,0.0f,2.0f));
camera.mouseTest = false;

glEnable(GL_DEPTH_TEST);

Model model("../Textures/grindstone/scene.gltf");

DebugStruct background = {0.4f,0.5f,1.0f, false};
```

Model constructor reads the gltf file passed in (is in JSON format)



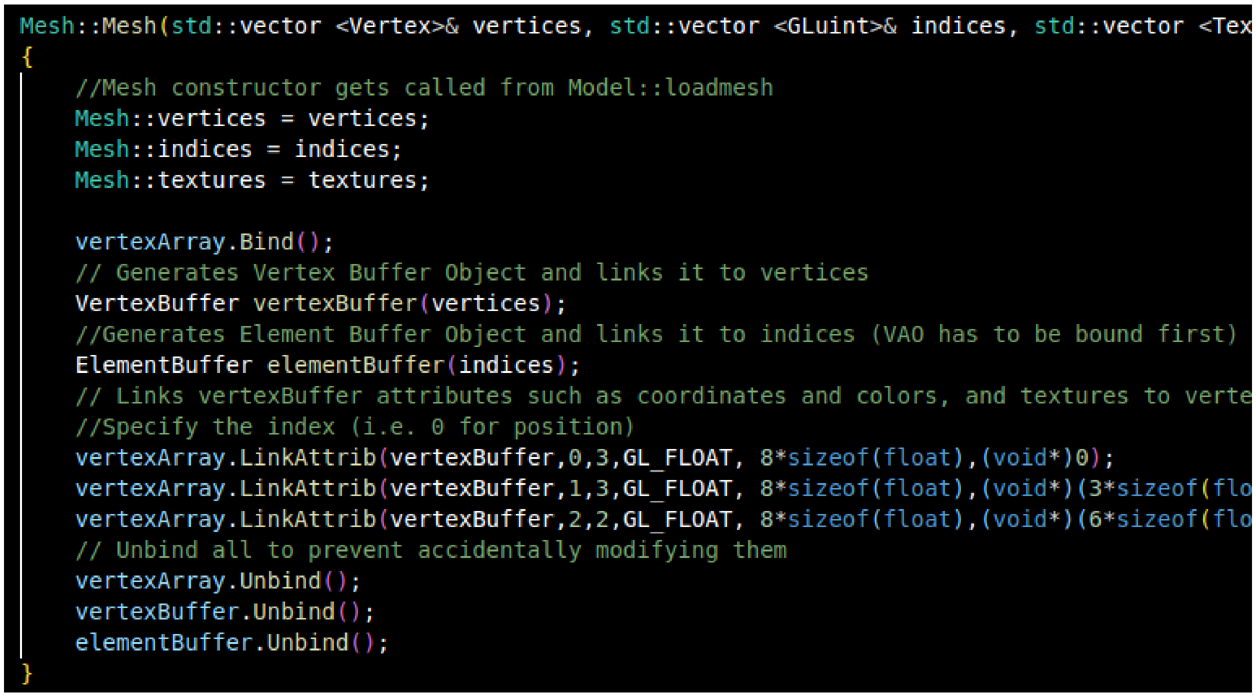
```cpp
Model::Model(const char* file)
{
    // Make a JSON object and get file using the method in the shader class
    std::string text = get_file_contents(file);
    JSON = json::parse(text);

    // Get the binary data
    Model::file = file;
    data = getData();

    // Traverse all nodes
    traverseNode(0);
}
```

traverseNode calls loadMesh which adds a Mesh to meshes property of Model (assembleVertices combines properties into vertices to then be used by VAO)

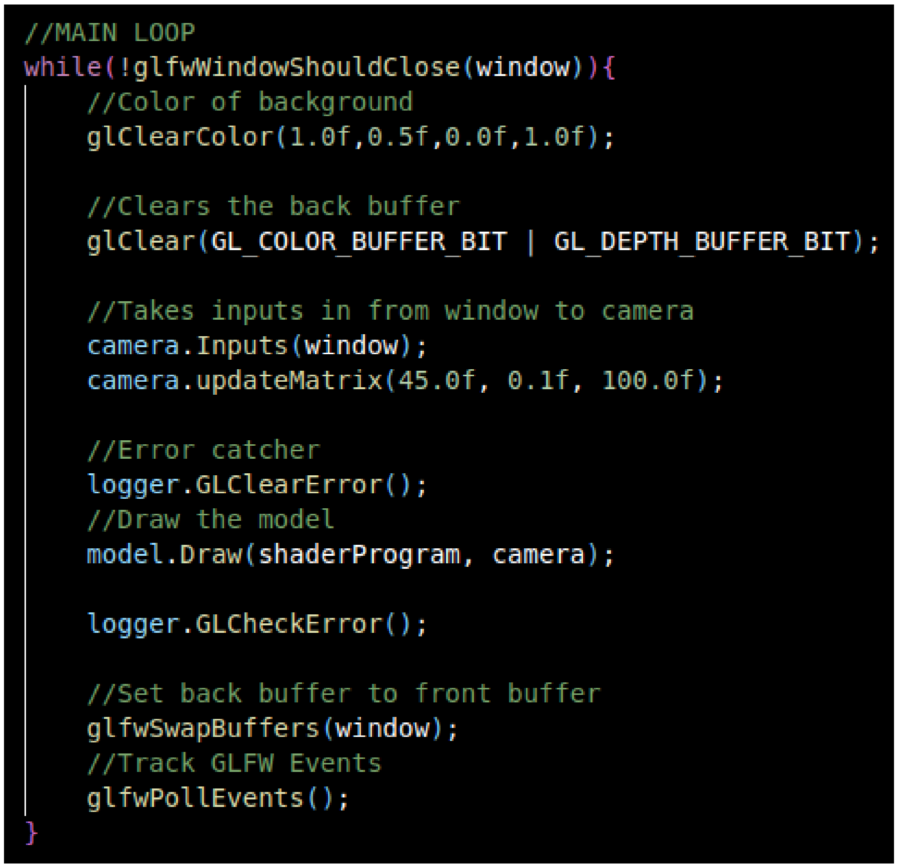


```cpp
void Model::loadMesh(unsigned int indMesh)
{
    // Get all accessor indices
    unsigned int posAccInd = JSON["meshes"][indMesh]["primitives"][0]["attributes"]["POSIT
    unsigned int normalAccInd = JSON["meshes"][indMesh]["primitives"][0]["attributes"]["NO
    unsigned int texAccInd = JSON["meshes"][indMesh]["primitives"][0]["attributes"]["TEXCO
    unsigned int indAccInd = JSON["meshes"][indMesh]["primitives"][0]["indices"];

    // Use accessor indices to get all vertices components
    std::vector<float> posVec = getFloats(JSON["accessors"][posAccInd]);
    std::vector<glm::vec3> positions = groupFloatsVec3(posVec);
    std::vector<float> normalVec = getFloats(JSON["accessors"][normalAccInd]);
    std::vector<glm::vec3> normals = groupFloatsVec3(normalVec);
    std::vector<float> texVec = getFloats(JSON["accessors"][texAccInd]);
    std::vector<glm::vec2> texUVs = groupFloatsVec2(texVec);

    // Combine all the vertex components and also get the indices and textures
    std::vector<Vertex> vertices = assembleVertices(positions, normals, texUVs);
    std::vector<GLuint> indices = getIndices(JSON["accessors"][indAccInd]);
    std::vector<Texture> textures = getTextures();

    // Combine the vertices, indices, and textures into a mesh and insert into meshes prop
    meshes.push_back(Mesh(vertices, indices, textures));
}
```

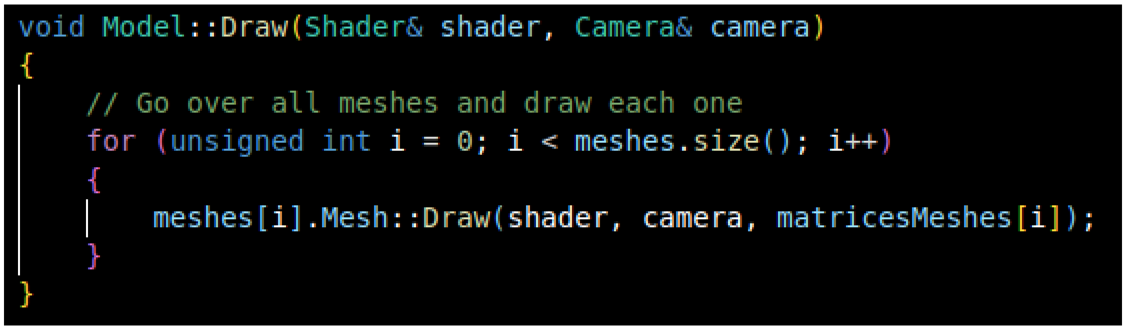
This calls the Mesh constructor which binds the vertexArray property in Mesh, and then sets the attributes up in the vertex array (in this case it’s linking coordinates, colors, textures)


```
Mesh::Mesh(std::vector <Vertex>& vertices, std::vector <GLuint>& indices, std::vector <Tex
{
    //Mesh constructor gets called from Model::loadmesh
    Mesh::vertices = vertices;
    Mesh::indices = indices;
    Mesh::textures = textures;

    vertexArray.Bind();
    // Generates Vertex Buffer Object and links it to vertices
    VertexBuffer vertexBuffer(vertices);
    //Generates Element Buffer Object and links it to indices (VAO has to be bound first)
    ElementBuffer elementBuffer(indices);
    // Links vertexBuffer attributes such as coordinates and colors, and textures to verte
    //Specify the index (i.e. 0 for position)
    vertexArray.LinkAttrib(vertexBuffer,0,3,GL_FLOAT, 8*sizeof(float),(void*)0);
    vertexArray.LinkAttrib(vertexBuffer,1,3,GL_FLOAT, 8*sizeof(float),(void*)(3*sizeof(flo
    vertexArray.LinkAttrib(vertexBuffer,2,2,GL_FLOAT, 8*sizeof(float),(void*)(6*sizeof(flo
    // Unbind all to prevent accidentally modifying them
    vertexArray.Unbind();
    vertexBuffer.Unbind();
    elementBuffer.Unbind();
}
```

Main while loop

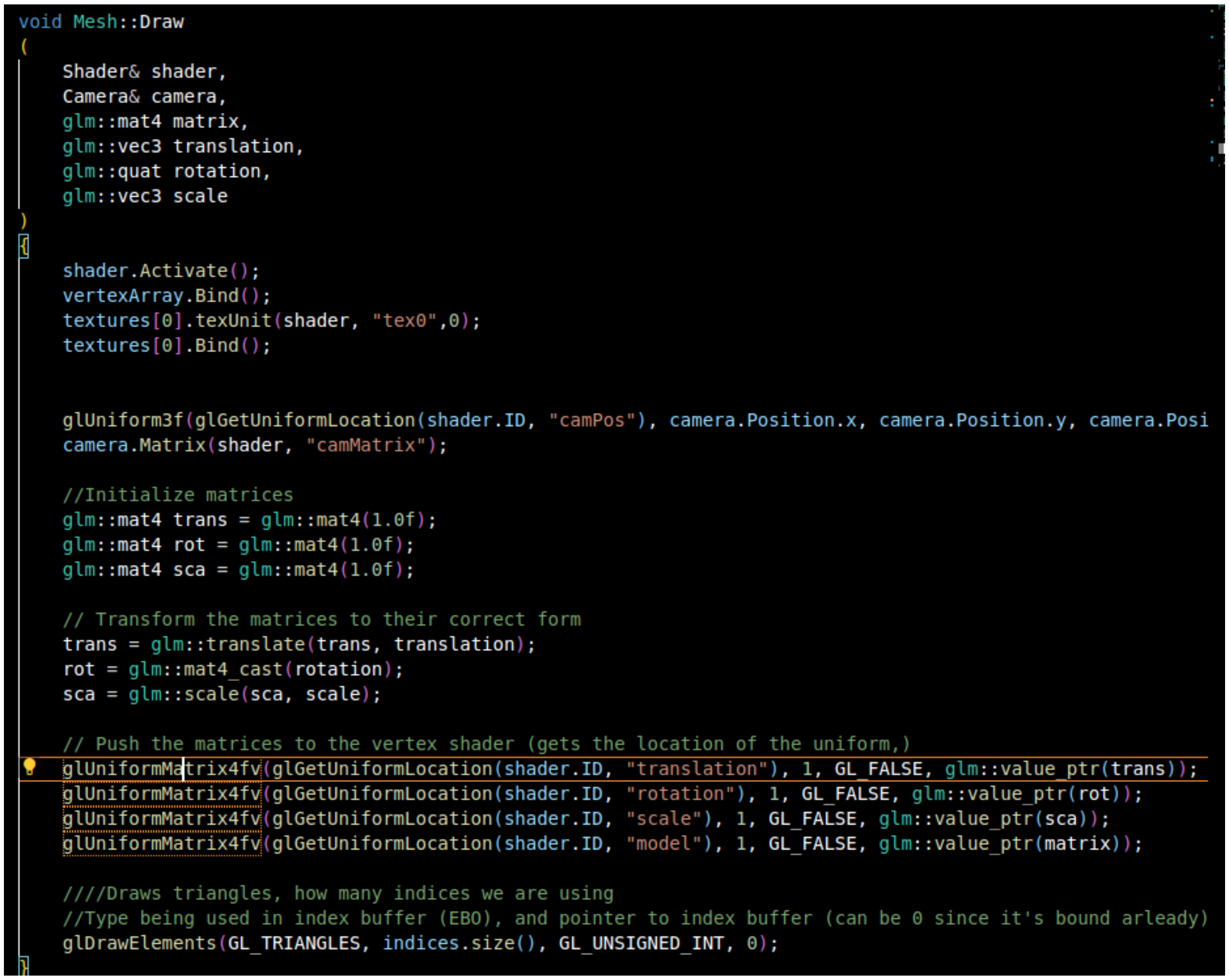

```
//MAIN LOOP
while(!glfwWindowShouldClose(window)){
    //Color of background
    glClearColor(1.0f,0.5f,0.0f,1.0f);

    //Clears the back buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //Takes inputs in from window to camera
    camera.Inputs(window);
    camera.updateMatrix(45.0f, 0.1f, 100.0f);

    //Error catcher
    logger.GLClearError();
    //Draw the model
    model.Draw(shaderProgram, camera);

    logger.GLCheckError();

    //Set back buffer to front buffer
    glfwSwapBuffers(window);
    //Track GLFW Events
    glfwPollEvents();
}
```

Model.Draw goes over the meshes that were pushed to the vertex



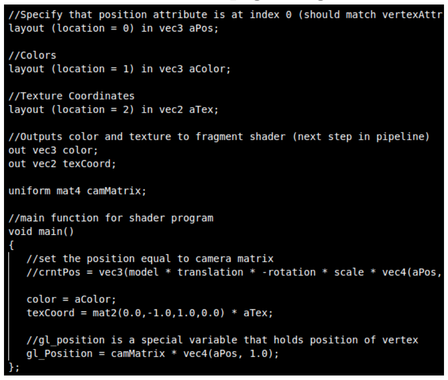
```cpp
void Model::Draw(Shader& shader, Camera& camera)
{
    // Go over all meshes and draw each one
    for (unsigned int i = 0; i < meshes.size(); i++)
    {
        meshes[i].Mesh::Draw(shader, camera, matricesMeshes[i]);
    }
}
```

Mesh.Draw - updates the shader uniforms and then draws it



```cpp
void Mesh::Draw
(
    Shader& shader,
    Camera& camera,
    glm::mat4 matrix,
    glm::vec3 translation,
    glm::quat rotation,
    glm::vec3 scale
)
{

    shader.Activate();
    vertexArray.Bind();
    textures[0].texUnit(shader, "tex0",0);
    textures[0].Bind();


    glUniform3f(glGetUniformLocation(shader.ID, "camPos"), camera.Position.x, camera.Position.y, camera.Posi
    camera.Matrix(shader, "camMatrix");

    //Initialize matrices
    glm::mat4 trans = glm::mat4(1.0f);
    glm::mat4 rot = glm::mat4(1.0f);
    glm::mat4 sca = glm::mat4(1.0f);

    // Transform the matrices to their correct form
    trans = glm::translate(trans, translation);
    rot = glm::mat4_cast(rotation);
    sca = glm::scale(sca, scale);

    // Push the matrices to the vertex shader (gets the location of the uniform,)
    glUniformMatrix4fv(glGetUniformLocation(shader.ID, "translation"), 1, GL_FALSE, glm::value_ptr(trans));
    glUniformMatrix4fv(glGetUniformLocation(shader.ID, "rotation"), 1, GL_FALSE, glm::value_ptr(rot));
    glUniformMatrix4fv(glGetUniformLocation(shader.ID, "scale"), 1, GL_FALSE, glm::value_ptr(sca));
    glUniformMatrix4fv(glGetUniformLocation(shader.ID, "model"), 1, GL_FALSE, glm::value_ptr(matrix));

    ////Draws triangles, how many indices we are using
    //Type being used in index buffer (EBO), and pointer to index buffer (can be 0 since it's bound arleady)
    glDrawElements(GL_TRIANGLES, indices.size(), GL_UNSIGNED_INT, 0);
}
```

(the trans, rot, and sca vars can and are removed because i removed them from shaders [lighting not implemented yet]
In vertex shader, vars are pulled from vertexAttribute shader positions



```glsl
//Specify that position attribute is at index 0 (should match vertexAttr
layout (location = 0) in vec3 aPos;

//Colors
layout (location = 1) in vec3 aColor;

//Texture Coordinates
layout (location = 2) in vec2 aTex;

//Outputs color and texture to fragment shader (next step in pipeline)
out vec3 color;
out vec2 texCoord;

uniform mat4 camMatrix;

//main function for shader program
void main()
{
   //set the position equal to camera matrix
   //crntPos = vec3(model * translation * -rotation * scale * vec4(aPos,

   color = aColor;
   texCoord = mat2(0.0,-1.0,1.0,0.0) * aTex;

   //gl_position is a special variable that holds position of vertex
   gl_Position = camMatrix * vec4(aPos, 1.0);
};
```