7-1 Final Project Reflection and Review:

I learned a lot throughout this project. I think the best way of structuring this document is by covering each of the milestones and anything notable I did and learned, and finally how I put it all together in the final milestone.

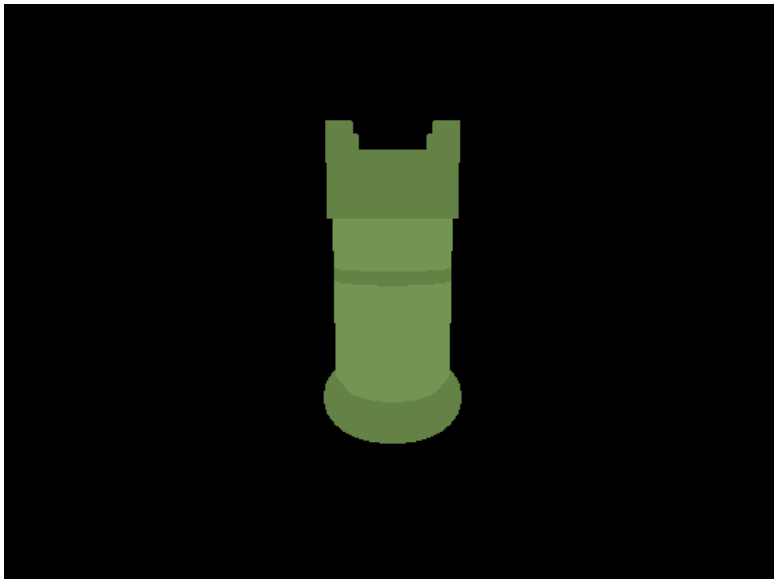
2-3 Milestone One: Project Proposal:

We started with our Project Proposal. After a bit of brainstorming, I thought about going with a chessboard as it would satisfy all the requirements and be quite fun, almost like making my own chess game. I knew that doing all the pieces would be a bit troublesome, so I decided to go with something significant. That's when I decided on a check in one. I modeled it on a 2D chessboard and decided that's what I'd do. If I couldn't do a knight piece, I decided to replace it with a pawn because it would have the same meaning. I did try to make the Knight, but it didn't look as good as the others, so I decided to focus on the pawn. I mainly followed my own ideas and instructions from this document to choose the pieces, textures, and positioning.

3-3 Milestone Two: Beginning a 3D Scene:

This was our first coding milestone. We were tasked with creating one of the 3D objects in our chosen scene. I decided to go with the rook. The rook was actually one of the more complex pieces, so I'm happy I started out with this one.

This was our result:



I decided to go with the green color because it made it easier to see in the dark void template we were given, but also because it is a staple of the well-known chess.com website.

Coding Design:

For this milestone, I decided to code the rook by gathering the objects we discussed in our proposal and then using positioning and scaling to make it into a proper piece. While doing so, we created most of the conventions we would utilize in the future.

One of the most notable pieces of code would have to go to our crown:

```
// Rook Crown formation (4 small boxes)
{

    scaleXYZ = glm::vec3(0.4f, 0.8f, 0.4f);

    XrotationDegrees = 0.0f;  YrotationDegrees = 0.0f;  ZrotationDegrees = 0.0f;

    float crownY = 6.0f;

    // trying to form the crown formation via 4 box location

    glm::vec3 offsets[4] = {
        glm::vec3(0.8f, crownY,  0.8f),
        glm::vec3(-0.8f, crownY,  0.8f),
        glm::vec3(-0.8f, crownY, -0.8f),
        glm::vec3(0.8f, crownY, -0.8f),
    };

    // applying necessary transforms configs

    for (int i = 0; i < 4; ++i)
    {
        positionXYZ = offsets[i];
        SetTransformations(scaleXYZ, XrotationDegrees, YrotationDegrees, ZrotationDegrees, positionXYZ);

        SetShaderColor(100.0f / 255.0f, 130.0f / 255.0f, 70.0f / 255.0f, 1.0f);  // RGB(100,130,70) AKA slightly darker than base

        m_basicMeshes->DrawBoxMesh();
    }
}
}
```

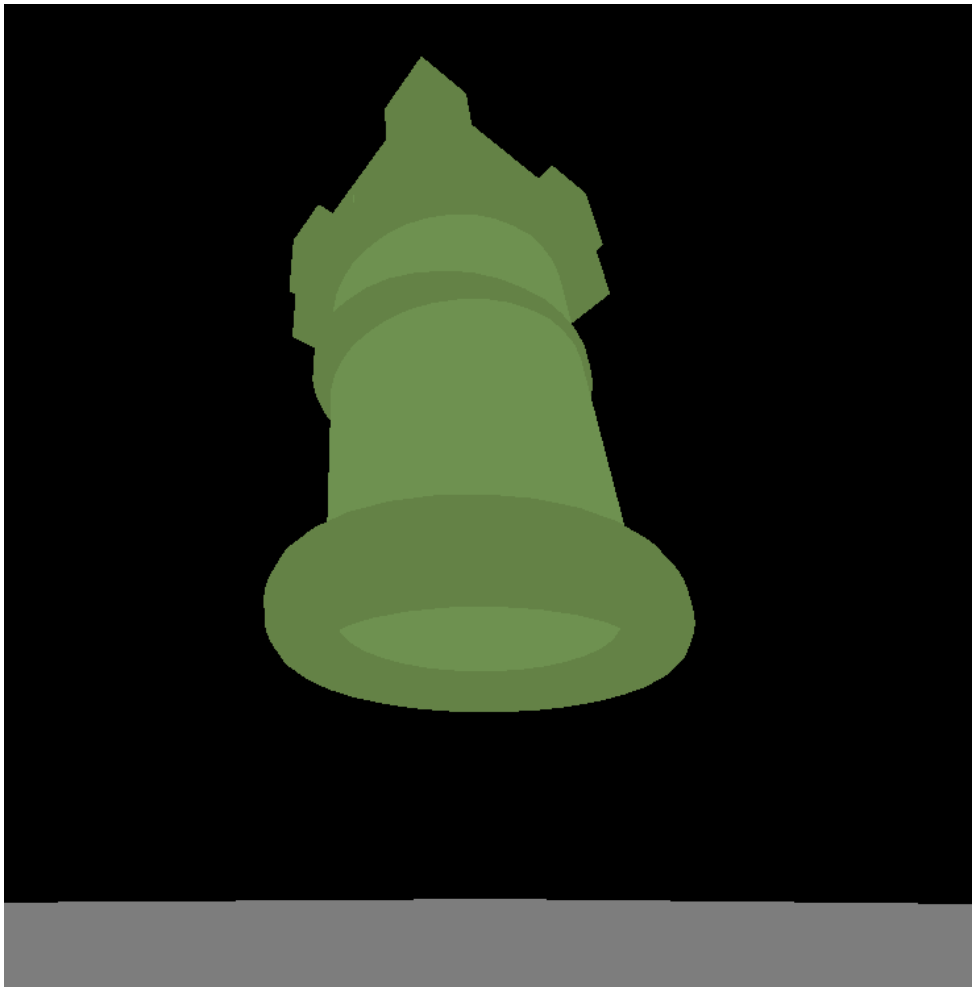
Essentially, we used a loop to place the four boxes in related positions. It took a while to do correctly. We used a combination of darker and normal greens for the piece, but once textures came along I had no need for it. That said, I decided to add a little stylish ring to all of my pieces.

4-3 Milestone Three: Interactivity in a 3D Scene:

For this milestone we worked on movement. Now, this is something I think I got pretty well. I was familiar with movements in game applications and I knew what to do to create a real responsive camera. I followed the criteria and instructions, but I also decided to think outside of the box and apply scrolling speed changes to more pieces that weren't required, such as overall camera movement. Even if it wasn't required, this made everything a lot smoother.

Making this camera is what actually allowed me to see some of the flaws that I couldn't earlier with my rook chess piece. I decided to fix those issues as they were now visible and added a plane as required. I painted it grey so it would be noticeable.

Cool Angle (NOT POSSIBLE BEFORE):



Coding Design:

I mostly followed the instructions of the previous assignment to make the controls. However, I also decided to do a couple of changes. Mainly, I used my gSpeedFactor variable that was a speed multiplier in more places than required to make it more smooth overall.

```
// movement speed multiplier (changed by scroll) MY ADDITION
float gSpeedFactor = 1.0f;
```

```
void ViewManager::Mouse_Position_Callback(GLFWwindow* window, double xMousePos, double yMousePos)
{
    // On first mouse move, initialize the last-position
    if (gFirstMouse)
    {
        gLastX = (float)xMousePos;
        gLastY = (float)yMousePos;
        gFirstMouse = false;
    }

    // Compute offset since last frame
    float xOffset = (float)xMousePos - gLastX;
    float yOffset = gLastY - (float)yMousePos; // invert Y

    // Update last positions
    gLastX = (float)xMousePos;
    gLastY = (float)yMousePos;

    // Pass scaled offsets to camera
    g_pCamera->ProcessMouseMovement(
        xOffset * gSpeedFactor,
        yOffset * gSpeedFactor
    );

    std::cout << "Mouse moved " << "X:" << xOffset << ", " << "Y:" << yOffset << std::endl;
}
```

```
void ViewManager::Mouse_Scroll_Callback(GLFWwindow* window, double xOffset, double yOffset)
{
    gSpeedFactor += static_cast<float>(yOffset) * 0.1f;
    gSpeedFactor = glm::clamp(gSpeedFactor, 0.1f, 10.0f);
    std::cout << "Speed = " << gSpeedFactor << std::endl;
}
```

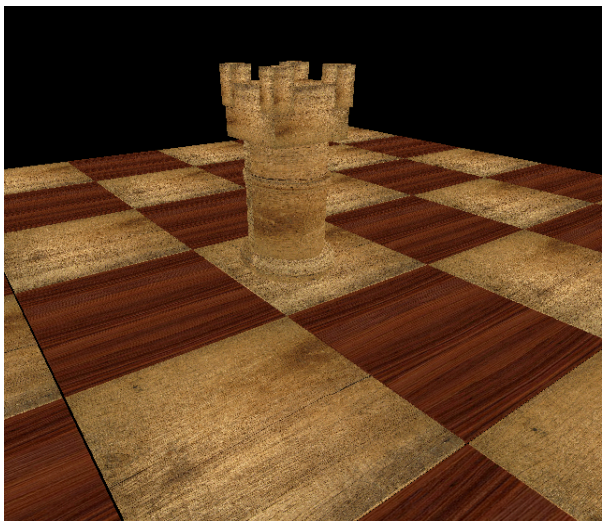
Because of our use of these standardized controls, they can easily be set up and utilized with various input devices with proper mapping such as controllers or touchscreens.

5-3 Milestone Four: Texturing Objects in a 3D Scene:

This milestone required us to add a texture to our object and use proper orientation and other practices to make it happen. I remember having some issues in this milestone due to my JPG textures not being supported. I followed everything on a helper document regarding advanced Google searches to get my textures; however, I ended up corrupting the file type and even though it said it was JPG it wasn't true JPG. This is something I didn't solve during this milestone though, I thought something was wrong with the code and wasted hours on it. Eventually, I ended up going with some default textures that were already inside the folder.

I decided to add the chessboard pattern in this milestone as well, which was quite difficult. I kept running into issues and eventually decided to create real separate tiles with a very small gap to prevent any issues regarding how it was seen. I don't have a picture of the issues, but I do have one of the end result.

Here is the result:



Coding Design:

The most notable thing I did here was my approach to the chessboard. I was quite proud of finishing it because it did take several tries to do so.

```
/* CREATING CHESSBOARD THIS LOOP TOOK WAY LONGER THAN I THOUGHT TO PERFECT*/

// I'll just create easily modifiable vars because this is taking me way too long.

float squareSize = 6.25f;
float gap = 0.05f;        // space between tiles (ended up being the solution)
float height = 0.1f;      // thickness of each tile to match our board
float uvRepeat = 1.0f;    // texture repeat per tile to match our tiles


// The main goal of this loop is to create separte tiles for different meshes and manage them with a small gap
// to avoid any flickers. I wish I documented this more but I was just running and coding solutions until
// something worked. I rushed this but I should improve this next time.

for (int x = 0; x < 8; ++x) {
    for (int z = 0; z < 8; ++z) {
        float posX = (x - 3.5f) * (squareSize + gap);
        float posZ = (z - 3.5f) * (squareSize + gap);
        float posY = height / 2.0f;

        SetTransformations(
            glm::vec3(squareSize, height, squareSize), // thickness added in Y
            0.0f, 0.0f, 0.0f,
            glm::vec3(posX, posY, posZ)
        );

        if ((x + z) % 2 == 0)
            SetShaderTexture("wood1");
        else
            SetShaderTexture("wood2");

        SetTextureUVScale(uvRepeat, uvRepeat);
        m_basicMeshes->DrawBoxMesh();
    }
}
```

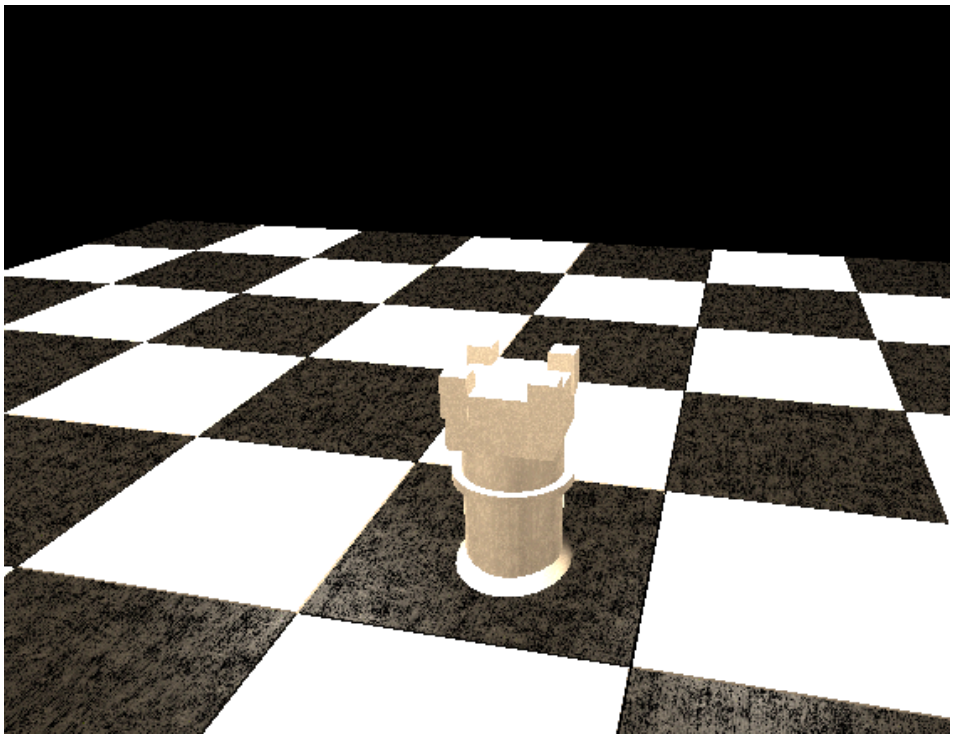
We created an 8x8 board with 6.25f square sizes and distributed our texture via an if/else loop doing a classic even/odd check (C++ classes came in handy!?)

6-3 Milestone Five: Lighting Complex Objects:

Our final milestone before our project submission had to do with lighting. We had to use different light sources to illuminate our image. I will admit this took longer than expected, and I still made some minor modifications in the final submission.

Other than the light changes, this is when I figured out the problem regarding my textures not functioning. I was determined to use wood-like textures, but I went with darker ones I modified via GIMP to make it look more like the original white/black pattern that I later modified.

Our Result:



Might be subtle, but you can see our application of materials and lighting and how some squares react to this specific angle.

Coding Design:

I mainly followed the instructions in the PDF resource for this module; however, I quickly found out the values utilized, and some of the lights utilized, did not look good at all. For full visibility

I had to come up with the North/South/West/East lights (didn't name them that until the final submission), which were lights that came from every corner of the chessboard.

```
void SceneManager::SetupSceneLights()
{
    // We use four lights, one in each board direction and location to illuminate the entire scene.
    // Top-front-right light
    m_pShaderManager->setVec3Value("lightSources[0].position", 10.0f, 14.0f, 10.0f);
    m_pShaderManager->setVec3Value("lightSources[0].ambientColor", 0.02f, 0.02f, 0.02f);
    m_pShaderManager->setVec3Value("lightSources[0].diffuseColor", 0.15f, 0.15f, 0.15f);
    m_pShaderManager->setVec3Value("lightSources[0].specularColor", 0.08f, 0.08f, 0.08f);
    m_pShaderManager->setFloatValue("lightSources[0].focalStrength", 32.0f);
    m_pShaderManager->setFloatValue("lightSources[0].specularIntensity", 0.04f);

    // Top-front-left light
    m_pShaderManager->setVec3Value("lightSources[1].position", -10.0f, 14.0f, 10.0f);
    m_pShaderManager->setVec3Value("lightSources[1].ambientColor", 0.02f, 0.02f, 0.02f);
    m_pShaderManager->setVec3Value("lightSources[1].diffuseColor", 0.15f, 0.15f, 0.15f);
    m_pShaderManager->setVec3Value("lightSources[1].specularColor", 0.08f, 0.08f, 0.08f);
    m_pShaderManager->setFloatValue("lightSources[1].focalStrength", 32.0f);
    m_pShaderManager->setFloatValue("lightSources[1].specularIntensity", 0.04f);

    // Top-back-right light
    m_pShaderManager->setVec3Value("lightSources[2].position", 10.0f, 14.0f, -10.0f);
    m_pShaderManager->setVec3Value("lightSources[2].ambientColor", 0.02f, 0.02f, 0.02f);
    m_pShaderManager->setVec3Value("lightSources[2].diffuseColor", 0.15f, 0.15f, 0.15f);
    m_pShaderManager->setVec3Value("lightSources[2].specularColor", 0.08f, 0.08f, 0.08f);
    m_pShaderManager->setFloatValue("lightSources[2].focalStrength", 32.0f);
    m_pShaderManager->setFloatValue("lightSources[2].specularIntensity", 0.04f);

    // Top-back-left light
    m_pShaderManager->setVec3Value("lightSources[3].position", -10.0f, 14.0f, -10.0f);
    m_pShaderManager->setVec3Value("lightSources[3].ambientColor", 0.02f, 0.02f, 0.02f);
    m_pShaderManager->setVec3Value("lightSources[3].diffuseColor", 0.15f, 0.15f, 0.15f);
    m_pShaderManager->setVec3Value("lightSources[3].specularColor", 0.08f, 0.08f, 0.08f);
    m_pShaderManager->setFloatValue("lightSources[3].focalStrength", 32.0f);
    m_pShaderManager->setFloatValue("lightSources[3].specularIntensity", 0.04f);

    // Enable lighting globally to avoid any issues (make sure it's on)
    m_pShaderManager->setBoolValue("bUseLighting", true);
}
```

7-1 Final Project Submission

Now, this is our final milestone, our actual submission. To perfect the submission I had a lot of work to do. This included well-documented comments and better-structured code through the use of facilitating functions and variables, as well as many adjustments to perfect the new look of the scene.

In this milestone I changed and edited the textures utilized and decided to go with a wooden chessboard. I additionally made a frame/border for aesthetic purposes. As a result of this change, I had to change the lights to match them a bit better.

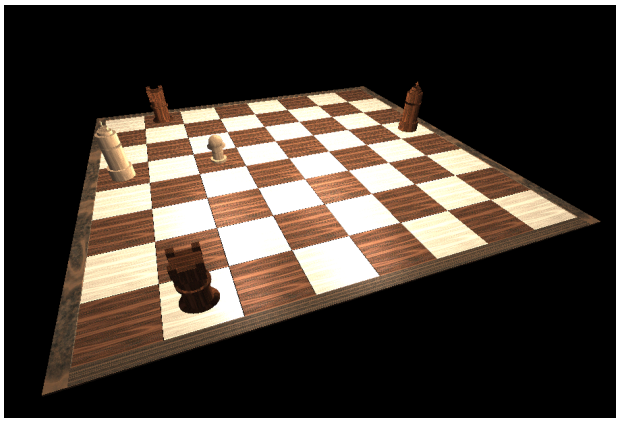
I also made my Rook that I drew in the RenderScene() into its own function with all required parameters including size, color, and positional changes. Once I did that I also made a function for the other pieces and also created them, which included the King and Pawn. I really improved everything in this milestone and could talk about many instances of my changes. Fortunately I documented most of it in my comments, so I will just talk about the newest feature that makes everything a lot easier for my purposes.

My NotationToWorld function:

```
/*
* Notation To World took a few tries so I didn't document everything but here is the gist of it. We want to use chess notation
* which usually goes like "e4" or "b4" for specific positions. I originally had a real world coordinate but the parameters and
* arguments got messy and I knew it would be an issue so I decided to do this. A chessboard is 8x8 we took our specific size
* which is 6.25 per square and the gaps I used to prevent weird collisions and put it into this function. I used chars and arrays
* to identify the positions and as the code shows I use known defaults to make it so that it converts correctly. This makes it easy
* to use and position pieces.
*/
glm::vec3 SceneManager::NotationToWorld(const std::string& square, float pieceHeight /*=0.0f*/)
{
    if (square.size() != 2) return glm::vec3(0.0f);

    char column = tolower(square[0]); // column 'a' to 'h'
    char row = square[1];          // row '1' to '8'

    // Convert column 'a' to 'h' -> 0 to 7 b/c 8x8
    int xIndex = column - 'a';
    // Convert row '1' to '8' -> 0 to 7 b/c 8x8
    int zIndex = row - '1';

    float squareSize = 6.25f;
    float gap = 0.05f;

    // Translating to world coordinates
    float posX = (xIndex - 3.5f) * (squareSize + gap);
    float posZ = (zIndex - 3.5f) * (squareSize + gap);

    return glm::vec3(posX, pieceHeight, posZ);
}
```

The comments explain it; essentially, with this function we can grab the real world coordinates of our board and apply them to our parts in function calls via its notation. So we can position a created or called piece into “e4” just by adding it to the parameter. Definitely my most notable addition other than the countless fixes and modifications.

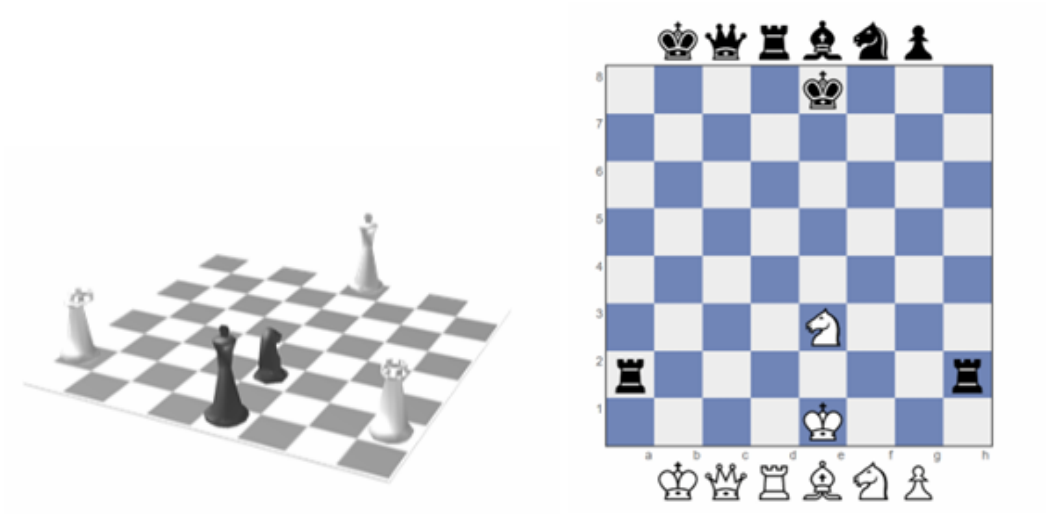
Here is our final result:



Final Reflection:

After comparing our final result to the original goal below, I would say we have satisfied all requirements and made my vision a reality.

Original Goal:



Working on this project was actually quite fun and I enjoyed documenting what I recalled in this document. To summarize, we managed to successfully create low-polygon 3D representations of my chessboard aka real-world objects, apply accurately projected textures to those models, apply lighting to create a polished visualization of the scene, place objects appropriately, create navigation inputs with a smooth camera that utilized WASD and QE keys and other conventions and therefore applying nuanced camera controls as well as creating perspective/orthographic options, and commenting, formatting, the code using best practices that applied to my situation.