Zense Project

Rudransh Dixit IMT2020056

3D Renderer

NOTE: PLEASE READ README.MD TO KNOW HOW TO SETUP THE PROJECT ON YOUR MACHINE.

Motivation

The motivation to build the 3D Renderer was from wanting to learn to build the Game Engine, I wanted to start small by understanding the basics of Graphics Programming. The easiest graphics API is OpenGL, so I decided to learn OpenGL basics while also learning the basics of the graphics pipeline. Using that I was able to build this 3D Renderer by abstracting the OpenGL specifics in various classes. The initial motivation to build a Game Engine still remains and I will try to make it into a game engine in the future. Since many game engines already exist and are way better and advanced then, why another one? I am making this to learn the basics of computer graphics so that in the future I can contribute to open-source game engines like Godot, Blender (not a game engine but basics of 3D graphics are also used here) and eventually build a career as a game developer.

Features

Some of the main features of the 3D renderer are: -

1. It can render models out of blender, the feature is there in Model and Mesh classes but after exporting the blender data to .obj format we need to make some changes to make the mapping to correct textures, as the class uses relative pathing but usually blender exports with absolute pathing.
2. We can add any number of objects and lights of various shapes like squares, rectangles, spheres, and any basic shape by generating vertex data and index data. These both can be fed into a scene.
3. We can add shaders to these objects very easily, we just need the shaders and need to load them in a Scene and then map them to correct object.
4. We can add various lights, which may or may not have textures but we need to make sure to create appropriate shaders for them and take these lighting into account in shaders of objects as well. We can also add various types of lights, such as directional light and spot light with some basic changes in shaders and code.

5. All the things, textures, vertex data, index data, and shaders are separated into different folders and are read automatically to construct the scene.

Libraries Used

I have used many libraries for various things in the project, they are listed below with the reasons why I have used then and what feature do they help with

1. GLFW -> This library is used for setting up the OpenGL context and creating the window. This allows us to abstract the window creation instead of using win API to create window. GLFW is also multi-platform.
2. GLAD -> This library is used for getting the OpenGL functions. Since OpenGL is implemented by graphics driver manufacturer in a dll library so we need to query for these functions at runtime, fortunately GLAD does this automatically allowing us to use OpenGL function calls without any trouble.
3. GLM -> This is a math library for maintaing various matrices and vectors such as view, projection, lookat matrices and positions vectors and many more. Also this library is made to specifically comply with OpenGL specifications so we need not change matrices to match with OpenGL specifications.
4. Dear Imgui -> This is a basic library for setting up GUI interface in OpenGL context. We use this for setting some features such as scene loading, and in future I plan to implement feature so that we can directly control the scene using the dear imgui interface.
5. Assimp -> This library can load .obj files and load models from the 3D model creators such as blender, maya etc to our 3D renderer. Not used currently but can be used for loading model by adding a object of Model class in scene.
6. Stbi image library -> This is image loading header library, it can load any image extension, then we can pass the image data to OpenGL textures to store it in GPU.

Things Learnt

Some of the important things that I have learnt from this project:

1. Way OpenGL works. Since OpenGL is one of the easiest graphics API but the fundamentals learnt here about Vertex Buffers, Vertex Array, Index Buffers etc. and graphics pipeline will carry over to any other graphics pipeline such as Vulkan or DirectX. This gives a good understanding of how GPU does things, and how we can use these APIs to create complex graphics.

2. A lot more about C++. Since I made the project in C++ and tried using some advanced and newer features of C++ such as shared pointers, little bit of OOPs, adding dynamic and static libraries, adding more include directories etc.
3. Maths. Since computer graphics uses a lot of matrices, vectors and we just studies this stuff in last semester, it really taught me how we can practically apply mathematics. Understanding maths behind projection matrix taught a lot about connecting the 3D graphics with matrices, it was not necesarry as glm library has function to create 2 types of projection matrix but understanding the details gives me in depth knowledge as to how I can change it to get various effects like changing pov in projection matrix create a zoom effect. So This gives a lot of information about understanding the practical aspects of maths in computer graphics.

Problems Faced

Some problems faced by me while making this project are:

1. Getting the libraries setup. One of the most frustrating thing was to setup libraries. In C++ there are 2 types of library dynamicaly linked and statically linked. They were hard to get working and understand how they exactly work.
2. Setting up proper framework for scene system. Since scene class handles most of stuff it was very problematic to figure out which things will go where. Like how will I map shaders, textures to objects and light. Also light need to be a object but with more features, a lot of stuff still is very counter intuitive to work with, and I still plan on improving the framework to make things more intuitive for others to load scenes easily.
3. Getting models to load. Since models from blender have lot of data into them with textures placed at various places so they are very hard to import sucessfully.
4. Well the biggest problem was RAM usage. Since for the most part I was using 8k textures when they were decompressed by image loading library the ram usage shot upto 2.5GB. Well for this basic engine we do not want that high ram usage 🤣. So I added up adding 2k textures which reduced ram usage. The RAM usage is high due to some bug in Nvidia driver as when we send the textures to VRAM I clear the RAM texture but for some reason OpenGL saves the copy of data. To takle this problem there is a texture compression system in OpenGL but it is very slow. To takle this the best thing to do is to used pre compressed textures, or compress them on cpu then send to gpu. I do plan on adding this feature in future. Right now to reduce RAM usage it is recommended to use low quality textures.
5. Sorry but this has no shadows as of yet. I do plan on adding shadow maps next.

Future Plans

This project is no where near polished and a lot of performance improvements, quality improvements are needed to be made. Some of my future plans are:-

1. Shadow Maps. Shadows are very improtant for a scene to look real and I do plan on adding shadows as soon as possible.
2. Batch Rendering. My laptop is pretty high end so engine works fine on this machine but it will start to struggle once I start adding more vertex data, for this reason a technique called batch rendering is used. This makes it so we draw multiple objects in single draw call. SInce draw calls to OpenGL are pretty slow so if we reduce the number of draw calls we can get significant performance upgrade. Batch Rendering allows us to achieve this optimization.
3. Automated Shader generation. Since we cannot expect every game developer to worry about shaders as everyone might not know GLSL so I do plan on doing this automatically, as then they would not have to worry about lighting maths and can be done automatically. This will allow them to focus more on making good games.
4. PHYSICS. There is no physics yet so I do plan on adding it. Also with this I would like to mention a optimization of adding quaternions. They are better way to do rotation, currently I am performing rotations using euler angles which has some limitations like gimbal lock. So this will make engine more robust.
5. Adding advanced lighting.

Issues

Some of the Issues that might be faced are:-

1. Making a new Scene is pain, as we need to setup shader, vertex data, index data, textures manually and create a function to link everything together. I do plan on fixing this asap.
2. RAM usage is high, as of current state, app uses about 500 MB ram which is lot for what I am trying to achieve. This could be due to app running in debug mode in visual studio, this might get fixed in release mode as texture copying would be avoided.
3. Windows only. As of now when I tried running this on a system without Visual Studio installed it gives out some DLL errors, even when the build is done in release mode. So if you want to run this application you need to have Visual studio installed. I tried searching for many fixes but as of yet no success. For linux and macOS I do plan on adding them to work. This could be done by using Cmake but not sure will have to research more on this.
4. Clunky GUI. I do have to setup proper gui to make this application easier to use.

Sources

Some resources which helped me a lot are:-

1. Learn OpenGL, extensive tutorial resource for learning Modern OpenGL :- This is just best site to learn opengl. I learnt almost all of the things from there except some of the indepth maths as the website is targetted to beginers who don't have much maths background thus they seem to abstract everything, so for maths I read the sources mentioned on this website.
2. The Cherno - YouTube -> This guy is absolute legend, he has videos on OpenGL, making game engine, and C++. I learn most of the C++ from him. He has videos on batch rendering as well which I intend to add on to this project asap.
3. OpenGL (songho.ca) -> I understood most of the advanced maths from this website. He has very nice tutorial on projection matrix, normalized coordinates, homogeneous coordinates.
4. docs.gl -> Documentation for OpenGL. Very handy.

Thank you For reading this report. I hope you liked this project. This is something I am passionate about, so I tried my best to get a finished application in as good working condition as possible. I made this project because I wanted to learn OpenGL properly for a long time and I do plan on continuing this project's development to make it a Game Engine by even adding some advanced graphics API like vulkan and directX. Thank you zense for providing me an opportunity to show what I can make.