MAJOR PROJECT

AI-Enabled Game Engine

Bachelor of Engineering

In

Computer Science Engineering | Artificial Intelligence & Robotics

Proposed By

RIDHEESH AMARTHYA
RAHUL KUMAR
ARYAN BHARDWAJ

Enrollment No.

A505112518005
A50105218083
A505112518007

Under the guidance of

Dr Charu Jain
Assistant Professor

Dr Aarti Chugh
Assistant Professor

Department of Computer Science & Engineering
Amity School of Engineering & Technology
Amity University Haryana

Department of Computer Science and Engineering

Amity School of Engineering and Technology

Declaration

We, ***Ridheesh Amarthya A505112518005, Rahul Kumar A50105218083, Aryan Bhardwaj A505112518007***, student of Bachelor of Technology in Department of Computer Science and Engineering, Amity School of Engineering and Technology, Amity University Haryana, hereby declare that We are fully responsible for the information and results provided in this project report titled ***AI ENABLED GAME ENGINE"*** submitted Department of Computer Science and Engineering, Amity School of Engineering and Technology, Amity University Haryana, Gurgaon for the partial fulfillment of the requirement for the award of the degree of ***Bachelor of Technology in Computer Science and Engineering | Artificial Intelligence and Robotics***. We have taken care in all respects to honor intellectual property rights and have acknowledged the contributions of others for using them. We further declare that in case of any violation of intellectual property rights or copyrights, We as candidates will be fully responsible for the same. My supervisor, Head of the department, and the Institute should not be held for full or partial violation of copyrights if found at any stage of my degree.

Rahul Kumar	**Aryan Bhardwaj**	**Ridheesh Amarthya**
A50105218083	**A505112518007**	**A505112518005**

Department of Computer Science and Engineering

Amity School of Engineering and Technology

Certificate

This is to certify that the work in the project report entitled ***"AI ENABLED GAME ENGINE"*** by ***Ridheesh Amarthya A505112518005, Rahul Kumar A50105218083, Aryan Bhardwaj A505112518007.*** is a bonafide record of project work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering in the Department of Computer Science and Engineering, Amity School of Engineering and Technology, Amity University Haryana, Gurgaon. Neither this project nor any part of it has been submitted for any degree or academic award elsewhere.

Signature of Supervisor(s)

Dr Charu Jain	**Dr Aarti Chugh**
Assistant Professor	**Assistant Professor**

ABSTRACT

Game engines are primarily designed for the development of video games but can also be used as rendering engines for 2D and 3D graphics. They include all relevant libraries and supported programs. Developers can use them to construct games for game consoles and other types of computer systems. They also aid in the porting of games to multiple platforms. Producers of game engines decide how they allow users to utilize their products. Similar to Game engines there are Rendering Engines used for computer-generated graphics. Both these types of software are difficult to run on everyday machines, especially laptops, limiting their use to professionals with dedicated systems.

The game engine we will be designing will have all the standard features of a rendering engine and all the code will be open source and published in a public domain open to all. This will be done under the Apache License 2.0 which enables the editing and re-using of the code with appropriate acknowledgements. The AI aspect of the engine requires rendering of visual data at a lower resolution to improve performance and then using trained Deep Networks to upscale the image to the native resolution of the user's monitor. The AI will ensure that upscaling will be adding detail rather than averaging the nearby pixels. This will enable the AI-Enabled Game Engine to be accessible to all.

The developed game engine named the Tookivi Rendering Engine (Editor) met all expected performance metrics, averaging 60+ frames per second on a generic everyday system. The AI upscaler was able to enhance the image up to 4 times the resolution with an average frame rate ranging between 15 to 20 frames per second. The expected performance of 30+ frames per second needs more optimization and is left for future development under the future scope of the project.

LIST OF FIGURES

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Game Engine

A game engine is a software used for the development of games or in some cases known for its use as a real-time rendering engine for computer graphics. Some notable engines include the Unreal Engine and Unity Game engine. These game engines are known to be "heavy" applications. They have extensive requirements such as high system memory and a dedicated graphics processor, because of this, not every system can run these engines. Limiting them to professionals and those who can afford the equipment.

"AI-Enabled Game Engine" is a fully functional game engine designed to be able to render real-time graphics for gameplay purposes, this functionality is enhanced by Artificial Intelligence in terms of enhanced performance on lower specification machines.

The Game Engine has the following functionality:

- Texture management
- Layers
- Input management
- Shaders
- Camera control
- 3D renderer
- Batch Rendering
- Model Loading

1.2 Image Upscaling

There are several algorithms for the upscaling of images, one of the most common ones is the averaging or multiplication of nearby pixels to approximate the image. This approach however cannot add detail to the larger image but rather just increases the size. This is the "dumb" or the standard way of upscaling images. With the advent of neural networks and AI, it is now possible to upscale images using a trained deep network in real-time. The model will add detail to the image rather than simply averaging the pixels, this is done based on its training on similar data, preferably on the target game. The artificial intelligence system is enabled by machine learning and deep learning algorithms to implement a system allowing the game engine to render images at a lower resolution thus, enabling smooth performance on lower specification machines, the image will be upscaled in real-time to the native screen resolution of the display.

1.3 Objective

The major objectives of the project are to develop a game engine from its very foundation and implement AI upscaling features. The project will hope to achieve the following objectives:

- Implement a fully functional Rendering Engine
- Have a user-friendly UI
- The engine should support importing of textures and .OBJ files
- Python overlay to upscale the output of the game engine
- Performance of 30+ FPS or a 20% performance improvement. (Whichever is higher)

The main goal of the project was the implementation of a system that will allow users to run heavy rendering and gameplay applications on their lower specification hardware. This is achieved by taking advantage of the fact that the majority of the work is done by the graphics card while the CPU has the performance to spare. Our engine implements a Deep Layered Network Trained on Lower resolution data and uses it to upscale to the native viewing resolution of the user's monitor. This if implemented properly will show a noticeable improvement in the performance of lower specification or older machines thus, extending their life cycle and making them more productive.

The second major objective of the program is to understand enterprise-level programming standards such as the use of header files and namespaces in C++ which will help give the learners an edge in a competitive hiring environment. The implementation of a Game Engine is no easy task, it takes an in-depth knowledge of Graphics APIs and Rendering Pipeline to implement an efficient system that will be able to perform well. This project forces a better understanding of modern computer graphics.

CHAPTER 2

BACKGROUND OF PROJECT

2.1 Background Study

Game engines are tools enabling easy, fast, and reliable development of games on any platform. With a forecasted industry worth exceeding $250 Billion, game development is an in-demand skill, are game engines.[1] Game engines are real-time rendering software that allows them to be used not only for game development but also as a 3D rendering tool for graphic design. There are two major types of rendering:

- 2D rendering
- 3D rendering

3D Game engines are often expected to be able to render both. Another expectation from a game engine is to be able to create platform-independent games. Games developed should be able to run on Mobile Phones, Computers, and Laptops no matter the Operating System and have to be limited only by the hardware. Some game engines adopt visual scripting, and among these tools are such as GameMaker and GDevelop.[2]

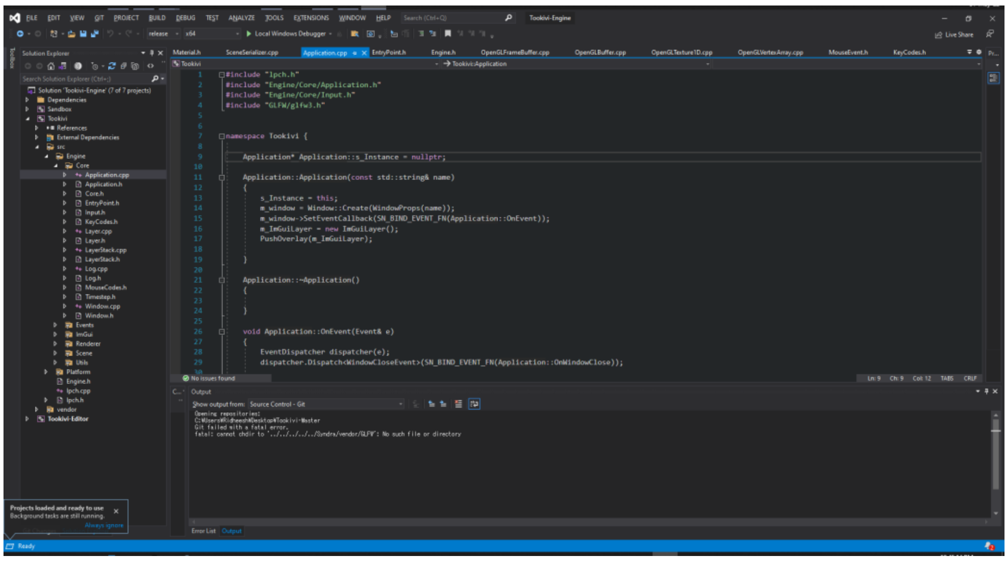
Some notable game engines are Unreal Engine used to make games such as Valorant and Unity which is the more popular game engine for independent developers. Major game engines come at varying prices, whether it may be in form of subscription fees or license payments. Some game engines like the one we are developing are open source under the GNU open use license.[2]

Hardware limitation has been a major limiting factor, as older hardware is not fast enough to be able to render the high-resolution images and textures demanded by the game. This limitation can be overcome by the use of Artificial Intelligence,[5] machine learning can be used to upscale the images from a lower resolution to a higher resolution.[6] The challenge is latency. The upscaling is an extra step in rendering, games render at an average speed of 60 frames per second, to introduce an upscaling algorithm that can keep up with the highly demanding frame rates is a challenging task. Latency must be kept at a minimum.

2.2 Choice of an IDE (Integrated Development Environment)

The IDE while inconsequential in most projects plays a vital role in enterprise-level programming practices,

- Keeping that in mind Visual Studio 2019 was the selected IDE for the C++ object-oriented programming as it is the industry standard for graphics programming. The selection of visual studio as the IDE enabled the project to be built using Dynamic Linked Libraries where the header files can be abstracted. This enables efficient Programming.
- Similarly for the AI aspect of the AI ENABLED GAME ENGINE the choice of IDE was PyCharm as it has proven to be extremely similar to Visual Studio which we have experience with. As Python is an interpreted language there are a few notable differences, the most significant of these being the lack of the compilation step.



```cpp
#include "lpch.h"
#include "Engine/Core/Application.h"
#include "Engine/Core/Input.h"
#include "GLFW/glfw3.h"


namespace Tookivi {

	Application* Application::s_Instance = nullptr;

	Application::Application(const std::string& name)
	{
		s_Instance = this;
		m_window = Window::Create(WindowProps(name));
		m_window->SetEventCallback(SN_BIND_EVENT_FN(Application::OnEvent));
		m_ImGuiLayer = new ImGuiLayer();
		PushOverlay(m_ImGuiLayer);

	}

	Application::~Application()
	{

	}

	void Application::OnEvent(Event& e)
	{
		EventDispatcher dispatcher(e);
		dispatcher.Dispatch<WindowCloseEvent>(SN_BIND_EVENT_FN(Application::OnWindowClose));
```

```
Opening repositories:
C:¥Users¥Ridheesh¥Desktop¥Tookivi-Master
Git failed with a fatal error.
fatal: cannot chdir to '../../../../../Syndra/vendor/GLFW': No such file or directory
```

Figure 2.1 Visual Studio 2019 IDE sample program

CHAPTER 3

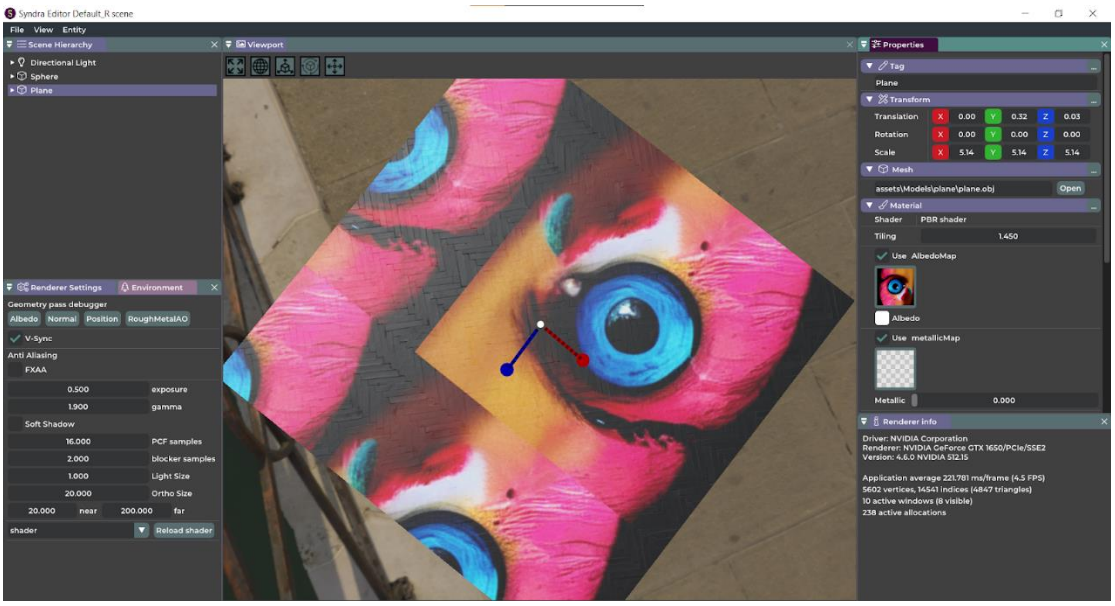
TOOLS AND TECHNOLOGIES

3.1 Shaders

Shaders are written in the C-like language GLSL. GLSL is tailored for use with graphics and contains useful features specifically targeted at vector and matrix manipulation. Shaders always begin with a version declaration, followed by a list of input and output variables, uniforms, and its main function. Each shader's entry point is at its main function where we process any input variables and output the results in its output variables.

3.2 Textures

A texture is a 2D image (even 1D and 3D textures exist) used to add detail to an object; think of a texture as a piece of paper with a nice brick image (for example) on it neatly folded over your 3D house so it looks like your house has a stone exterior. Because we can insert a lot of detail in a single image, we can give the illusion the object is extremely detailed without having to specify extra vertices.



```
V-Sync: ✓
Anti Aliasing
FXAA
0.500     exposure
1.900     gamma
Soft Shadow
16.000    PCF samples
2.000     blocker samples
1.000     Light Size
20.000    Ortho Size
20.000  near  200.000  far
shader
```

```
Tag
Plane
Transform
Translation  X 0.00  Y 0.32  Z 0.03
Rotation     X 0.00  Y 0.00  Z 0.00
Scale        X 5.14  Y 5.14  Z 5.14
Mesh
assets\Models\plane\plane.obj
Material
Shader   PBR shader
Tiling   1.450
✓ Use AlbedoMap
Albedo
✓ Use metallicMap
Metallic  0.000
```

```
Renderer info
Driver: NVIDIA Corporation
Renderer: NVIDIA GeForce GTX 1650/PCIe/SSE2
Version: 4.6.0 NVIDIA 512.15

Application average 221.781 ms/frame (4.5 FPS)
5602 vertices, 14541 indices (4847 triangles)
10 active windows (8 visible)
238 active allocations
```

Figure 3.1 Loaded Textures in Tookivi Game Engine

3.3 Transformation

Any game object whether it is in 2D, or 3D can be performed with three transformations: translation, rotation, and scaling. [3]

3.3.1 Translation Transformation

Translation relates to the movements and the position of a game object and is specified by a 2D vector that the person in the previous section moved on the map. The important feature of a translation is whenever the x, y, or z values of an object are modified. [3] The values can be changed all at once with a vector or one at a time. To move an object in the x-direction by 8 is:

3.3.2 Rotation Transformation

A game object can be rotated about its x, y, or z-axis of the world x, y, or z axes. Combined rotations are also supported. Rotations in 3D are specified with an angle and a rotation axis. The angle specified will rotate the object along the rotation axis given. Try to visualize this by spinning your head to a certain degree while continually looking down a single rotation axis.

A rotation matrix is defined for each unit axis in 3D space where the angle is represented as the theta symbol θ.[3]

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ \cos\theta \cdot y - \sin\theta \cdot z \\ \sin\theta \cdot y + \cos\theta \cdot z \\ 1 \end{pmatrix}$$

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot x + \sin\theta \cdot z \\ y \\ -\sin\theta \cdot x + \cos\theta \cdot z \\ 1 \end{pmatrix}$$

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot x - \sin\theta \cdot y \\ \sin\theta \cdot x + \cos\theta \cdot y \\ z \\ 1 \end{pmatrix}$$

Figure 3.2 Rotation of matrices

3.3.3 Scaling Transformation

When we're scaling a vector we are increasing the length of the arrow by the amount we'd like to scale, keeping its direction the same. Since we're working in either 2 or 3 dimensions we can define scaling by a vector of 2 or 3 scaling variables, each scaling one axis (x, y, or z).

3.4 Camera and View-Space

When we're talking about camera/view space we're talking about all the vertex coordinates as seen from the camera's perspective as the origin of the scene: the view matrix transforms all the world coordinates into view coordinates that are relative to the camera's position and direction. [3] To define a camera we need its position in world space, the direction it's looking at, a vector pointing to the right, and a vector pointing upwards from the camera. [3] A careful reader may notice that we're going to create a coordinate system with 3 perpendicular unit axes with the camera's position as the origin.

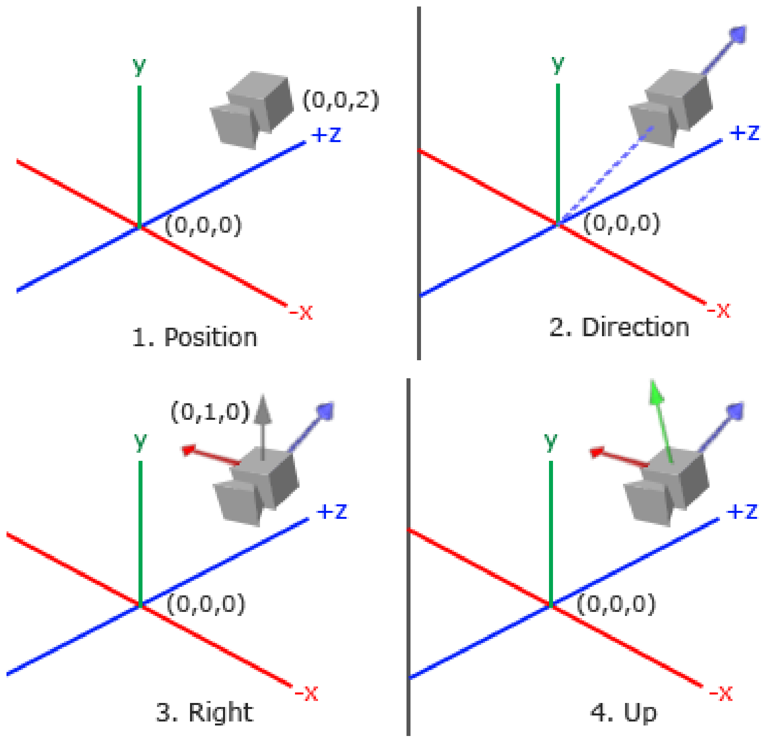


Figure 3.3 Camera perspective and view space

3.5 C++ Dependencies

3.5.1 OpenGL

OpenGL (Open Graphics Library) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering.[8]

OpenGL is mainly considered an API (an Application Programming Interface) that provides us with a large set of functions that we can use to manipulate graphics and images.[8]

Routines in the OpenGL graphics library can be called from most major languages. Window management is not supported in the basic version of OpenGL but there are additional libraries built on top of OpenGL that you can use for many of the things which OpenGL itself does not support.

3.5.2 GLFW

GLFW is a library, written in C, specifically targeted at OpenGL. GLFW gives us the bare necessities required for rendering goodies to the screen. It allows us to create an OpenGL context, define window parameters, and handle user input, which is plenty enough for our purposes.

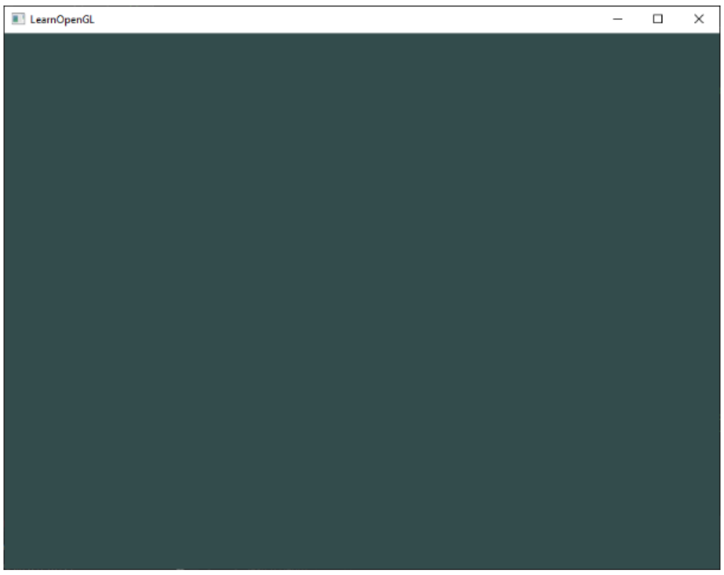


Figure 3.4 Empty GLFW window init

3.5.3 GLAD

OpenGL is only really a standard/specification it is up to the driver manufacturer to implement the specification to a driver that the specific graphics card supports. Since there are many different versions of OpenGL drivers, the location of most of its functions is not known at compile-time and needs to be queried at run-time. It is then the task of the developer to retrieve the location of the functions he/she needs and store them in function pointers for later use.

GLAD is an open-source library that manages all that cumbersome work we talked about. GLAD has a slightly different configuration setup than most common open source libraries. GLAD uses a web service where we can tell GLAD for which version of OpenGL we'd like to define and load all relevant OpenGL functions according to that version.

3.5.4 imGUI

imGui is a bloat-free graphical user interface library for C++. It outputs optimized vertex buffers that you can render anytime in your 3D pipeline-enabled application. It is fast, portable, renderer agnostic, and self-contained (no external dependencies). imGui is designed to enable fast iterations and to empower programmers to create content creation tools and visualization / debug tools (as opposed to UI for the average end-user). It favours simplicity and productivity toward this goal and lacks certain features normally found in more high-level libraries.

imGui is particularly suited to integration in games engines (for tooling), real-time 3D applications, fullscreen applications, embedded applications, or any applications on consoles platforms where operating system features are non-standard.

3.5.5 GLM

GLM is a C++ mathematics library for graphics software based on the OpenGL Shading Language (GLSL) specification. GLM provides classes and functions designed and implemented with the same naming conventions and functionalities as GLSL so that when a programmer knows GLSL, he knows GLM as well which makes it easy to use.

This library works perfectly with OpenGL but it also ensures interoperability with other third party libraries and SDK. It is a good candidate for software rendering (raytracing / rasterisation), image processing, physics simulations and any development context that requires a simple and convenient mathematics library.

```
#include <glm/vec3.hpp> // glm::vec3
#include <glm/vec4.hpp> // glm::vec4
#include <glm/mat4x4.hpp> // glm::mat4
#include <glm/gtc/matrix_transform.hpp> // glm::translate, glm::rotate,
glm::scale, glm::perspective


glm::mat4 camera(float Translate, glm::vec2 const & Rotate)
{
glm::mat4 Projection = glm::perspective(glm::radians(45.0f), 4.0f / 3.0f,
0.1f, 100.f);
glm::mat4 View = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f, -
Translate));
View = glm::rotate(View, Rotate.y, glm::vec3(-1.0f, 0.0f, 0.0f));
View = glm::rotate(View, Rotate.x, glm::vec3(0.0f, 1.0f, 0.0f));
glm::mat4 Model = glm::scale(glm::mat4(1.0f), glm::vec3(0.5f));
return Projection * View * Model;
}
```

Figure 3.5 GLM Sample program

3.6 Python Dependencies

3.6.1 TensorFlow

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on the training and inference of deep neural networks. TensorFlow was developed by the Google Brain team for internal Google use in research and production. It can be used with many programming languages such as C++, Python, and Java just to name a few. It was released in 2015 with its latest stable release being updated in 2022. [10]

3.6.2 NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was created by Jim Hugunin with contributions from several other developers. NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms are written for this version of Python often run much slower than compiled equivalents due to the absence of compiler optimization. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays; using these requires rewriting some code, mostly inner loops, using NumPy.[11]

3.6.3 OpenCV (CV2)

OpenCV (Open Source Computer Vision Library) is an open-source software library for computer vision and machine learning. OpenCV was created to provide a shared infrastructure for applications for computer vision and to speed up the use of machine perception in consumer products. OpenCV, as a BSD-licensed software, makes it simple for companies to use and change the code. There are some predefined packages and libraries that make our life simple and OpenCV is one of them.[9]

Gary Brodsky invented OpenCV in 1999 and soon the first release came in 2000. This library is based on optimized C / C++ and supports Java and Python along with C++ through an interface. The library has more than 2500 optimized algorithms, including an extensive collection of computer vision and machine learning algorithms, both classic and state-of-the-art. Using OpenCV it becomes easy to do complex tasks such as identify and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D object models, generate 3D point clouds from stereo cameras, stitch images together to generate an entire scene with a high-resolution image and many more.[9]

3.7 Requirements

The game engine section of the project is implemented in Visual Studio 2019, users who wish to build their own engine need to download and run the premake file included with the game engine on GIT. This will generate all the needed files with the paths generated to the user's computer. This script is named “GenerateProjects.bat” and will download all the needed dependencies, not including the Vulkan SDK which we expect the user to install separately as it is several gigabytes and requires an installation to move the needed .dll to system32 for the smooth performance of the Game Engine.

The Engine Run Time and Compile-Time were tested on the following kitted machine:

- Intel i5 4690K 3.9 GHz
- Nvidia GeForce GTX 960 2GB VRAM
- DDR3 8 GB RAM clocked at 1844 MHz
- 1920×1080 60 Hz Display

3.7.1 Run Time Hardware Requirements

- Processor: 1.3 GHz.
- RAM: 4 GB
- AMD Vega or Better Graphics Processing Unit
- Free Space required on hard disk: 2 GB

3.7.2 Run-Time Software Requirements

- C++ Runtime v11.0 framework package for Desktop Bridge (Project Centennial)
- Python3 run-time environment

3.7.3 Compile Time Hardware Requirements

- Processor: 3.5 GHz.
- RAM: 8 GB
- AMD Vega or Better Graphics Processing Unit
- Free Space required on hard disk: 10 GB

3.7.4 Compile Time Software Requirements

- C++ 11 Compiler and Python3 Interpreter
- GLAD, imGUI, VulkanSDK, GLFW Libraries and OpenCV, TensorFlow

CHAPTER 4

METHODOLOGY

4.1 Tookivi Game Engine

Game engines are primarily designed for the development of video games, but can also be used as rendering engines for 2D and 3D graphics. They include all relevant libraries and supported programs. Developers can use them to construct games for game consoles and other types of computer systems. They also aid in the porting of games to multiple platforms. Producers of game engines decide how they allow users to utilize their products. The game engine we will be designing will have all the standard features of a rendering engine but all the code will be open source and published in a public domain open to all. This will be done under the Apache License 2.0 which enables the editing and re-using of the code with appropriate acknowledgements.

4.2 AI Image Upscaling

Image upscaling algorithms when viewed as a function of input and output is simple, the function takes in the input of a low-resolution image and outputs a higher-resolution “upscaled” image.[5] These can also be used to increase the size of the image, not to be confused with image scaling which refers to higher resolution images with the same level of detail.

4.3 Working

A fully working and functional game engine will be implemented that includes tools for creating games or other 3D applications and a renderer capable of rendering real-time graphics. Included are all the standard tools needed for game development. The second major component of this project includes an image upscaling algorithm at the output end of the renderer, which enables the game engine to render games or graphics at a lower resolution allowing them to run smoother on older hardware, which is taken as the input for the image upscaling algorithm which in turn converts these images or frames to the native resolution of the user's monitor.

4.4 WORK PROPOSAL

“AI-Enabled Game Engine” provides a tool that enables developers to create games and 3D rendering programs which are compatible with older hardware, allowing them to issue draw calls at a lower resolution which are then upscaled by a lightweight CPU based program thus, freeing up the load from the graphics card.

The aim is to upscale images with as little latency as possible at a speed of 60 frames per second requiring the upscaling time to be less than or equal to 16 ms, this is over the time required to render the image which varies from 25 ms to 50 ms.

Deep learning neural networks boost frame rates and generate crisp and sharp images in games and other rendering programs. Existing technologies such as DLSS need special cores known as Tensor Cores which are exclusive to Nvidia RTX graphics cards. The proposed system will be optimized to run on all systems with any graphics cards.[7]

Tookivi, the rendering engine, comes featured with all the standard requirements of any rendering API with features such as:

- 3D viewport
- .obj File imports
- Textures and Bitmaps
- Dynamic lighting capability
- Animations
- Translations, Rotation, and Scaling options for models
- Tree style object hierarchy

All these features make it a viable alternative for someone looking to make a simple game.

This implementation is done using OpenGL as our choice of the graphics API, the choice came as it was the simplest of the alternatives such as Direct 3D or Vulcan by Microsoft and AMD respectively. OpenGL is also a state machine, making the binding and unbinding of buffers and textures less complex.

CHAPTER 5

DESIGN AND LAYOUT

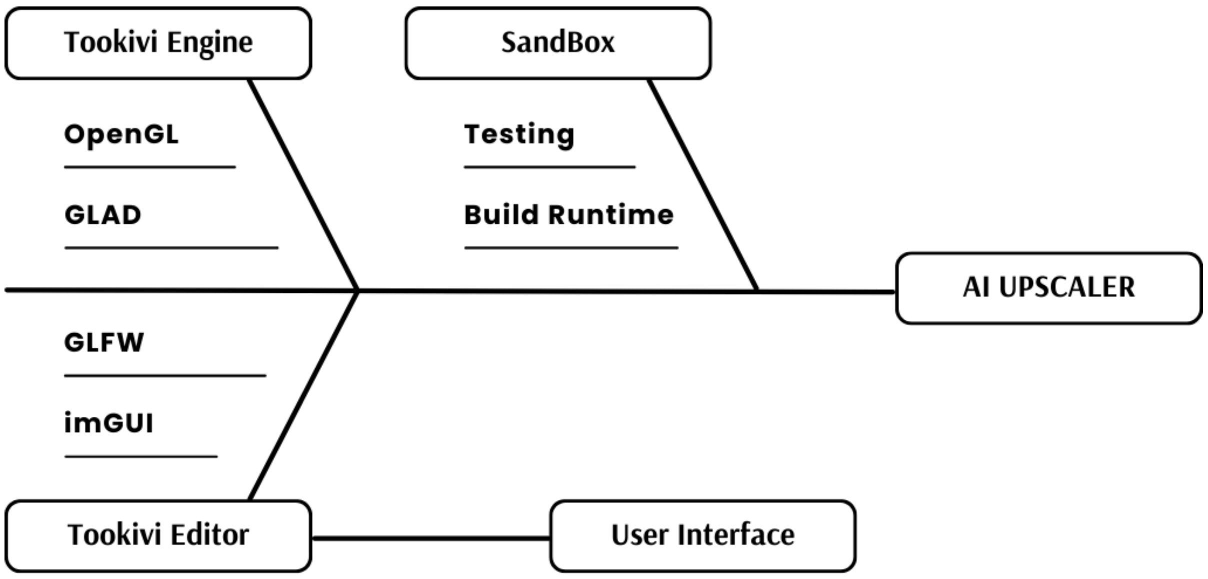


Figure 5.1 Project Flow Chart

5.1 Game Engine

The game engine as implemented in Visual C++ consists of four main parts (one of which is used for developmental purposes). Each of these parts consists of several subparts each making up one of the functionalities of the engine. Each of the segments of segments is implemented separately and will be linked during compile-time .dll files will be generated which will be the abstraction layer for the project. The generated .dll files must be in the same directory as the .exe file of the editor for the engine to run successfully. The various sections of the Game Engine include the following:

- Tookivi Engine
- Tookivi Editor
- User Interface
- Sandbox
- Dependencies

5.1.1 Tookivi Engine

This is the main section of the project. The engine is at the heart of all of the functioning of the complex machine. It is further divided into three more folders which each consist of fragmented parts of the engine, these are:

- External Dependencies – This section consists of all the files needed by the dependencies of the engine. These are the dependencies of the dependency. As we are not concerned with these files during our implementation these have been abstracted off to a separate folder for better documentation and clarity in the workspace.
- src (source folder) – This folder consists of all the implementations of the various features of the game engine. These include but are not limited to the core functions of the engine, event manager, graphical user interface manager, renderer, scene manager, utility functions, etc. All these are the functioning of the engine and are thus abstracted away from the developer wishing to only design a game. They need not worry about the workings of the engine and may focus on the task at hand.
- Vendor – These are the dependencies of the game engine. All the major dependencies were discussed in an earlier part of this paper, but some more that are of note included in this folder are ImGuizmo, stb_image and the Vulkan SDK used to load textures.

5.1.2 Tookivi Editor

This is the editor of the game engine, the part of the project the end consumer or in this case the developer interacts with. All the functioning classes and functions of the main engine have been abstracted away in the Tookivi Engine folder as mentioned before, this enables ease of use for the developer.

This folder following older patterns is again divided into multiple sections:

- External Dependencies
- User Interface
- Editor layer and Application

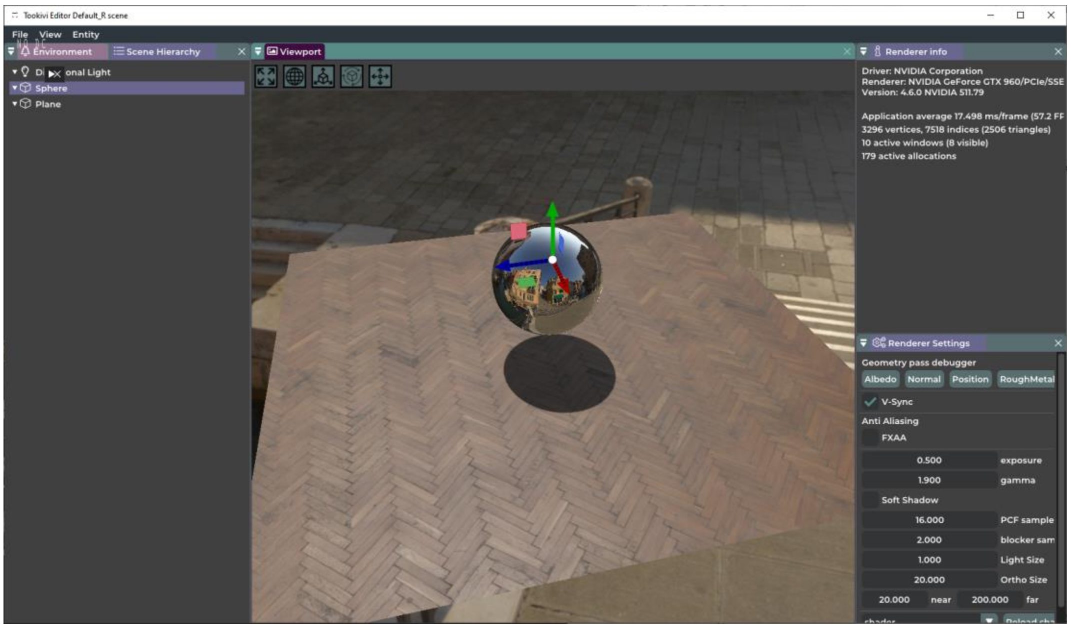
Similar to the folder of the Tookivi Engine, the dependencies folder consists of the many libraries used by the editor's dependencies. The UI folder simply contains the UI elements and style programming needed for the engine. The editor layer defines the layout of the interface.

5.1.2.1 User Interface

The user interface is a subpart of the Tookivi Editor, as the editor consists of all the backend programming of the visuals, the User Interface implemented with imGUI gives the developer or the user of the game engine a visual medium of interaction with the engine rather than having the code. This makes it more user-friendly.

5.1.3 SandBox

The Sandbox is what its name implies, a sandbox to test the performance of the editor and the rendering engine. It is used only for development purposes and will not be included in the final release of the AI-ENABLED GAME ENGINE. The Sandbox can also be used for debugging any logical errors which make through the compile stage of the program. If any part of the engine is not working as intended, We need to load the whole engine into the memory and re-compile the whole project, rather we may only compile, run and test that single feature in the sandbox.



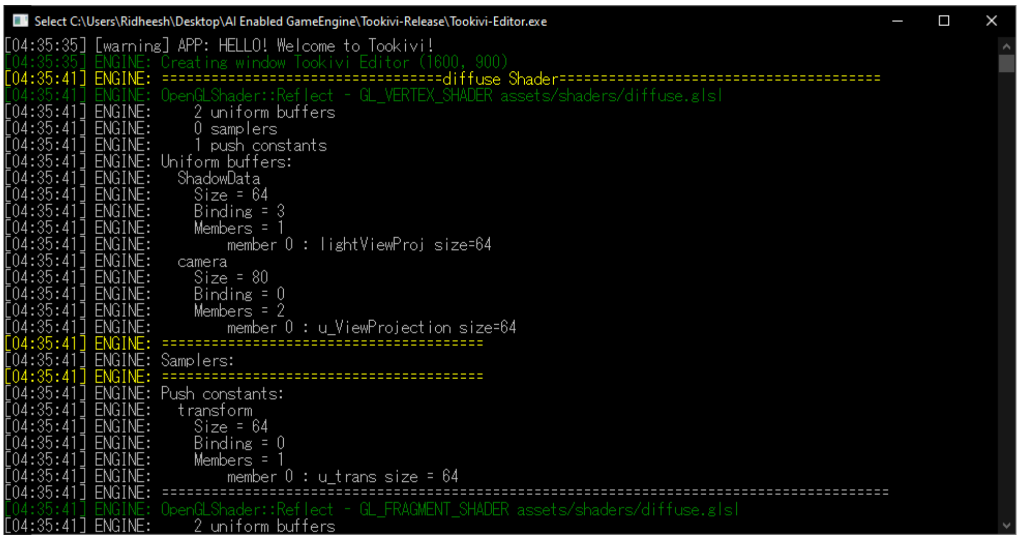
```
Driver: NVIDIA Corporation
Renderer: NVIDIA GeForce GTX 960/PCIe/SSE
Version: 4.6.0 NVIDIA 511.79

Application average 17.498 ms/frame (57.2 FF
3296 vertices, 7518 indices (2506 triangles)
10 active windows (8 visible)
179 active allocations
```

```
✓ V-Sync
Anti Aliasing
  FXAA
0.500    exposure
1.900    gamma
  Soft Shadow
16.000   PCF sample
2.000    blocker sam
1.000    Light Size
20.000   Ortho Size
20.000 near  200.000 far
```

Figure 5.2 Tookivi Editor Running

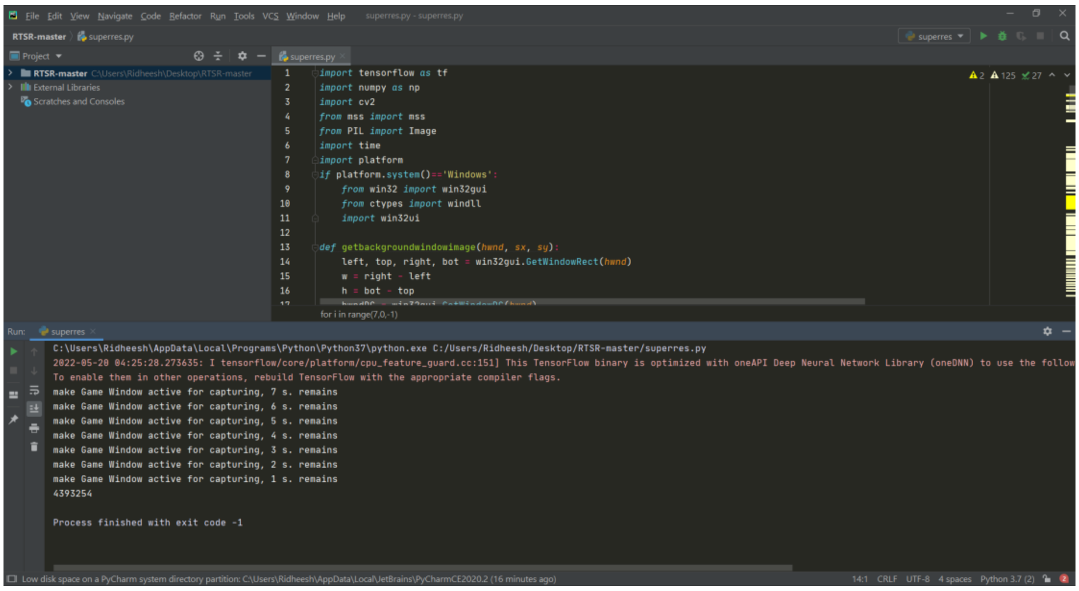
5.2 AI Upscaler

The AI Upscaler as implemented in Python upscales the image from a lower rendered resolution which can run smoothly on the target machine to the native resolution of the machine. This is done using Deep Learning, the model has been trained on data sets acquired from recordings of games and general desktop usage and stored. The model is called when the program is run and upscaled the image by 4 times. Upscaling from a rendered resolution of 1366×768 to the native resolution of 1920×1080 of the monitor, the resultant 15-20 fps is below expectation and further optimization is required.



```
Select C:\Users\Ridheesh\Desktop\AI Enabled GameEngine\Tookivi-Release\Tookivi-Editor.exe
[04:35:35] [warning] APP: HELLO! Welcome to Tookivi!
[04:35:35] ENGINE: Creating window Tookivi Editor (1600, 900)
[04:35:41] ENGINE: ==================================diffuse Shader=====================================
[04:35:41] ENGINE: OpenGLShader::Reflect - GL_VERTEX_SHADER assets/shaders/diffuse.glsl
[04:35:41] ENGINE:     2 uniform buffers
[04:35:41] ENGINE:     0 samplers
[04:35:41] ENGINE:     1 push constants
[04:35:41] ENGINE: Uniform buffers:
[04:35:41] ENGINE:   ShadowData
[04:35:41] ENGINE:     Size = 64
[04:35:41] ENGINE:     Binding = 3
[04:35:41] ENGINE:     Members = 1
[04:35:41] ENGINE:         member 0 : lightViewProj size=64
[04:35:41] ENGINE:   camera
[04:35:41] ENGINE:     Size = 80
[04:35:41] ENGINE:     Binding = 0
[04:35:41] ENGINE:     Members = 2
[04:35:41] ENGINE:         member 0 : u_ViewProjection size=64
[04:35:41] ENGINE: =======================================
[04:35:41] ENGINE: Samplers:
[04:35:41] ENGINE: =======================================
[04:35:41] ENGINE: Push constants:
[04:35:41] ENGINE:   transform
[04:35:41] ENGINE:     Size = 64
[04:35:41] ENGINE:     Binding = 0
[04:35:41] ENGINE:     Members = 1
[04:35:41] ENGINE:         member 0 : u_trans size = 64
[04:35:41] ENGINE: ==========================================================================================
[04:35:41] ENGINE: OpenGLShader::Reflect - GL_FRAGMENT_SHADER assets/shaders/diffuse.glsl
[04:35:41] ENGINE:     2 uniform buffers
```

Figure 5.3 Tookivi Editor Logs (For Resolution)



```python
import tensorflow as tf
import numpy as np
import cv2
from mss import mss
from PIL import Image
import time
import platform
if platform.system()=='Windows':
    from win32 import win32gui
    from ctypes import windll
    import win32ui

def getbackgroundwindowimage(hwnd, sx, sy):
    left, top, right, bot = win32gui.GetWindowRect(hwnd)
    w = right - left
    h = bot - top
```

```
C:\Users\Ridheesh\AppData\Local\Programs\Python\Python37\python.exe C:/Users/Ridheesh/Desktop/RTSR-master/superres.py
2022-05-20 04:25:28.273635: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the follow
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
make Game Window active for capturing, 7 s. remains
make Game Window active for capturing, 6 s. remains
make Game Window active for capturing, 5 s. remains
make Game Window active for capturing, 4 s. remains
make Game Window active for capturing, 3 s. remains
make Game Window active for capturing, 2 s. remains
make Game Window active for capturing, 1 s. remains
4393254

Process finished with exit code -1
```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
superres.py - superres.py
RTSR-master
superres.py
Project
RTSR-master C:\Users\Ridheesh\Desktop\RTSR-master
External Libraries
Scratches and Consoles
for i in range(7,0,-1)
Run: superres
Low disk space on a PyCharm system directory partition: C:\Users\Ridheesh\AppData\Local\JetBrains\PyCharmCE2020.2 (16 minutes ago)
14:1 CRLF UTF-8 4 spaces Python 3.7 (2)

Figure 5.4 AI Engine implementation

CHAPTER 6

IMPLEMENTATION AND PERFORMANCE

6.1 Tookivi Engine

The Tookivi Engine was implemented in Visual Studio 2019 and can be found in the “Tookivi-Master” folder under “Tookivi”. It is further divided into sections:

- External Dependencies
- SRC
  - Engine
    - Core
    - Events
    - ImGUI
    - Renderer
    - Scene
    - Utils
  - Platform
    - OpenGL
    - Windows
- Vendor
  - GLM
  - ImGUIzmo
  - Stb_image

The External Dependencies and the Vendor folder contain all the dependencies and their specifications. Thus, are of no concern to the implementation, it is just important that is properly installed and present at the time of compiler execution. SRC folder, names for Source Files, contains all the files of the Tookivi Engine which are discussed in detail. The major folder here to keep in mind is the Renderer subfolder inside the SRC\Engine folder which contains the majority of the implementation of the engine including but not limited to the rendering engine, event manager, texture and lighting manager, etc.

6.1.1 Engine

The Engine consists of the files responsible for all the back end calculations for the rendering process. As the name suggests, it is the engine in the rendering engine. Responsible for rendering, UI implementations, Core functionality like Translation, Rotations and Scaling, Scene layout manager and all the Utility managers.

6.1.1.1 Core

The core folder consists of both .h and .cpp files. It is responsible for the following:

6.1.1.1.1 Entry Point of the Renderer

This is the entry point of the program consisting of the main() function. This acts as the Application or the Game. So, the implementation here will be the final game output of the Game Engine.

6.1.1.1.2 Input/Output management

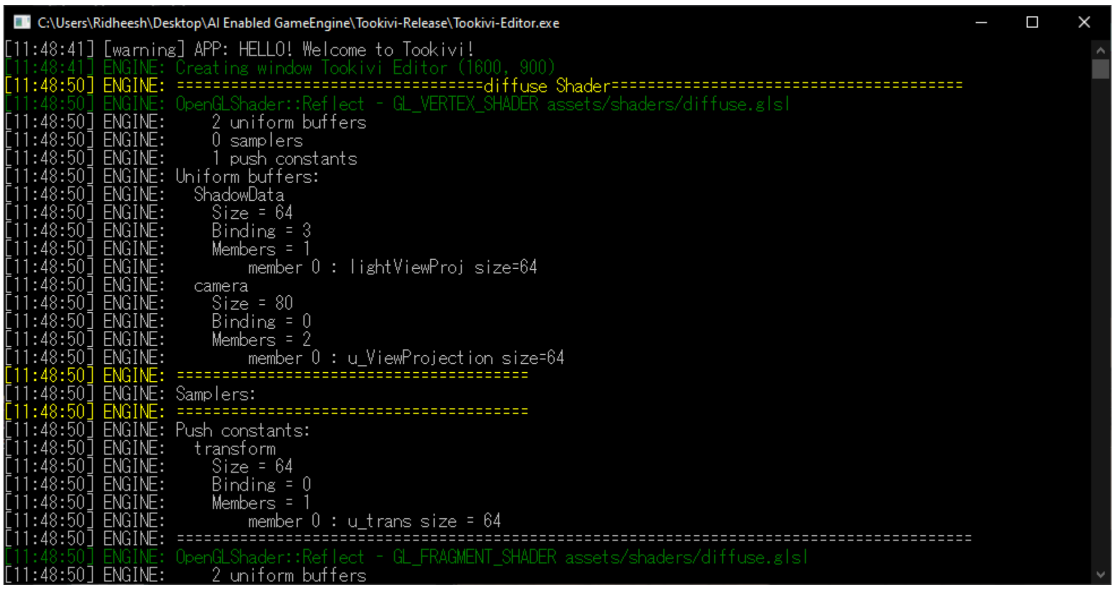
As the name implies, the Core is also responsible for managing the input and output from the Keyboard and Mouse. The key values are mapped to a premade map of values and called whenever the key is pressed. This is handled as an interrupt to the program.

6.1.1.1.3 Window management

This consist of the GLFW library, implementation and empty window to draw upon. It is the application window and is opened separately from the command line window. This screen is controlled by the core of the renderer. It is also responsible for managing all the different layers of the game engine generated by the GUI and the renderer along with polling for events such as input events or any generated interrupts by the engine or the User Interface.

6.1.1.1.4 Event logging in CMD

Managed by an external library all the events and occurrences of the game engine are logged to the CMD opened alongside the GLFW window. This can prove to be a useful tool not only for debugging during development but also as a source of log files for the end-user.



```
C:\Users\Ridheesh\Desktop\AI Enabled GameEngine\Tookivi-Release\Tookivi-Editor.exe
[11:48:41] [warning] APP: HELLO! Welcome to Tookivi!
[11:48:41] ENGINE: Creating window Tookivi Editor (1600, 900)
[11:48:50] ENGINE: ==================================diffuse Shader======================================
[11:48:50] ENGINE: OpenGLShader::Reflect - GL_VERTEX_SHADER assets/shaders/diffuse.glsl
[11:48:50] ENGINE:     2 uniform buffers
[11:48:50] ENGINE:     0 samplers
[11:48:50] ENGINE:     1 push constants
[11:48:50] ENGINE: Uniform buffers:
[11:48:50] ENGINE:   ShadowData
[11:48:50] ENGINE:     Size = 64
[11:48:50] ENGINE:     Binding = 3
[11:48:50] ENGINE:     Members = 1
[11:48:50] ENGINE:         member 0 : lightViewProj size=64
[11:48:50] ENGINE:   camera
[11:48:50] ENGINE:     Size = 80
[11:48:50] ENGINE:     Binding = 0
[11:48:50] ENGINE:     Members = 2
[11:48:50] ENGINE:         member 0 : u_ViewProjection size=64
[11:48:50] ENGINE: ======================================
[11:48:50] ENGINE: Samplers:
[11:48:50] ENGINE: ======================================
[11:48:50] ENGINE: Push constants:
[11:48:50] ENGINE:   transform
[11:48:50] ENGINE:     Size = 64
[11:48:50] ENGINE:     Binding = 0
[11:48:50] ENGINE:     Members = 1
[11:48:50] ENGINE:         member 0 : u_trans size = 64
[11:48:50] ENGINE: =======================================================================================
[11:48:50] ENGINE: OpenGLShader::Reflect - GL_FRAGMENT_SHADER assets/shaders/diffuse.glsl
[11:48:50] ENGINE:     2 uniform buffers
```

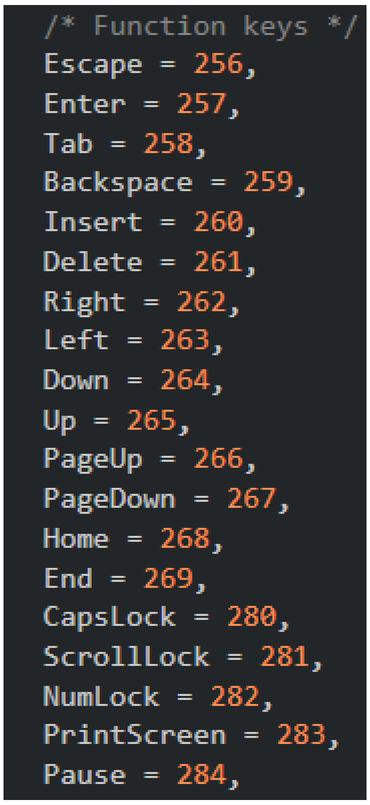
Figure 6.1 Tookivi Editor Event Logs

6.1.1.1.1 Layer Stack handler

As the program consists of multiple layers of item objects, this file manages the Layer data and keeps track of the layers by making use of the data structure stack.

6.1.1.1.5 Key Code mapping for both mouse & keyboard buttons,

As discussed before, these files help in the mapping of key presses to predefined values for handling key inputs from both the mouse and the keyboard. These can be mapped to an integer value ranging from the standard range of INT data type. Although it is preferable to not use negative INTs or unsigned INTs to increase the performance of the system as this is all run in real-time. Each key is pressed an interrupt is generated.



```
/* Function keys */
Escape = 256,
Enter = 257,
Tab = 258,
Backspace = 259,
Insert = 260,
Delete = 261,
Right = 262,
Left = 263,
Down = 264,
Up = 265,
PageUp = 266,
PageDown = 267,
Home = 268,
End = 269,
CapsLock = 280,
ScrollLock = 281,
NumLock = 282,
PrintScreen = 283,
Pause = 284,
```

Figure 6.2 Key Map Examples

6.1.1.1.1 Application file (The actual Game)

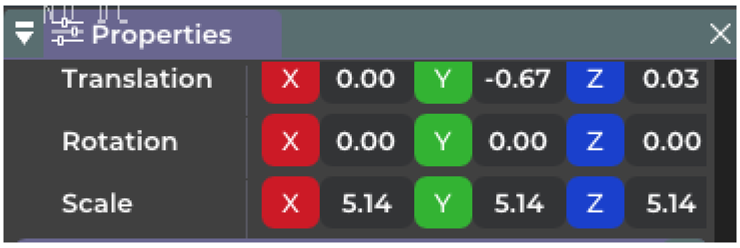
This is the file that will be exported alongside the entry point and is responsible for generating the game files. As we do not want to include the whole rendering engine while the game is being exported to save bot size and complexity, only the required libraries and the application portion of the engine are exported when the game is designed and ready to be tested.

6.1.1.1.6 Timer and Time management

As many of the aspects of the engine are linked to the direct performance of the engine, rather than link it to the performance, aspects such as event registering and input/output are mapped to a timer. This will ensure that a game will run the same on all computers and the performance of the game will not affect any mechanics. This is also responsible for synchronizations.

6.1.1.1.7 Core Files and functionality

These files consist of the core functionality of the game engine. Features such as scaling, translation and rotation are implemented here. It also takes care of lighting and HDR rendering. This file is also needed for linking together all the files in the folder.



Properties

Translation	X	0.00	Y	-0.67	Z	0.03
Rotation	X	0.00	Y	0.00	Z	0.00
Scale	X	5.14	Y	5.14	Z	5.14

Figure 6.3 GUI implementation of CORE functionality

6.1.2 Events

This is the folder responsible for triggering certain pre-defined functions when a certain event takes place. It is the interrupt folder. This folder consists of several run time programs which may throw exceptions.

It consists of the application event handler which notifies the rendering engine when any UI elements are interacted with along with any interaction with the game engine itself.

Key events and Mouse events are responsible for calling interrupts when a keyboard key or a mouse button is pressed. It checks the keymap and sends the command to the core folder to be processed accordingly.

6.1.3 ImGUI

ImGUI as the name implies consists of the functionality of the imGUI library. All the implementation of the User Interface can be found in this folder including but not limited to the buttons, menu, text, icons, etc. It consists of an imGUI layer which is rendered on top of the rendering layer as the engine is running.

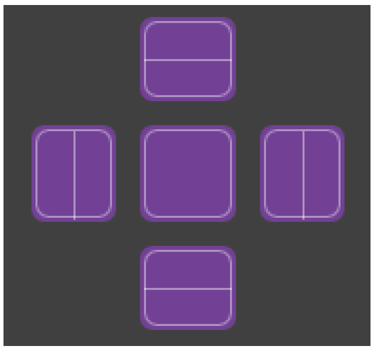
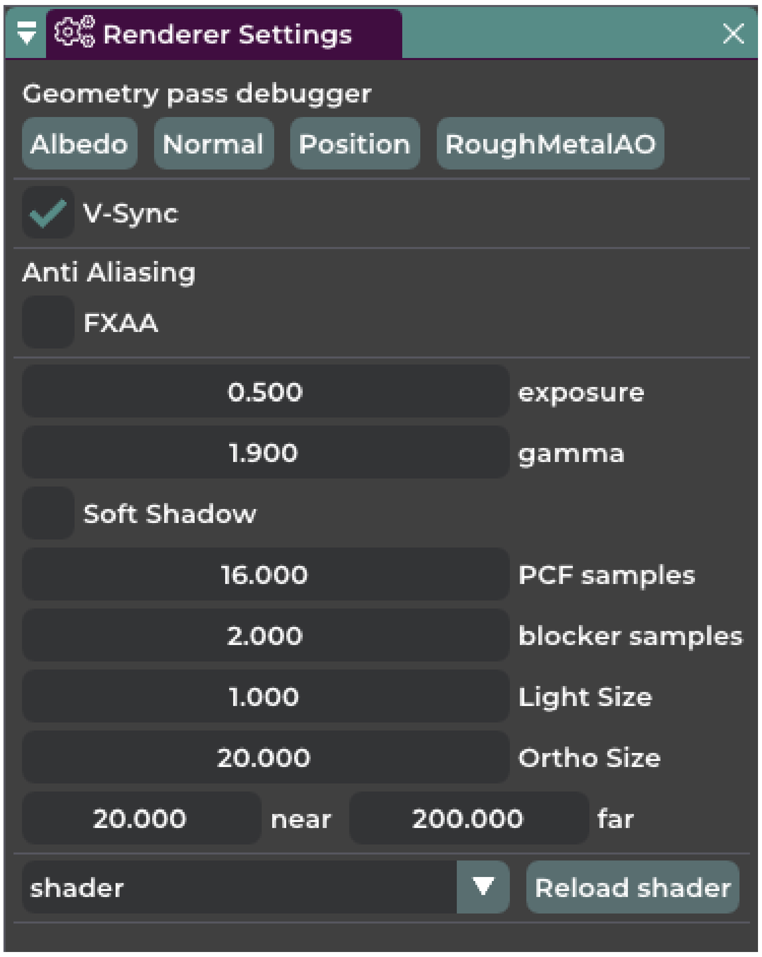


Figure 6.4 imGUI Drag and Drop Window

6.1.4 Renderer

This is the main section of the project, the renderer consists of the part of the code which draws and renders the models, textures and shaders onto the screen. As this is the Major part of the engine it consists of numerous files.



Renderer Settings

Geometry pass debugger

Albedo Normal Position RoughMetalAO

✓ V-Sync

Anti Aliasing

FXAA

0.500 exposure

1.900 gamma

Soft Shadow

16.000 PCF samples

2.000 blocker samples

1.000 Light Size

20.000 Ortho Size

20.000 near 200.000 far

shader ▼ Reload shader

Figure 6.5 Renderer Settings

6.1.4.1 Buffer Management

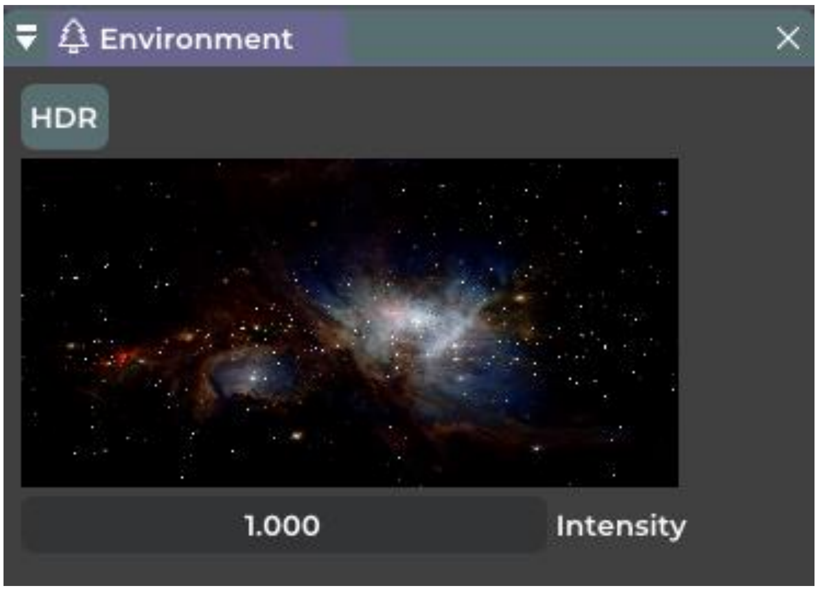
This file is responsible for the management of the buffer, the buffer stores the buffer data of any models loaded into the engine. All the buffer data is then passed on and loaded into the GPU VRAM and called when needed.

6.1.4.2 Camera

Cameras' functionality and workings are managed by this file. It selected the orientation of the camera. Based on the camera's field of view or FOV, the renderer draws the image. As nothing that is not in the view of the camera need to be drawn by the renderer to save resources.

6.1.4.3 Environment

File responsible for creating, and maintaining a functional environment where everything is rendered. This works hand in hand with the renderer and the imGUI file. It updates the GUI when something changes or lets the renderer know any needed information. It is also responsible for importing and displaying the HDR content of the game engine.



Intensity: 1.000

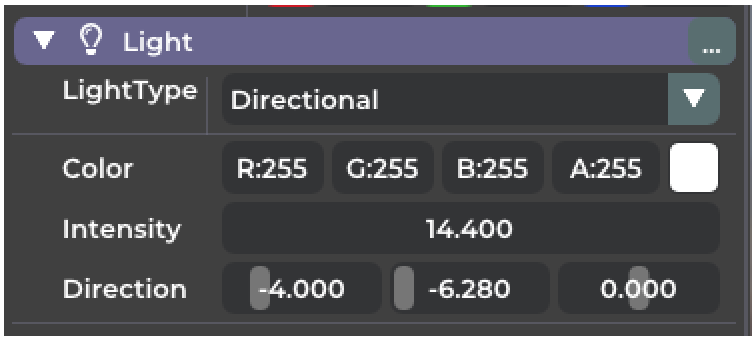
Figure 6.6 Environment Manager GUI

6.1.4.4 Frame Buffer Management

When all the data is calculated by the GPU it stores a frame buffer with all the RGB (red, green and blue) pixel values along with lumen data in the frame buffer. As soon as a frame buffer is finished drawing to the screen another should be loaded and ready to go for smooth performance of the engine.

6.1.4.5 Lighting Manager

This is the master light manager file, responsible for global illumination, spotlighting and reflections of the surfaces. OpenGL functions are made use of for the implementation of illumination in the engine which is then calculated by the GPU to be drawn on the screen.



Light

LightType: Directional

Color: R:255 G:255 B:255 A:255

Intensity: 14.400

Direction: -4.000 -6.280 0.000

Figure 6.7 Lighting Manager GUI

6.1.4.6 Material Handler

All models have attributes such as tiling, normal, roughness, metallic, etc. All these are known as materials and are controlled by this file. They are implemented either as a slider in the GUI or as maps which can be loaded as image files.

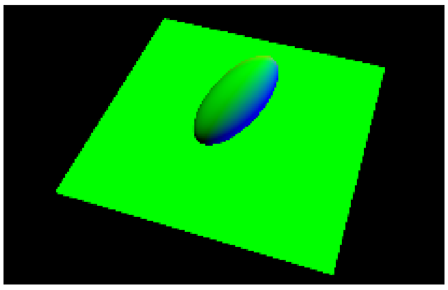


Figure 6.8 Normal Map

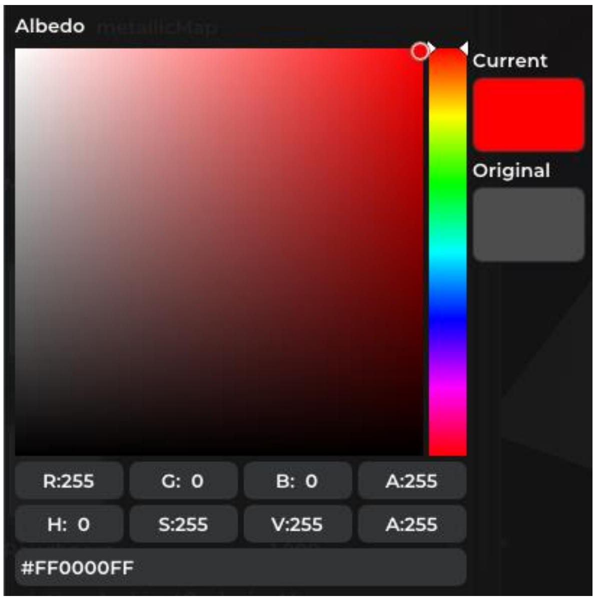
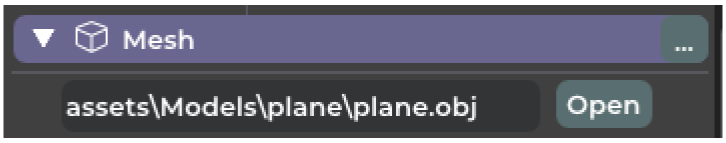


Figure 6.9 Albedo Color Map

6.1.4.7 Mesh Handler

Any .obj file or a 3D model consists of a series of vertices which when together make up a mesh. This mesh when joined together forms the final model. This file is needed for creating, loading and drawing meshes in the engine.



assets\Models\plane\plane.obj

Figure 6.10 Mesh Importer

6.1.4.8 Model Loader

This file is needed for the importing and loading of the models from external sources such as Blender. These models can be developed in an external environment and then imported into the engine to be programmed to move. The supported format for the importing the models is .obj

6.1.4.9 Perspective and Orthographic Camera implementation

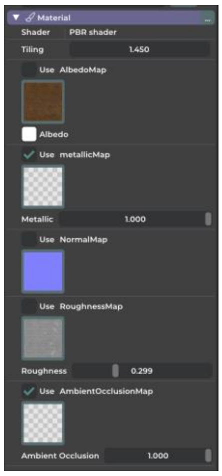
As the name implies, this is the implementation of the Orthographic and the perspective camera modes. These additional camera modes give the game developed more freedom to explore new possibilities.

6.1.4.10 Renderer

The Core of the Renderer aptly named the Renderer is not a standalone file but a group of files consisting of the Rendering API which interacts with the GLAD functions, the Rendering command acceptor, the Scene Renderer, Rendering Pass Generator and more. All these parts together make it possible to render all the vertex, texture and shader data requested by the developer.

6.1.4.11 Texture Manager

Textures are any image files loaded into the engine to be used on top of an object. These also are image maps for normal mapping or HDR data loading for the environmental illumination based on an image.



Shader PBR shader
Tiling 1.450
Use AlbedoMap
Albedo
✓ Use metallicMap
Metallic 1.000
Use NormalMap
Use RoughnessMap
Roughness 0.299
✓ Use AmbientOcclusionMap
Ambient Occlusion 1.000

Figure 6.11 Material and Texture Manager GUI

6.1.4.12 Uniform Buffer Array Generator

As passing the Buffer data alone will cause too many function calls, this file makes use of the Uniform Buffer Array feature of

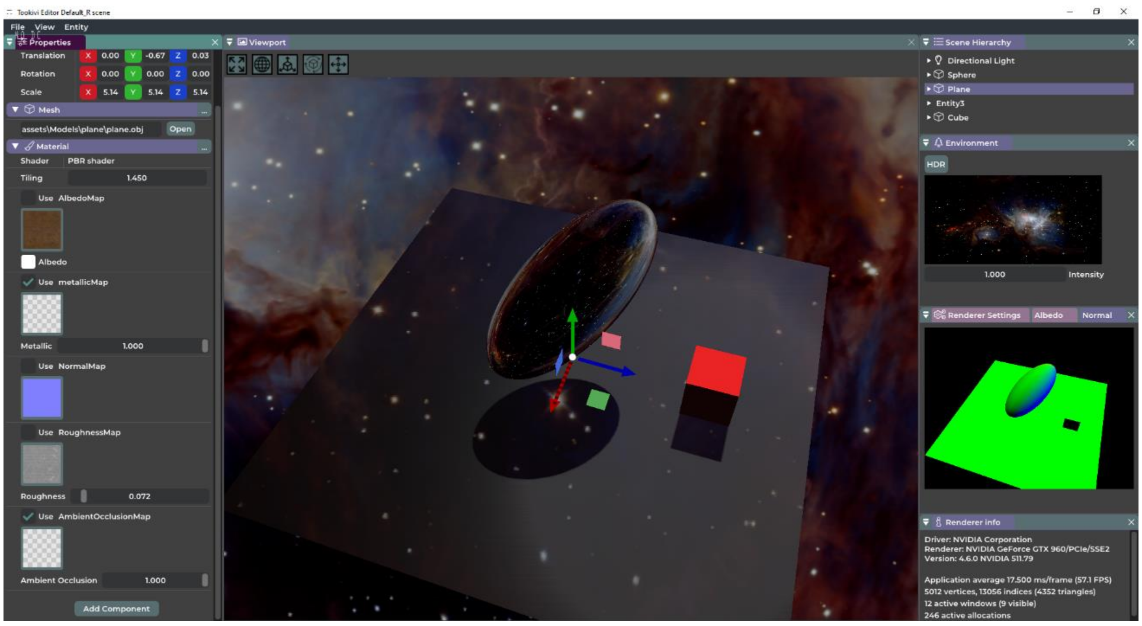
OpenGL to take advantage of the performance boost provided by this implementation.

6.1.4.13 Vertex Array Manager

This file is responsible for managing the vertices in the engine, these can be manually placed or a part of a .obj file. They are then loaded onto the GPU VRAM similar to the buffer object data.

6.1.5 Scene and Utils

This folder is responsible for scene generation. It is similar to the environment generator but rather is responsible for the rendering environment, this is responsible for the whole GLFW screen. It manages the output of both the GUI and the renderer and seamlessly merges the two to be shown as one window.



Translation X 0.00 Y -0.67 Z 0.03
Rotation X 0.00 Y 0.00 Z 0.00
Scale X 5.14 Y 5.14 Z 5.14

Mesh
assets\Models\plane\plane.obj

Material
Shader PBR shader
Tiling 1.450
Use AlbedoMap
Albedo
✓ Use metallicMap
Metallic 1.000
Use NormalMap
Use RoughnessMap
Roughness 0.072
✓ Use AmbientOcclusionMap
Ambient Occlusion 1.000

1.000 Intensity

Driver: NVIDIA Corporation
Renderer: NVIDIA GeForce GTX 960/PCIe/SSE2
Version: 4.6.0 NVIDIA 511.79

Application average 17.500 ms/frame (57.1 FPS)
5012 vertices, 13056 indices (4352 triangles)
12 active windows (9 visible)
246 active allocations

Figure 6.12 Scene GUI

The name of the folder “Utils” is an abbreviation for Utilities, this stores files responsible for math operations which are used throughout the project. Implementation for matrix multiplication, the sum of all values, Platform-specific operations, Poisson generator, Noise generator and more are some of its many features. It may also prove useful for the implementation of maybe such general utility features.

6.2 Tookivi Editor

Similar to the Tookivi Rendering Engine, the Tookivi Editor is also implemented in Visual Studio 2019 and can be thought of as the Front-End of the project. This visual studio solution is responsible for the editor layer of the game engine along with any interactions. It is divided into the following sections:

- External Dependencies
- UI
    - Panels
- Editor Layer
- Editor Application

When compared to the Tookivi Rendering solution, it is a much simpler solution. Due to this, there is an inherent lack of explanation needed.

6.2.1 External Dependencies

Same as the Dependencies folder of the renderer this is a collection of the dependencies needed to run the files in the solution, these include but are not limited to inGUI, GLFW, GLAD, etc.

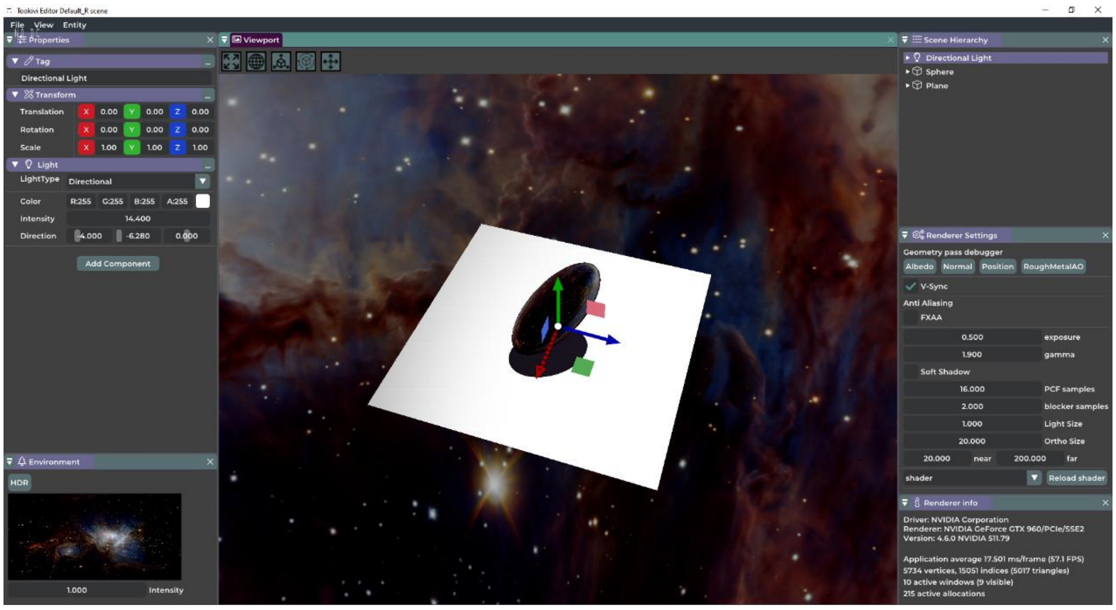
6.2.2 User Interface

This folder includes the files needed to render the front end of the user interface which includes the .h files for the UI and the scene panels.

The Scene Panels include the following six sections which are visible in the graphical user interface:

- Camera Panel
- Content Browser
- Light Panel
- Materials Panel
- Mesh Panel
- Scene Panel

These panels can be dragged around are repositioned in the graphical user interface when the GLFW window is open. This enables customizability for the game developer to set up the environment as he or she pleases.



Tag: Directional Light

Transform

	X	Y	Z
Translation	0.00	0.00	0.00
Rotation	0.00	0.00	0.00
Scale	1.00	1.00	1.00

Light

LightType	Directional			
Color	R:255	G:255	B:255	A:255
Intensity	14.400			
Direction	-4.000	-6.280	0.000	

Environment: 1.000 Intensity

Renderer Settings

Geometry pass debugger

✓ V-Sync

Anti Aliasing

FXAA

0.500	exposure
1.900	gamma

Soft Shadow

16.000	PCF samples
2.000	blocker samples
1.000	Light Size
20.000	Ortho Size

20.000 near 200.000 far

shader

Renderer info

```
Driver: NVIDIA Corporation
Renderer: NVIDIA GeForce GTX 960/PCIe/SSE2
Version: 4.6.0 NVIDIA 511.79

Application average 17.501 ms/frame (57.1 FPS)
5734 vertices, 15051 indices (5017 triangles)
10 active windows (9 visible)
215 active allocations
```

Figure 6.13 UI Scene Panels

6.3 Results and Performance Metrics

A fully functional game engine fulfilling all the objective requirements has been implemented in visual studio 2019 and C++, the performance of the game engine is acceptable at a stable 60 frames per second (idle) on a standard machine. The AI upscaler has been implemented using Python and PyCharm has been achieved. The upscaler works as an overlay over the game engine window upscaling it to the native resolution of the user's monitor. The performance of the upscaler is limited, under standard conditions on an HD monitor upscaling from SD it performs with 15-20 frames per second. Further optimization of an overhaul of the upscaler is required for the full release but is acceptable as an alpha product.

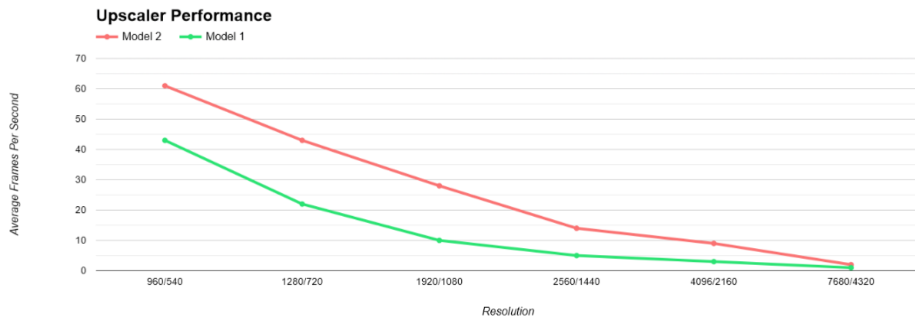


Figure 6.14 Upscaler Engine Performance comparison

The performance of the game engine was tested based on the metric of frames per second, relative to industry-standard game engines such as the Unity and Unreal Engine. The relative performance of the game engine is shown in the below graph:

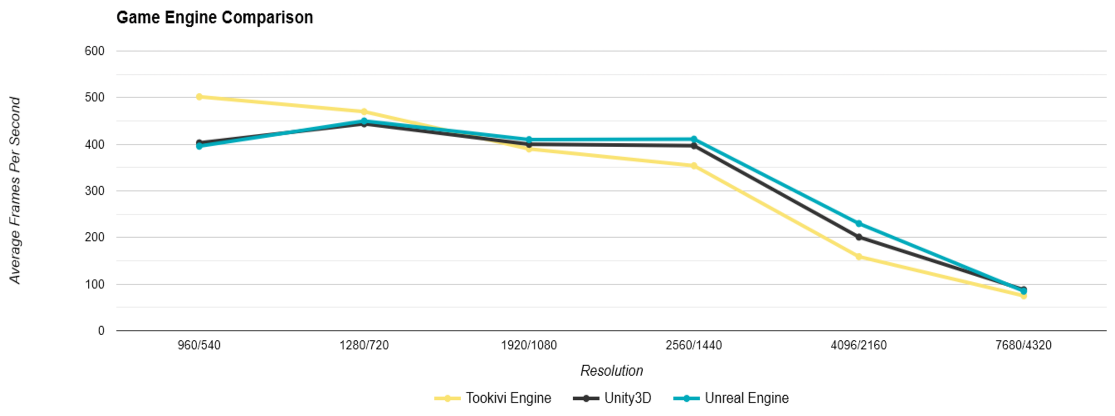
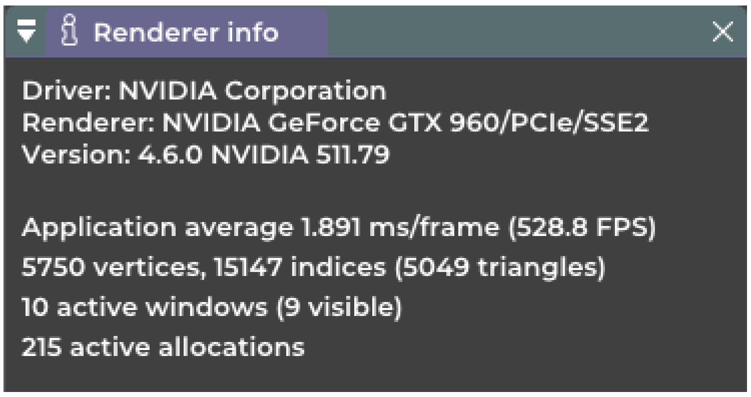


Figure 6.15 Game Engine Performance comparison

The Tookivi Game Engine performs better at a lower rendering resolution, as intended for AI upscaling, and then the performance is comparable at higher resolutions with the industry standard. Although this performance of the Tookivi engine is being compared to a fully-featured game engine with much more dependencies and memory usage which may cause the testing to favour the Tookivi Engine.



Driver: NVIDIA Corporation
Renderer: NVIDIA GeForce GTX 960/PCIe/SSE2
Version: 4.6.0 NVIDIA 511.79

Application average 1.891 ms/frame (528.8 FPS)
5750 vertices, 15147 indices (5049 triangles)
10 active windows (9 visible)
215 active allocations

Figure 6.16 Tookivi Engine Performance on an Nvidia GeForce GTX 960 GPU

The above image shows the rendering information of the machine. All testing was performed on a machine with the following specifications:

- Intel i5 4690K (4th Generation)
- Nvidia GeForce GTX 960 2GB VRAM
- 8GB DDR3 RAM
- 1920×1080 60Hz Display

CHAPTER 7
Conclusion and Future Scope

The project “AI-Enabled Game Engine” has been successfully implemented. The Game engine section of the project was implemented in two major parts:

- Tookivi Engine
- Tookivi Editor

The game engine has implemented all of the proposed features and has a satisfactory run time and render time performance.

The AI Upscaler of the project has been implemented in Python and under real-life test cases in upscaling from a rendered resolution of 1366×768 to the native resolution of 1920×1080 of the monitor, the resultant 15-20 frames per second are below expectation and further optimization is required for a stable frame rate of 30+ frames per second.

The Game Engine, though successfully implemented still has much room for improvement, notable features like Sound Development and Cross-Platform access are yet to be implemented in the future, causing the game engine to come short of being a viable product for the current competitive market. These features and more will be implemented in the future to make it a competitor to today’s standards of game engines. The AI Upscaling section of the project has a subpar performance of 15-20 frames per second and much optimization is needed or even a full redesign of the AI model could be warranted for improved performance. The target performance is 30 frames per second. The future scope of the project may include:

- Optimizations of AI Upscaler
- Multi-Platform Support
- Sound Engine implementation in the Tookivi Game Engine
- Dynamic processor adaptation between CPU and GPU
- Additional AI-enabled features include but are not limited to animations, movement, In-game player design, etc.
- Implementation of missing features such as scripting, animation and frame-based rendering.

REFERENCES

[1] “Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices” [December 2017] by Stelios Xinogalos ISSN 2384-8766

[2] “Game Engine Solutions” [2018] by Anis Zarrad

[3] “Computer Graphics: Principles and Practice” [Third Edition] by James D. Foley ISBN-13: 978-0-321-39952-6 ISBN-10: 0-321-39952-8

[4] “Real-Time Rendering” by Tomas Akenine-Möller ISBN-13: 978-1138627000, ISBN-10: 1138627003

[5] “Image upscaling technique” by Hao Pan

[6] “Corrections to “Real-Time Artifact Free Image Upscaling” [Oct 11, 2760-2768]” by Andrea Giachetti and Nicola Asuni IEEE Transactions on Image Processing 21(4):2361-2361

[7] "NVIDIA DLSS Technology for Incredible Performance". NVIDIA. [2022-02-07]

[8] "The OpenGL Graphics System: A Specification" [March 11, 2010] (PDF). 4.0 (Core Profile)..

[9] OpenCV Developer Site: http://code.opencv.org Archived [2013-01-13]

[10] Dean, Jeff; Monga, Rajat; et al. [November 9, 2015]. "TensorFlow: Large-scale machine learning on heterogeneous systems". Google Research.

[11] Pine, David [2014]. "Python resources". Rutgers University. Retrieved 2017-04-07.

[12] Real-Time Video Super-Resolution on Smartphones With Deep Learning, Mobile AI 2021 Challenge: Report

[13] Andrey Ignatov, Andres Romero, Heewon Kim, Radu Timofte; Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2021, pp. 2535-2544