A

REPORT

ON

3D MODELLING AND RENDERING OF DHARAHARA AND GHANTAGHAR

Submitted To:

Anil Verma

Department of Electronics and

Computer Engineering

IOE, Pulchowk Campus

Submitted By:

Sandeep Acharya (075BCT074)

Sangam Chaulagain (075BCT078)

Saujan Tiwari (075BCT083)

August 26, 2021

Acknowledgement

First and foremost, We would like to express our sincere gratitude to our lecturer Mr. Anil Verma for providing us the opportunity to undertake this project work entitled “3D Modelling and Rendering of Dharahara and Ghataghar”. It was a great opportunity for us to implement the algorithms and techniques we studied in the Computer Graphics. Without his guidance, Our project would never have been completed in time.

We are grateful to the Department of Electronics and Computer Engineering for including this course in the syllabus. It helped us to gain insight to the working of graphics with which we interact in our daily life. It is a great pleasure to understand how the inner things work with different algorithms involved that we are unaware of when using it.

Last but not the least, We would like to thank everyone who is directly or indirectly involved in the project.

Abstract

Computer graphics is a field of computer science that studies methods for digitally synthesizing and manipulating visual content.Computer graphics deals with generating images with the aid of computers. Today, computer graphics is a core technology in digital photography, film, video games, cell phone and computer displays, and many specialized applications.

This report presents the use of graphics algorithms in order to render a 3D model of Dharahara and Ghantaghar using OpenGl. The 3D model consists of two historical monuments of Nepal: Dharahara, also known as Bhimsen Stambha and Ghantaghar, the first public clock tower of Nepal. We have detailed the techniques used in computer graphics to make the objects look more realistic. The theoretical aspects of these algorithms and techniques that are needed to understand the project are well covered in the report.

Additionally the report explores the implementation of these algorithms using C++, OpenGL and GLSL. It is hoped that this report will help all the individuals interested in computer graphics to know more about the use of OpenGl to create and render 3D models using efficient graphics algorithms.

Table of Contents

Acknowledgement **i**

Abstract **ii**

Table of Contents **iii**

List of Figures **iv**

List of Symbols and Abbreviation **v**

1. Introduction 1
- 1.1. Background 1
- 1.2. Objectives 1
- 1.3. Motivations 1
- 1.4. Scope 2

2. Literature Review **3**
- 2.1. Background Theories 3

3. Methodologies **11**
- 3.1 Project Block Diagram 14
- 3.2 Application Flow Chart 14
- 3.3 Algorithms and Implementations 15
- 3.4 Languages and Tools 25

4. Result **26**

5. Conclusion **28**

6. Limitations and Future Works **29**

7. References **30**

List of Figures

1. Blender Logo
2. OpenGL Logo
3. GLFW Logo
4. Diffuse Lighting
5. Specular Lighting
6. Graphics Pipeline
7. Project Block Diagram
8. Assimp Components Tree
9. Flow Chart
10. 3D Translation
11. Equation for 3d Translation
12. Equation for 3d Rotation along X-axis
13. Equation for 3d Rotation along Y-axis
14. Equation for 3d Rotation along Z-axis
15. Equation for 3d Scaling
16. Road Scene with Perspective Projection used
17. Perspective Projection
18. Transformation Matrix For Perspective Projection
19. Viewing Transformation
20. Translation Matrix for Viewing Transformation
21. Rotation Matrix for Viewing Transformation
22. uvn Unit Vectors
23. Equation for Viewing Transformation
24. Phong Illumination Model
25. Z-buffer Test
26. Day Scene (Close)
27. Night Scene (Close)
28. Day Scene (Far)
29. Night Scene (Far)

List of Symbols and Abbreviation

2D	:	Two Dimensions
3D	:	Three Dimensions
OpenGL	:	Open Graphics Library
GLSL	:	OpenGL Shading Language
GLFW	:	Graphics Library Framework
ARB	:	Architecture Review Board
CPU	:	Central Processing Unit
GPU	:	Graphical Processing Unit
COP	:	Center of Projection

1. Introduction

1.1. Background

Our project entitled "3D modelling and rendering of Dharahar and Ghantaghar" is the simulation of the imaginary model consisting of two major attractions Dharahara and Ghantaghar with the near surroundings. The landscape features a road with street lights, benches on footpaths and the boundary consisting of these two towers. Two different views are included: Day view and Night view. Users have the ability to switch between the views by using a keyboard, Z for day view and X for night view. Camera is incorporated to navigate the 3D model by using the keys: A, W, S, D and the mouse.

1.2. Objectives

The major objectives of our project are:

1. To design and render a beautiful 3D Dharahara and Ghantaghar model.
2. To be familiar with blender for making 3D models.
3. To learn about OpenGl for rendering 3D objects.
4. To implement the graphics algorithms to create real life objects.
5. To understand the concept of shaders and lighting models.
6. To work with the team members in mutual collaboration.

1.3. Motivations

Dharahara, also known as Bhimsen Tower is a twenty-two storey tall tower at the center of Sundhara, Kathmandu, Nepal. It was destroyed by the 2015 earthquake and the reconstruction is near completion at this time. Dharahara is one of the major centers of attraction of the people living both inside and outside the Kathmandu valley. While choosing our project work, We decided to render Dharahara which is being constructed. The problem was we didn't fully know the design of the dharahara being reconstructed and we weren't allowed to go there as there was a lockdown due to the ongoing Covid-19. So, We decided to redesign the dharahara by ourselves. First of all we collected the pictures of the tower being constructed and the destroyed tower. In the course of building the dharahara model, we decided to add a clock-tower with it to make the view more beautiful.

1.4. Scope

The model used in our project is the combination of Dharahara and Ghantaghar. In reality, Dharahara and Ghantaghar are two different objects at two different places. We simply combined both to develop a model that could be studied and implemented by the designers in real life. As our project is open source and freely available on the internet, it can be studied, used and improved by any interested individual in computer graphics. This model could also be used in game development by the developers community as it gives the true glimpse of Nepal. This project could be the source of inspiration for our juniors to understand the workings of the algorithms and to generate the potential project ideas.

2. Literature Review

2.1. Background Theories

2.1.1. Blender

Fig 1: Blender Logo

Blender is the free and open source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, video editing and 2D animation pipeline. It can be used to create complex models and extract the vertices, normal coordinates, textures coordinates and faces in .obj file format which can be imported in OpenGL to render the model.

2.1.2. OpenGL

Fig 2: OpenGL Logo

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms. OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment.

2.1.3. GLFW

Fig 3: GLFW Logo

GLFW is a lightweight utility library for use with OpenGL. GLFW stands for Graphics Library Framework. It provides programmers with the ability to create and manage windows and OpenGL contexts, as well as handle joystick, keyboard and mouse input.

2.1.4. GLAD

Glad is a multi-language loader generator for GL/GLES/EGL/GLX/WGL based on the official Khronos Group specifications. It generates a loader for the developer needs based on the up-to-date specifications and can easily be extended to other languages.

2.1.5. GLSL

GLSL is a high level shading language with a syntax based on the C programming language. It was created by OpenGL ARB to give developers more direct control of the graphics pipeline without having to use ARB assembly language or hardware- specific languages.

2.1.6. Shaders

Shaders are little programs that rest on the GPU. These programs are run for each specific section of the graphics pipeline. In a basic sense, shaders are nothing more than programs transforming inputs to outputs.

There are mainly two types of shaders and they are:

I. Vertex Shader
II. Fragment Shader

2.1.6.1. Vertex Shader

Vertex shaders manipulate coordinates in a 3D space and are called once per vertex. The purpose of the vertex shader is to set up the gl_Position variable — this is a special, global, and built-in GLSL variable. gl_Position is used to store the position of the current vertex.

The void main() function is a standard way of defining the gl_Position variable. Everything inside void main() will be executed by the vertex shader. A vertex shader yields a variable containing how to project a vertex's position in 3D space onto a 2D screen.

2.1.6.2. Fragment Shader

Fragment shaders define RGBA (red, green, blue, alpha) colors for each pixel being processed — a single fragment shader is called once per pixel. The purpose of the fragment shader is to set up the gl_FragColor variable. gl_FragColor is a built-in GLSL variable like gl_Position.

The calculations result in a variable containing the information about the RGBA color.

2.1.7. Textures

A texture is a 2D image used to add detail to an object. Textures are basically used as images to decorate 3D models and can be used to store many different kinds of data. It is used to add detail to an object. Aside from images, textures can also be used to store a large amount of data to send to the shaders.

2.1.8. Lighting

Lighting deals with assigning the color to the surface points of objects. The scene is illuminated with light based upon a lightning model. Lighting model computes the color in terms of intensity values.

The luminous intensity or color of a point depends on the properties of the light source, the properties of the surface on which the point lies such as its reflectance, refraction, and the position, orientation of the surface with respect to the light source.

2.1.8.1. Ambient Light

Ambient light is a simple way to model the combination of the light reflections from various surfaces to produce a uniform illumination. Ambient light has no spatial or directional characteristics. The amount of the ambient light incident is constant for all the surfaces and in all directions. The position of the viewer is not important for modeling the ambient light.

I = IaKa

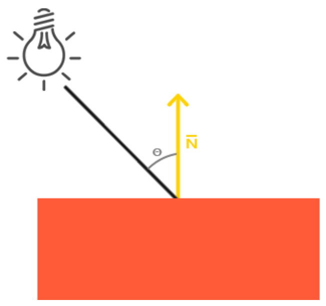
I: intensity

Ia: intensity of Ambient light

Ka: object's ambient reflection coefficient, 0.0 - 1.0 for each of R, G, and B

2.1.8.2. Diffuse Light

Diffuse light is modelled using a point light source.The light comes from a specific direction. This light reflects off the dull surface also. It is reflected with equal intensity in all the directions. The brightness depends upon the angle theta which is the angle between the surface normal and the direction to the light source. The position of the viewer is not considered for diffuse light.



$$I_{diff} = K_a I_a + K_d I_l (\vec{N}.\vec{L})$$

Fig 4: Diffuse Lighting

I = Ip Kd cos(theta) or I = Ip Kd(N' * L')

I: intensity

Ip: intensity of point light

Kd: object's diffuse reflection reflection coefficient, 0.0 - 1.0 for each of R, G, and B

N': normalized surface normal

L': normalized direction to light source

*: represents the dot product of the two vectors

It is rare that we have an object in the real world illuminated only by a single light. Even on a dark night there is some ambient light. To make sure all sides of an object get at least a little light we add some ambient light to the diffuse light. The combined equation for the intensity of light considering both the ambient light and diffuse light becomes

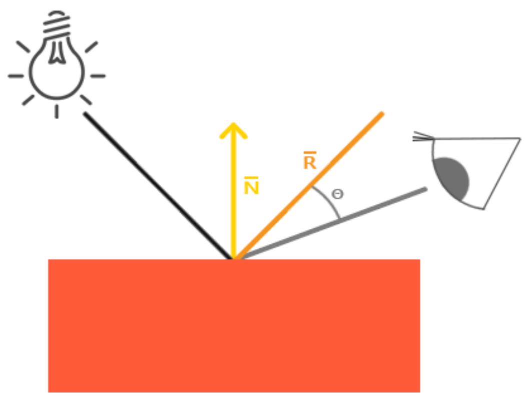
I = Ia Ka + Ip Kd(N' * L')

If the light source attenuation factor is also considered .i.e the intensity of the light source varying with the distance then the equation becomes

I = Ia Ka + Fatt Ip Kd(N' * L')

2.1.8.3. Specular Light

Specular light gives the reflection off the shiny surfaces. The position of the viewer is important in case of the specular reflection. Surfaces with the higher specular component such as metals and plastic give more specular intensity whereas surfaces with lower specular components such as chalk, sand do not reflect much specular light.



$$I_{spec} = K_s I_l (\vec{V}.\vec{R})^{n_s}$$

Fig 5: Specular Lighting

I = Ip cos^n(a) W(theta)

I: intensity

Ip: intensity of point light

n: specular-reflection exponent (higher is sharper falloff)

W: gives specular component of non-specular materials

If we put together ambient light, diffuse light and the specular light, the intensity equation becomes

I = Ia Ka + Ip Kd(N' * L') + Ip cos^n(a) W(theta)

2.1.9. Shading Models

The use of the lightning models individually to calculate the intensity of each fragment on the surface of the polygon mesh becomes computationally expensive. For the faster operation, shading models are used which are based upon use of the interpolation for the calculation of the surface intensity.

2.1.9.1. Flat Shading

This shading model considers only the ambient light. All the points of the surface have the same intensity when flat shading is applied. It is the fastest shading model. It is applicable only when the object the model is to be used is at a far distance from the light source or the viewer is at far distance.

2.1.9.2. Gouraud Shading

In this shading model the average unit normal vector at each vertex of the polygon is calculated. The lightning equation is used at each vertex. It makes use of the linear interpolation to calculate the intensities of the intermediate vertices in the edges. Colours are interpolated across the polygon. The Mach Band effect is caused by the use of Gouraud shading. It can be eliminated by increasing the number of vertices of the polygon. It is slower than the flat shading model.

2.1.9.3. Phong Shading

The Phong shading model is similar to the Gouraud shading model. Where Gouraud shading uses normals at the vertices and then interpolates the resulting colours across the polygon, Phong shading goes further and interpolates the normals. Linear interpolation of the normal values at each vertex are used to generate normal values for the pixels on the edges. Linear interpolation across each scan line is used to then generate normals at each pixel across the scan line. Whether we are interpolating normals or colours the procedure is the same. It is better at dealing with the highlights than the Gouraud shading model. It is slower than the Gouraud shading model.

2.1.10 Coordinate Systems in Computer Graphics

5 coordinate systems that are important in Computer Graphics :

- Local space - original coordinates of the object, relative to object’s origin
- World space - all coordinates relative to a global origin.
- View space (or Eye space)-all coordinates as viewed from a camera's perspective.
- Clip space- all coordinates as viewed from the camera's perspective but with projection applied
- Screen space- all coordinates as viewed from the screen. Coordinates range from 0 to screen width/height.

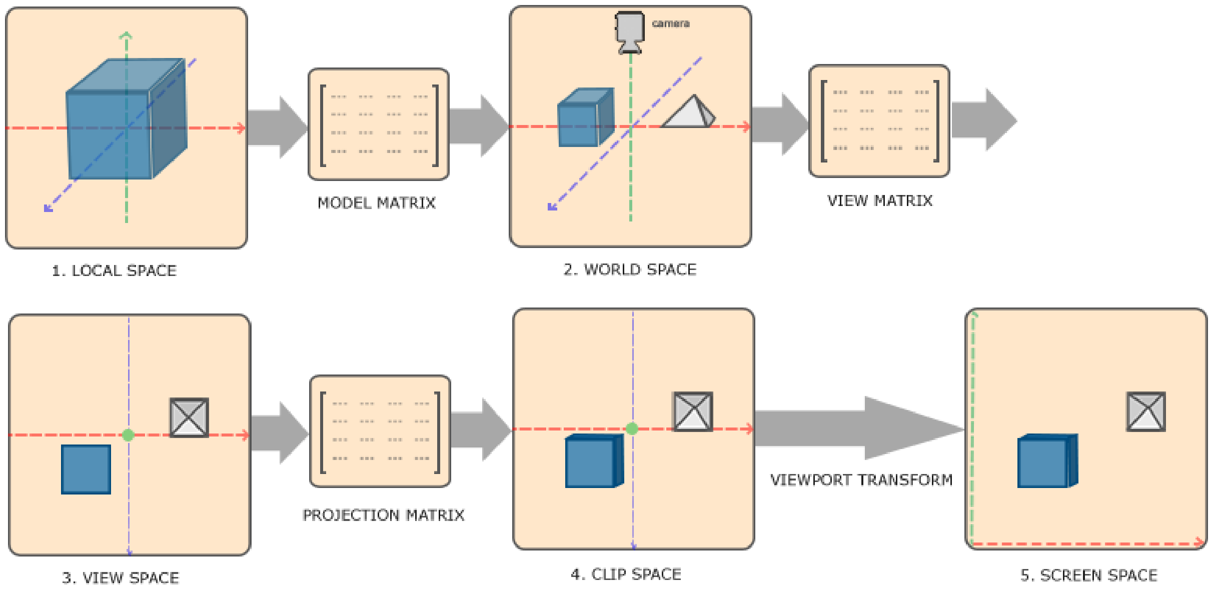


Fig 6: Graphics Pipeline

3. Methodologies

3.1 Block Diagram of Project Methodology

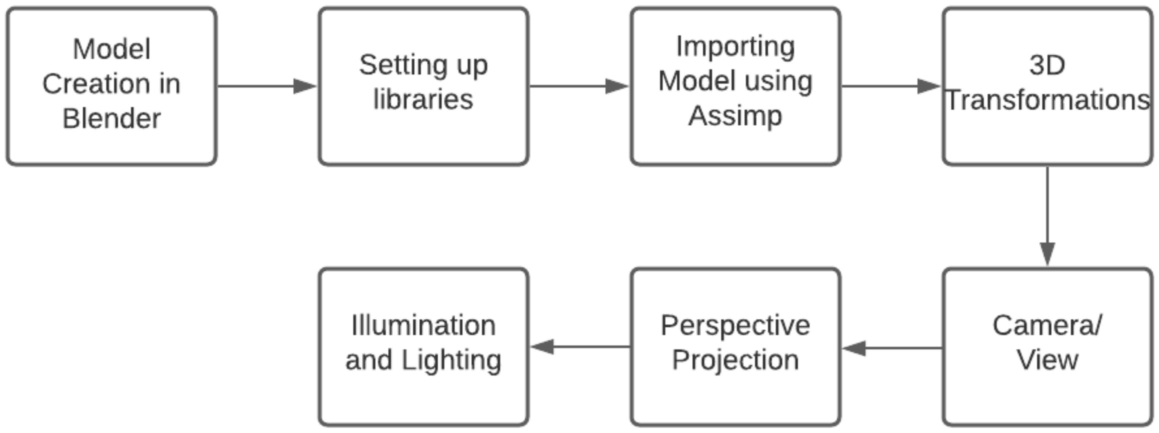

Model Creation in Blender
Setting up libraries
Importing Model using Assimp
3D Transformations
Camera/ View
Perspective Projection
Illumination and Lighting

Fig 7: Project Block Diagram

3.1.1. Creating Model in Blender

First of all, models of Dharahara and Ghantaghar were created in Blender using different meshes, curves and functionalities available in Blender. Different textures and colors were applied to the model and finally exported to the obj file.This generated obj file contained all the vertices, texture coordinates, normals and faces of the model in triangulated mesh form. Material file(mtl) was also generated from blender which linked various properties of materials.

3.1.2 Setting up libraries

There are many libraries used in our project for accomplishing specific tasks. The creation and management of windows along with processing user inputs is handled by GLFW. GLAD is used in order to query and load OpenGL extensions. Also, stb_image is used in order to load different textures. All the required libraries required in our project were initialized.

3.1.3 Importing Model using Assimp

The exported model from Blender in obj file format was imported using Assimp which stands for Open Asset Import Library. When importing a model via Assimp it loads the entire model into the Assimp data structure which consists of Scene objects. Assimp then has a collection of nodes where each node contains indices to data stored in the scene object where each node can have any number of children. A model of Assimp's structure is shown below:

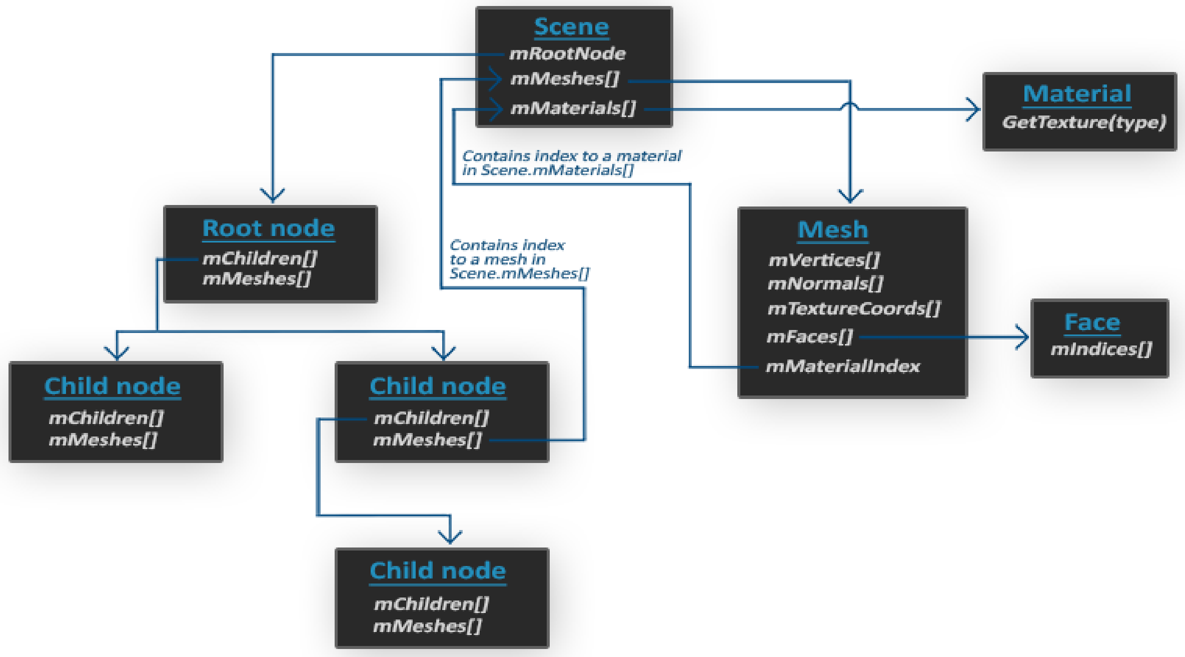


Fig 8: Assimp Components Tree

3.1.4 Transformations and Projection

After importing the model, various transformations like translation, scaling, rotation etc. were performed in order to position the model in the required place in 3D space by creating a model matrix. Similarly, the view matrix was created using an instance of Camera class. Finally, the model was projected to the screen using perspective projection in order to obtain a real world viewing experience by defining a projection matrix. All these matrices were combined and multiplied with the position of the object in the vertex shader to obtain the required realistic view.

3.1.5 Camera

In order to obtain a realistic view of the object, camera position can be changed by pressing keys or moving the mouse position.These positions are sensed and camera position is changed accordingly for changing the view of an object.

3.1.6 Lighting

Many different types of point lights are positioned in different positions by defining coordinates of the light source.For object color to produce ambient lighting, object color, light color and ambient strength was multiplied. For diffuse lighting, the lighting changes with angle. The dot product of normal and light coordinate vectors was multiplied to the object color. For specular lighting, object color changes with view position and the lighting intensity is obtained by multiplying color with dot product of reflection and the view direction.

3.2 Application Flow Chart

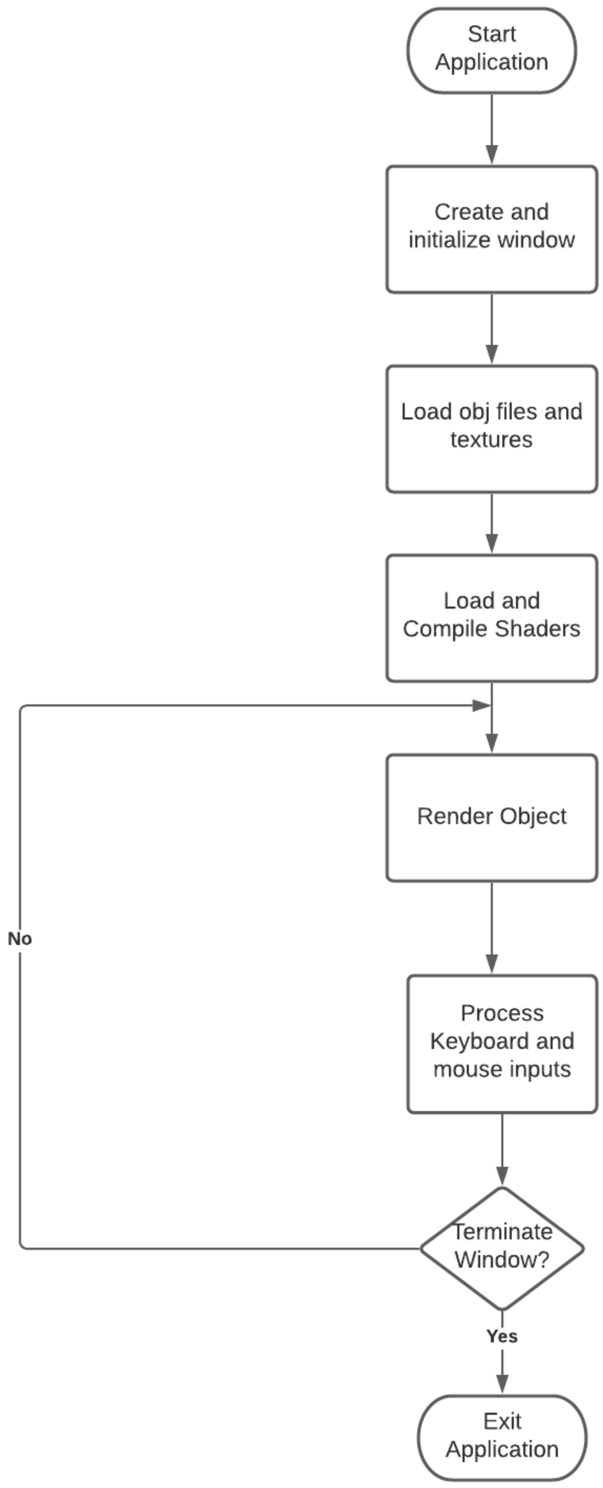


Fig 9: Flow Chart

3.3 Algorithms and Implementations

3.3.1 Transformations - Translation, Scaling, Rotation

3.3.1.1 Translation:

In Computer graphics, 3D Translation is a process of moving an object from one position to another in a three dimensional plane.

Consider a point object O that has to be moved from one position to another in a 3D plane.

Let,

Initial coordinates of the object O = (Xold, Yold, Zold)

New coordinates of the object O after translation = (Xnew, Ynew, Zold)

Translation vector or Shift vector = (Tx, Ty, Tz)

Given a Translation vector (Tx, Ty, Tz)-

Tx defines the distance the X-old coordinate has to be moved.

Ty defines the distance the Y-old coordinate has to be moved.

Tz defines the distance the Z-old coordinate has to be moved.

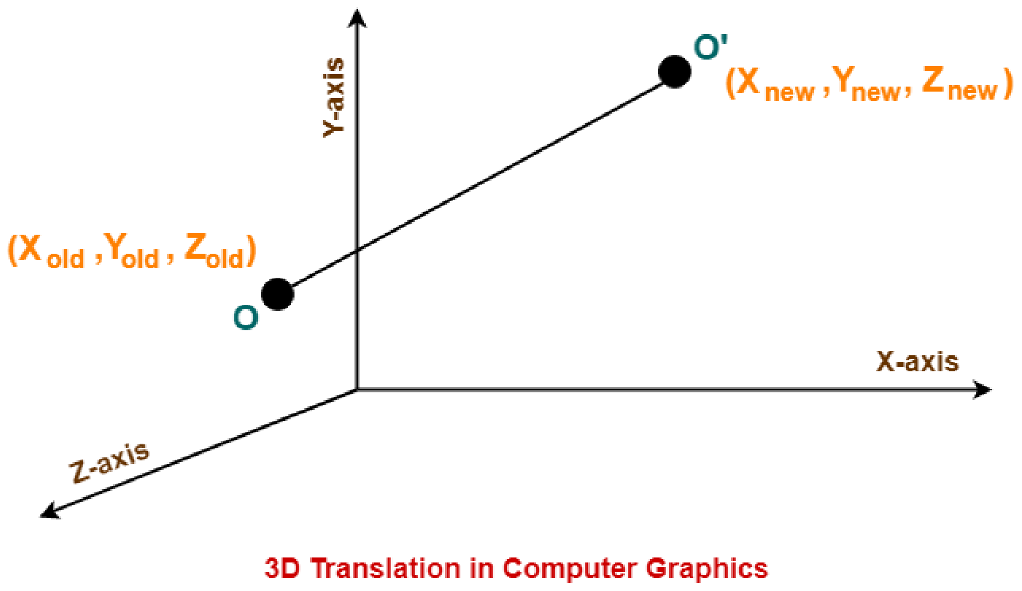

Y-axis
O'
$(X_{new}, Y_{new}, Z_{new})$
$(X_{old}, Y_{old}, Z_{old})$
O
X-axis
Z-axis
3D Translation in Computer Graphics

Fig 10: 3d Translation

This translation is achieved by adding the translation coordinates to the old coordinates of the object as-

Xnew = Xold + Tx (This denotes translation towards X axis)
Ynew = Yold + Ty (This denotes translation towards Y axis)
Znew = Zold + Tz (This denotes translation towards Z axis)

In Matrix form, the above translation equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

3D Translation Matrix

Fig 11: Equation for 3d Translation

3.3.1.2 Rotation

In Computer graphics, 3D Rotation is a process of rotating an object with respect to an angle in a three dimensional plane.

Consider a point object O that has to be rotated from one angle to another in a 3D plane.
Let-
Initial coordinates of the object O = (Xold, Yold, Zold)
Initial angle of the object O with respect to origin = Φ
Rotation angle = θ

New coordinates of the object O after rotation = (Xnew, Ynew, Znew)

In 3 dimensions, there are 3 possible types of rotation-

X-axis Rotation

Y-axis Rotation

Z-axis Rotation

For X-Axis Rotation-

This rotation is achieved by using the following rotation equations-

Xnew = Xold

Ynew = Yold x cosθ – Zold x sinθ

Znew = Yold x sinθ + Zold x cosθ

In Matrix form, the above rotation equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

3D Rotation Matrix

(For X-Axis Rotation)

Fig 12: Equation for 3D Rotation Along X-axis

For Y-Axis Rotation-

This rotation is achieved by using the following rotation equations-

- $X_{new} = Z_{old}$ x sinθ + X_{old} x cosθ
- $Y_{new} = Y_{old}$

- $Z_{new} = Y_{old} \times \cos\theta - X_{old} \times \sin\theta$

In Matrix form, the above rotation equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

3D Rotation Matrix

(For Y-Axis Rotation)

Fig 13: Equation for 3D Rotation Along Y-axis

For Z-Axis Rotation-

This rotation is achieved by using the following rotation equations-

- $X_{new} = X_{old} \times \cos\theta - Y_{old} \times \sin\theta$
- $Y_{new} = X_{old} \times \sin\theta + Y_{old} \times \cos\theta$
- $Z_{new} = Z_{old}$

In Matrix form, the above rotation equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

3D Rotation Matrix

(For Z-Axis Rotation)

Fig 14: Equation for 3D Rotation Along Z-axis

3.3.1.3 Scaling:

In computer graphics, scaling is a process of modifying or altering the size of objects. Scaling may be used to increase or reduce the size of objects. Scaling subjects the coordinate points of the original object to change.

Scaling factor determines whether the object size is to be increased or reduced.
If scaling factor > 1, then the object size is increased.
If scaling factor < 1, then the object size is reduced.

Consider a point object O that has to be scaled in a 3D plane.
Let-
Initial coordinates of the object O = (Xold, Yold,Zold)
Scaling factor for X-axis = Sx
Scaling factor for Y-axis = Sy
Scaling factor for Z-axis = Sz
New coordinates of the object O after scaling = (Xnew, Ynew, Znew)

This scaling is achieved by using the following scaling equations-

Xnew = Xold x Sx
Ynew = Yold x Sy
Znew = Zold x Sz

In Matrix form, the above scaling equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

3D Scaling Matrix

Fig 15: Equation for 3D Scaling

3.3.2 Perspective Projection

Perspective view gives the realistic representation of the appearance of the 3D object. Object positions are transformed to the view plane along the line that converges to a point called COP. In this projection farther the object from the viewer, small it appears. Two main characteristics of perspective are vanishing points and perspective foreshortening. On the basis of vanishing point, perspective projection is classified as:

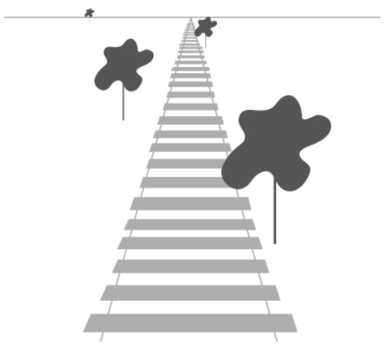


Fig 16: Road Scene with Perspective Projection used

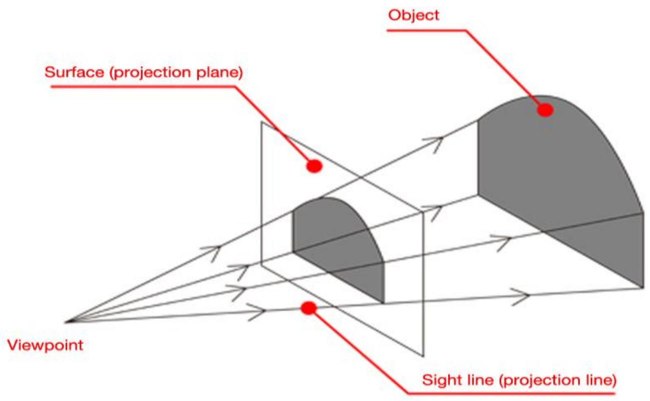


Fig 17: Perspective Projection

The perspective projection matrix is defined as:

$$\begin{bmatrix} \frac{1}{aspect * \tan(\frac{fov}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{fov}{2})} & 0 & 0 \\ 0 & 0 & -\frac{far + near}{far - near} & -\frac{2 * far * near}{far - near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Fig 18 : Transformation Matrix For Perspective Projection

3.3.3 3D Viewing Transformations

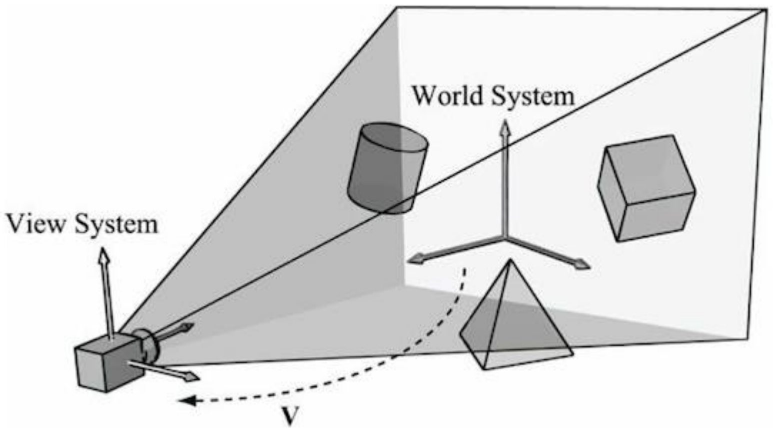


Fig 19: Viewing Transformation

The viewing transformation is done to transform the world coordinates to the view-space coordinates in such a way that each object is as seen from the viewer's point of view. It is the first step of the 3D viewing pipeline. The viewing transformation is performed in two steps. First, translate the viewing-coordinate origin at world position P0 = (x0, y0, z0) to the origin of the world coordinate system. The matrix for translating the viewing origin to the world origin is

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fig 20: Translation Matrix for Viewing Transformation

Now, the rotation is performed to align viewing-space coordinate axes with the world coordinate axes. The composite matrix for the rotation is

$$\mathbf{R} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fig 21: Rotation Matrix for Viewing Transformation

The view-plane normal N defines the direction for the viewing-space z axis and the view-up vector V is used to obtain the direction for its y axis. Using the input values for N and V, we can compute a third vector, U, that is perpendicular to both N and V with a vector cross product. Vector U then defines the direction for the positive x axis of the viewing-space. The unit vectors along the directions U,V and N i.e u, v and n represent the viewing-space coordinate axes.

$$\mathbf{n} = \frac{\mathbf{N}}{|\mathbf{N}|} = (n_x,\ n_y,\ n_z)$$
$$\mathbf{u} = \frac{\mathbf{V} \times \mathbf{n}}{|\mathbf{V} \times \mathbf{n}|} = (u_x,\ u_y,\ u_z)$$
$$\mathbf{v} = \mathbf{n} \times \mathbf{u} = (v_x,\ v_y,\ v_z)$$

Fig 22: uvn Unit Vectors

The final transformation matrix for the transformation from world coordinates to viewing coordinates is obtained as the product of this rotation matrix and translation matrix.

$$\mathbf{M}_{WC,\,VC} = \mathbf{R} \cdot \mathbf{T}$$

Fig 23: Equation For Viewing Transformation

The viewing-space coordinates obtained from the viewing transformation are later transformed to the 2D view plane using projection transformation.

3.3.4 Phong Reflection Model

Phong shading may also refer to the specific combination of Phong interpolation and the Phong reflection model, which is an empirical model of local illumination. It describes the way a surface reflects light as a combination of the diffuse reflection of rough surfaces with the specular reflection of shiny surfaces. It is based on Bui Tuong Phong's informal observation that shiny surfaces have small intense specular highlights, while dull surfaces have large highlights that fall off more gradually. The reflection model also includes an ambient term to account for the small amount of light that is scattered about the entire scene.

$$I = I_a k_a + f_{att} I_p [k_d (\overline{N}.\overline{L}) + k_s (\overline{V}.\overline{R})^n]$$

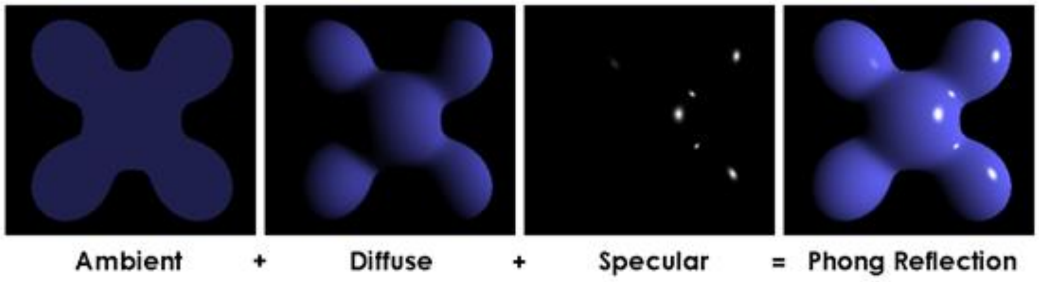


Fig 24: Phong Illumination Model

3.3.5 Depth Testing:

The depth testing is performed using the z-buffer method. It is done in order to detect the visible surfaces so that they are rendered on the screen. In this method, the depth value of each pixel in the projection plane is compared for the detection of the visible surface. The Depth is the distance of the surfaces of polygons in the scene along the z-axis from the view plane.

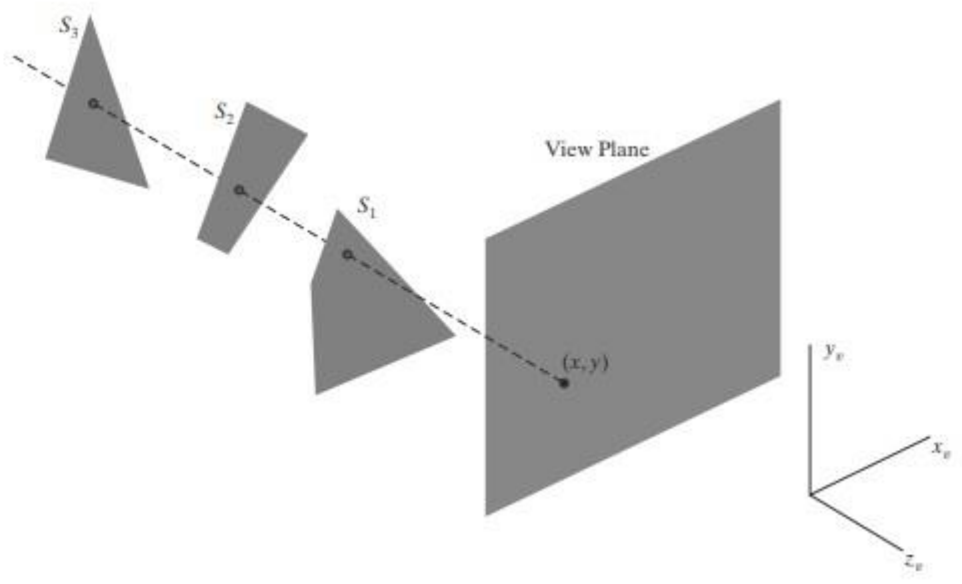


Fig 25: Z-buffer Test

The algorithm is implemented using a Depth buffer, to store the depth value of the fragments and a fragment buffer, to store the color values for storing the color of the fragments. The depth buffer is automatically created by the windowing system to store the depth values and initialized to 1.0 for all the screen fragments. Depth testing is enabled in opengl calling the function glEnable with flag GL_DEPTH_TEST..

The opengl maintains the depth values of the fragments in the depth buffer. During the rendering process, opengl compares the z-value of the fragments with the contents of the depth buffer. The fragment is rendered and its depth value is updated in the depth buffer whenever the fragment passes the depth test else the fragment is discarded. The Depth buffer is cleared before rendering each frame.

3.4 Languages and Tools

Languages used:

- C++
- GLSL

Libraries Used:

- OpenGl
- GLAD
- GLFW
- Assimp

Tools Used:

- Blender
- Visual Studio 2019
- Notepad

4. Result

Output Images:

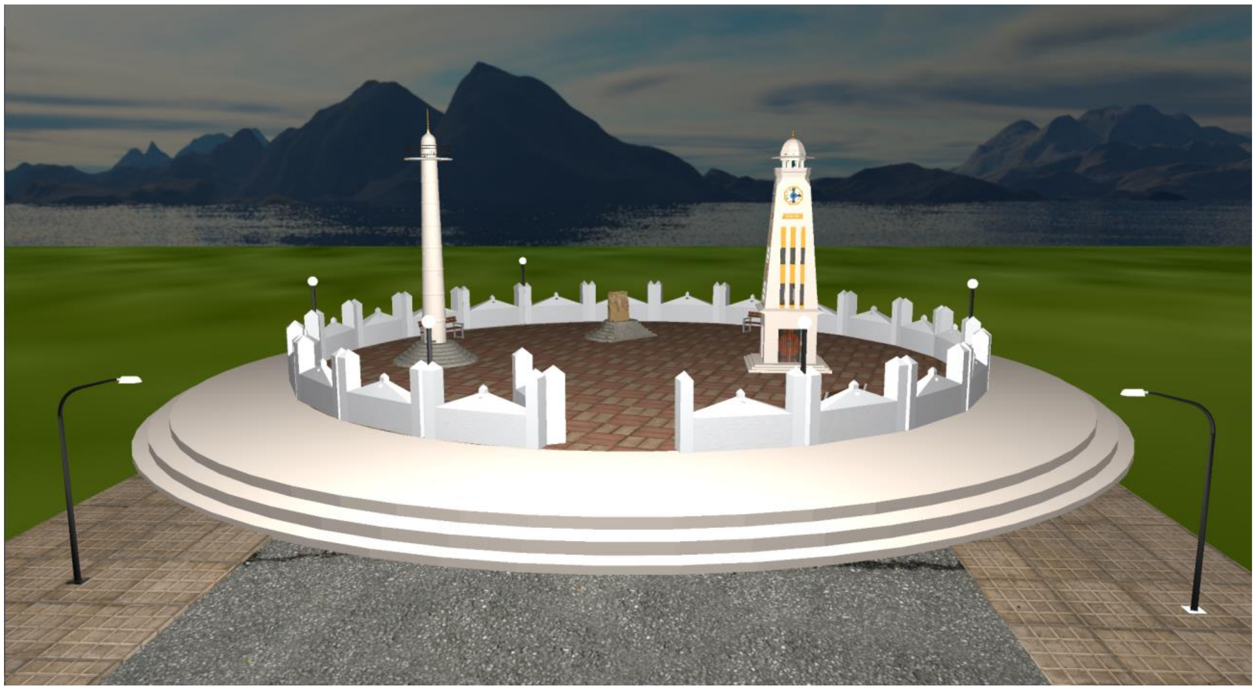


Fig 26: Day Scene (Close)

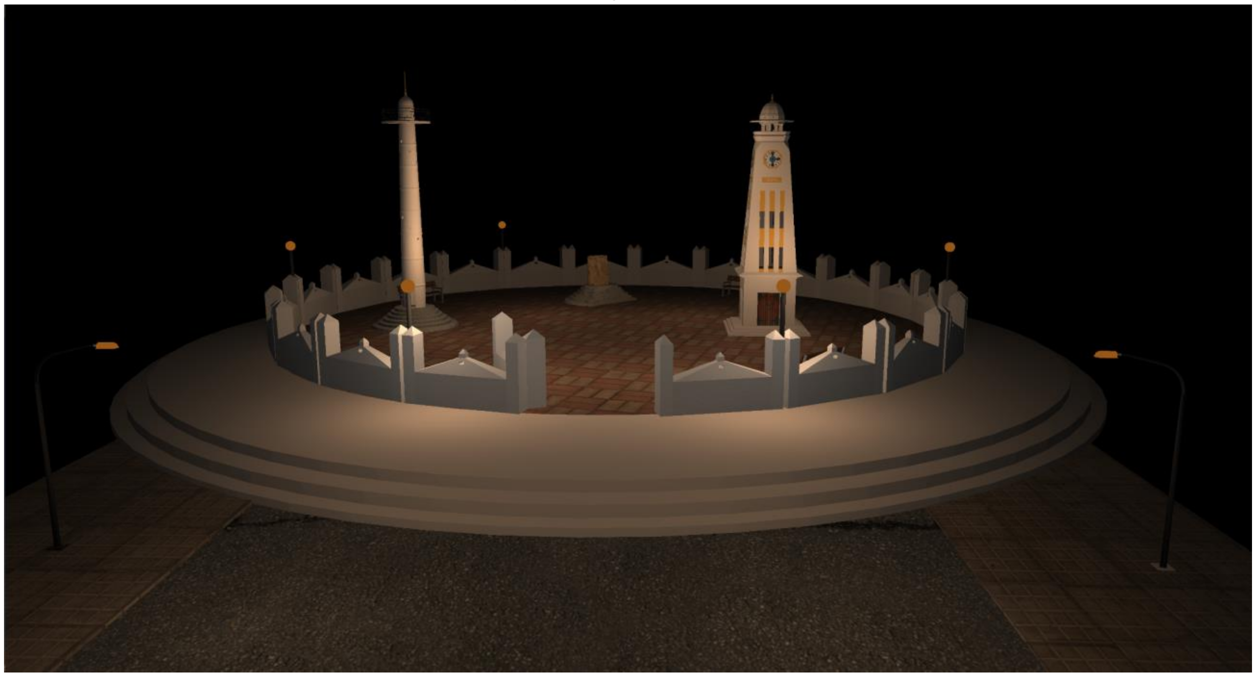


Fig 27: Night Scene (Close)

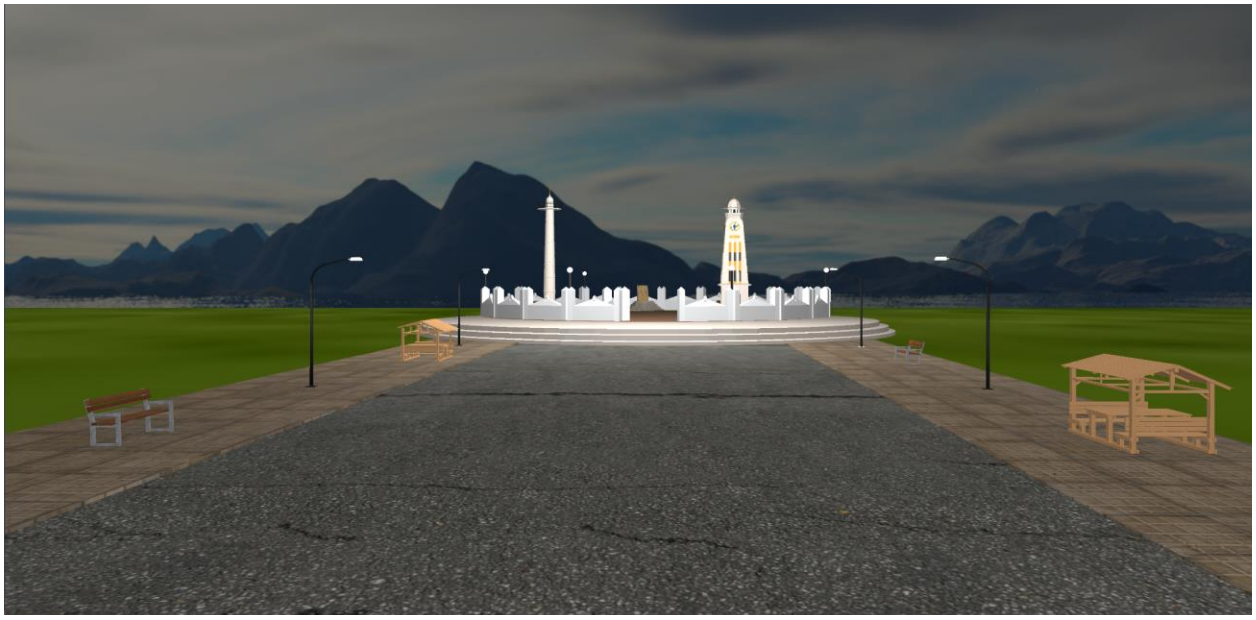


Fig 28: Day Scene (Far)

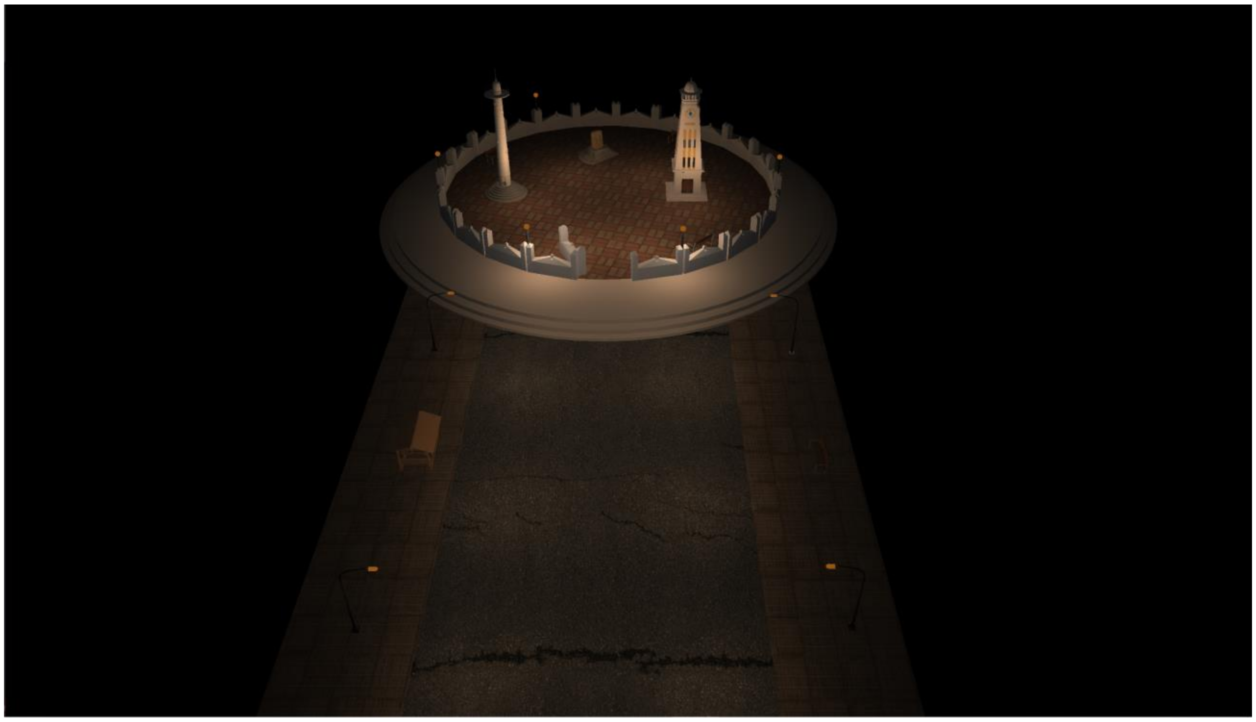


Fig 29: Night Scene (Far)

5. Conclusion

In this way we were able to render a 3D model of Dharahara and Ghantaghar. We successfully applied the algorithms that we studied in our course. Our project demonstrates the implementation of graphics algorithms like transformation, scaling, rotation. We created our own transformation matrix to do these composite transformations. For the lighting model, we used a phong shading model as it was most suitable for the object like our’s. For the visible surface detection, We used the z-buffer algorithm. We learnt to use blender for creating realistic models. Also, In this journey we learnt the importance of teamwork.

6. Limitations and Future Works

The application currently features a fly style camera for moving freely around the scene. It can move across the objects in the scene and realism is missing. Collision detection algorithms can be used to avoid it and make it more realistic.

The meshes for the scenes modeled using blender have a large number of vertices. No checking of the double vertices has been done. Thus, presence of the redundant vertices might have hampered the performance causing lags. We can redesign the same models eliminating the redundant vertices.

The current application is designed with the use of opengl api. Better graphics api such as Vulkan can be used for designing the application with faster performance and better image quality at low overhead.

Other future enhancements include redesigning the application as a 3D game with the addition of the characters, levels, physics, audio, First person camera, etc. The scene in the application can also be used in other open world games.

7. References

1. Donald D. Hearn and M. Pauline Baker, “Computer Graphics C version (2nd edition)”
2. Foley, Van Dam, Feiner, Hughes “Computer Graphics principles and practice (Second Edition in C”)
3. https:://learnopengl.com, Learn OpenGL, extensive tutorial resource for learning Modern OpenGL