3D MODELING O F SWOYAMBHUNATH STUPA

A simple graphics renderer

Date : 2021/09/31

Project Report - Computer Graphics

TITLE: Rendering Swayambhunath Temple

Submitted by:

Name :Bishal Bashyal(075BCT026)
Name :Bishal Katuwal(075BCT028)
Name :Dimple Saraogi(075BCT034)
Name: :Janak Sharma (075BCT040)

Submitted To:

Basanta Joshi, Ph.D.
Department Of Electronics And Computer Engineering

ABSTRACT

Computer graphics is a branch of computer science that investigates ways for creating and modifying visual images digitally. The field of computer graphics is concerned with the creation of visuals using computers. Computer graphics are now used in digital photography, cinema, video games, mobile phone and computer displays, and a wide range of other applications.

The 3D model includes the famous Nepalese monuments: Swayambhunath, also known as Swayambhu, an ancient religious complex atop a hill in the Kathmandu Valley. We've gone over the computer graphics techniques that were employed to make the things appear more realistic. The report thoroughly covers the theoretical parts of these algorithms and techniques that are required to comprehend the project.

For the major part of this project we will be focusing on the physical visible structures of Swayambhunath stupa while trying to approximately represent it in terms of a 3d Model. The planning of this project assumed the use of colour fill, shape geometry and camera implementation.

Table of Contents

Introduction 7
Background 7
Objectives 8
Motivations 9
Scope 10
Literature Review 11
Related Theories 11
Related Works 17
Methodologies 18
System Block diagram 18
Implementations 19
Languages and Tools 25
Result 26
Output Images 27
Conclusion 33
Limitations and Future Works 34
References 35

List of Figures

1. Figure 1: Plotting of Vertex Data
2. Figure 2: Graphics Pipeline
3. Figure 3: Projected Pixel for the main Stupa
4. Figure 4: Wireframe for main Stupa
5. Figure 5: Flat Shading
6. Figure 6: Smooth shading and Z-buffer
7. Figure 7: Light on top
8. Figure 8: Light on front
9. Figure 9: Light on Back
10. Figure 10: Top View after some ambient lightning

List of Symbols and Abbreviations

C.E: Common Era
2D : Two Dimensional
3D : Three Dimensional
API: Application Programming Interface
OpenGl: Open Graphics Library
BCE: Before Common Era

Introduction

Background

Our 3D rendering project is focused on the Swayambhunath temple which holds significant importance as one of the religious sites of nepal.

The temple was founded by the great-grandfather of King Mānadeva (464-505 CE), *King Vṛsadeva*, about the beginning of the 5th century CE. This seems to be confirmed by a damaged stone inscription found at the site, which indicates that King Vrsadeva ordered work done in 640 CE.
However, Emperor Ashoka is said to have visited the site in the third century BCE and built a temple on the hill which was later destroyed.
Although the site is considered Buddhist, the place is revered by both Buddhists and Hindus.

Although this heritage is rich in its presence decorated by many sub-temples, monuments , statues and its own atmosphere of things, Our project is limited to approximately modelling and rendering only some of the important aspects of this temple. This includes the temple itself as the focal point, 2 big sub-temples on the side and other temples on the same plane.

Objectives

The objectives as enlisted before the initiation of our project are:

- To understand the concepts of Computer Graphics
- To learn 3D rendering and geometry.
- To understand 2D and 3D graphing concepts
- To learn about OpenGL specifications.
- To improve the ability to work and cooperate in a team.
- To understand the concept of shaders and lighting models.
- To understand the basic implementation and usage of camera, color fill and lighting.

Motivations

The main motivation behind this project was to understand the concepts of 3D rendering starting just from scratch pixels and vertex data. Swyambhunath was chosen as a subject of importance to understanding this concept because of its religious importance and also the structural complexity in its vertices and face orientations which would make the project visually appealing.

Getting in further towards the importance of this project. Any 3d rendering engine cannot be built without following the steps involved in creating a graphics pipeline. This project, though minimalist in its scope and application, was extremely helpful in conceptually implementing the different phases of constructing a 3D renderer.

Scope:

The main scope of this project lies in the underlying constructs that need to be done before being able to produce a 3d output on a 2d screen.As explained throughout this report,the main motive of this project is to learn how the real life 3D objects are transformed on the screen. The code will be modular thus it will help for extension of this project in future as we will be able to add different objects and manipulate the scene accordingly.
This project however can go beyond modeling of Swayambhunath and may include different places surrounding that object to better reflect the real life scenario of the assumed model.

Literature Review

Related Theories

Blender

Blender is a free and open source 3D modeling program. Modeling, rigging, animation, simulation, rendering, compositing, and motion tracking, as well as video editing and 2D animation pipelines, are all supported. It may be used to generate complicated models and then export the vertices, normal coordinates, texture coordinates, and faces as.obj files, which can then be loaded into OpenGL to render the model.

Open Gl

OpenGL is the industry standard for creating portable, interactive 2D and 3D graphics apps. OpenGL has been the industry's most commonly used and supported 2D and 3D graphics application programming interface (API) since its inception in 1992, delivering hundreds of programs to a range of computer platforms. By embracing a large collection of rendering, texture mapping, special effects, and other sophisticated visualization functions, OpenGL stimulates innovation and accelerates application development. Developers may take advantage of OpenGL's capabilities across a wide range of desktop and workstation platforms, assuring widespread application deployment.

GLUT

The OpenGL Utility Toolkit (GLUT) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system. Functions performed include window definition, window control, and monitoring of keyboard and mouse input.

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 is

determined by the light source's attributes, as well as the properties of the surface on which the point is located, such as reflection and refraction, as well as the surface's position and orientation in relation to the light source.

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

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 = 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

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

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 = 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)

Shading Models

It becomes computationally expensive to employ individual lightning models to determine the intensity of each piece on the surface of the polygon mesh. For a speedier operation, shading models are utilized, which are based on the usage of interpolation for surface intensity calculation.

Flat Shading

This shading model solely takes into account ambient light. When flat shading is used, all of the points on the surface have the same intensity. It's the quickest shading method. It is only usable when the modeled item is located far away from the light source or when the viewer is located far away.

Gouraud Shading

The average unit normal vector at each vertex of the polygon is determined in this shading scheme. At each vertex, the lightning equation is employed. The intensities of the intermediate vertices in the edges are calculated using linear

interpolation. The polygon's colors are interpolated. The use of Gouraud shading causes the Mach Band effect. It can be removed by increasing the polygon's number of vertices. It is more time consuming than the flat shading model.

Phong Shading

The shading model used by Phong is comparable to the shading model used by Gouraud. Phong shading interpolates the normals, whereas Gouraud shading employs normals at the vertices and then interpolates the resulting colors across the polygon. To create normal values for the pixels on the edges, linear interpolation of the normal values at each vertex is utilized. The normals at each pixel along each scan line are generated via linear interpolation over each scan line. The process is the same whether we're interpolating normals or colors. It is superior to the Gouraud shading model in terms of handling highlights. It is more time consuming than the Gouraud shading model.

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.

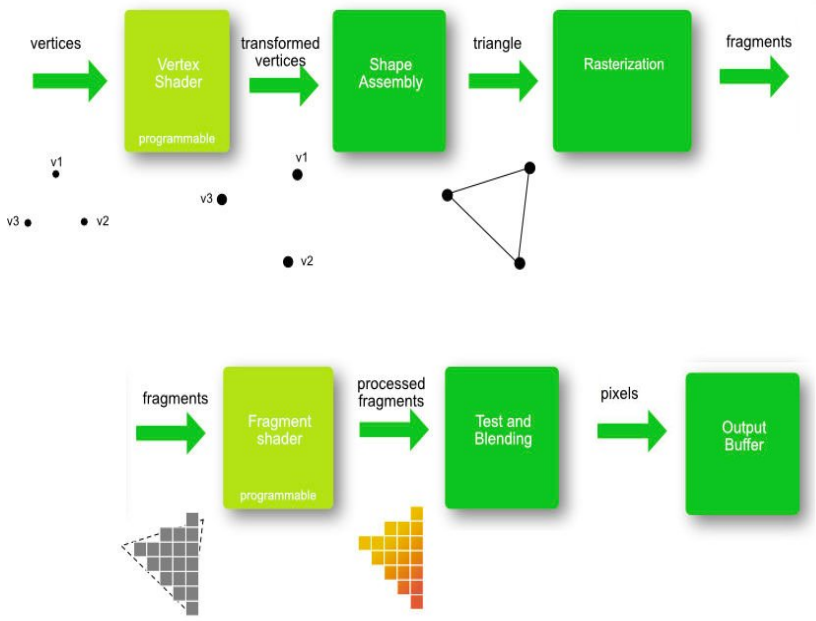


Fig 2:Graphics Pipeline

Most of the 2D and 3D graph concepts have already been established. Thus most, if not all, rendering software run on the same graphing theories and concepts.
In short, the related theories can be listed as follows:

1. Bresenham Line Drawing Algorithm
2. Homogeneous matrix forms
3. Scan Line Rasterization
4. Barycentric Coordinates(Rasterization and Interpolation)
5. z-Buffer Algorithm
6. Flat Shading
7. Gouraud Shading
8. Phong Shading

Related Works

There are plenty of 3D rendering software out there in the world. The most common being Blender. However for 3D to 2D graphing, following projects were studied:

Blender 3D :https://github.com/blender

TinyRenderer : https://github.com/ssloy/tinyrenderer

Methodologies

System Block diagram

This system block diagram depicts the basic workflow of how 3d models are rendered in opengl, which will be the base of our implementation of this graphics project.

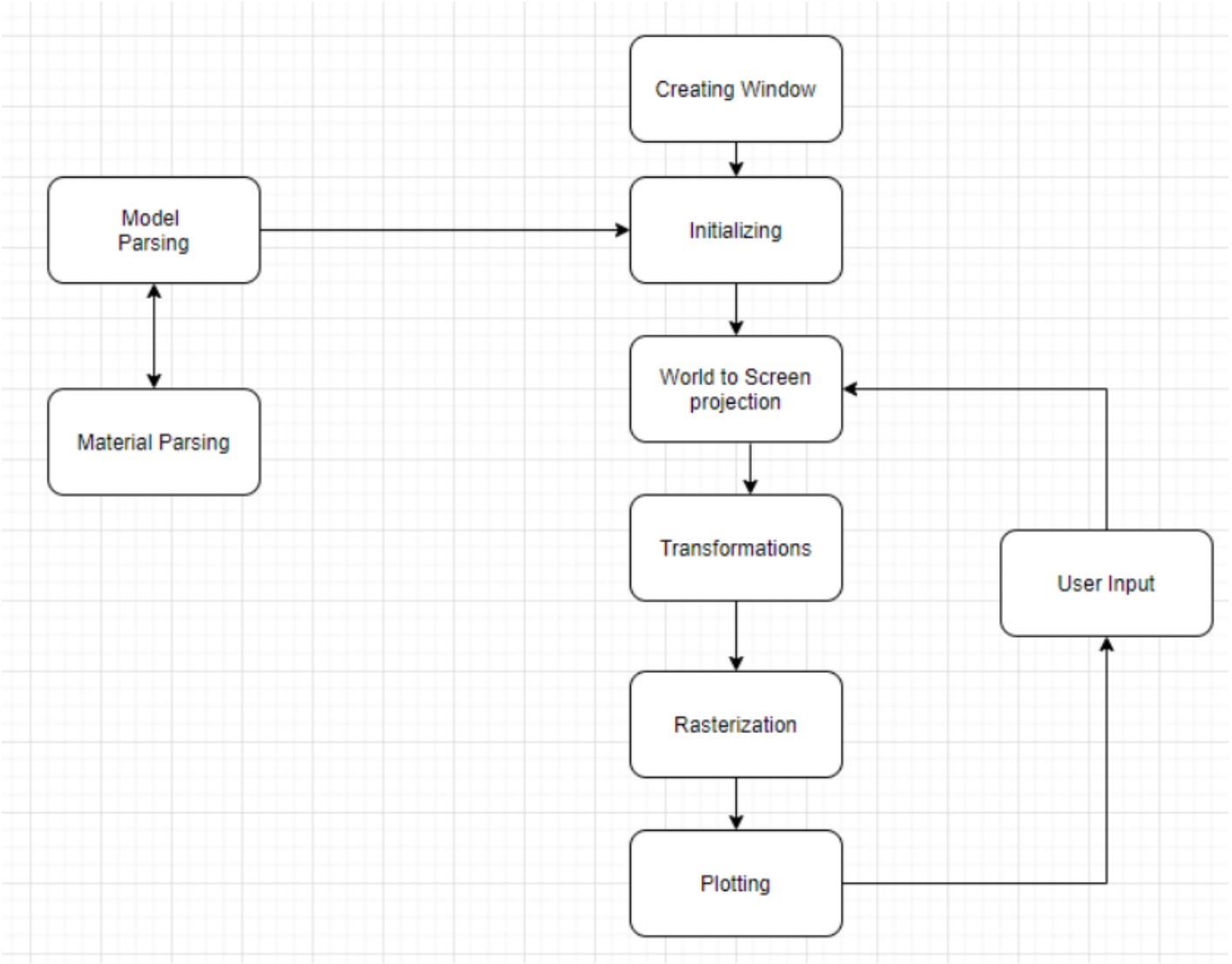


Figure 1: Plotting of Vertex Data

Algorithms and Implementations

We will be using OpenGL API to call the putpixel functions. Besides that everything is written from scratch.

The model to be displayed will be created in the program Blender from which we will import the model vertices and material properties

For camera movement we will need a controller which will be either a mouse to a keyboard in our case. The scene will move accordingly with the instruction that will be obtained from the above mentioned devices.

Following Algorithms are involved:

1. **World to Screen Transformation:**

A vector space is a mathematical structure that is defined by a given number of linearly independent vectors, also called base vectors; the number of linearly independent vectors defines the size of the vector space, therefore a 3D space has three base vectors, while a 2D space would have two. These base vectors can be scaled and added together to obtain all the other vectors in the space. Our models live in one specific vector space, which goes under the name of Model Space and it's represented with the canonical 3D coordinates system .

When an artist authors a 3D model he creates all the vertices and faces relatively to the 3D coordinate system of the tool he is working in, which is the Model Space. All the vertices are relative to the origin of the Model Space, so if we have a point at coordinates (1,1,1) in Model Space, we know exactly where it is. Every model in the game lives in its own Model Space and if you want them to be in any spatial relation you need to transform them into a common space (which is what is often called World Space).

The projection space is obtained from :

$$\begin{bmatrix} \frac{1}{width} & 0 & 0 & 0 \\ 0 & \frac{1}{height} & 0 & 0 \\ 0 & 0 & -\frac{2}{Z_{far}-Z_{near}} & -\frac{Z_{far}+Z_{near}}{Z_{far}-Z_{near}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and the perspective projection is given by:

$$\begin{bmatrix} \tan^{-1}(\frac{FOVx}{2}) & 0 & 0 & 0 \\ 0 & \tan^{-1}(\frac{FOVy}{2}) & 0 & 0 \\ 0 & 0 & -\frac{Z_{far}+Z_{near}}{Z_{far}-Z_{near}} & -\frac{2\,(Z_{near}\,Z_{far})}{Z_{far}-Z_{near}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

2. Transformation matrix

If we want to represent a transformation from one 3D space to another we will need a 4x4 Matrix. We will assume from here on a column vector notation, as in OpenGL. In order to apply the transformation we have to multiply all the vectors that we want to transform against the transformation matrix. If vectors were in Space A and the transformation was describing a new position of Space A relative to Space B, after the multiplication all the vectors would then be described in Space B.

The transformation matrix can be shown as

$$\begin{bmatrix} Transform\_XAxis.x & Transform\_YAxis.x & Transform\_ZAxis.x & Translation.x \\ Transform\_XAxis.y & Transform\_YAxis.y & Transform\_ZAxis.y & Translation.y \\ Transform\_XAxis.z & Transform\_YAxis.z & Transform\_ZAxis.z & Translation.z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where Transform_XAxis is the XAxis orientation in the new space, Transform_YAxis is the YAxis orientation in the new space, Transform_ZAxis is the ZAxis orientation in the new space and **Translation** describes the position where the new space is going to be relatively to the active space.

If we want simple transformations rather than a complex transformation matrix, we have

a. Translation

$$\begin{bmatrix} 1 & 0 & 0 & Translation.x \\ 0 & 1 & 0 & Translation.y \\ 0 & 0 & 1 & Translation.z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

b. Scaling

$$\begin{bmatrix} Scale.x & 0 & 0 & 0 \\ 0 & Scale.y & 0 & 0 \\ 0 & 0 & Scale.z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

c. Rotation about x-axis

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

d. **Rotation about Y-axis**

$$\begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

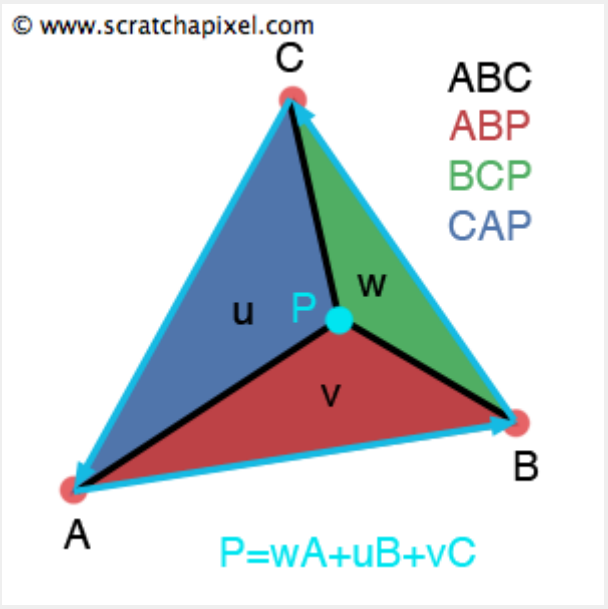
e. **Rotation about z-axis**

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.Barycentric interpolation

Barycentric coordinates can be used to express the position of any point located on the triangle with three scalars. The location of this point includes any position inside the triangle, any position on any of the three edges of the triangles, or any one of the three triangle's vertices themselves. To compute the position of this point using barycentric coordinates we use the following equation (1):

$$P=uA+vB+wC.$$



3. Shading

Each polygon surface is rendered with Gouraud Shading by performing the following calculations:

1. Determining the average unit normal vector at each polygon vertex.
2. Apply an illumination model to each vertex to determine the vertex intensity.
3. Linear interpolate the vertex intensities over the surface of the polygon.

Fig: The normal vertex at vertex V is calculated as the average of surface normal for each polygon sharing the vertex.

Thus, for any vertex position V, we acquire the unit vertex normal with the calculation

$$\mathbf{N_V} = \frac{\sum_{k=1}^{n} \mathbf{N_k}}{\left|\sum_{k=1}^{n} \mathbf{N_k}\right|}$$

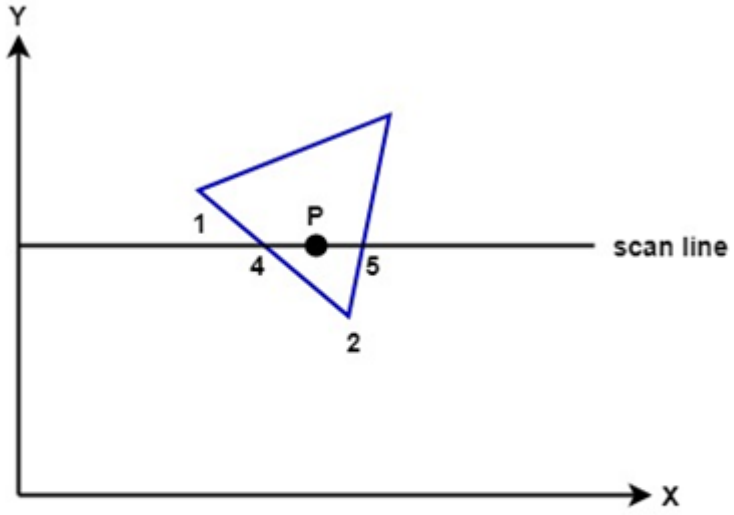


Fig: For Gouraud Shading, the intensity at point 4 is linearly interpolated from the intensities at vertices 1 and 2. The intensity at point 5 is linearly interpolated from intensities at vertices 2 and 3. An interior point P is then assigned an intensity value that is linearly interpolated from intensities at position 4 and 5.

$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_4}{y_1 - y_2} I_2$$

Similarly, the intensity at the right intersection of this scan line (point 5) is interpolated from the intensity values at vertices 2 and 3. Once these bounding intensities are established for a scan line, an interior point (such as point P in the previous fig) is interpolated from the bounding intensities at point 4 and 5 as

$$I_P = \frac{x_5 - x_p}{x_5 - x_4} I_4 + \frac{x_p - x_4}{x_5 - x_4} I_5$$

4. Z-buffer

For all pixels on the screen, depth of [x, y] is set to to 1.0 and intensity [x, y] to a background value.

For each polygon in the scene, find all pixels (x, y) that lie within the boundaries of a polygon when projected onto the screen. For each of these pixels:

(a) Calculate the depth z of the polygon at (x, y)

(b)

If z < depth [x, y], this polygon is closer to the observer than others already recorded for this pixel. In this case, set depth [x, y] to z and intensity [x, y] to a value corresponding to polygon's shading.

If instead z > depth [x, y], the polygon already recorded at (x, y) lies closer to the observer than does this new polygon, and no action is taken.

3. After all, polygons have been processed; the intensity array will contain the solution.

Languages and Tools:

- C++
- OpenGL
- Blender
- Visual Studio 2019

Result

Our entire Swayambhunath is a single model with distinct structures. The structures are:

- The Domed stupa
- Two distinct large temples
- Other smaller structures

These 3 major structures are assumed to be the building block for this project and the approximate representation for the Temple itself. We have also built some approximation of surrounding structures in the project.

Output Images

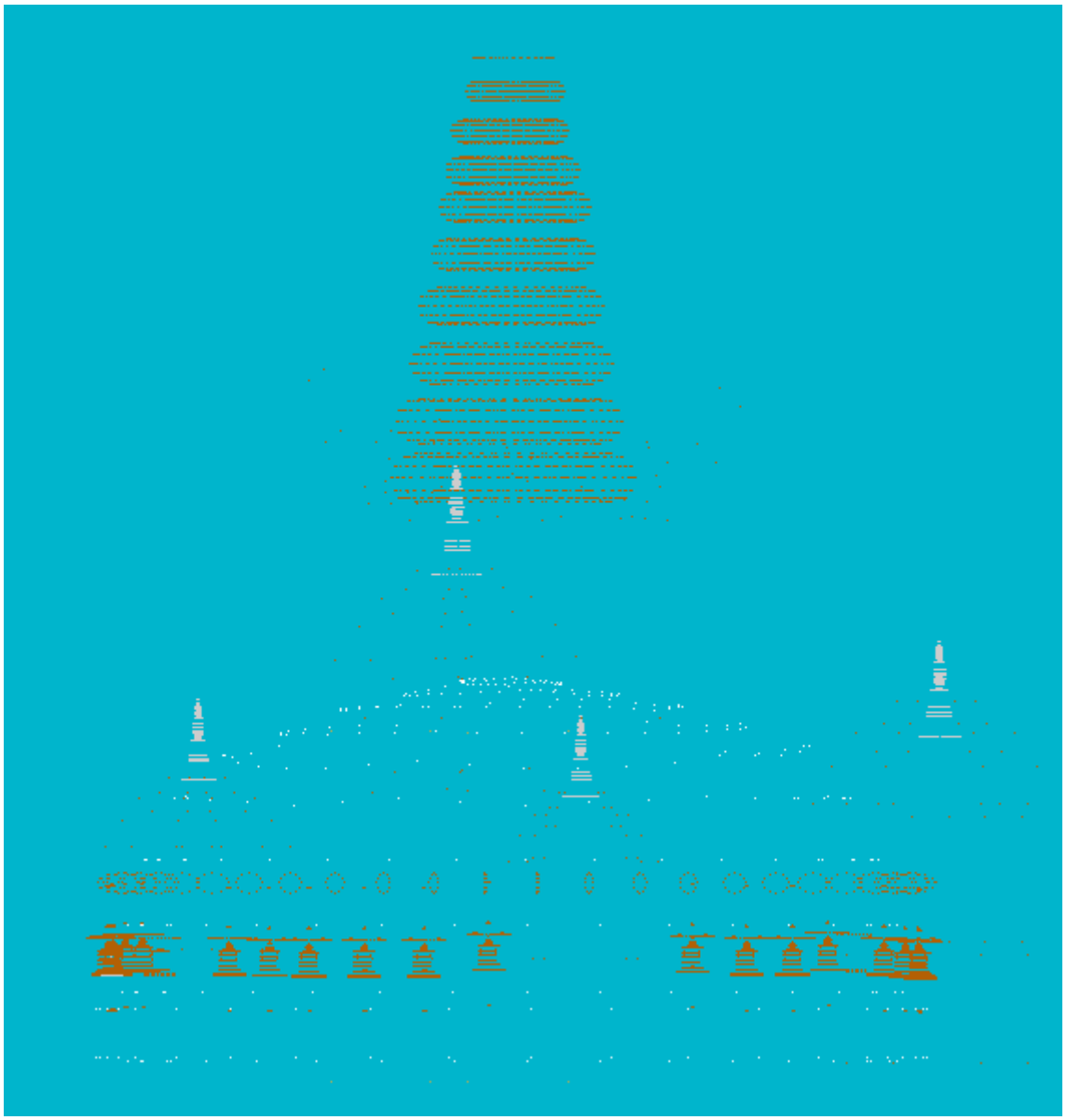


Figure 3: Projected Pixel for the main Stupa

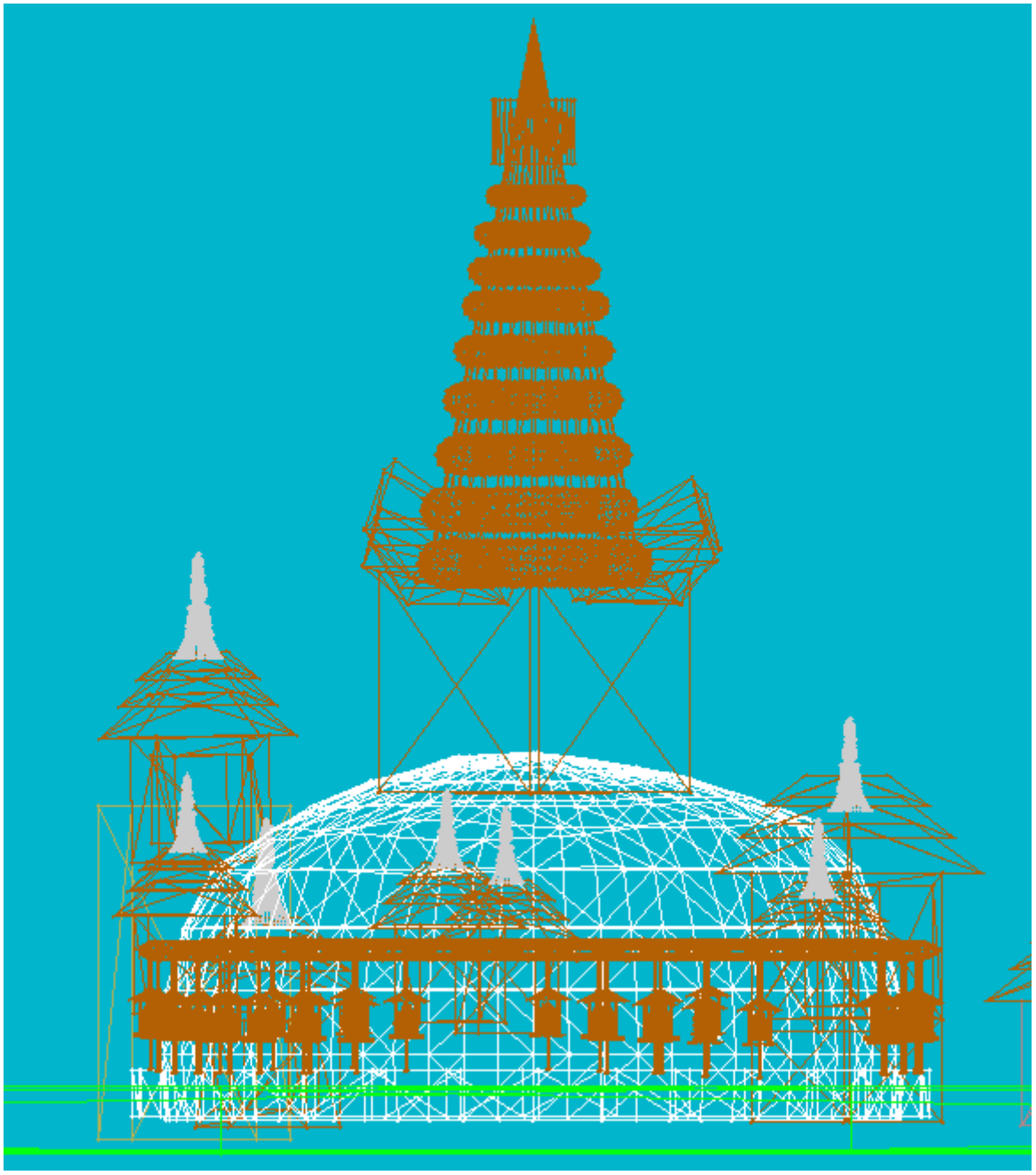


Figure 4: Wireframe for main Stupa

Figure 5: Flat Shading

Figure 6: Smooth shading and Z-buffer

Some figures with varying light source position:

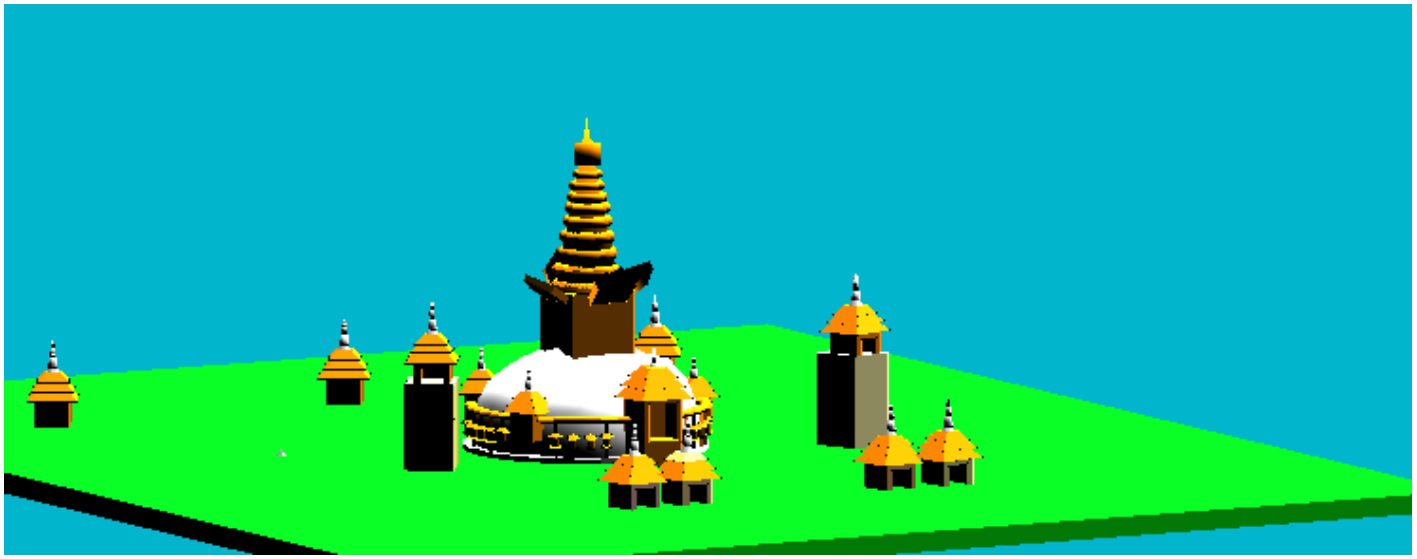


Figure 7: Light on top

Figure 8: Light on front

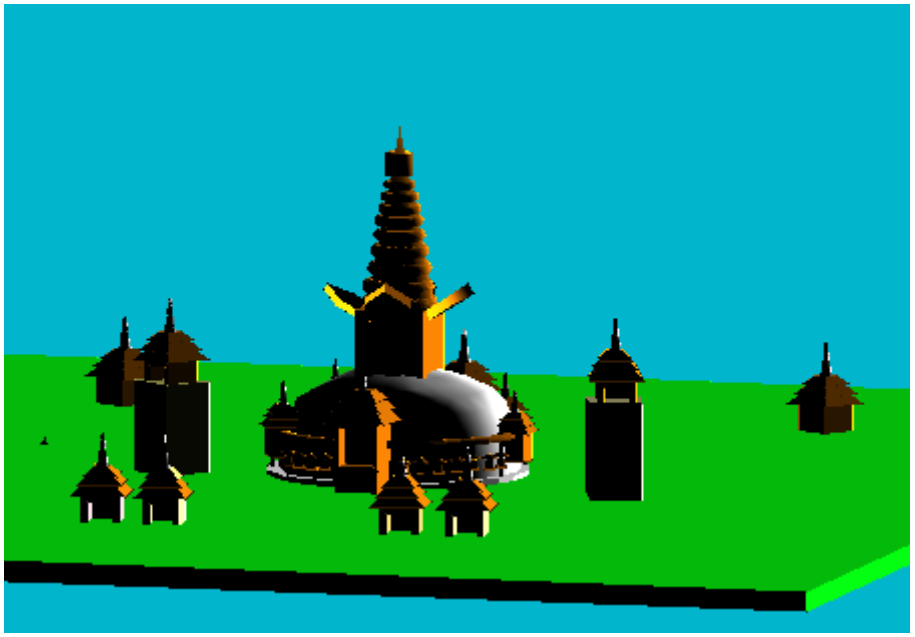


Figure 9: Light on Back

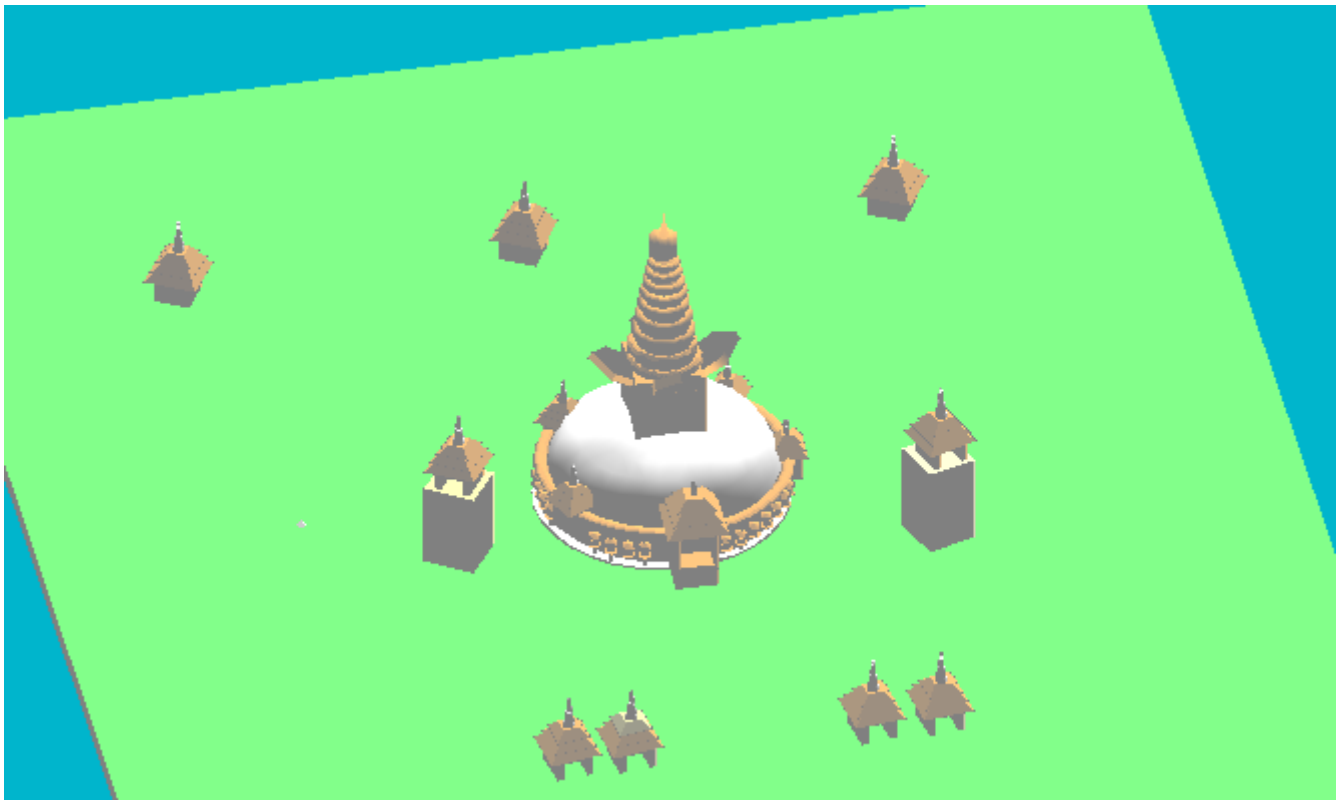


Figure 10: Top View after some ambient lightning

Conclusion

The project titled "Swayambhunath 3D Rendering" can indeed be considered a major first milestone in terms of our development into the retrospect of Computer Graphics, 3D Rendering and Graphics Pipeline.Starting with modelling an approximate representation of Swayambhunath temple, we parsed the obj files into vertex data. The vertex data had many properties including normals and material datas. Reading the necessary data from the file. We passed the data to various transformation functions. The vertex data was also processed by the camera and the perspective projection before finally passing it to our rasterizing algorithms. We used different shaders like flat, gouraud and phong to be able to substantially differentiate the visible outputs in the project. The camera controls provided us with the access to dynamically change our view planes using the point at and look at matrices.

Hence, the project did indeed cover almost all objectives that were set before the initiation of the project. However, due to time constraints, learning curve and skill barrier. Some aspects like accurate representation of the model, texture mapping, and a more dynamic camera control has been given space in the near future.

Putting on the final remarks, this project helped us understand 3D rendering basics and also tried to cover the approximate representation of the Swayambhunath temple.

Limitations and Future Works

The biggest limiting factor of our project is time and skill constraints. We learned the basics of graphics few weeks prior to starting the project.The main limitations are:

a. Time and skill constraints
b. Lack of modeling experience
c. Single threaded and non-optimized code
d. Only a handful of features available

Although the renderer does fine as a starter rendering project, there are definitely some improvements for the future. Possible future plans could include:

a. Texture mapping
b. Illumination
c. Shadow mapping
d. Transparency
e. Reflection and Refraction

References

1. Wikipedia-https://en.wikipedia.org/
2. Blender-https://github.com/blender
3. LearnOpenGl-https://learnopengl.com/
4. TinyRenderer-https://github.com/ssloy/tinyrenderer
5. Scratchpixel-https://www.scratchapixel.com/
6. OLC series-(https://www.youtube.com/watch?v=ih20l3pJoeU)
7. Donald D. Hearn and M. Pauline Baker, “Computer Graphics C version”