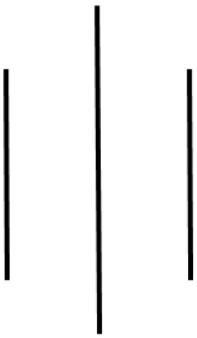
Tribhuvan University

Institute Of Engineering

Central Campus, Pulchowk



A Report On:

Graphics Project (Dharahara 3D Model)

Submitted By:

Anish Sah	**(PUL074BEX005)**
Pradipta Pradhanang	**(PUL074BEX022)**
Pratik Sharma	**(PUL074BEX029)**
Sukirti Bhattarai	**(PUL074BEX045)**

Submitted To: **Department of Electronics and Computer Engineering**

Falgun, 22 2076

Acknowledgement

This report on Graphics Project (Dharahara 3D Model) has been prepared for the partial fulfillment of the course requirement of Bachelors in Electronics and Communication Engineering. We would like to thank the Department of Electronics and Communications Engineering for including this project in our curriculum which helps us to be familiar with real world applications of the knowledge that we gain in our engineering course. We would like to express our sincere gratitude to our lecturer Dr. Basant Joshi for his support and guidance to conduct this study. His guidance and encouragement has played a significant role in the successful completion of this project. We also like to express our gratitude to the Department of Electronics and Computer Engineering for providing us the work and environment and resources.

We also express our deepest gratitude to all the people who helped us directly or indirectly during our working on this project without whom it wouldn't have been possible.

Sincerely,

Anish Sah

Pradipta Pradhanang

Pratik Sharma

Sukirti Bhattarai

Abstract

In this graphics project, we made the 3D model of Dharahara which is currently being constructed at Sundhara, Kathmandu with brief details about the surroundings. This 3D model of Dharahara consists of newly build 10 storied Dharahara along with the old demolished Dharahara which was destroyed by the earthquake and also wooden fence surrounding the premise of the Dharahara. We used various softwares like Blender 2.8 to build the object and Python IDLE 3.8 to program the object build by the Blender using python programming language to add features namely: smoothing the object, perspective viewing, camera controlled by the mouse and keyboard, lighting and shadowing effect.

Table of Contents

1. INTRODUCTION

Dharara, one of the greatest pride of Nepalese people. A symbol of a great history showcasing the bravery of the great. Our project is to make a 3d model of Dharara, ten-storey, tower at the center of Sundhara in Kathmandu. This model will focus on the newly proposed model of Dharara by the government. We want to showcase the beauty and elegance of Dharara.So, we have tried to make our project as unique and raw as possible. Most of the tower collapsed in the 25 April 2015 Nepal earthquake, but the base remains. Reconstruction of the tower commenced in June 2018. We have tried to make our project very realistic and it also include both the remains and the new darahara.

Here Blender was used to model the object, opengl was use to generate all the effects in the model, and python was used the progeamming language.

1.1. Objective

The main objective of our project is to

- To make a 3D model of Dharara.
- To simulate the real-life experience in Dharara.
- To contribute to the ongoing movement (Visit Nepal 2020).
- To also simulate the new features in Dharara proposed by the government.
- To be familiar with 3D mode making using blender.
- To learn about opengl for rendering 2d and 3d object.
- To be familiar with python programming.

The project will help in realizing the estatic and experience of visiting the Dharara.

1.2. Requirements

Software requirement:

Operating system like Windows 10 is required to make 2d and 3D model.

A built in graphics library like gult, gult32, glfw etc are required to create graphics layout.

Pycharm and its associated package.

Hardware requirement:

The used software can run in most of the device so the hardware requirement is minimum.

Processor: 32-bit dual core 2 GHz CPU with SSE2 support

Processor Speed: 500mhz and above

RAM: 64MB or above storage capacity of 4GB

Monitor used: 1280×768 pixels, 24-bit color

2. BACKGROUND THEORY

2.1. Visible surface detection:

When a picture that contains the non-transparent objects and surfaces are viewed, the objects that are behind the objects that are closer cannot be viewed. To obtain a realistic screen image, these hidden surfaces need to be removed. This process of identification and removal of these surfaces is known as Hidden-surface problem.

2.2. Depth buffer

It is an image-space approach developed by Cutmull. The Z-depth of each surface is tested for determining the closest surface.

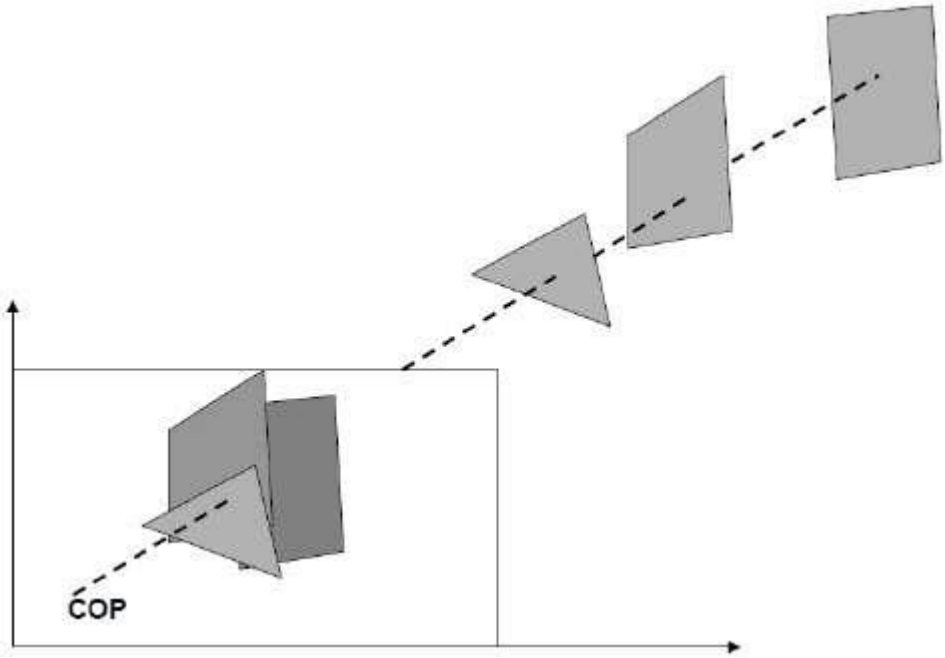
Across the surface, one pixel position is processed at a time. The colour that is to be displayed on the frame buffer is determined by the closest (smallest z) surface, by comparing the depth values for a pixel.

The closer polygons are override by using two buffers namely frame buffer and depth buffer.

Depth buffer is used to store depth values for (x, y) position, as surfaces are processed ($0 \leq \text{depth} \leq 1$).

The frame buffer is used to store the intensity value of color value at each position (x, y).

The z-coordinates are usually normalized to the range [0, 1]. The 0 value for z-coordinate indicates back clipping pane and 1 value for z-coordinates indicates front clipping pane.



Advantages:

Implementation is easy.

The problems related to speed is reduced once it is implemented in the hardware.

It processes one object at a time.

Disadvantages:

Large memory is required.

It is a time consuming process.

The Edge Table − It contains coordinate endpoints of each line in the scene, the inverse slope of each line, and pointers into the polygon table to connect edges to surfaces.

The Polygon Table − It contains the plane coefficients, surface material properties, other surface data, and may be pointers to the edge table.

2.3. Shading:

2.3.1. Fragment Shader:

A Fragment Shader is the Shader stage that will process a Fragment generated by the Rasterization into a set of colors and a single depth value.

The fragment shader is the OpenGL pipeline stage after a primitive is rasterized. For each sample of the pixels covered by a primitive, a "fragment" is generated. Each fragment has a Window Space position, a few other values, and it contains all of the interpolated per-vertex output values from the last Vertex Processing stage.

Fragment shaders also have access to the discard command. When executed, this command causes the fragment's output values to be discarded. Thus, the fragment does not proceed on to the next pipeline stages, and any fragment shader outputs are lost. The discard command will also prevent any image store and atomic operations and Shader Storage Buffer Object writes (issued before the discard) from working.

The inputs to the fragment shader are either generated by the system or passed from prior fixed-function operations and potentially interpolated across the surface of the primitive.

The user-defined inputs received by this fragment shader will be interpolated according to the interpolation qualifiers declared on the input variables declared by this fragment shader. The fragment shader's input variables must be declared in accord with the interface matching rules between shader stages. Specifically, between this stage and the last Vertex Processing shader stage in the program or pipeline object.

2.3.2. Phong Shading (per fragment)

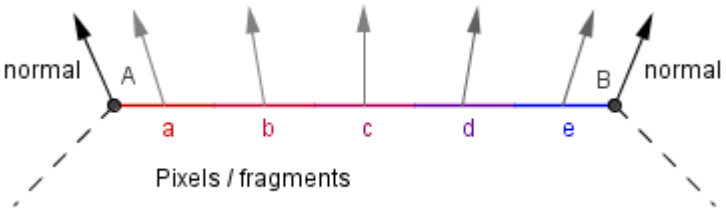
A more accurate method for rendering a polygon surface is to interpolate the normal vector and then apply the illumination model to each surface point. This method developed by Phong Bui Tuong is called Phong Shading or normal vector

Interpolation Shading. It displays more realistic highlights on a surface and greatly reduces the Match-band effect.

A polygon surface is rendered using Phong shading by carrying out the following steps:

1. Determine the average unit normal vector at each polygon vertex.
2. Linearly & interpolate the vertex normals over the surface of the polygon.
3. Apply an illumination model along each scan line to calculate projected pixel intensities for the surface points.

This was another improvement in order to account for the specular reflection. Main idea is that the normal from the vertices is interpolated. Color is calculated per fragment, taking into account the interpolated normal. For an approximation of a sphere, this is quite ideal, because the interpolated normals would be exactly those that a perfect sphere would have. Because the color is calculated based on the normals, it will be calculated as if it were a perfect sphere.



2.3.3. Shading in blender

Wireframe:- Objects appear as a mesh of lines representing the edges of faces and surfaces.

Solid:- The default drawing mode using solid colored surfaces and simple lighting.

Textured:- Shows meshes with an image applied using the mesh's active UV map. For Cycles materials, the image is the last one selected in the Node Editor. For other render engines, the UV map's applied face texture will be shown.

Material:- A fast approximation of the applied material.

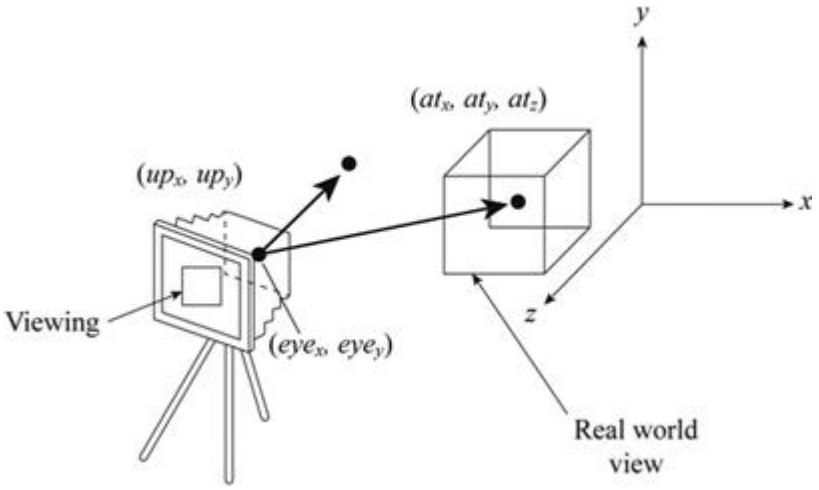
Rendered:- An accurate representation using the selected *Render Engine* and lit with the visible scene lights.

2.4. Texture Mapping in blender

In computer graphics, the application of a type of surface to a 3D image. A texturecan be uniform, such as a brick wall, or irregular, such as wood grain or marble. The common method is to create a 2D bitmapped image of the texture, called a "texture map," which is then "wrapped around" the 3D object. An alternate method is to compute the texture entirely via mathematics instead of bitmaps. The latter method is not widely used, but can create more precise textures especially if there is great depth to the objects being textured.

2.5. Window-to-viewport Mapping

View port is a rectangular section of the interface which defines the image appearance location.



The window-to-viewport mapping is a process of transforming or mapping the two dimensional or world coordinate view into device coordinate. The object which is available inside of the clipping window or world is mapped into the viewport and is displayed on the interface window screen, or the clipping window selects the piece of the scene from the display and view port positions it in the output device.
The formula for window-to-viewport mapping is

$$t_x = \frac{XW_{max} XV_{min} - XW_{min} XV_{max}}{XW_{max} - XW_{min}}$$

$$t_y = \frac{YW_{max} YV_{min} - YW_{min} YV_{max}}{YW_{max} - YW_{min}}$$

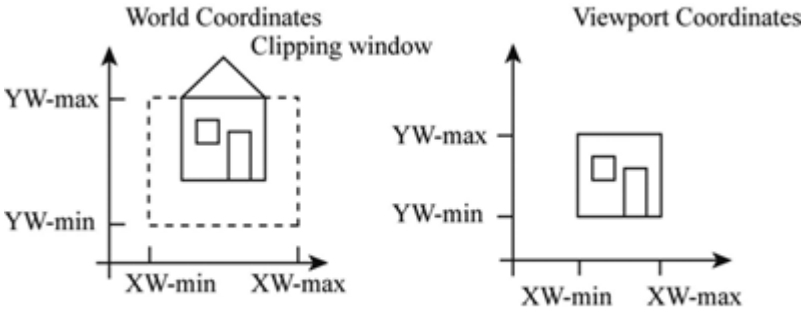


Figure to show the window-to-viewport mapping

2.6. Functions used in opengl

2.6.1. Built in functions used

glClearColor(float red, float green, float blue, float alpha) - set the clear color to (red, green, blue) with value alpha.

glClear(GLbitfield mask) - clear the buffer indicated by mask using the current clear color. Values for mask are:

GL_COLOR_BUFFER_BIT - color buffer

GL_DEPTH_BUFFER_BIT - depth buffer

GL_ACCUM_BUFFER_BIT - accumulation buffer

GL_STENCIL_BUFFER_BIT - stencil buffer

glColor3f(float red, float green, float blue) - set the current drawing color to (red, green, blue).

gluLookat(GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble aimx, GLdouble aimy, GLdouble aimz, GLdouble upx, GLdouble upy, GLdouble upz) - defines the viewing matrix, setting the current eye position (eyex, eyey, eyez), the aim point along the desired line of sight (aimx, aimy, aimz),

and the desired "up" direction (upx, upy, upz). Default values are (0,0,0), (0,0,-1), and (0,1,0).

glScalef(float x, float y, float z) - scale by (x, y, z). Also: glScaled( ).

glTranslatef(float x, float y, float z) - translate by (x, y, z). Also: glTranslated( ).

glRotatef(float angle, float x, float y, float z) - rotate counterclockwise by angle degrees around the line through the origin with direction vector (x, y, z)

glMatrixMode(GLenum mode) - sets the matrix to be worked on. Values for mode are:

GL_MODELVIEW - use the modelview matrix

GL_PROJECTION - use the projection matrix

GL_TEXTURE - use the texture matrix

glLoadIdentity(void) - set the current matrix to the identity

glfw_key - A keyboard key identifier, which can be either an uppercase printable ISO 8859-1 (Latin 1) character (e.g. 'A', '3' or '.'), or a special key identifier.

glUniformMatrixf4 (GLint location GLsizei count GLboolean transpose const GLfloat *value)

Glfw_swap_buffer (**GLFWwindow** * *window*)

This function swaps the front and back buffers of the specified window. If the swap interval is greater than zero, the GPU driver waits the specified number of screen updates before swapping the buffers.

2.6.2. User defined function used

ObjLoader.load_mode(file,status) – this is used to load the obj file created from blender.

Load_texture(file,texture) – this is used to load the texture to each object. Here the file is in png formate.

Search_data(data_values,coordinate,skip,data_type)

Create_stored_vertex_buffer(indices_data,vertices,texture,normals)

Compile__shader(vs,fs)

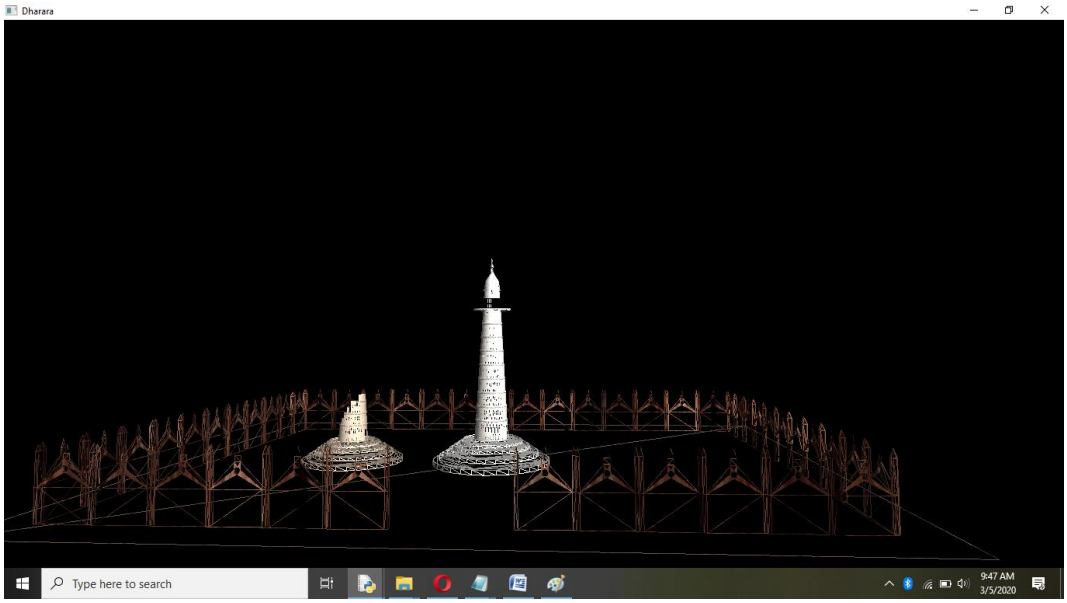
3. METHODOLOGY

For our project we first went through the basic tutorial for opengl and blender since both were new to us. After learing about it we then modeled the darahara amd its environment as per the convinience in blender. Then the loading of the model in opengl using python as the programming language was done. Uv mapping for texture was generated from blender and loaded in opengl. The all the necessary effects that was requied in our project was done through coding in opengl.

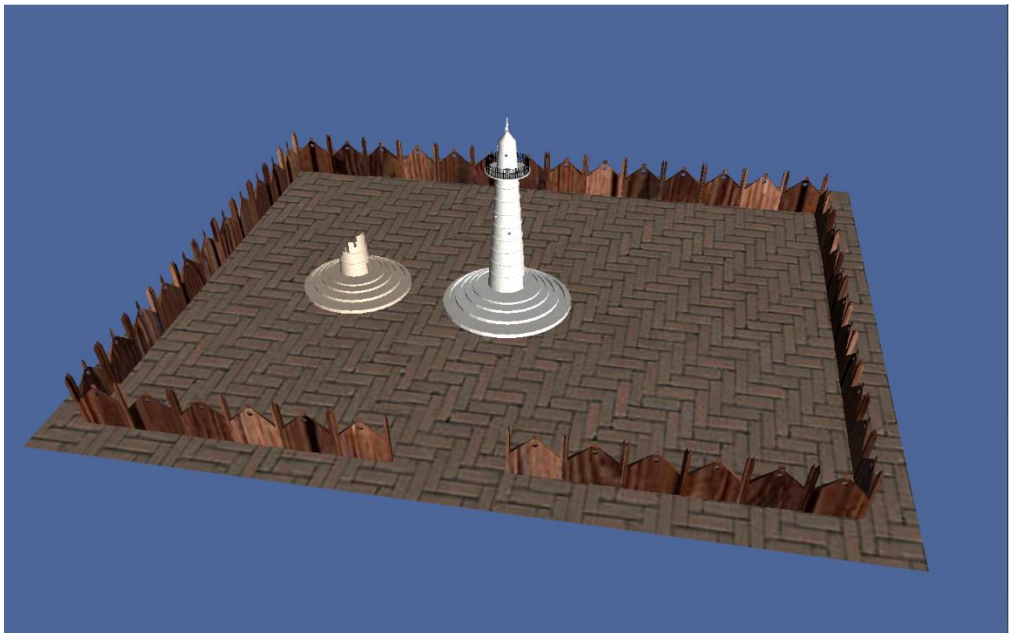
For loading

- Initialization of GLFW library
- Window object is created
- .obj file is used to store the vertices,texture coordinates,normals in a list
- A VAO is created and binded
- A VBO is created and binded
- Attriburepointer were allocated for each vertices,texture coordinates,normals
- .png file is read for the texture.
- Shader file is used for required position calculation which are then stored in related matrices with required transformation
- We use a main window loop which runs until the application closes
- input is checked if any from keyboard or mouse for the required change in position.
- Object is drawn to the window with required shading
- The window is displayed.

3.1. Initial object



3.2. Final Object



3.3. Project Details:

We have used different programs and algorithms to achieve the required objective. As we know, 3D modeling is a technique in computer graphics for producing a 3D digital representation of any object or surface. An artist uses special software to manipulate points in virtual space (called vertices) to form a mesh: a collection of vertices that form an object. 3D models are used for a variety of mediums including video games, movies, architecture, illustration, engineering, and commercial advertising. We will be using the following:

1)Opengl: Open graphics library (OpenGL) is a cross language, cross-platform application
programming interface (API) for rendering 2D and 3D vectors graphics. It provides a common
set of commands that can be used to manage graphics in different applications and on multiple
platforms. OpenGL commands include drawing polygons, assigning colors to shapes, applying
textures to polygons (texture mapping), zooming in and out, transforming polygons, and rotating
objects. OpenGL is also used for managing lighting effects, such as light sources, shading, and
shadows. It can also create effects like haze or fog, which can be applied to a single object or an
entire scene.
Here we have used python as our programming language so some of the library we have used pf our puthon programming.

The following are the library used in our program:-

- GLFW – A cross-platform windowing and keyboard-mouse-joystick handler; is more game-oriented.

- a user interface library. It allows the programmer to create top-level windows with OpenGL contexts. No menus, no buttons.
- a Windows-only library. Requests for features that cannot be portably implemented will be denied unless they are unobtrusive, like the Windows port looking for a GLFW_ICON resource at window creation.
- a threading library. There are already good cross-platform threading libraries and threading has been added to both the C11 and C++11 standard libraries.
- an image loading library. There are already good cross-platform image loading libraries.
- capable of rendering text. There are already several libraries that render text with OpenGL and consistent cross-platform text rendering cannot depend on the platform's text rendering facilities anyway.
- capable of rendering anything at all. Rendering is up to the programmer and/or other libraries.

- Pyrr:- Pyrr is another library function used in this program. Since we are using python as our programming language so this library is required. Pyrr is a Python mathematical library. It id used to provide function for manipulating and creating 3D vectors.
- Pillow:- Pillow is the friendly PIL fork by Alex Clark and Contributors. PIL is the Python Imaging Library by Fredrik Lundh and Contributors. It was based on the PIL code, and then evolved to a better, modern and more friendly version of PIL. It adds support for opening, manipulating, and saving many different image file formats. A lot of things work the same way as the original PIL.
- Numpy:- NumPy is the fundamental package for scientific computing with Python. It contains among other things:

  - a powerful N-dimensional array object
  - sophisticated (broadcasting) functions

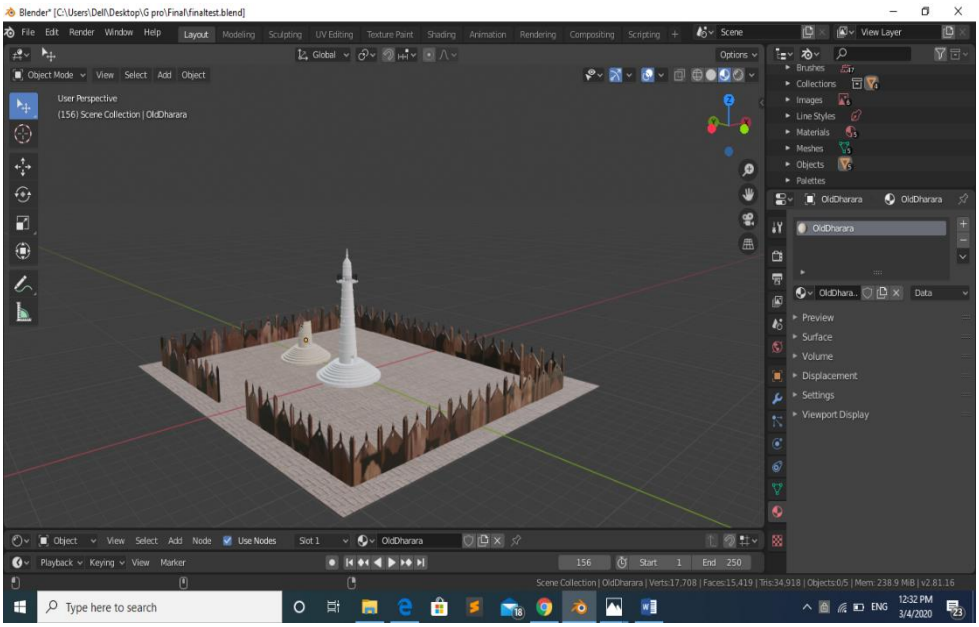
  A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the *rank* of the

array; the *shape* of an array is a tuple of integers giving the size of the array along each dimension.

2)Blender: Blender is the free and open source 3D creation suite. Blender is a free and open-source 3D computer graphics software toolset used for creating animated films, visual effects, art, 3D printed models, motion graphics, interactive 3D applications, and computer games. Blender's features include 3D modeling, UV unwrapping, texturing, raster graphics editing, rigging and skinning, fluid and smoke simulation, particle simulation, soft body simulation, sculpting, animating, match moving, rendering, motion graphics, video editing, and compositing.

Among the above mentioned features of blender the features that we used while modeling this darahara model are 3D modeling, UV unwrapping, texturing.In blender we did the following process:

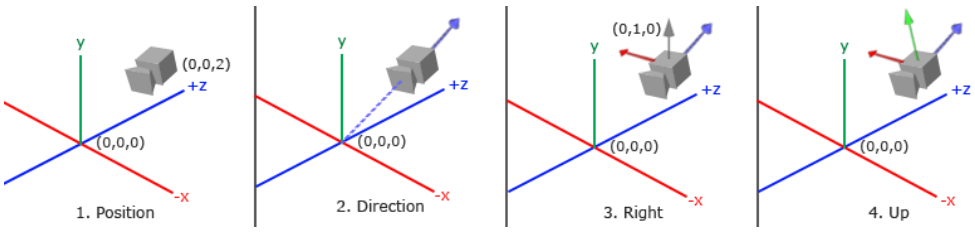
- At first a simple model was modled then the required texture was added for each of the items.
- Then, the obj and mtl file was taken from blender to load in the program.
- For texturing, we generated a png file from uv mapping and loaded the texture in the model. And finally the mode was obtained.



Hence, blender was one of the best place for the generation requied 3D model.

3.4. Camera

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. 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. A careful reader might notice that we're actually going to create a coordinate system with 3 perpendicular unit axes with the camera's position as the origin.



1. Camera position

Getting the camera position is easy. The camera position is a vector in world space that points to the camera's position. We set the camera at the same position we've set the camera in the previous tutorial:

```
glm::vec3 cameraPos = glm::vec3(0.0f, 0.0f, 3.0f);
```

.

2. Camera direction

The next vector required is the camera's direction e.g. at what direction it is pointing at. For now we let the camera point to the origin of our scene: (0,0,0). Remember that if we subtract two vectors from each other we get a vector that's the difference of these two vectors? Subtracting the camera position vector from the scene's origin vector thus results in the direction vector we want. For the view matrix's coordinate system we want its z-axis to be positive and because by convention (in OpenGL)

the camera points towards the negative z-axis we want to negate the direction vector. If we switch the subtraction order around we now get a vector pointing towards the camera's positive z-axis:

```
glm::vec3 cameraTarget = glm::vec3(0.0f, 0.0f, 0.0f);
```

```
glm::vec3 cameraDirection = glm::normalize(cameraPos -
cameraTarget);
```

3. Right axis

The next vector that we need is a *right* vector that represents the positive x-axis of the camera space. To get the *right* vector we use a little trick by first specifying an *up* vector that points upwards (in world space). Then we do a cross product on the up vector and the direction vector from step 2. Since the result of a cross product is a vector perpendicular to both vectors, we will get a vector that points in the positive x-axis's direction (if we would switch the cross product order we'd get a vector that points in the negative x-axis):

```
glm::vec3 up = glm::vec3(0.0f, 1.0f, 0.0f);
```

```
glm::vec3 cameraRight = glm::normalize(glm::cross(up,
cameraDirection));
```

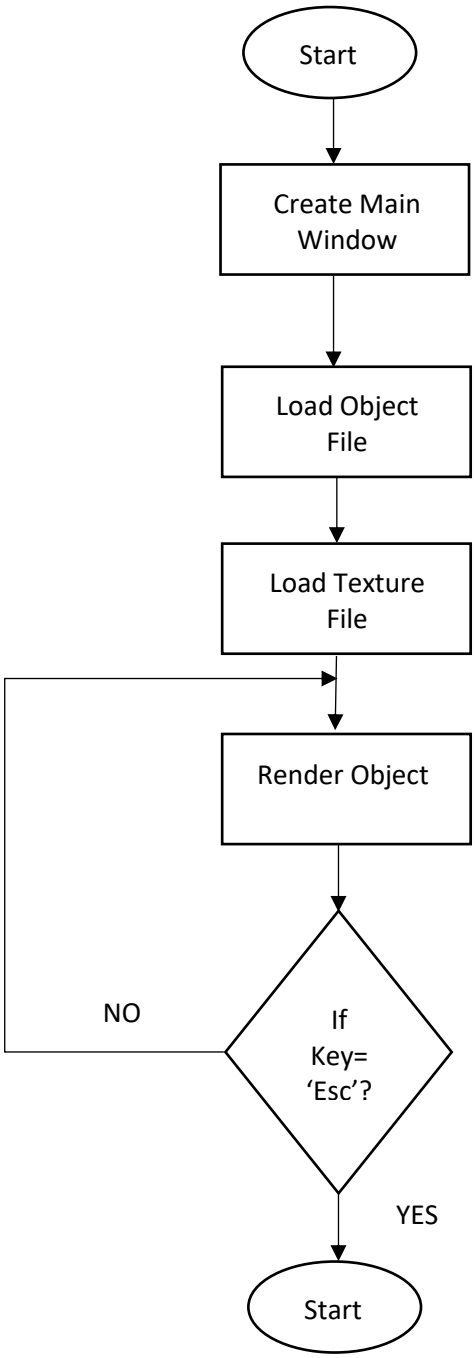
4. Up axis

Now that we have both the x-axis vector and the z-axis vector, retrieving the vector that points to the camera's positive y-axis is relatively easy: we take the cross product of the right and direction vector:

```
glm::vec3 cameraUp = glm::cross(cameraDirection,
cameraRight);
```

With the help of the cross product and a few tricks we were able to create all the vectors that form the view/camera space. For the more mathematically inclined readers, this process is known as the Gram-Schmidt process in linear algebra. Using these camera vectors we can now create a LookAt matrix that proves very useful for creating a camera.

3.5. Flowchart



4. PROJECT SCHEDULE AND WORK DIVISION

The work was successfully divided among our four group members. We were concentrated to finish our work on time. We shared our individual work among the group members. Like this way, we were able to check our project progress and could complete the project as per our plan. Hence, we could all share our knowledge and prepare our group project.

For the distribution of our project we divide the project work into two group among which one part was completed by two members of the group while the other part was completed by the other two. Here, the first part includes modeling the object in blender, texture and uv mapping and loading the object in opengl whereas the second part include camera positioning, shading and shadow in opengl. Both the part was constantly shared in among our group members so that the all the part of the project was familiar between all the members of the group.

5. DISCUSSION:

The idea for our project eas raw and unique so to find te reference for our project was difficult to find. Our project can be implemented in real life to create some 3D model.

We used blender to create the model as it is easy in blender for rendering, camera and object motion tracking, modeling, simulation, visual effects. Blender users enjoy a sped-up rendering process since rather than using the CPU, Cycles users a graphics card to perform rendering. Because this lets modern GPUs to do a lot of crunching instead, the rendering process is considerably quicker. Also, it also has some disadvantage as the "bad" of Blender is that because of the 100% customizable interface, a lot of "casual" users cannot get the hang of it. The users, who starts learning Blender, finds out Blender has a steep learning curve because you have to relearn and get used to a whole new interface. It brings us right back to the usability and workflow of the users.

Opengl was considered for loading the object as it has been stable for more than seven years across a wide variety of platforms. Additions to the specification are well controlled, and proposed updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure that existing applications do not become obsolete. OpenGL applications offer reliability and provide consistent visual display results on any OpenGL API-compliant hardware, regardless of operating system or windowing system. Supported platforms and operating system include:(All Unix workstations, BeOS, Linux, Mac OS, OPENStep, OS/2,Python, Window 95/98/NT/2000). Supported windowing systems include:(Presentation Manager, Win32, X/Window System). 5- OpenGL applications and services can be called from a variety of languages. OpenGL offers complete independence from network protocols and topologies. OpenGL offers scalability with applications that can execute on systems running the gamut from consumer electronics to PCs, workstations, and supercomputers. Consequently, applications can scale to any class of machine that the developer choose to target.

Python was considered for our project because of the Python Package Index (PyPI) contains numerous third-party modules that make Python capable of interacting with most of the other languages and platforms. It provides a large standard library. Python language is developed under an OSI-approved open source license, which makes it free to use and distribute, including for commercial purposes. Python offers excellent readability and uncluttered simple-to-learn syntax which helps beginners to utilize this programming language ad is user friendly structures.

5.1. Result

We tried to create the 3D model of daraharar very realistic. We have also tried to follow the new model of darahara and given all the texture the is different fro the old darahara. Also we have managed the camera angle and perspective projection to see the model by aplyong all the effects thhet is zoom in, zoom out, rotation translation, shading.

The motion is given by the following way:-

W for forward motion

S for backward motion

A for left motion

D for right motion

R for rotation

Also these all the motion can be controlled by mouse as well.

Hence our result has the lighting effect and shading effect and camera movement effect as well.

5.2. Conclusion:

In this project we were able to model a 3d object and then apply the algoritms that we have studied in our syllabus. By doing this project we were also able to know about the importance of team work. This project is suitable for modeling a 3D object and it uses graphics functationality. Out of many features, our project demonstrate the features of blender and opengl like rendering, translation, rotation, uv mapping, shading. Hence, here we tried to use most of the effects of our syllabus and create the realistic model as possible. Hence, all the work was sucessfully completed by all our group members. We tried to learn about opengl and blender through this project. Therefore, we tried to achieve all the possible objective.

6. REFERENCES

- https://www.raywenderlich.com/2604-how-to-export-blender-models-to-opengl-es-part-1-3
- https://stackoverflow.com/questions/3886611/blender-vs-unity
- https://kathmandupost.com/miscellaneous/2016/08/01/designs-of-new-dharahara-madepublic
- https://www.youtube.com/watch?v=nseD3RN9tkU
- wikipedia.com
- https://www.blender.org/
- https://learnopengl.com/Getting-started/Camera?fbclid=IwAR32kgikSdqx3hU0imX52SJ4dlI680nsb4OlmR6KQE-kd3wNc0Ta9uef1lQ