

**TRIBHUVAN UNIVERSITY**

**IOE**

**PULCHOWK CAMPUS**

A REPORT ON

**COMPUTER GRAPHICS**

**(3D HOUSE MODEL RASTERIZATION)**

**Submitted by:**

Pooja Sah(074/BEX/421)

Prava Basnet(074/BEX/430)

Rabindra Kumar Yadav(074/BEX/432)

Sabin Sharma Gaire(074/BEX/436)

**Submitted to:**

DEPARTMENT OF COMPUTER

AND ELECTRONICS ENGINEERING

Lalitpur, Nepal

5 mar, 2020

**Acknowledgement**

We would like to express our special thanks of gratitude to our respected sir, **Mr Dr.Basanta Joshi**, for giving us special opportunity to do project on “**3-D model of house**”. The project definitely enlighten our knowledge regarding graphics and programming language. As well as we were able to think practically. Doing this project would be productive and fruitful for all of us. Not only the project broadens knowledge it also encouraged us for team spirit. We are really grateful toward our teacher for giving us platform to present our project and also for encouraging us.

**Abstract**

This project visualizes House implementing 3D Modeling algorithms and methods. The project shows the 3D model of house using various techniques and algorithms along with their efficient implementation. The algorithms are implemented using C++. GLFW is used for window creation, context rendering and pixel plotting. We used Sketchup to design the model and exported it in .obj file format. We created a class using Assimp that reads and stores the vertices, edges, surfaces, normal as objects of respective classes. Various 2D and 3D transformations are done to realize 3D house in 2D plane using perspective projection. Z buffer algorithm is used to determine the visible surface.

# 

# **Table of Contents**

[**Acknowledgement**](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.woy47zirynfz)[**1**](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.woy47zirynfz)

[**Abstract**](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.p23glwtqwusj) **2**

[**Introduction**](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.3kqi68fe9vi) **4**

**Objective 5**

**Theory** [**6**](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.ali2ci8ir178)

**Literature Review 9**

[**Methodology**](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.z659918b1mkw) **10**

[OpenGL](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.gofjgif0b0rf) 10

Shader 11

Texture Wrapping [1](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.nuucerxbqwu3)2

Scaling [12](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.b4cmmlpb4o4g)

Translation [13](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.fmhiajxf6opz)

Rotation [14](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.u1gt9enmkyy7)

[Perspective Projection](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.s53d9t4646s0) [15](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.s53d9t4646s0)

[Depth Buffer](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.uao14le4irnw) [1](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.uao14le4irnw)6

[Shadow Mapping](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.w1vkfxkr61fz) [1](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.w1vkfxkr61fz)7

[Phong Reflection Model](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.su2zjanv8ly6) [18](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.su2zjanv8ly6)

3D Modeling [1](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.sm9v8mtwqgbp)8

[Model Rendering](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.55xh5w87pg1a) 19

**Implementations 20**

[**Output Screenshots**](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.tfml0cbqmhc1)[**21**](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.tfml0cbqmhc1)

**Problems Faced 22**

[**Limitations**](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.sodt8u20uaeu) **and Future Enhancements 22**

[**Conclusions and Recommendations**](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.caeis37kmd26)[**2**](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.caeis37kmd26)**2**

[**References**](https://docs.google.com/document/d/1Bh9KcTH6_hDcrnfTlHtiHnl_rn74qI8oz2OpLHsWZ3g/edit#heading=h.fgorb0xan2k9) **23**

**Introduction**

Computer graphics have become a powerful tool for the rapid and economical production of pictures. The use of computer graphics is so widespread that there is virtually no area in which graphical displays cannot be used. Its major applications are:

1. **Computer aided design:** a major use of computer graphics is in the design process particularly in engineering and architectural systems but now almost all products are computer designed.
2. **Computer art:** Computer graphics are widely used in both fine and commercial art applications.
3. **Presentation graphics:** Computer graphics are also used to produce illustrations for reports or generate slides or transparencies for projectors.
4. **Visualizations:** Computer graphics are used to analyze large amounts of information to study the behavior of certain processes.
5. **Education and training:** Computer generated models of physics, finance and economics are often used as educational aids.
6. **Image processing:** Computer graphics are used on image processing to modify or interpret existing pictures.
7. **Graphical user interfaces**

This project is hence, an application of many of the graphics algorithms and processes we studied in the syllabus. The project follows the graphics rendering pipeline to renderer 3D view on the screen.

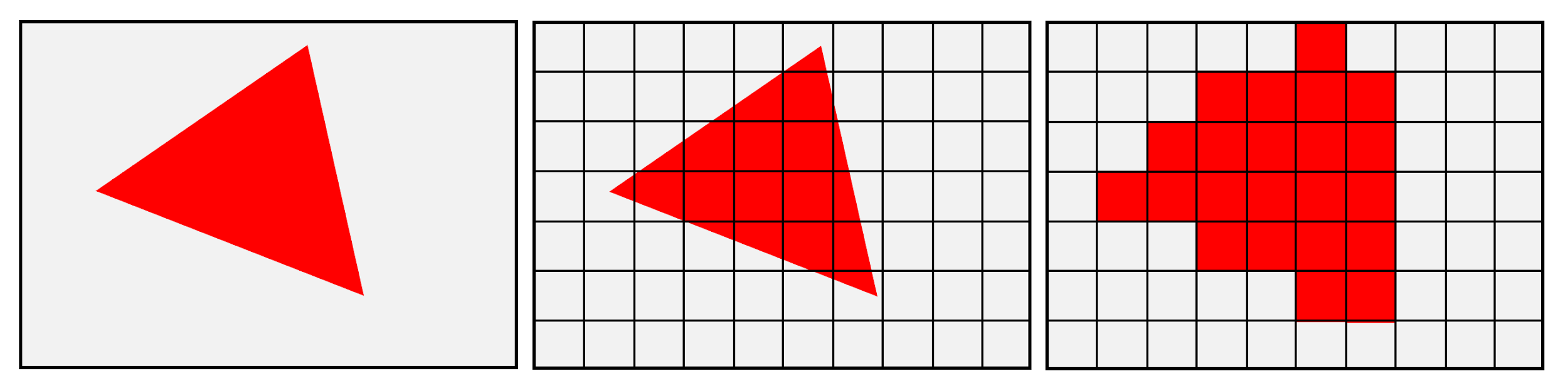
**Objectives**

The prime objectives of this project can be listed as follows:

1. To learn and be familiar with the concepts of Computer Graphics
2. To get familiar with 2D and 3D rendering pipeline
3. To build an interactive way of modelling any 3D object
4. To learn practical implementation of mathematical equations in digital world
5. To learn about the programs that resides on the GPU i.e. Shaders
6. To perform different transformations like rotation about coordinate axes, translation, scaling, etc. on the object.
7. To color and create lighting effects for the objects.
8. To learn the essence and ethics of working as a team

**Theory**

## **Rasterization**

*Figure: Steps involving rasterization of a primitive (triangle) into viewport (grid of pixels)*

According to OpenGL specifications, “Rasterization is the process by which a primitive is converted to a two-dimensional image. Each point of this image contains such information as color and depth. Thus, rasterizing a primitive consists of two parts. The first is to determine which squares of an integer grid in window coordinates are occupied by the primitive. The second is assigning a color and a depth value to each such square.”

Rasterization is the task of taking an image described in a vector graphics format (shapes) and converting it into a raster image (a series of pixels, dots or lines, which, when displayed together, create the image which was represented via shapes). The rasterized image may then be displayed on a computer display, video display or printer, or stored in a bitmap file format. Rasterization may refer to either the conversion of models into raster files, or the conversion of 2D rendering primitives such as polygons or line segments into a rasterized format.

Rasterization is the process by which most modern display systems turn electronic data or signals into projected images, such as video or still graphics. This is typically a process of identifying the needs of a specific media configuration, then allocating resources so that images are efficiently and optimally projected on the display device.

Rasterized graphics are often compared with image vectors. While rasterization is typically a process of compiling scan lines or pixels on a bitmap, in contrast, vectors incorporate mathematical functions in order to create images based on geometric shapes, angles and curves.

### 

### **Rasterization of 3D images**

Compared with other rendering techniques such as ray tracing, rasterization is extremely fast. However, rasterization is simply the process of computing the mapping from scene geometry to pixels and does not prescribe a particular way to compute the color of those pixels. Shading, including programmable shading, may be based on physical light transport, or artistic intent.

The process of rasterizing 3D models onto a 2D plane for display on a computer screen ("screen space") is often carried out by fixed function hardware within the graphics pipeline. This is because there is no motivation for modifying the techniques for rasterization used at render time and a special-purpose system allows for high efficiency.

#### 

#### 

#### **Triangle rasterization**

A common representation of digital 3D models is polygonal. Before rasterization, individual polygons are broken down into triangles, therefore a typical problem to solve in 3D rasterization is rasterization of a triangle. Properties that are usually required from triangle rasterization algorithms are that rasterizing two adjacent triangles (i.e. those that share an edge)

1. leaves no holes (non-rasterized pixels) between the triangles, so that the rasterized area is completely filled (just as the surface of adjacent triangles). And
2. no pixel is rasterized more than once, i.e. the rasterized triangles don't overlap. This is to guarantee that the result doesn't depend on the order in which the triangles are rasterized. Overdrawing pixels can also mean wasting computing power on pixels that would be overwritten.

This leads to establishing rasterization rules to guarantee the above conditions. One set of such rules is called a top-left rule, which states that a pixel is rasterized if and only if

1. its center lies completely inside the triangle. Or
2. its center lies exactly on the triangle edge (or multiple edges in case of corners) that is (or, in the case of corners, all are) either top or left edge.

A top edge is an edge that is exactly horizontal and lies above other edges, and a left edge is a non-horizontal edge that is on the left side of the triangle.

**Literature Review**

By surveying the web and going through different articles, tutorials and documentation, we acquired enough knowledge of rendering pipeline and OpenGL functions, then finally used it for our project development. Tutorials and explanations given on the website learnopengl.com were extremely helpful for our project which provides the best reference and workflow for our project. We referred to the Youtube Channel “The Cherno Project '' which provided us great insights about the OpenGL library.

As per the advice of seniors and experts of Computer Graphics “The Red Book” was also recommended but as a beginner we found it difficult to understand and take reference from “Computer Graphics with OpenGL” by F.S. HILLS.Hence all these references contribute to the completion of our project.

**Methodology**

**OpenGL**

OpenGL is mainly considered an API that provides us with a large set of functions that we can use to manipulate graphics and images. However, OpenGL by itself is not an API, but merely a specification, developed and maintained by the Khronos Group.

OpenGL is by itself a large state machine: a collection of variables that define how OpenGL should currently operate. The state of OpenGL is commonly referred to as the OpenGL context. When using OpenGL, we often change its state by setting some options, manipulating some buffers and then render using the current context.

Whenever we tell OpenGL that we now want to draw lines instead of triangles for example, we change the state of OpenGL by changing some context variable that sets how OpenGL should draw. As soon as we change the context by telling OpenGL it should draw lines, the next drawing commands will now draw lines instead of triangles.

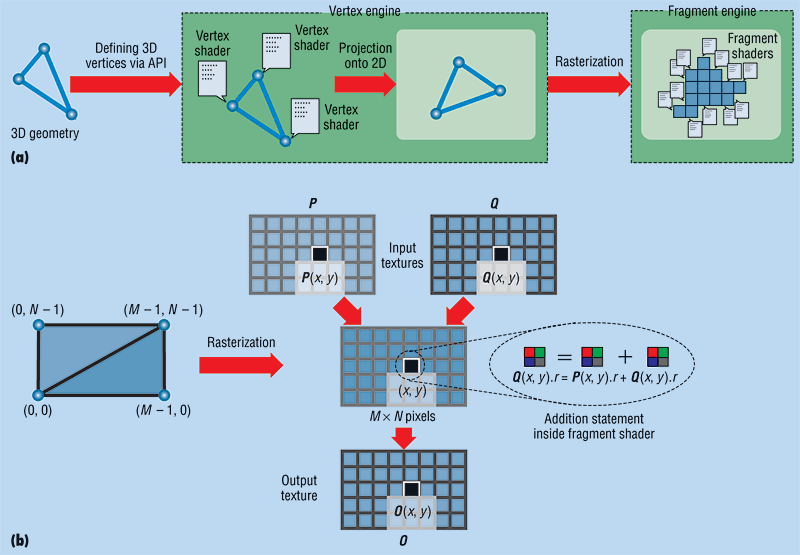


Fig.3D Rendering Pipeline

**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. Shaders are also very isolated programs in that they're not allowed to communicate with each other; the only communication they have is via their inputs and outputs.Shaders in our project are:

**Vertex Shader**

The first part of the pipeline is the vertex shader that takes as input a single vertex. The main purpose of the vertex shader is to transform 3D coordinates into different 3D coordinates (more on that later) and the vertex shader allows us to do some basic processing on the vertex attributes.

**Fragment Shader**

A fragment in OpenGL is all the data required for OpenGL to render a single pixel.

The main purpose of the fragment shader is to calculate the final color of a pixel and this is usually the stage where all the advanced OpenGL effects occur. Usually the fragment shader contains data about the 3D scene that it can use to calculate the final pixel color (like lights, shadows, color of the light and so on).

**Texture Wrapping**

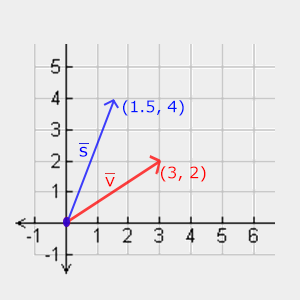
## Texture coordinates usually range from (0,0) to (1,1) but what happens if we specify coordinates outside this range? The default behavior of OpenGL is to repeat the texture images (we basically ignore the integer part of the floating point texture coordinate), but there are more options OpenGL offers:

1. **GL\_REPEAT**: The default behavior for textures. Repeats the texture image.
2. **GL\_MIRRORED\_REPEAT**: Same as **GL\_REPEAT** but mirrors the image with each repeat.
3. **GL\_CLAMP\_TO\_EDGE**: Clamps the coordinates between 0 and 1. The result is that higher coordinates become clamped to the edge, resulting in a stretched edge pattern.
4. **GL\_CLAMP\_TO\_BORDER**: Coordinates outside the range are now given a user-specified border color.

**Scaling**

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).

Let's try scaling the vector v=(3,2). We will scale the vector along the x-axis by 0.5, thus making it twice as narrow; and we'll scale the vector by 2 along the y-axis, making it twice as high. Let's see what it looks like if we scale the vector by (0.5,2) as s:

Fig: Scaling

**Translation**

Translation is the process of adding another vector on top of the original vector to return a new vector with a different position, thus moving the vector based on a translation vector..

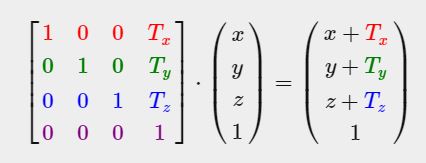
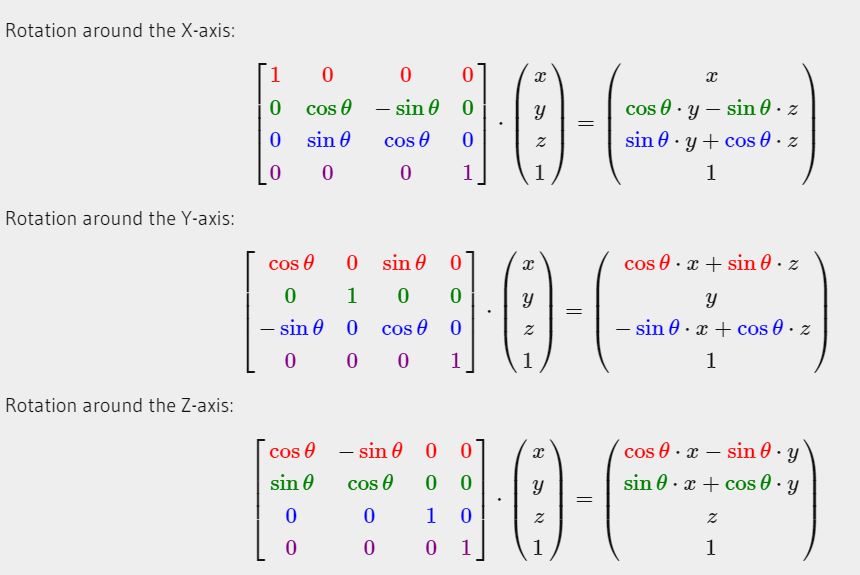
Just like the scaling matrix there are several locations on a 4-by-4 matrix that we can use to perform certain operations and for translation those are the top-3 values of the 4th column. If we represent the translation vector as (Tx,Ty,Tz) we can define the translation matrix by:

Fig: Translation

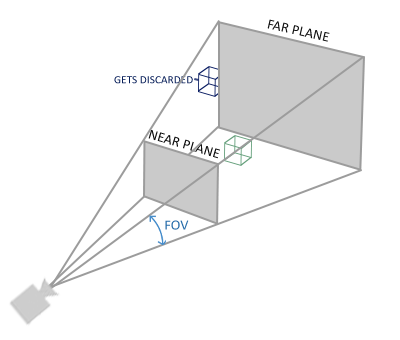
**Rotation**

A rotation is a transformation that turns a figure about a fixed point called the center of rotation. An object and its rotation are the same shape and size, but the figures may be turned in different directions. Rotations may be clockwise or counterclockwise.fig:Rotation Matrix

**Perspective Projection**

Perspective projection is a linear projection where three dimensional objects are projected on a picture plane. This has the effect that distant objects appear smaller than nearer objects.

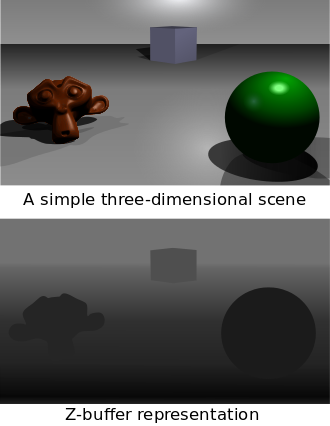
It also means that lines which are parallel in nature (that is, meet at the point at infinity) appear to intersect in the projected image, for example if railways are pictured with perspective projection, they appear to converge towards a single point, called the vanishing point. Photographic lenses and the human eye work in the same way, therefore perspective projection looks most realistic.Perspective projection is usually categorized into one-point, two-point and three-point perspective, depending on the orientation of the projection plane towards the axes of the depicted object.

Fig: Perspective Frustum

**Depth Buffer**

Depth buffering, is the management of image depth coordinates in 3D graphics, usually done in hardware, sometimes in software. It is one solution to the visibility problem, which is the problem of deciding which elements of a rendered scene are visible, and which are hidden.

In a 3d-rendering engine, when an object is projected on the screen, the depth (z-value) of a generated pixel in the projected screen image is stored in a buffer (the z-buffer or depth buffer). A z-value is the measure of the perpendicular distance from a pixel on the projection plane to its corresponding 3d-coordinate on a polygon in world-space.

Fig: Depth Buffer Representation

**Shadow Mapping**

Shadow mapping or shadowing projection is a process by which shadows are added to 3D computer graphics .

The first step renders the scene from the light's point of view. For a point light source, the view should be a perspective projection as wide as its desired angle of effect (it will be a sort of square spotlight). For directional light (e.g., that from the Sun), an orthographic projection should be used.

From this rendering, the depth buffer is extracted and saved. This depth map is often stored as a texture in graphics memory. This depth map must be updated any time there are changes to either the light or the objects in the scene, but can be reused in other situations, such as those where only the viewing camera moves.

The second step is to draw the scene from the usual camera viewpoint, applying the shadow map. This process has three major components, the first is to find the coordinates of the object as seen from the light, the second is the test which compares that coordinate against the depth map, and finally, once accomplished, the object must be drawn either in shadow or in light.

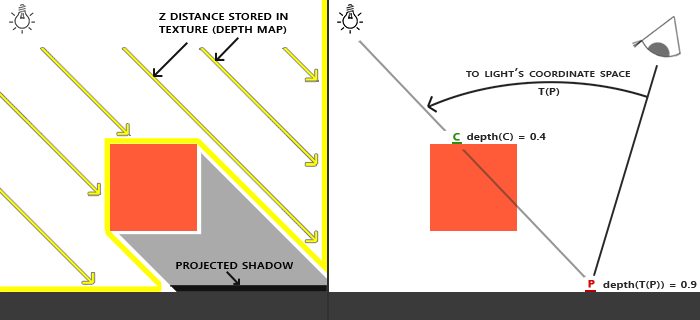


Fig: Shadow Mapping

.

**Phong Reflection Model**

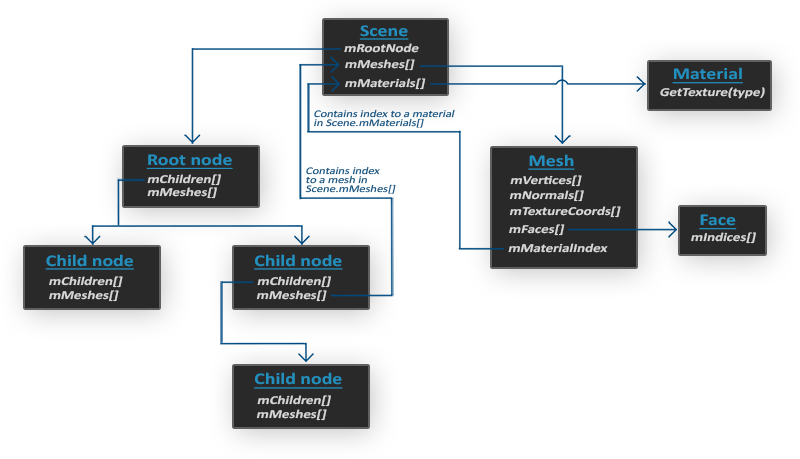
Phong shading may also refer to the specific combination of Phong interpolation and the Phong reflection model, which is an empirical model of local illumination. It describes the way a surface reflects light as a combination of the diffuse reflection of rough surfaces with the specular reflection of 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.



**3D Modelling**

In 3D computer graphics, 3D modeling is the process of developing a mathematical representation of any surface of an object (either inanimate or living) in three dimensions. The product is called a 3D model that represents a physical body using a collection of points in 3D space, connected by various geometric entities such as triangles, lines, curved surfaces, etc. Being a collection of data (points and other information), 3D models can be created manually, algorithmically (procedural modeling), or by scanning. It can be displayed as a two-dimensional image through a process called 3D rendering or used in a computer simulation of physical phenomena. The model can also be physically created using 3D printing devices

**Model Rendering**



* All the data of the scene/model is contained in the Scene object like all the materials and the meshes. It also contains a reference to the root node of the scene.
* The Root node of the scene may contain children nodes (like all other nodes) and could have a set of indices that point to mesh data in the scene object's mMeshes array. The root node's mMeshes array contains the actual Mesh objects, the values in the mMeshes array of a node are only indices for the scene's meshes array.
* A Mesh object itself contains all the relevant data required for rendering, think of vertex positions, normal vectors, texture coordinates, faces and the material of the object.
* A mesh contains several faces. A Face represents a render primitive of the object (triangles, squares, points). A face contains the indices of the vertices that form a primitive. Because the vertices and the indices are separated, this makes it easy for us to render via an index buffer.
* Finally a mesh also contains a Material object that hosts several functions to retrieve the material properties of an object. Think of colors and/or texture maps (like diffuse and specular maps).

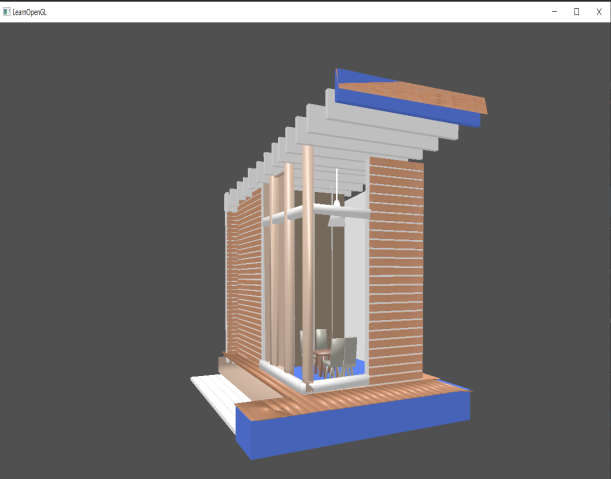
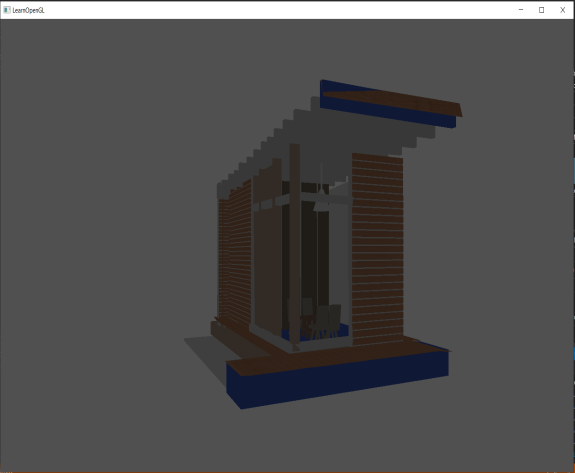
**Implementations**

As discussed earlier all the rendering code is written in C++ programming language using OpenGL library.We use two shaders: vertex shader and fragment shader. Vertex shader stores all the textures coordinates,normals in certain memory locations. These vertices are passed to fragment shader as well. Light source, direction of light and its intensity, texture to a particular location all are handled by fragment shader. Based on types of light source,its intensity, distance of object from light source and material body of textures, its lighting effect is determined.

In the project the whole object is not rendered at once.After modelling in the Sketch Up, we use .obj file where we get meshes i.e. parts of model. All these meshes and their coordinates indices and normals are provided after exporting.We simply use these values ,separate types of Material: Wooden, Metallic and so on, then lighting effect is calculated and then rendered.

Various mathematical concepts are necessary for rendering. All this is provided by the OpenGL mathematics library-glm. Mainly 3D coordinates are projected to 2D coordinates and rendered on our 2D screen. Matrix operation is very essence which is used for translation, scaling, rotation and other calculations.

**Output Screenshot**

****

****

****

**Problem Faced**

The development of 3D modeling led us through a lot of problems regarding shader programs and rendering.

* The logic of model loading was difficult while combining all the meshes
* We didn’t find some easy methods for shadow rendering.
* It took a lot of time for setup and to draw triangles as the way OpenGL works is quite difficult.
* We were not very much familiar with the GPU programs
* We find difficult to make different textures in blender and spent lots of time finding error as some textures has no coordinates

**Limitations and Future Enhancement**

Due to time limitations and OpenGL being a hard topic there are some limitations in our project. We can extend its features and make it even better. We don't consider much about advanced lighting.We do not use our own framebuffers for storing data and do not use other topics like Gamma Correction,Cube Maps, Advanced Data, HDR and Bloom.All these can be added to our recent project and enhance the beauty of model and make it more realistic.

**Conclusion and Recommendations**

Hence, with a lot of hardwork and failures, finally a sweet model was created using blender and OpenGL library in Visual Studio. If anyone has a great desire to know the logic and mathematics behind 3D Graphics pipeline, OpenGL is best to learn. It makes everyone familiar with GPU programming and helps to understand everything from scratch.

# **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](https://learnopengl.com/)