TRIBHUVAN UNIVERSITY
Institute Of Engineering
CENTRAL CAMPUS
PULCHOWK

COMPUTER GRAPHICS
PROJECT REPORT
ON
3D VIEW OF ICTC BUILDING

Submitted To

Anil Verma
Department of Electronics and Computer Engineering

Submitted by

Anusandhan Pokhrel (073BCT507)

Ashim Sharma (073BCT508)

Ashish Agarwal (073BCT509)

Date: 2075/11/22

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our teacher Anil Verma for giving us this platform to convert our theoretical knowledge into practical form. This project on computer graphics which is taught to us by our Anil sir to whom we are immensely grateful.

We would like to thank our lab teacher who assisted us to complete this project by providing us all the necessary ideas and help that we needed.

We became more familiar with this subject while working on this project.

Finally, special thanks to our every friends and family members for their supports and assistance.

Yours' Sincerely

Anusandhan Pokhrel

Ashim Sharma

Ashish Agarwal

Objectives

- To design the 3D model of ICTC building.
- To implement the features of computer graphics learnt into the project.
- To implement the 3D rotational, scaling effects.
- To implement the color filling and lightning effects.

Introduction

We used different graphical algorithms to realize a 3D complex object in this project. The algorithms are implemented in C++ using Open GL graphics library. Visual Studio and CLion are used as IDEs.

To realize the 3D object into 2D plane, we used algorithm for one point perspective projection. In the same way, other important geometrical transformation such as translations, rotations and scaling are implemented. The whole wireframe model was drawn using DDA Line Drawing Algorithm.

In order to give more realistic view, light intensity was calculated using Gouraud shading. And visible surfaces were detected using depth buffer (Z-Buffer) algorithm.

Perspective Projection

In perspective projection, the distance from the center of projection to project plane is finite and the size of the object varies inversely with distance which looks more realistic.

The distance and angles are not preserved and parallel lines do not remain parallel. Instead, they all converge at a single point called center of projection or projection reference point. There are 3 types of perspective projections which are shown in the following chart.

One point perspective projection is simple to draw.

Two point perspective projection gives better impression of depth.

Three point perspective projection is most difficult to draw.

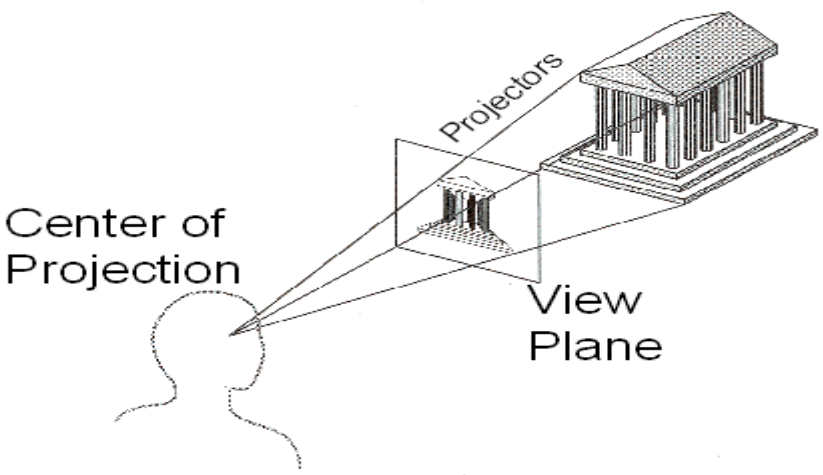


Fig: Perspective Projection

Z BUFFER

To accurately represent a complex model, it is imperative that those surfaces normally invisible from a certain point, be also invisible in the computer generated image. This problem normally called the visible surface problem or the hidden-surface problem and has been the fundamental research problem in computer graphics over the past 20 years. The effects of this algorithm are easily seen, even in relatively simple pictures. If we consider the following illustration, we can obtain much more information about the relative positions of the objects by using the right hand figure, rather than using the left hand one. Of all algorithms for visible surface determination, the depth-buffer is perhaps the simplest, and is the most widely used. For each pixel on the display, we keep a record of the depth of the object in the scene that is closest to the viewer, plus a record of the intensity that should be displayed to show the object. When a new polygon is to be processed, a z-value and intensity value are calculated for each pixel that lies within the boundary of the polygon. If the z-value at a pixel indicates that the polygon is closer to the viewer than the z-value in the z-buffer, the z-value and the intensity values recorded in the buffers are replaced by the polygon's values. After processing all polygons, the resulting intensity buffer can be displayed. The z-buffer, from which this algorithm derives its name, is an n × n array for which the (i, j) th element corresponds to the (i, j) th pixel. This array holds the image-space z value of the currently visible object at the pixel. There is also another n × n array whose elements correspond to the color that is to be assigned to the pixel.

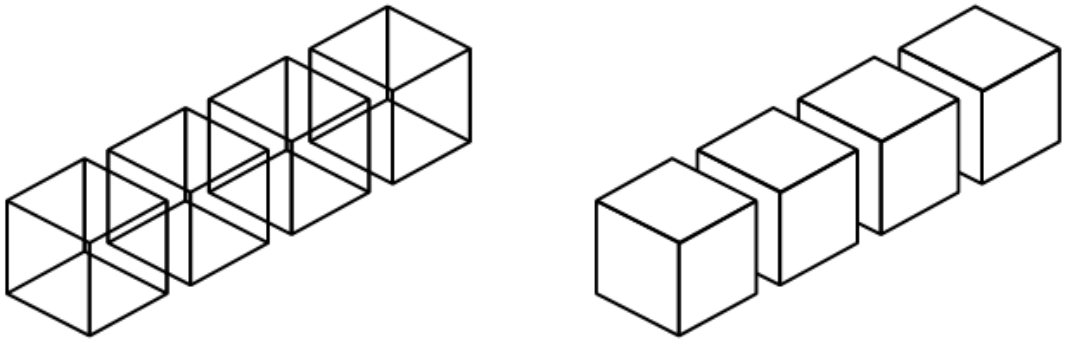


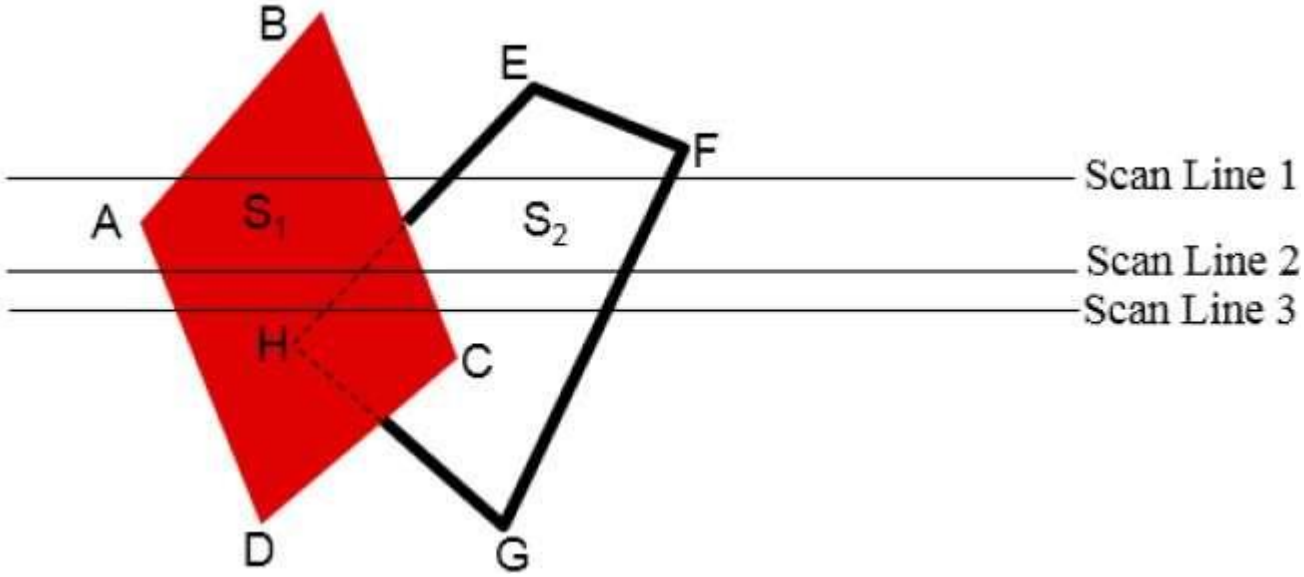
Fig. Z (Depth) Buffer

Scan-Line Method

It is an image-space method to identify visible surface. This method has a depth information for only single scan-line. In order to require one scan-line of depth values, we must group and process all polygons intersecting a given scan-line at the same time before processing the next scan-line. Two important tables, edge table and polygon table, are maintained for this.

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.



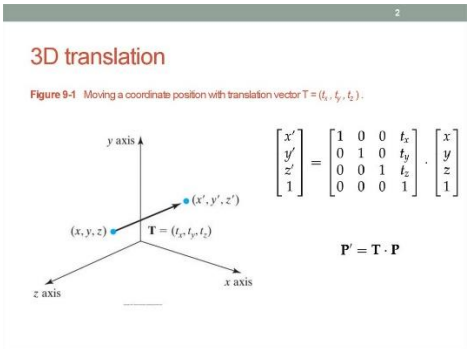
To facilitate the search for surfaces crossing a given scan-line, an active list of edges is formed. The active list stores only those edges that cross the scan-line in order of increasing x. Also a flag is set for each surface to indicate whether a position along a scan-line is either inside or outside the surface.

Pixel positions across each scan-line are processed from left to right. At the left intersection with a surface, the surface flag is turned on and at the right, the flag is turned off. You only need to perform depth calculations when multiple surfaces have their flags turned on at a certain scan-line position.

TRANSLATION

Translation can best be described as linear change in position. This change can be represented by a delta vector [Tx, Ty, Tz], where Tx represents the change in the object's x position, Ty represents the change in its y position, and Tz represents its change in z position.

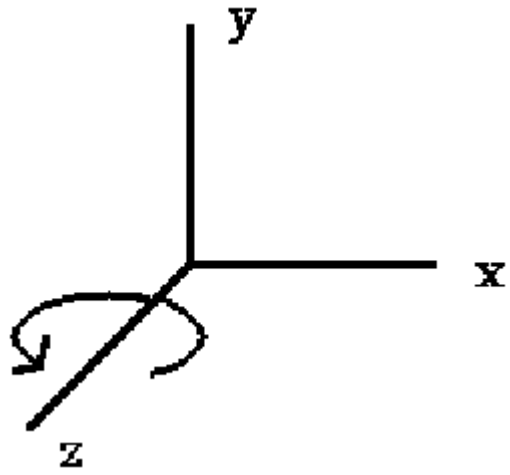
Translation is defined by a delta vector v = (Tx, Ty, Tz) and by transformation matrix.



ROTATION

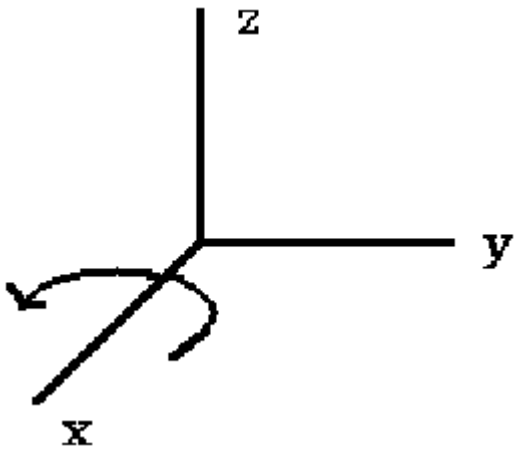
Z-Axis Rotation

Z-axis rotation is identical to the 2D case:



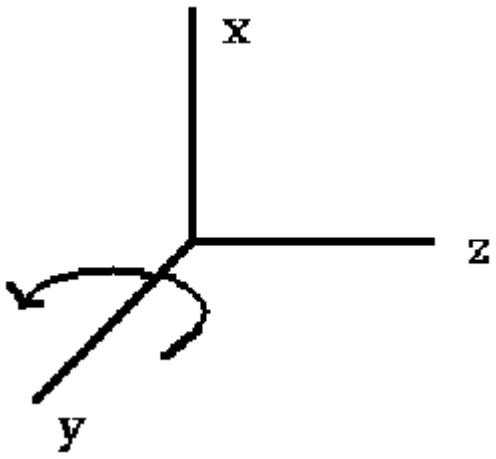
x' = x*cos q - y*sin q
y' = x*sin q + y*cos q
z' = z

X-Axis Rotation



y' = y*cos q - z*sin q
z' = y*sin q + z*cos q
x' = x

Y-Axis Rotation



z' = z*cos q - x*sin q
x' = z*sin q + x*cos q
y' = y

SCALING

We can change the size of an object using scaling transformation. In the scaling process, we either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result. In 3D scaling operation, three coordinates are used. Let us assume that the original coordinates are (X, Y, Z), scaling factors are (S_X, S_Y, S_Z) respectively, and the produced coordinates are (X', Y', Z').

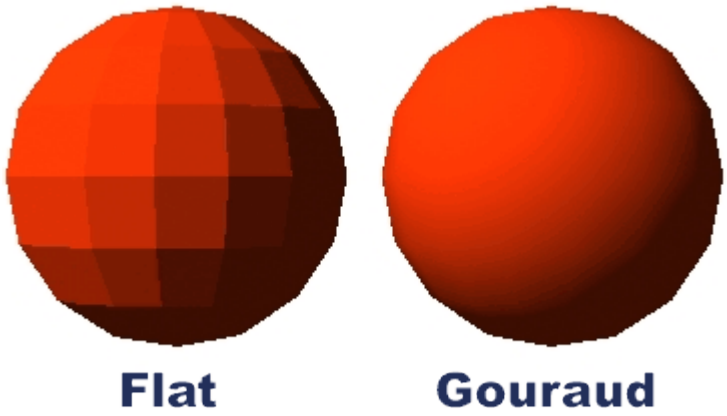
Mathematically, $X' = X . S_X$; $Y' = Y . S_Y$; $Z' = Z . S_Z$;

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

GOURAUD SHADING

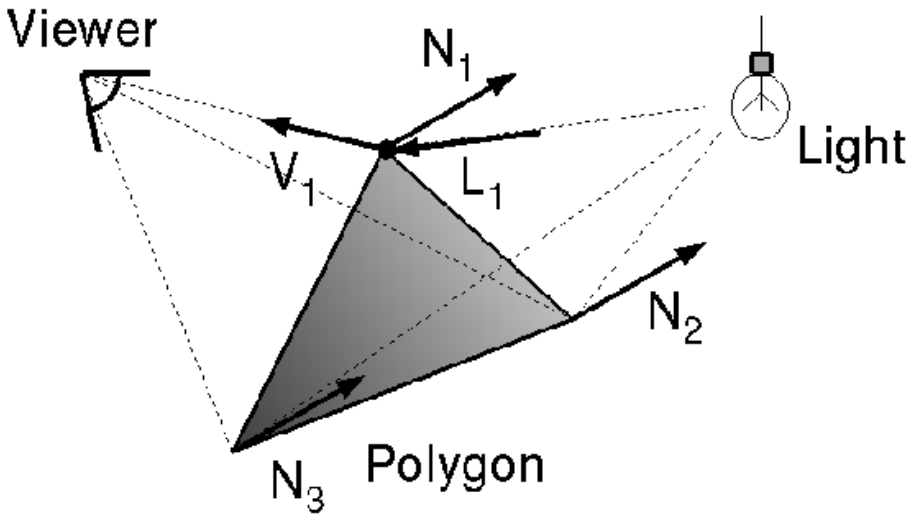
Gouraud shading is considered superior to flat shading and requires significantly less processing than Phong shading, but usually results in a faceted look.

In comparison to Phong shading, Gouraud shading's strength and weakness lies in its interpolation. If a mesh covers more pixels in screen space than it has vertices, interpolating colour values from samples of expensive lighting calculations at vertices is less processor intensive than performing the lighting calculation for each pixel as in Phong shading. However, highly localized lighting effects (such as specular highlights, e.g. the glint of reflected light on the surface of an apple) will not be rendered correctly, and if a highlight lies in the middle of a polygon, but does not spread to the polygon's vertex, it will not be apparent in a Gouraud rendering; conversely, if a highlight occurs at the vertex of a polygon, it will be rendered correctly at this vertex (as this is where the lighting model is applied), but will be spread unnaturally across all neighboring polygons via the interpolation method.

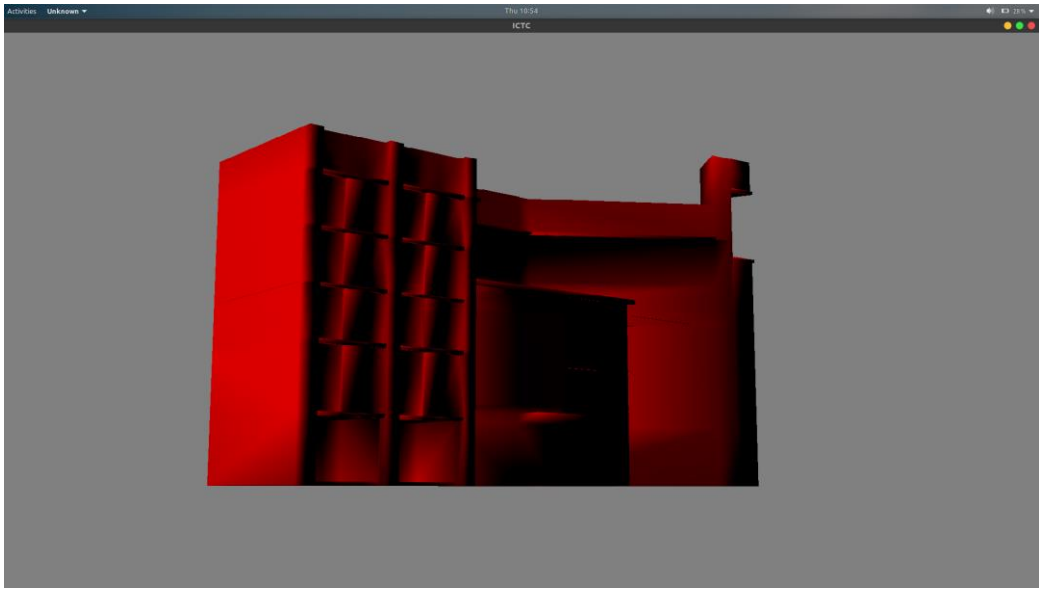


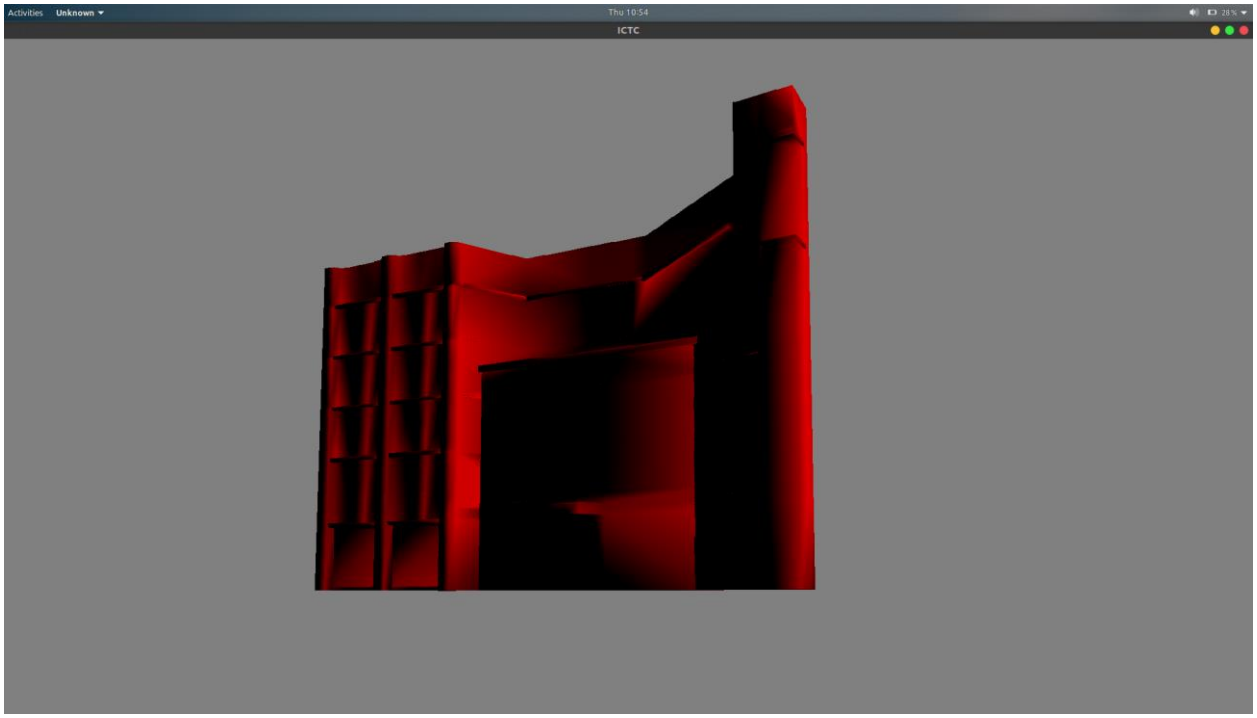
Gouraud Shading

- One lighting calculation per vertex
  - Assign pixels inside polygon by interpolating colors computed at vertices



Some Screen Shots





Conclusion

After completing this project we now became familiar with some basic concepts use in computer graphics. In this project we have used the basic ideas of computer graphics like, rotation, scaling, surface rendering, z buffer method to find the depth of the surfaces of the object. We used rotation about x-axis, y-axis and z-axis, so that our object can be rotate about all three major axis. For zooming in and zooming out, we used scaling concepts. And to hide one surface with another surface we used Z buffer technique. By comparing the value of the z coordinate of different surfaces with same value of the x and y, we can show one surface by hiding the other surfaces behind it. For the purpose of surface rendering we used the gouraud shading method. Since it applies illuminates a surface using the unit normal vector of a point it is better than other method of illumination technique. Hence by applying these basic concepts of computer graphics we completed a small project which was very useful.