BMS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DODDABALLAPUR MAIN ROAD, BANGALORE-560 064

A Mini Project Report on

"Graphical implementation of *3D view of an orthographic projection*"

Submitted in partial fulfillment of the requirements for the award of the degree of
Bachelor of Engineering in

COMPUTER SCIENCE & ENGINEERING

Submitted By:

Akshob. G
1BY08CS005

Rohin Rajan
1BY08CS037

Under the guidance of:

Mrs. Bharathi. R
Asst. Professor, Dept. of CSE

Mr. Muneshwara. M. S
Lecturer, Dept. of CSE

YEAR: 2010 - 2011

BMS INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that **Akshob G,** USN **1BY08CS005,** has successfully completed the mini project entitled "**Graphical implementation of 3D View of an Orthographic Projection"** in partial fulfillment for the award of Bachelor of Engineering in Computer science and Engineering of the VISVESVARAIAH TECHNOLOGICAL UNIVERSITY, BELGAUM during the year 2010-2011. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the Bachelor of Engineering degree.

……..	……..	…….
Signature of the Guide	**Signature of the Guide**	**Signature of the HOD**
Mrs. Bharathi. R	**Mr. Muneshwara.M.S**	**Mr. Anil .G.N**
Asst. Professor, Dept. of CSE	*Lecturer, Dept. of CSE*	*HOD,Dept.of CSE*

Signature of the examiners

Internal	**External**

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crowned our effort with success.

I express my sincere gratitude to our Principal **Dr. S. Venkateshwaran**, BMS Institute of Technology for providing facilities.

I wish to place on record my grateful thanks to **Mr. Anil .G .N**, Asst. professor and Head of the Department, Computer Science and Engineering, BMS Institute of Technology, Bangalore for providing the encouragement and guidance.

I hereby like to thank **Mrs. Bharathi. R** and **Mr. Muneshwara M S** on their periodic inspection, time to time evaluation of the project and help to bring the project to the present form.

Also I would like to thank the members of the faculty of CS&E department whose suggestions enabled me to surpass many of the seemingly impossible hurdles.

Finally, I thank my parents and my sister who have given me financial and spiritual support. Thank you for all my friends and classmates who have given comments and supports throughout the project.

ABSTRACT

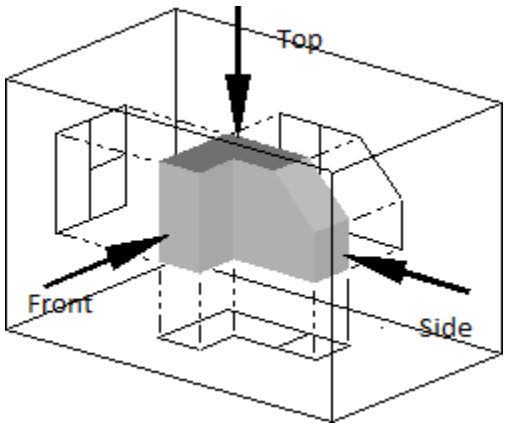
When people want to buy a flat, the flat may not be always available for the buyers to visit the flat in person. So, it is necessary for them to see the floor plan of the desired flat. However it is not possible to imagine for the non-technical buyers by just seeing the 2D floor plan. Therefore 3D model of the flat can be used. It is found that 3D modeling of a flat is not commonly used by normal users nowadays. The main reason is the common existing 3D visualization tools are difficult for non-technical users to create a 3D model of a flat with interior design. Most of the tools do not have a graphics recognition feature to automate the 3D model generation. Also most of the tools require a high end system and trained user to work on it.

The main objective of this project is to create 3D objects of simpler objects like cone, cube, cylinder etc...

Using the idea of this a fully working model of software can be written which can import a blue print and generate a full-fledged 3D model of the flat.

Orthographic Projection (or orthogonal projection) is a means of representing a three-dimensional object in two dimensions. It is a form of parallel projection, where all the projection lines are orthogonal to the projection plane, resulting in every plane of the scene appearing in affine transformation on the viewing surface. It is divided into multiview orthographic projections and axonometric projections.

With multiview orthographic projections, up to six pictures of an object are produced, with each projection plane parallel to one of the coordinate axes of the object. The views are positioned relative to each other according to either of two schemes: First-angle or Third-angle projection.



Top View, Front View and Side view of an object.

Using these projections the 3D object can be constructed.

CONTENTS

Chapter1.

Introduction

1.1. COMPUTER GRAPHICS

Today, computers and computer-generated images touch many aspects of daily life. Computer imagery is found on television, in newspapers, for example in weather reports, or for example in all kinds of medical investigation and surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. In the media "such graphs are used to illustrate papers, reports, theses", and other presentation material.

Many powerful tools have been developed to visualize data. Computer generated imagery can be categorized into several different types: 2D, 3D, 5D, and animated graphics. As technology has improved, 3D computer graphics have become more common, but 2D computer graphics are still widely used. Computer graphics has emerged as a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Over the past decade, other specialized fields have been developed like information visualization, and scientific visualization more concerned with "the visualization of three dimensional phenomena (architectural, meteorological, medical, biological, etc.), where the emphasis is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic (time) component".

HISTORY

The advance in computer graphics was to come from one MIT student, Ivan Sutherland. In 1961 Sutherland created another computer drawing program called

Sketchpad. Using a light pen, Sketchpad allowed one to draw simple shapes on the computer screen, save them and even recall them later.

The first major advance in 3D computer graphics was created at UU by these early pioneers, the hidden-surface algorithm. In order to draw a representation of a 3D object on the screen, the computer must determine which surfaces are "behind" the object from the viewer's perspective, and thus should be "hidden" when the computer creates (or renders) the image.

1.2. IMAGE TYPES

2D computer graphics:

2D computer graphics are the computer-based generation of digital images—mostly from two-dimensional models, such as 2D geometric models, text, and digital images, and by techniques specific to them.2D computer graphics are mainly used in applications that were originally developed upon traditional printing and drawing technologies, such as typography, cartography, technical drawing, and advertising. Two-dimensional models are preferred, because they give more direct control of the image than 3D computer graphics, whose approach is more akin to photography than to typography.

There are two approaches to 2D graphics: vector and raster graphics.

- Pixel art: Pixel art is a form of digital art, created through the use of raster graphics software, where images are edited on the pixel level.
- Vector graphics: Vector graphics formats are complementary to raster graphics, which is the representation of images as an array of pixels, as it is typically used for the representation of photographic images.

3D computer graphics:

With the birth of the workstation computers(like LISP machines, paintbox computers and Silicon Graphics workstations)came the 3D computer graphics.3D computer graphics in contrast to 2D computer graphics are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering 2D images.

Some major advances in 3D computer graphics since then have been:

- **Flat shading:** A technique that shades each polygon of an object based on the polygon's "normal" and the position and intensity of a light source.
- **Gouraud shading:** Invented by Henri Gouraud in 1971, a fast and resource-conscious technique used to simulate smoothly shaded surfaces by interpolating vertex colors across a polygon's surface.
- **Texture mapping:** A technique for simulating surface detail by mapping images (textures) onto polygons.
- **Phong shading:** Invented by Bui Toc Phong, a smooth shading technique that approximates curved-surface lighting by interpolating the vertex normals of a polygon across the surface; the lighting model includes glossy reflection with a controllable level of gloss.
- **Bump mapping:** Invented by Jim Blinn, a normal-perturbation technique used to simulate bumpy or wrinkled surfaces.
- **Raytracing:** A method based on the physical principles of geometric optics that can simulate multiple reflections and transparency.
- **Radiosity:** a technique for global illumination that uses radiative transfer theory to simulate indirect (reflected) illumination.

APPLICATIONS OF COMPUTER GRAPHICS

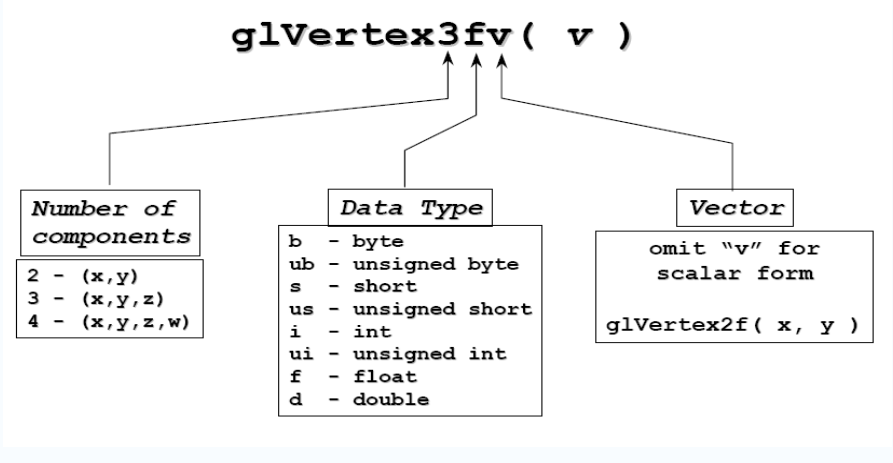
- Computational biology
- Computational physics

- Computer-aided design
- Computer simulation
- Digital art
- Education
- Graphic design
- Video Games
- Virtual reality
- Web design

1.3. OPENGL

As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer. These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images). OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

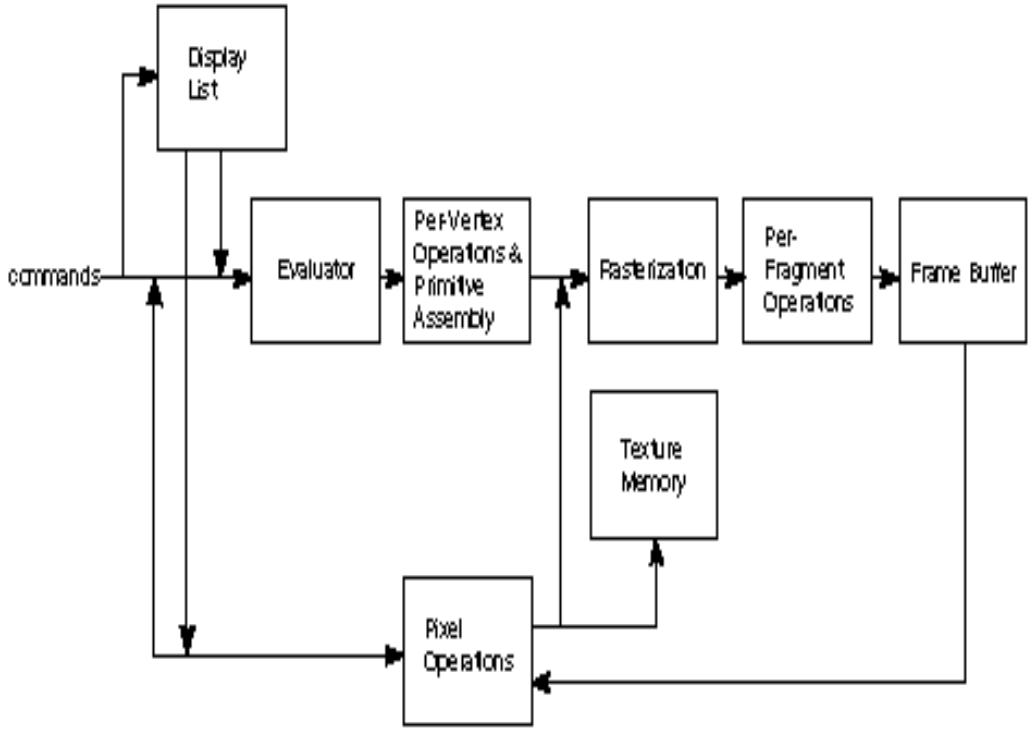
- **OpenGL Command Format**



Basic OpenGL Operation

The figure shown below gives an abstract, high-level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.

OpenGL Block Diagram



As shown by the first block in the diagram, rather than having all commands proceed immediately through the pipeline, you can choose to accumulate some of them in a *display list* for processing at a later time.

The *evaluator* stage of processing provides an efficient means for approximating curve and surface geometry by evaluating polynomial commands of input values.

During the next stage, *per-vertex operations and primitive assembly*, OpenGL processes geometric primitives—points, line segments, and polygons, all of which are described by vertices. Vertices are transformed and lit, and primitives are clipped to the viewport in preparation for the next stage.

Rasterization produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each *fragment* so produced is fed into the last stage, *per-fragment operations*, which performs the final operations on the data before it's stored as pixels in the *frame buffer*. These operations include conditional updates to the frame buffer based on incoming and previously stored z-values (for z-buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values.

Input data can be in the form of pixels rather than vertices. Such data, which might describe an image for use in texture mapping, skips the first stage of processing described above and instead is processed as pixels, in the *pixel operations* stage. The result of this stage is either stored as *texture memory*, for use in the rasterization stage, or rasterized and the resulting fragments merged into the frame buffer just as if they were generated from geometric data.

All elements of OpenGL state, including the contents of the texture memory and even of the frame buffer, can be obtained by an OpenGL application.

GLUT

GLUT (pronounced like the *glut* in *gluttony*) is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming.

GLUT provides a portable API so you can write a single OpenGL program that works on both Win32 PCs and X11 workstations.

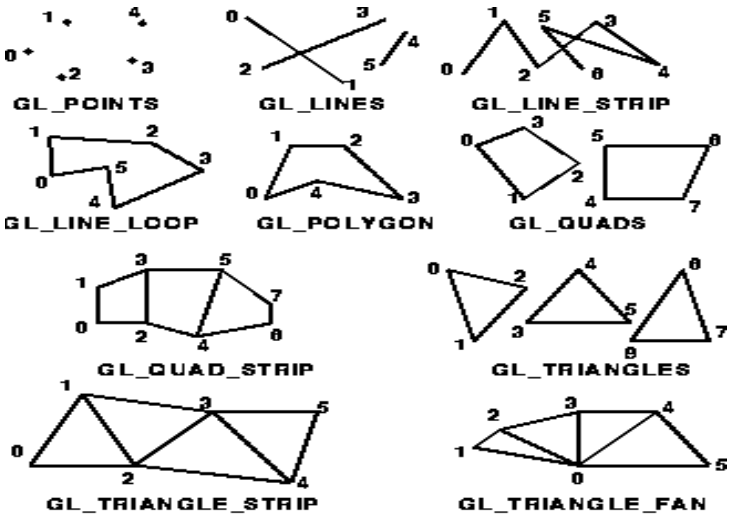
GLUT is designed for constructing small to medium sized OpenGL programs. While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits like Motif. GLUT is simple, easy, and small. My intent is to keep GLUT that way.

The GLUT library supports the following functionality:

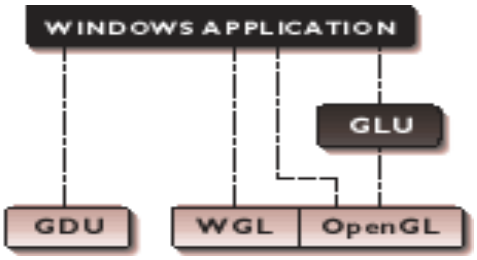
- Multiple windows for OpenGL rendering.
- Callback driven event processing.
- An `idle' routine and timers.
- Utility routines to generate various solid and wire frame objects.
- Support for bitmap and stroke fonts.
- Miscellaneous window management functions.

OPENGL PRIMITIVES

The programmer is provided the following primitives for use in constructing geometric objects.



API HIERARCHY



~ OpenGL applications use the window system's window, input, and even the mechanism.

~ GLU supports quadrics, NURBS, complex polygons, matrix utilities, and more.

APPLICATIONS OF OPENGL

1 Virtual reality
2 CAD
3 Scientific Visualization
4 Information Visualization
5 Flight simulation
6 Video games, where it competes with Direct3D
7 Animations
8 Interfacing with the User.

Chapter 2.

Requirements

2.1. Software Requirements:

- Microsoft Windows XP, Windows 7.
- OpenGL runtime libraries – glut, glu and gl.
- Any Compiler like Cygwin, MinGW, UNIX compilers.
- Any IDE (Integrated Development Environment) like Microsoft Visual C++, Netbeans, Eclipse.

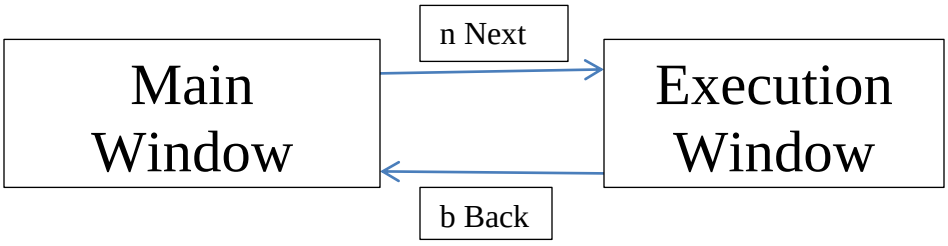
2.2. Hardware Requirements:

- Intel Dual Core Processor or higher.
- 512 MB of RAM for Windows XP and 1 GB RAM for Windows 7.
- Keyboard, Mouse, Monitor.

Chapter 3.

Design

Design is the planning that lays the basis for the making of every object or system. As a verb, "to design" refers to the process of originating and developing a plan for a product, structure, system, or component with intention. As a noun, "a design" is used for either the final (solution) plan (e.g. proposal, drawing, model, description) or the result of implementing that plan in the form of the final product of a design process. Designing often requires a designer to consider the aesthetic, functional, and many other aspects of an object or a process, which usually requires considerable research, thought, modeling, interactive adjustment, and re-design. With such a broad definition, there is no universal language or unifying institution for designers of all disciplines.



Main Window contains the title of the project, college name and other details.
Execution Window is divided into views using lines.

1. Front View – This is how the object looks when viewed from the front.
2. Top View – This is how the object looks when viewed from the top. Also called bird's eye view.
3. 3D View – This is the 3D object constructed form using the two views.
4. Menu Bar – Selection of objects. It can also be selected using right click.
5. Status Bar – Contains the details like instructions, mouse co-ordinates and which sub-window the mouse pointer is in.

Chpater 3.

Implementation

3.1. Introduction

The program is implemented using OpenGL library. OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using. Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, you must build up your desired model from a small set of *geometric primitives* - points, lines, and polygons sophisticated library that provides these features could certainly be built on top of OpenGL. The OpenGL Utility Library (GLU) provides many of the modeling features, such as quadric surfaces and NURBS curves and surfaces. GLU is a standard part of every OpenGL implementation. Also, there is a higher-level, object-oriented toolkit, Open Inventor, which is built atop OpenGL, and is available separately for many implementations of OpenGL.

3.2. The **inbuilt functions** used in the project are as follows:

1. **void glutInit(int argc, char **argv)**

   Initializes GLUT. The arguments passed from main can be used by the applicaions.

2. **void glutInitDisplayMode(unsigned int mode)**

   Requests a display with properties in mode. The value of the mode is determined by logical OR operaion.

   Mode values used are GLUT_DOUBLE, GLUT_RGB, GLUT_DEPTH.

3. **void glutInitWindowSize(int width, int height)**

   Specifies the initial width and height of the window in pixels.

4. **int glutCreateWindow(const char *title)**

   Gives a name to the window which is created.

5. **void glutDisplayFunc(void (*func)(void))**

   Registers the display function ‘func’ that is executed when the window needs to be redrawn.

6. **void glutMouseFunc(void (*func)(int button, int state, int x, int y))**

   The callback function returns the button, GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON, GLUT_MIDDLE_BUTTON, the state of the button after the event GLUT_UP or GLUT_DOWN and the position of the mouse relative to the top corner of the window.

7. **void glutKeyboardFunc(void (*func)(int key, int x, int y))**

   The callback function returns the key pressed with the coordinates of the mouse relative to the top corner of the window.

8. **void glutPassiveMotionFunc(void (*func)(int x, inty))**

   This function sets the callback for the current window. The motion callback for a window is called when the mouse moves within the window while no mouse buttons are pressed. The x and y callback parameters indicate the mouse position relative to the left corner of the window.

9. **void glutIdleFunc(void (*func)(void))**

   glutIdleFunc sets the global idle callback to be func so a GLUT program can perform background processing tasks or continuous animation when window system events are not being received. If enabled, the idle callback is continuously called when events are not being received. The callback routine has no parameters. The current window and current menu will not be changed before the idle callback. Programs with multiple windows and/or menus should explicitly set the current window and/or current menu and not rely on its current setting.

   The amount of computation and rendering done in an idle callback should be minimized to avoid affecting the program's interactive response. In general, not more than a single frame of rendering should be done in an idle callback.

10. void glutReshapeFunc(void (*func)(int width, int height))

glutReshapeFunc sets the reshape callback for the *current window*. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established. The width andheight parameters of the callback specify the new window size in pixels. Before the callback, the *current window* is set to the window that has been reshaped.

If a reshape callback is not registered for a window or NULL is passed to glutReshapeFunc (to deregister a previously registered callback), the default reshape callback is used. This default callback will simply call glViewport(0,0,width,height) on the normal plane (and on the overlay if one exists).

If an overlay is established for the window, a single reshape callback is generated. It is the callback's responsibility to update both the normal plane and overlay for the window (changing the *layer in use* as necessary).

When a top-level window is reshaped, subwindows are not reshaped. It is up to the GLUT program to manage the size and positions of subwindows within a top-level window. Still, reshape callbacks will be triggered for subwindows when their size is changed using glutReshapeWindow.

11. void myinit(void)

The initialization function. Usually used to initialize the matrix mode, set the background screen color, setting the clipping volume.

The clipping volume can be either perspective viewing or orthographic viewing.

12. glPushMatrix()

Saves the present matrix on the OpenGL stack.

13. glPopMatrix()

Restores the previous matrix which was pushed onto the stack.

14. glTranslatef(float x, float y, float z)

Translates to the coordinates given by the parameters.

15. glRotatef(float theta, float x, float y, float z)

Rotates the object with reference to the origin along the coordinate axes specified by x, y and z.

x = 1 indicates rotate the object along the x-axis and x = 0 indicates do not rotate the object along the x-axis. It also applies to the other axes as well.

The angle of rotation is specified by the first parameter theta.

3.3. These are the **user defined functions** used:

1. **void cone(void)**

   Handles the request for the front view and top view drawing of a cone.

2. **void cube(void)**

   Handles the request for the front view and top view drawing of a cube.

3. **void cylinder(void)**

   Handles the request for the front view and top view drawing of a cylinder.

4. **void tdcone(float a, float b, float h, float r)**

   Draws the 3D view of a cone. First two parameters are the starting point to draw the cone. ‘h’ is the height of the cone and ‘r’ is the radius of the base of the cone.

5. **void tdcube(float x, float y, float a)**

   Draws the 3D view of a cube. First two parameters specify the starting point from which the cube has to be drawn. ‘a’ specifies the length of the side.

6. **void tdcylinder(float a, float b, float h, float r)**

   Draws the 3D view of a cylinder. First two parameters specify the starting point from which the cylinder has to be drawn. ‘h’ is the height of the cylinder and ‘r’ is the radius of the cylinder.

7. **void square(float a, float b, float c, float d)**

   Given two coordinate points, (a,b) and (c,d) which are the opposite points of a diagonal of any square, it can be drawn.

8. **void circle(float r)**

   Given the radius of a circle, this function draws the circle. The initial point has to be translated before calling this function by using glTranslatef().

9. **void triangle(float a, float b, float c, float d)**

   Given the apex coordinate point (a,b) and the right coordinate point of the base (c,d) the triangle can be constructed.

10. **void rectangle(float a, float b, float c, float d)**

    Given opposite coordinate points of the diagonal of a rectangle, it can be constructed. This function handles that request.

11. **void Write(float x, float y, float scale, const char *)**

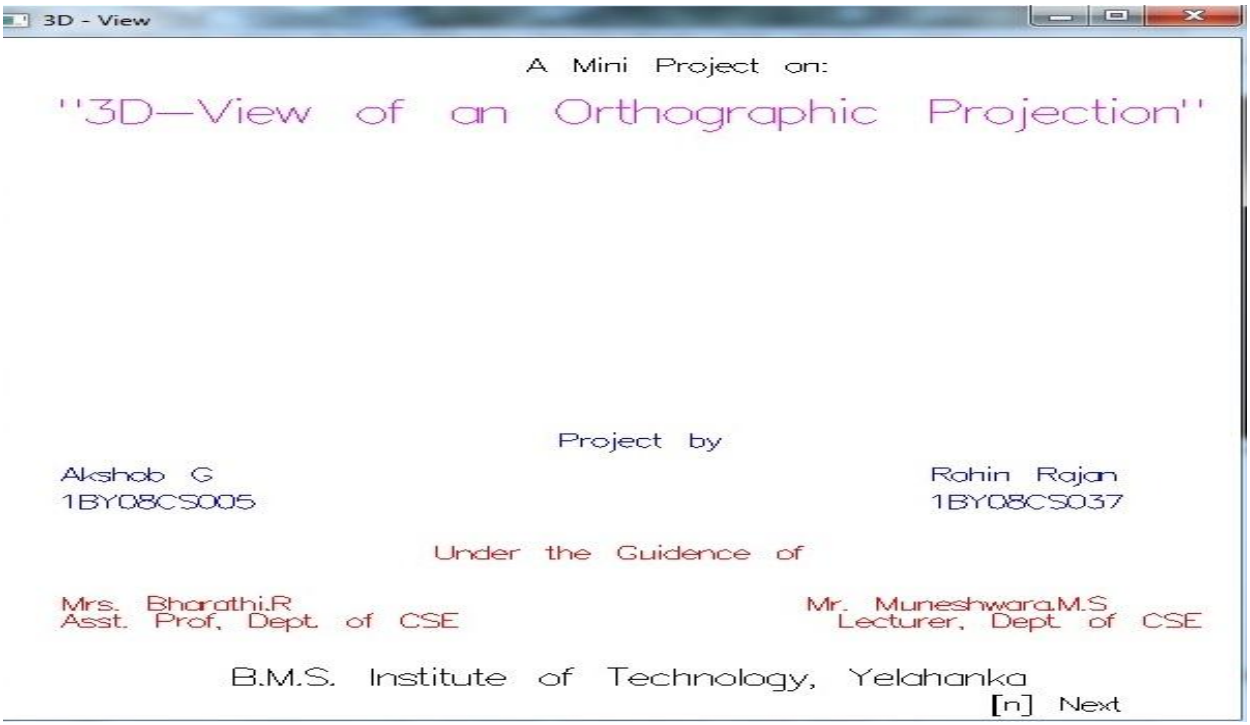
    First two parameters specify the coordinate points of the text to be drawn. scale specifies how big the text should look. Const char * is a pointer to the text to be drawn on the screen. This function uses glutStrokeCharacter() function to draw the text on the screen.

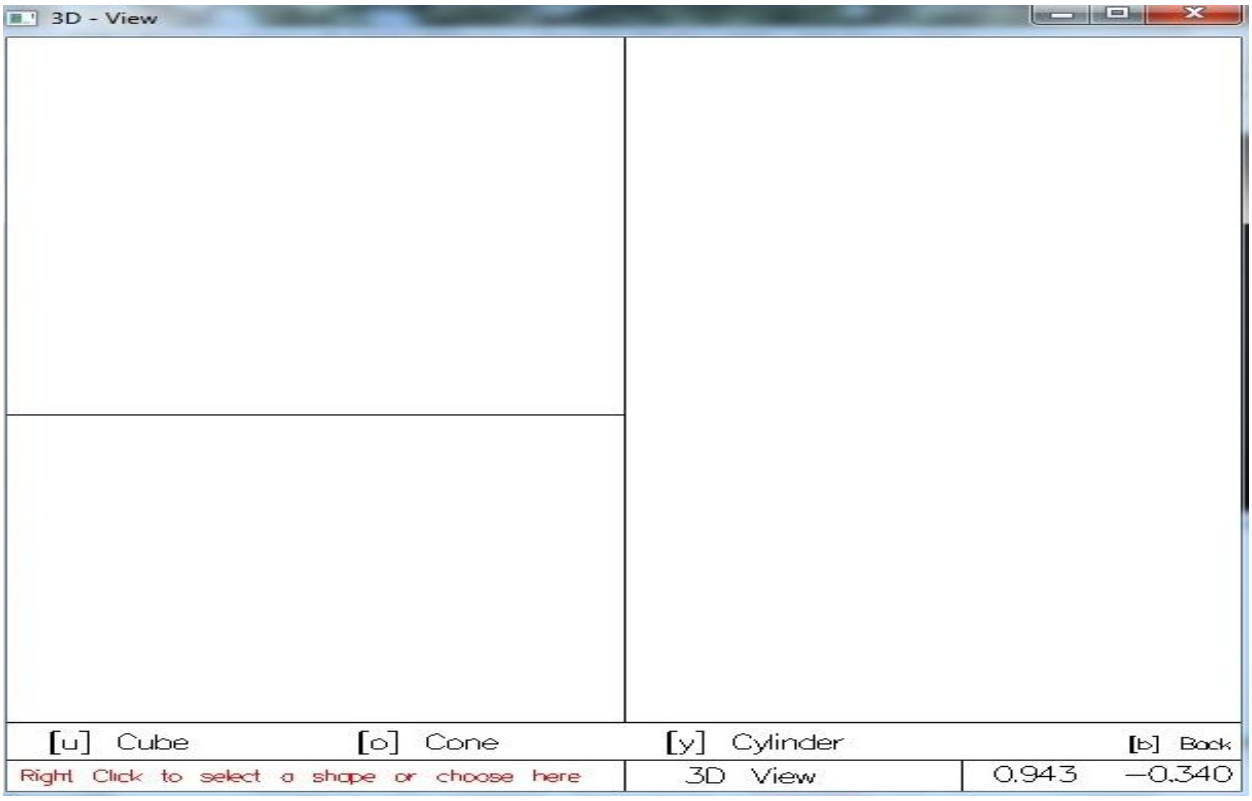
12. **void WriteNum(float x, float y, float scale, float num)**

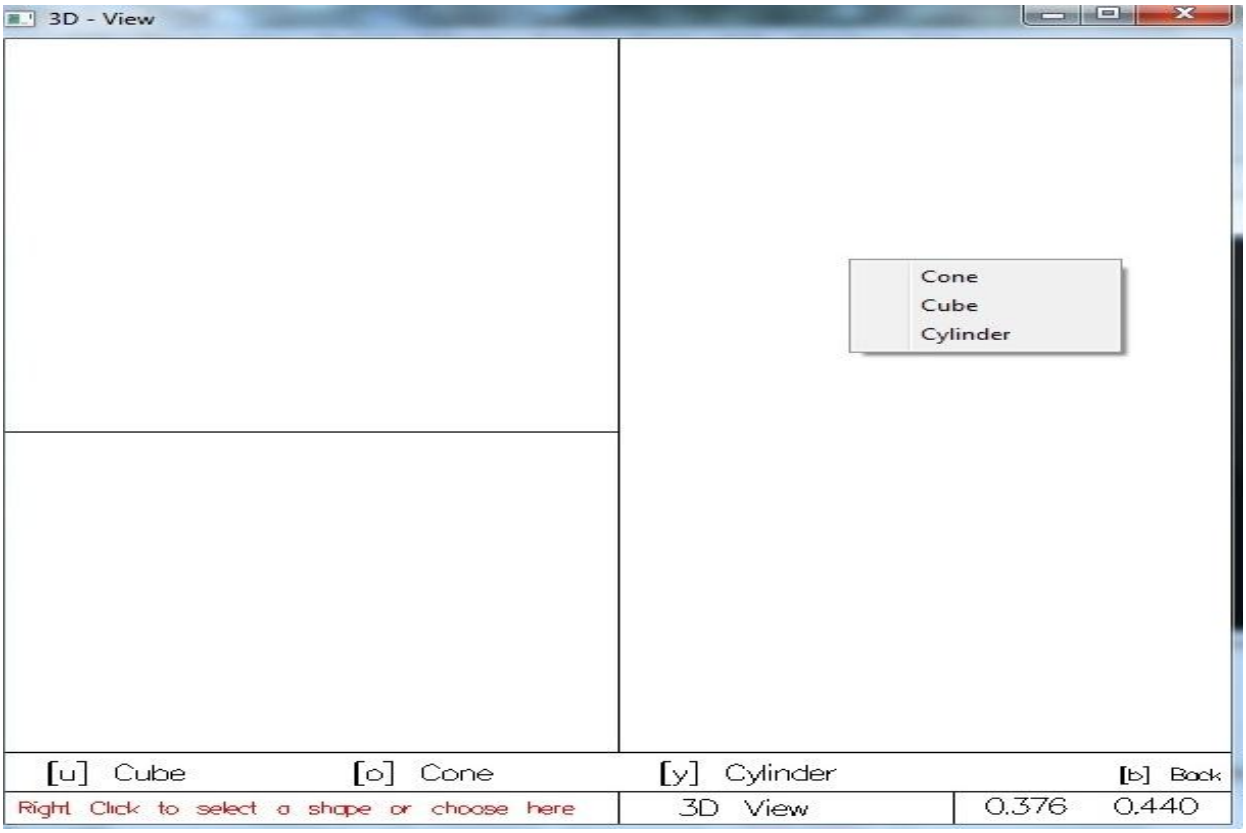
    This function is same as the Write function, except that this function takes a float point number as input which is to be printed instead of a string.

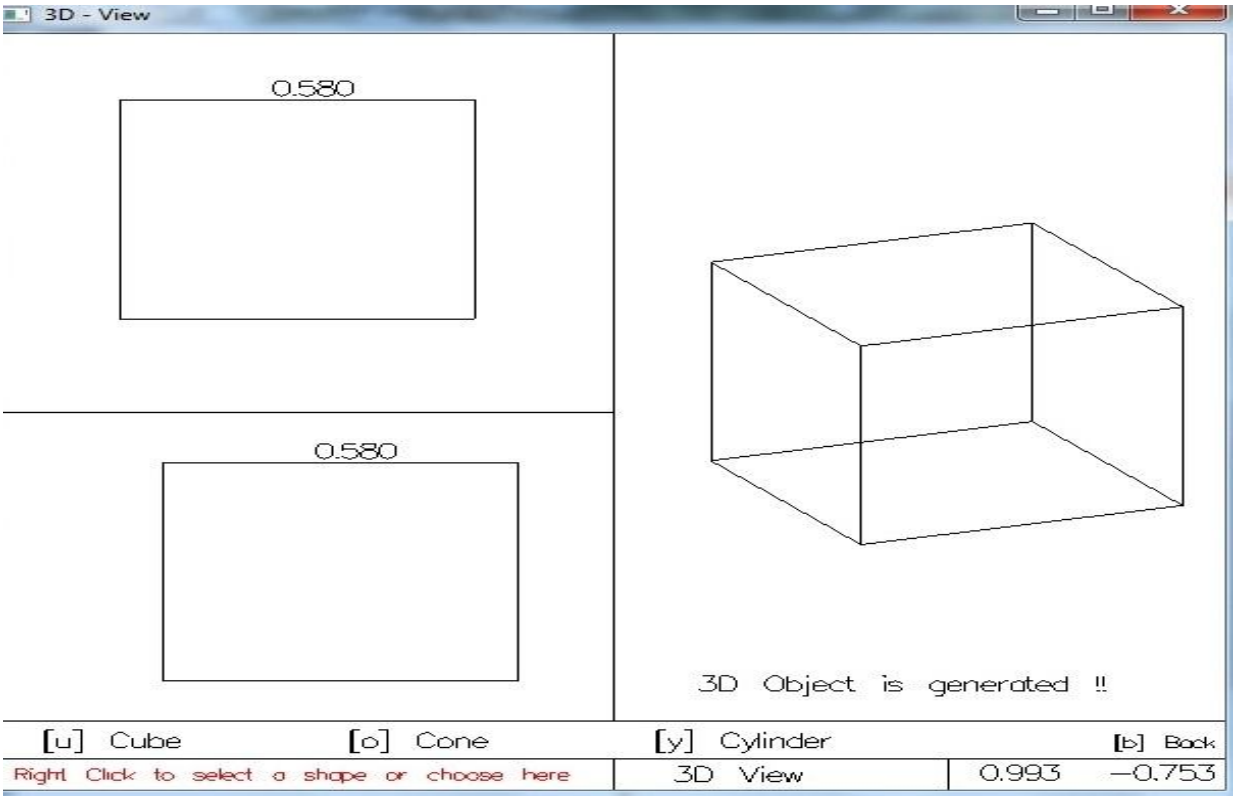
Chapter 4.

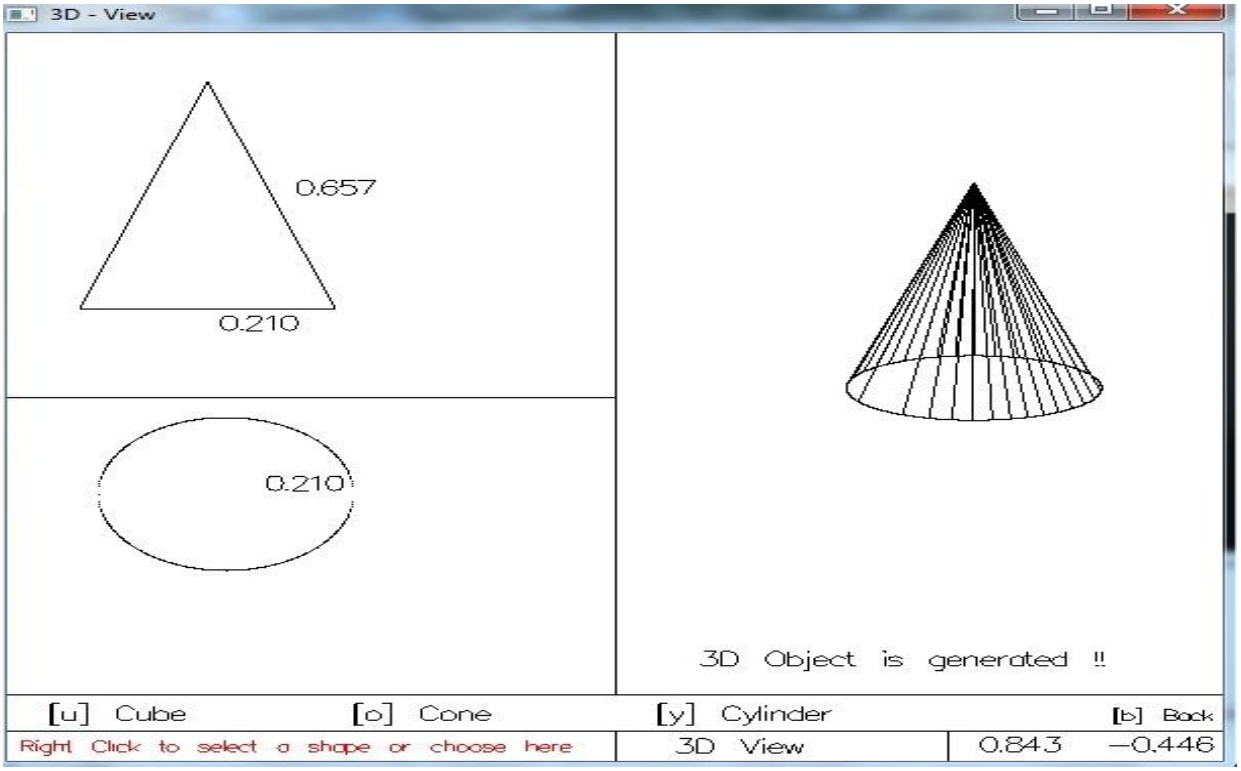
Screenshots

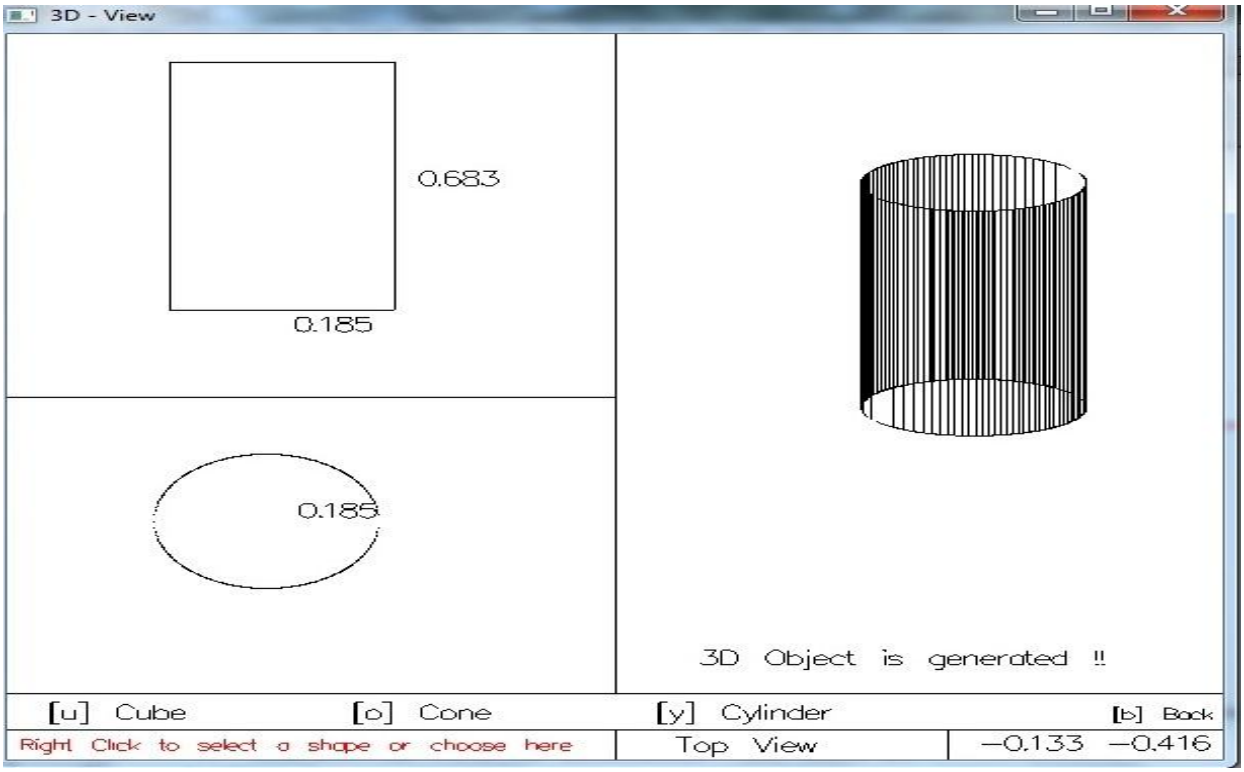





3D - View
Cone
Cube
Cylinder
[u] Cube
[o] Cone
[y] Cylinder
[b] Back
Right Click to select a shape or choose here
3D View
0.376
0.440


3D - View
0.580
0.580
3D Object is generated !!
[u] Cube
[o] Cone
[y] Cylinder
[b] Back
Right Click to select a shape or choose here
3D View
0.993
−0.753


3D - View
0.657
0.210
0.210
3D Object is generated !!
[u] Cube
[o] Cone
[y] Cylinder
[b] Back
Right Click to select a shape or choose here
3D View
0.843 −0.446


3D - View
0.683
0.185
0.185
3D Object is generated !!
[u] Cube
[o] Cone
[y] Cylinder
[b] Back
Right Click to select a shape or choose here
Top View
−0.133 −0.416

Chapter 5.

Conclusion

This project successfully builds a 3D visualization scheme for simple objects. The user draws the orthogonal projections of the object (top view and front view). This is easy-to-use feature which is similar to paint designed by Microsoft. The screen also has tips which serve as a guide as to how to proceed further.

For further development, the system can be extended to recognize floor plans and blue prints and generate a full 3D model. Also the system can be enhanced to support importing 2D floor plan of different file formats.

Bibliography

Books:

1. “Interactive Computer Graphics” by Edward Angel.

Websites:

1. www.opengl.org
2. www.wikipedia.org