



ESTD : 2001

An Institute with a Difference

RNS INSTITUTE OF TECHNOLOGY

(AICTE Approved, VTU Affiliated and NAAC 'A' Accredited)

UG Programs - CSE, ECE, ISE, EIE and EEE have been Accredited by NBA for three Academic years

DR. VISHNUVARDHAN ROAD, CHANNASANDRA, RR NAGAR POST, BENGALURU – 560 0

Department of Computer Science and Engineering

COMPUTER GRAPHICS LAB WITH MINI PROJECT MANUAL

**For Sixth Semester B.E
[VTU/CBCS, 2018-19 syllabus]**

Subject Code – 18CSL67

VISION AND MISSION OF INSTITUTION

Vision

Building RNSIT into a World Class Institution

Mission

To impart high quality education in Engineering, Technology and Management with a Difference, Enabling Students to Excel in their Career by

1. Attracting quality Students and preparing them with a strong foundation in fundamentals so as to achieve distinctions in various walks of life leading to outstanding contributions.
2. Imparting value based, need based, choice based and skill based professional education to the aspiring youth and carving them into disciplined, World class Professionals with social responsibility.
3. Promoting excellence in Teaching, Research and Consultancy that galvanizes academic consciousness among Faculty and Students.
4. Exposing Students to emerging frontiers of knowledge in various domains and make them suitable for Industry, Entrepreneurship, Higher studies, and Research & Development.
5. Providing freedom of action and choice for all the Stake holders with better visibility.

VISION AND MISSION OF CSE DEPARTMENT

Vision

Preparing better computer professionals for a real world

Mission

The Department of Computer Science and Engineering will make every effort to promote an intellectual and an ethical environment in which the strengths and skills of Computer Professionals will flourish by

1. Imparting Solid foundations and Applied aspects in both Computer Science Theory and Programming practices.
2. Providing Training and encouraging R&D and Consultancy Services in frontier areas of Computer Science with a Global outlook.
3. Fostering the highest ideals of Ethics, Values and creating Awareness on the role of Computing in Global Environment.
4. Educating and preparing the graduates, highly Sought-after, Productive, and Well-respected for their work culture.
5. Supporting and inducing Lifelong Learning practice

PREFACE

We have developed this comprehensive laboratory manual on **COMPUTER GRAPHICS LABORATORY WITH MINI PROJECT** with two primary objectives: To make the students comfortable with basic principles of problem solving and to train them in evolving as an efficient Computer Graphics programmer by strengthening their OpenGL programming abilities.

This material is divided into two parts: Part-A and Part-B, which provides the students an exposure to problem solving approaches and solution to number of problems using OpenGL API. The problems discussed in this manual comprises of an algorithm, programming solution, alternative logic and extensive test cases. Viva questions, frequently appeared examination questions and practicing programming problems constitute an indispensable part of this material.

Our profound and sincere efforts will be fruitful only when students acquire the extensive knowledge by reading this manual and apply the concepts learnt apart from the requirements specified in **COMPUTER GRAPHICS LABORATORY WITH MINI PROJECT** as prescribed by VTU, Belagavi.

Department of CSE

TABLE OF CONTENTS

SL.NO.	CONTENTS	PAGE NO.
---------------	-----------------	-----------------

PART A

1	Implement Brenham's line drawing algorithm for all types of slope.	9
2	Create and rotate a triangle about the origin and a fixed point.	13
3	Draw a color cube and spin it using OpenGL transformation matrices.	16
4	Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.	19
5	Clip a lines using Cohen-Sutherland algorithm.	22
6	To draw a simple shaded scene consisting of a tea pot on a table.	26
7	Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket.	30
8	Develop a menu driven program to animate a flag using Bezier Curve algorithm.	33
9	Develop a menu driven program to fill the polygon using scan line algorithm	37

PART B

Student should develop mini project on the topics mentioned below or similar applications using Open GL API. Consider all types of attributes like color, thickness, styles, font, background, speed etc., while doing mini project. (During the practical exam: the students should demonstrate and answer Viva-Voce)

Sample Topics: Simulation of concepts of OS, Data structures, algorithms etc.

Appendix –A : list of module-wise viva	41
---	-----------

SYALLABUS

SEMESTER VI

COMPUTER GRAPHICS LABORATORY WITH MINI PROJECT

Sub Code : 18CSL67

CIE Marks : 40

Hrs/ Week : 03

Exam Hours : 03

Credits : 01

SEE Marks : 60

Course Learning Objectives:

This course (18CSL67) will enable students to:

- Demonstrate simple algorithms using OpenGL Graphics Primitives and attributes.
- Implementation of line drawing and clipping algorithms using OpenGL functions
- Design and implementation of algorithms Geometric transformations on both 2D and 3D objects.

LABORATORY PROGRAMS

PART A

Design, develop, and implement the following programs using OpenGL API

1. Implement Brenham's line drawing algorithm for all types of slope.
2. Create and rotate a triangle about the origin and a fixed point.
3. Draw a color cube and spin it using OpenGL transformation matrices.
4. Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.
5. Clip a lines using Cohen-Sutherland algorithm.
6. To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene.
7. Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user.
8. Develop a menu driven program to animate a flag using Bezier Curve algorithm.
9. Develop a menu driven program to fill the polygon using scan line algorithm

PART B (MINI-PROJECT)

Student should develop mini project on the topics mentioned below or similar applications using Open GL API. Consider all types of attributes like color, thickness, styles, font, background, speed etc., while doing mini project. (During the practical exam: the students should demonstrate and answer Viva-Voce)

Sample Topics: Simulation of concepts of OS, Data structures, algorithms etc.

Laboratory Outcomes: The student should be able to:

- Apply the concepts of computer graphics
- Implement computer graphics applications using OpenGL
- Animate real world problems using OpenGL

Practical Examination Procedure:

- Experiment distribution
 - o For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
 - o For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution (*Courseed to change in accordance with university regulations*)
 - o) For laboratories having only one part
 - Procedure + Execution + Viva-Voce: $15+70+15=100$ Marks
 - o) For laboratories having PART A and PART B
 - i. Part A – Procedure + Execution + Viva = $6 + 28 + 6 = 40$ Marks
 - ii. Part B – Procedure + Execution + Viva = $9 + 42 + 9 = 60$ Marks

ACKNOWLEDGMENT

A material of this scope would not have been possible without the contribution of many people. We express our sincere gratitude to Dr. R N Shetty, Chairman, RNS Group of Companies for his magnanimous support in all our endeavors.

We are grateful to Dr. M K Venkatesha, Principal, RNSIT and Dr. P Kiran, HOD, CSE for extending their constant encouragement and support.

Our heartfelt thanks to Dr. M J Sudhamani and Mrs. S Mamatha Jajur for their unparalleled contribution throughout the preparation of this comprehensive manual. We also acknowledge our colleagues for their timely suggestions and unconditional support.

Department of CSE

COMPUTER GRAPHICS LABORATORY WITH MINI PROJECT [18CSL67]

1.1 Introduction

Computer graphics are graphics created using computers and, more generally, the representation and manipulation of image data by a computer hardware and software. It is a vast and recently developed area of computer science. The development of computer graphics, or simply referred to as CG, has made computers easier to interact with, and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized the animation and video game industry. 2D computer graphics are digital images—mostly from two-dimensional models, such as 2D geometric models, text (vector array), and 2D data. 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 images.

1.2 OpenGL

OpenGL is the most widely adopted 2D and 3D graphics API in the industry, bringing thousands of applications to a wide variety of computer platforms. It is window-system and operating-system independent as well as network-transparent. OpenGL enables developers of software for PC, workstation, and supercomputing hardware to create high-performance, visually compelling graphics software applications, in markets such as CAD, content creation, energy, entertainment, game development, manufacturing, medical, and virtual reality. OpenGL exposes all the features of the latest graphics hardware.

GLUT

GLUT 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.

The OpenGL Utility Toolkit (GLUT) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system. Functions performed include window definition, window control, and monitoring of keyboard and mouse input. Routines for drawing a number of geometric primitives (both in solid and wireframe mode) are also provided, including cubes, spheres and the Utah teapot. GLUT also has some limited support for creating pop-up menus. The two aims of GLUT are to allow the creation of rather portable code between operating systems (GLUT is cross-platform) and to make learning OpenGL easier. All GLUT functions start with the glut prefix (for example, glutPostRedisplay marks the current window as needing to be redrawn).

GLU Library

GLU is the OpenGL Utility Library. It consists of a number of functions that use the base OpenGL library to provide higher-level drawing routines from the more primitive routines that OpenGL provides. It is usually distributed with the base OpenGL package. This is a set of functions to map between screen- and world-coordinates, generation of texture mipmaps, drawing of quadric surfaces, NURBS, tessellation of polygonal primitives, interpretation of OpenGL error codes, an extended range of transformation routines for setting up viewing volumes and simple positioning of the camera, generally in more human-friendly terms than the routines presented by OpenGL. It also provides additional primitives for use in OpenGL applications, including spheres, cylinders and disks.

All GLU functions start with the glu prefix. An example function is gluOrtho2D which defines a two dimensional orthographic projection matrix.

Include Files

For all OpenGL applications, you want to include the gl.h header file in every file. Almost all OpenGL applications use GLU, the aforementioned OpenGL Utility Library, which also requires inclusion of the glu.h header file. So almost every OpenGL source file begins with:

```
#include <GL/gl.h>
```

```
#include <GL/glu.h>
```

If you are using the OpenGL Utility Toolkit (GLUT) for managing your window manager tasks, you should include:




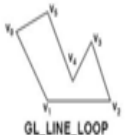
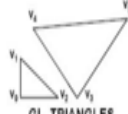
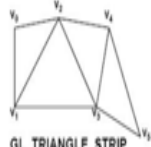
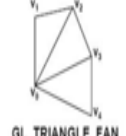



```
#include <GL/glut.h>
```

Note that glut.h guarantees that gl.h and glu.h are properly included for you so including these three files is redundant. To make your GLUT programs portable, include glut.h and do not include gl.h or glu.h explicitly.

1.3 OpenGL Primitives

OpenGL supports several basic primitive types, including points, lines, quadrilaterals, and general polygons. All of these primitives are specified using a sequence of vertices.

OpenGL primitives

Value	Meaning			
GL_POINTS	individual points		GL_POINTS	
GL_LINES	pairs of vertices interpreted as individual line segments		GL_LINES	
GL_LINE_STRIP	series of connected line segments		GL_LINE_STRIP	
GL_LINE_LOOP	same as above, with a segment added between last and first vertices		GL_LINE_LOOP	
GL_TRIANGLES	triples of vertices interpreted as triangles		GL_TRIANGLES	
GL_TRIANGLE_STRIP	linked strip of triangles		GL_TRIANGLE_STRIP	
GL_TRIANGLE_FAN	linked fan of triangles		GL_TRIANGLE_FAN	
GL_QUADS	quadruples of vertices interpreted as four-sided polygons		GL_QUADS	
GL_QUAD_STRIP	linked strip of quadrilaterals		GL_QUAD_STRIP	
GL_POLYGON	boundary of a simple, convex polygon		GL_POLYGON	

1.4 OpenGL Program Organization

- **main:**
 - find GL visual and create window
 - initialize GL states (e.g. viewing, color, lighting)
 - initialize display lists
 - loop
 - check for events (and process them)
 - if window event (window moved, exposed, etc.)
 - modify viewport, if needed
 - redraw
 - else if mouse or keyboard
 - do something, e.g., change states and redraw
- **redraw:**
 - clear screen (to background color)
 - change state(s), if needed
 - render some graphics
 - change more states
 - render some more graphics

1.5 Open GL program installation and execution steps

Step 1: Create a Visual Studio Project

To create an empty console project in Visual Studio, do the following:

1. Create a new project (File ---> New ---> --->Project)
2. In the Project Types: pane, select Visual C++, Win32. Then select Win 32 Console Application in the Templates: pane. Name your project, select the location for the project and click OK.
3. Click the Application Settings tab on the left, and check the Empty Project box. Then click Finish button

Step 2: Add Source Code

1. Select Project, Add New Item
2. In the Categories pane, select Visual C++, Code. Then select C++ File (.cpp) in the Templates: pane. Name your file, and then click Add.

Step 3: Compile and Run the project

- a. Compile the Program
From the Visual Studio's menu Build option (Build ---> Build Solution)
- b. Execute the program

From the Visual Studio's menu Debug option (Debug ---> Start Without Debugging)

1.6 Summary of OpenGL| GLUT functions

glutInit: initializes GLUT, must be called before other GL/GLUT functions. It takes the same arguments as the main().

```
void glutInit(int *argc, char **argv)
```

glutCreateWindow: Creates a display window (which is assigned an integer identifier) and specify a display-window title.

```
glutCreateWindow ("Title");
```

glutInitWindowSize: Specifies width and height for a display window.

```
glutInitWindowSize (winWidth, winHeight);
```

glutInitWindowPosition: Specifies coordinates for the top-left corner of a display window.

```
glutInitWindowPosition (100, 100);
```

Positions the top-left corner of the initial window at (x, y). The coordinates (x, y), in term of pixels, is measured in window coordinates, i.e., origin (0, 0) is at the top-left corner of the screen; x-axis pointing right and y-axis pointing down.

glutInitDisplayMode Selects parameters such as single/double buffer(GLUT_SINGLE,

GLUT_DOUBLE), color mode (GLUT_RGB, GLUT_RGBA, GLUT_INDEX) and depth (GLUT_DEPTH) for a display window.

glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

glutDisplayFunc: registers the callback function (or event handler) for handling window-paint event. In the example, we register the function displayFcn() as the handler. Invokes a function to create a picture within the current display window.

glutDisplayFunc (displayFcn);

glutMainLoop: At this time, display windows and their graphic contents are sent to the screen. The program also enters the GLUT processing loop that continually checks for new “events,” such as interactive input from a mouse or a graphics tablet

void glutMainLoop()

glClearColor Specifies a background RGB color for a display window.

glClearColor (1.0, 1.0, 1.0, 0.0);

glMatrixMode Specifies current matrix for geometric-viewing transformations, projection transformations, texture transformations, or color transformations.

void glMatrixMode (GLenum mode);

glLoadIdentity Sets current matrix to identity.

glLoadIdentity ();

glLoadMatrix Sets elements of current matrix.

glLoadMatrix* (elems);

The **glPushMatrix** and **glPopMatrix** functions push and pop the current matrix stack.

void glPushMatrix (void);

void glPopMatrix (void);

glutSetIconTitle Specifies a label for a display-window icon.

glutSetWindowTitle Specifies new title for the current display window.

glutPopWindow Moves current display window to the “top”; i.e., in front of all other windows.

glutPushWindow Moves current display window to the “bottom”; i.e., behind all other windows.

glutShowWindow Returns the current display window to the screen.

glutCreateSubWindow Creates a second-level window within a display window.

int glutCreateSubWindow(int win,int x,int y,int width,int height);

glPointSize function specifies the diameter of rasterized points.

void glPointSize (GLfloat size);

glEnable and **glDisable** functions enable or disable OpenGL capabilities.

void glEnable, glDisable();

glutBitmapCharacter function used for font style.

void glutBitmapCharacter(void *font, int character);

1.7 Summary of OpenGL 2D & 3D Viewing Functions

gluOrtho2D Specifies clipping-window coordinates as parameters for a two-dimensional orthogonal projection.

gluOrtho2D (0.0, 200.0, 0.0, 150.0);

glOrtho Specifies parameters for a clipping window and the near and far clipping planes for an orthogonal projection. It multiplies the current matrix by an orthographic matrix.

Void glOrtho(GLdoubleleft, GLdoubleright, GLdoublebottom, GLdoubletop,

GLdoublezNear, GLdoublezFar);

glViewport Specifies screen-coordinate parameters for a viewport.

glViewport (0, 0, newWidth, newHeight);

gluLookAt Specifies three-dimensional viewing parameters.

gluLookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);

gluPerspective Specifies field-of-view angle and other parameters for a symmetric perspective projection.

gluPerspective (theta, aspect, dnear, dfar);

glFrustum Specifies parameters for a clipping window and near and far clipping planes for a perspective projection (symmetric or oblique).

glFrustum (-1.0, 1.0, -1.0, 1.0, 2.0, 20.0);

1.8 Summary of OpenGL Input Functions

glutMouseFunc Specifies a mouse callback function that is to be invoked when a mouse button is pressed.

glutMouseFunc (mouseFcn);

This mouse callback procedure, which we named **mouseFcn**, has four arguments:

void mouseFcn (GLint button, GLint action, GLint xMouse, GLint yMouse)

glutMotionFunc Specifies a mouse callback function that is to be invoked when the mouse cursor is moved while a button is pressed.

glutMotionFunc (fcnDoSomething);

This routine invokes fcnDoSomething when the mouse is moved within the display window with one or more buttons activated. The function that is invoked in this case has two arguments:

void fcnDoSomething (GLint xMouse, GLint yMouse)

glutPassiveMotionFunc Specifies a mouse callback function that is to be invoked when the mouse cursor is moved without pressing a button.

glutKeyboardFunc Specifies a keyboard callback function that is to be invoked when a standard key is pressed.

glutKeyboardFunc (keyFcn);

The specified procedure has three arguments:

void keyFcn (GLubyte key, GLint xMouse, GLint yMouse)

glutSpecialFunc Specifies a keyboard callback function that is to be invoked when a special-purpose key (e.g., a function key) is pressed.

glutPostRedisplay Renews the contents of the current display window.

void glutPostRedisplay(void);

glutReshapeFunc Specifies a function that is to be invoked when display window size is changed.

glutReshapeFunc (winReshapeFcn);

1.9 Summary of OpenGL Geometric Transformation Functions

glTranslate* Specifies translation parameters.

glTranslatef (tx, ty, tz);

glRotate* Specifies parameters for rotation about any axis through the origin.

glRotatef (thetaDegrees, vx, vy, vz);

glScale* Specifies scaling parameters with respect to coordinate origin.

glScalef (sx, sy, sz);

PART – A

Brenham's line drawing algorithm

1. Implement Brenham's line drawing algorithm for all types of slope.

Bresenham's line algorithm is an accurate and efficient raster line-generating algorithm, developed by Bresenham, that uses only incremental integer calculations. In addition, Bresenham's line algorithm can be adapted to display circles and other curves.

Bresenham's Line-Drawing Algorithm for $|m| < 1.0$

1. Input the two line endpoints and store the left endpoint in (x_0, y_0) .
2. Set the color for frame-buffer position (x_0, y_0) ; i.e., plot the first point.
3. Calculate the constants Δx , Δy , $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as

$$p_0 = 2\Delta y - \Delta x$$

4. At each x_k along the line, starting at $k = 0$, perform the following test:
If $p_k < 0$, the next point to plot is $(x_k + 1, y_k)$ and

$$p_{k+1} = p_k + 2\Delta y$$

Otherwise, the next point to plot is $(x_k + 1, y_k + 1)$ and

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

5. Repeat step 4 $\Delta x - 1$ more times.

For $m > 1$, find out whether you need to increment x while incrementing y each time. After solving, the equation for decision parameter P_k will be very similar, just the x and y in the equation gets interchanged.

Purpose: To demonstrate Bresenham's line algorithm to draw lines.

Procedure: To read end points of lines from the user to demonstrate the working of a Bresenham's line algorithm

Input: End points of the line.

Expected Output: Draw a line between end points.

1.1 Program

```
/*Implement Bresenham's line drawing algorithm for all types
of slope.*/

#include<stdio.h>
#include<GL/glut.h>
#include<math.h>
```



```

int xx,yy,xend,yend;
void myinit()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0,500,0,500);
    glMatrixMode(GL_MODELVIEW);
}
void setPixel(int x,int y)
{
    glBegin(GL_POINTS);
    glVertex2f(x,y);
    glEnd();
    glFlush();
}
void Bresenhamline()
{
    int p, x, y;
    //to draw line from right to left
    if(xx>xend)
    {
        x=xend;
        y=yend;
        xend=xx;
        yend=yy;
    }
    else{//to draw line from left to right
        x=xx;
        y=yy;
    }

    int dx=abs(xend-x), dy=abs(yend-y);
    int twody=2*dy, twodyMinustwodx =2*(dy-dx);
    int twodx=2*dx, twodxMinustwody =2*(dx-dy);

    glColor3f(1,0,0); // Set color to red.
    glPointSize(2); //Set point size to 3
    if(dx > dy)
    {
        //For slope m<1
        p=2*dy-dx;
        setPixel(x,y);
        while(x<xend)
        {
            x=x+1;
            if(p<0)
                p=p+twody;
            else
            {
                y=y+1;
                p=p+twodyMinustwodx;
            }
        }
    }
}

```

```

        setPixel(x,y);
    }
}
else
{
//For slope m>1
p=2*dx-dy;
setPixel(x,y);
while(y<yend)
{
    y=y+1;
    if(p<0)
        p=p+twodx;
    else
    {
        x=x+1;
        p=p+twodxMinustwody;
    }
    setPixel(x,y);
}
}
}

void display()
{
    glClearColor(1,1,1,1);          //Specifies a background
    RGB color for a display window.
    glClear(GL_COLOR_BUFFER_BIT);    // Clear display window.
    glViewport(300,300,100,100);
    Bresenhamline();
    glFlush();          // Process all OpenGL routines as quickly
as possible.
}
void main()
{
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE); // Set
display mode.
    glutInitWindowPosition(100,100);             // Set
top-left display-window position.
    glutInitWindowSize(500,500);                 // Set
display-window width and height.
    glutCreateWindow("Bresenham's Line Drawing
Algorithm"); // Create display window.
    myinit();
    // Execute initialization procedure.

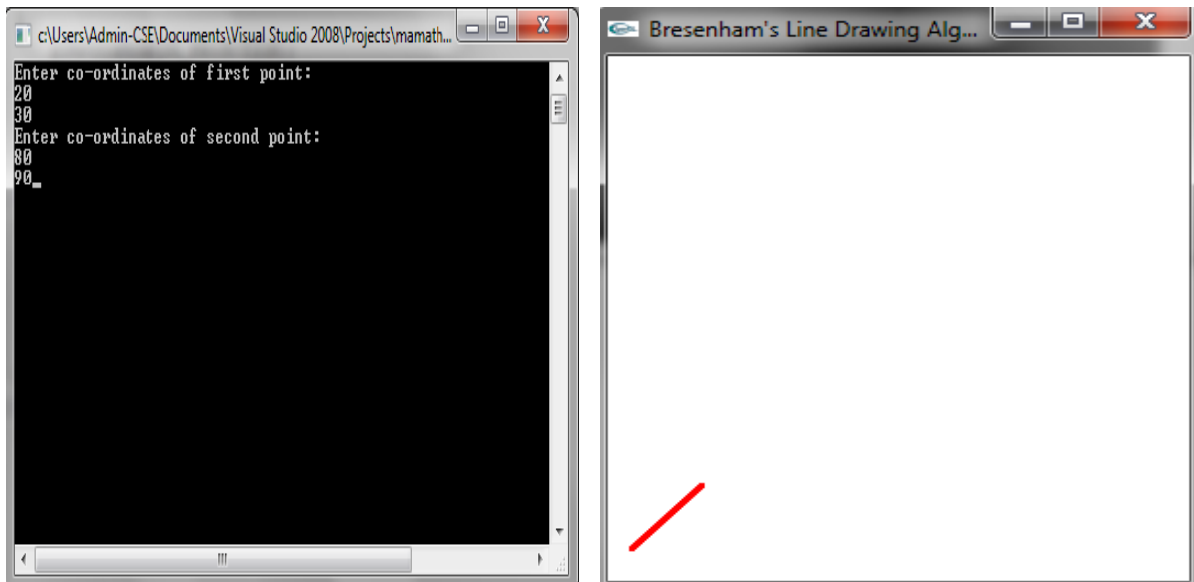
    printf("Enter co-ordinates of first point: ");
    scanf("%d %d", &xx, &yy);

    printf("Enter co-ordinates of second point: ");
    scanf("%d %d", &xend, &yend);
}

```

```
        glutDisplayFunc(display);  
        //Invokes a function to create a picture within the  
        current display window.  
        glutMainLoop();  
        //Executes the computer-graphics program.  
    }
```

1.2 Output



Create and Rotate a Triangle

2. Create and rotate a triangle about the origin and a fixed point

Purpose: To demonstrate 2D geometric transformations.

Procedure: To read angle of rotation and point of rotation from the user to demonstrate the transformation of triangle.

Input: Angle of rotation and point of rotation.

Expected Output: The triangle is drawn and rotated by given angle of rotation about the fixed point given.

2.1 Program

```
//Program 2 Rotate a Triangle
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
#include<math.h>
float p[9][2]={{20,20},{40,40},{60,20}};
float xp,yp,theta,rtheta;

void myinit()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-100,100,-100,100);
    glMatrixMode(GL_MODELVIEW);
}

void drawtriangle()
{
    glBegin(GL_TRIANGLES);
        glVertex2fv(p[0]);
        glVertex2fv(p[1]);
        glVertex2fv(p[2]);

    glEnd();
}

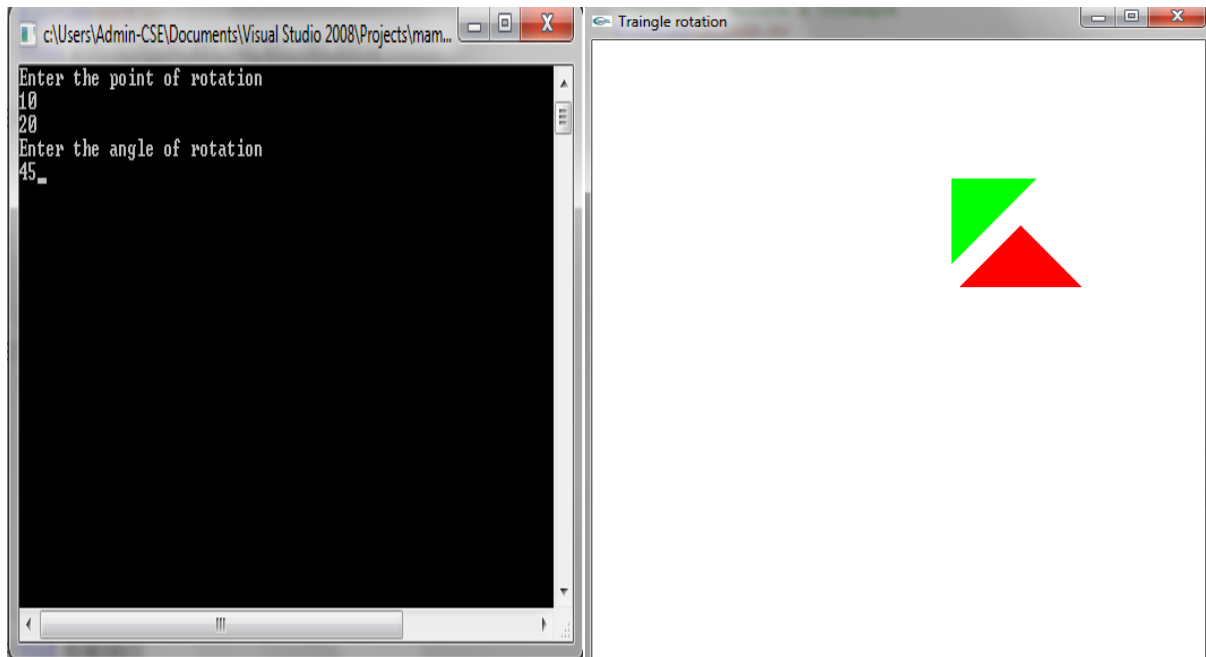
void display()
{
    float x,y;
    glClearColor(1,1,1,1);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1,0,0);
    drawtriangle();
    rtheta=(theta*3.142)/180;
```

```
        for(int i=0;i<3;i++)
        {
            x=p[i][0];
            y=p[i][1];
            p[i][0]= xp + (x-xp)*cos(rtheta) - (y-yp)*sin(rtheta);
            p[i][1]= yp + (x-xp)*sin(rtheta) + (y-yp)*cos(rtheta);
        }
        glColor3f(0,1,0);
        drawtriangle();
        glFlush();

    }

void main()
{
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(500,500);
    glutCreateWindow("Traingle rotation");
    myinit();
    printf("Enter the point of rotation\n");
    scanf("%f%f",&xp,&yp);
    printf("Enter the angle of rotation\n");
    scanf("%f",&theta);
    glutDisplayFunc(display);
    glutMainLoop();
}
```

2.2 Output



COLOR CUBE

3. Draw a color cube and spin it using OpenGL transformation matrices

Purpose: To demonstrate OpenGL transformation matrices.

Procedure: To draw a cube and use rotate transformation function to spin in different axis using mouse clicks.

Input: Mouse click to change the axis of rotation.

Expected Output: Cube is drawn and rotates in axis selected.

3.1 Program

```
//Spin Cube
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
#include<math.h>
float v[8][3]={{-1,-1,1},{-1,1,1},{1,1,1},{1,-1,1},
               {-1,-1,-1},{-1,1,-1},{1,1,-1},{1,-1,-1}};
float p[8][3]={0,0,1},{0,1,1},{1,1,1},{1,0,1},
               0,0,0},{0,1,0},{1,1,0},{1,0,0}};
float theta[3]={0,0,0};
int flag=2;
void myinit()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2,2,-2,2,-2,2);
    glMatrixMode(GL_MODELVIEW);
}

void idlefunc()
{
    theta[flag]++;
    if(theta[flag]>360)theta[flag]=0;
    glutPostRedisplay();
}

void mousefunc(int button,int status,int x,int y)
{
    if(button==GLUT_LEFT_BUTTON&&status==GLUT_DOWN)
        flag=2;
    if(button==GLUT_MIDDLE_BUTTON&&status==GLUT_DOWN)
        flag=1;
    if(button==GLUT_RIGHT_BUTTON&&status==GLUT_DOWN)
        flag=0;
```

```

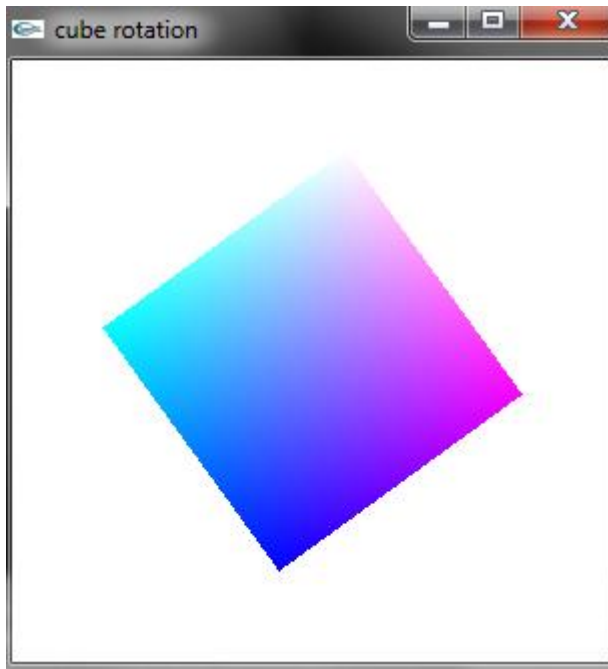
}
void drawpoly(int a,int b,int c,int d)
{
    glBegin(GL_POLYGON);
        glColor3fv(p[a]);
        glVertex3fv(v[a]);
        glColor3fv(p[b]);
        glVertex3fv(v[b]);
        glColor3fv(p[c]);
        glVertex3fv(v[c]);
        glColor3fv(p[d]);
        glVertex3fv(v[d]);
    glEnd();
}
void colorcube()
{
    drawpoly(0,1,2,3);
    drawpoly(0,1,5,4);
    drawpoly(1,5,6,2);
    drawpoly(4,5,6,7);
    drawpoly(3,2,6,7);
    drawpoly(0,4,7,3);
}
void display()
{
    glClearColor(1,1,1,1);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f(1,0,0);
    glEnable(GL_DEPTH_TEST);
    glLoadIdentity();
    glRotatef(theta[0],1,0,0); //x
    glRotatef(theta[1],0,1,0); //y
    glRotatef(theta[2],0,0,1); //z
    colorcube();
    glFlush();
    glutSwapBuffers();
}
void main()
{
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE
| GLUT_DEPTH);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(500,500);
    glutCreateWindow("cube rotation");
    myinit();
}

```



```
    glutDisplayFunc(display);  
    glutMouseFunc(mousefunc);  
    glutIdleFunc(idlefunc);  
    glutMainLoop();  
}
```

3.2 Output



COLOR CUBE WITH PERSPECTIVE VIEWING

4. *Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.*

Purpose: To demonstrate OpenGL transformation matrices with perspective viewing

Procedure: To draw a cube in 3D and use rotate transformation function to spin in different axis using mouse clicks.

Input: Mouse click to change the axis of rotation and press of keys move far/near.

Expected Output: Cube is drawn and rotates in axis selected and press of keys move far/near.

4.1 Program

```
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
#include<math.h>
float v[8][3]={{-1,-1,1},{-1,1,1},{1,1,1},{1,-1,1},{-1,-1,-1},{-1,1,-1},{1,1,-1},{1,-1,-1}};
float
c[8][3]={0,0,1},{0,1,1},{1,1,1},{1,0,1},{0,0,0},{0,1,0},{1,1,0},{1,0,0}};
float theta[3]={0,0,0};
int flag=2;

void myinit()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2,2,-2,2,2,20);
    glMatrixMode(GL_MODELVIEW);
}
int viewer[3]={0,0,2};
void mykey(unsigned char key,int x,int y)
{
    if(key=='x')viewer[0]--;
    if(key=='X')viewer[0]++;
    if(key=='y')viewer[1]--;
    if(key=='Y')viewer[1]++;
    if(key=='z')viewer[2]--;
    if(key=='Z')viewer[2]++;
    glutPostRedisplay();
}
void reshape(int w,int h)
{
    glViewport(0,0,w,h);
    glMatrixMode(GL_PROJECTION);
```

```

        glLoadIdentity();
        if(w<=h)
            glFrustum(2,2,2*(GLfloat)h/(GLfloat)w,
                      2*(GLfloat)h/(GLfloat)w,2,20);
        else
            glFrustum(2*(GLfloat)h/(GLfloat)w,
                      2*(GLfloat)h/(GLfloat)w,-2,2,2,20);
        glMatrixMode(GL_MODELVIEW);
    }
    void mousefunc(int button,int status,int x,int y)
    {
        if(button==GLUT_LEFT_BUTTON&&status==GLUT_DOWN)
            flag=2;
        if(button==GLUT_MIDDLE_BUTTON&&status==GLUT_DOWN)
            flag=1;
        if(button==GLUT_RIGHT_BUTTON&&status==GLUT_DOWN)
            flag=0;
        theta[flag]++;
        if(theta[flag]>360)theta[flag]=0;
        glutPostRedisplay();
    }
    void drawpoly(int a,int b,int c1,int d)
    {
        glBegin(GL_POLYGON);
            glColor3fv(c[a]);
            glVertex3fv(v[a]);
            glColor3fv(c[b]);
            glVertex3fv(v[b]);
            glColor3fv(c[c1]);
            glVertex3fv(v[c1]);
            glColor3fv(c[d]);
            glVertex3fv(v[d]);
        glEnd();
    }
    void colorcube()
    {
        drawpoly(0,1,2,3);
        drawpoly(0,1,5,4);
        drawpoly(1,5,6,2);
        drawpoly(4,5,6,7);
        drawpoly(3,2,6,7);
        drawpoly(0,4,7,3);
    }
    void display()
    {
        glClearColor(1,1,1,1);

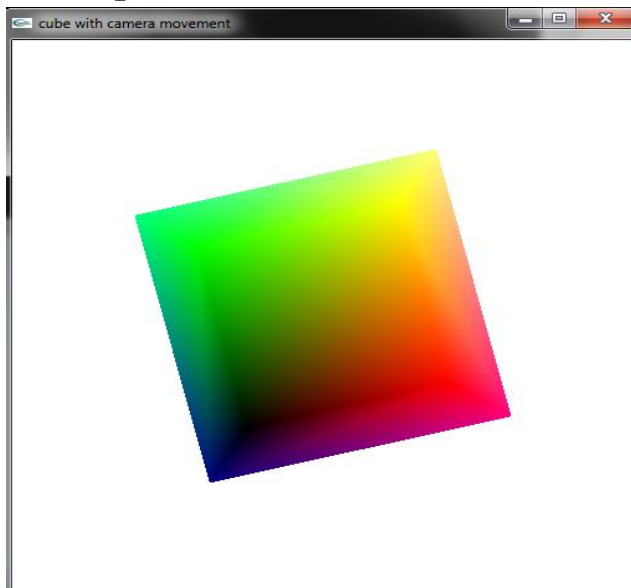
```

```

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f(1,0,0);
    glEnable(GL_DEPTH_TEST);
    glLoadIdentity();
    gluLookAt(viewer[0],viewer[1],viewer[2],0,0,0,0,1,0);
    glRotatef(theta[0],1,0,0);//x
    glRotatef(theta[1],0,1,0);//y
    glRotatef(theta[2],0,0,1);//z
    colorcube();
    glFlush();
    glutSwapBuffers();
}
void main()
{
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE GLUT_DEPTH);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(500,500);
    glutCreateWindow("cube with camera movement");
    myinit();
    glutDisplayFunc(display);
    glutMouseFunc(mousefunc);
    glutKeyboardFunc(mykey);
    glutReshapeFunc(reshape);
    glutMainLoop();
}

```

4.2 Output



COHEN SUTHERLAND LINE CLIPPING ALGORITHM

5. *Clip a lines using Cohen-Sutherland algorithm.*

Purpose: To demonstrate Line clipping using Cohen-Sutherland algorithm.

Procedure: To draw lines and clip the part of the lines which are outside the clipping window.

Input: (Xmin,Ymin) and (Xmax,Ymax) of the clipping window and display window. End points of the line.

Expected Output: Clipping window with actual line is drawn and display window with clipped line is drawn.

5.1 Program

```
//Cohen Sutherland Line clipping Algorithm
#include<stdio.h>
#include<GL/glut.h>
#define opcode int
enum {top=0x8,bottom=0x4,left=0x1,right=0x2};
float xmin,xmax,ymin,ymax,umin,umax,vmin,vmax;
float x1,y1,x2,y2;

void myinit()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0,500,0,500);
    glMatrixMode(GL_MODELVIEW);
}

opcode compopcode(float x,float y)
{
    int p=0;
    if(x<xmin)p=p|left;
    if(x>xmax)p=p|right;
    if(y<ymin)p=p|bottom;
    if(y>ymax)p=p|top;
    return p;
}

void cohensutherland()
{
    bool accept=false,done=false;
    int x=0,y=0;
    opcode p1,p2,p;
    p1=compopcode(x1,y1);
    p2=compopcode(x2,y2);
```

```

float m=(y2-y1)/(x2-x1);
do{
    if(!(p1|p2))
    {
        accept=true;
        done=true;
    }
    else if(p1&p2)
        done=true;
    else
    {
        p=p1?p1:p2;

        if(p&bottom)
        {
            y=ymin;
            x=x1+(ymin-y1)/m;
        }

        if(p&top)
        {
            y=ymax;
            x=x1+(ymax-y1)/m;
        }

        if(p&right)
        {
            x=xmax;
            y=y1+(xmax-x1)*m;
        }

        if(p&left)
        {
            x=xmin;
            y=y1+(xmin-x1)*m;
        }

        if(p==p2)
        {
            x2=x;
            y2=y;
            p2=comppopcode(x2,y2);
        }

        if(p==p1)

```

```

        {
            x1=x;
            y1=y;
            p1=compopcode(x1,y1);
        }
    }
}while(!done);

    if(accept)
    {
        float sx=(umax-umin)/(xmax-xmin);
        float sy=(vmax-vmin)/(ymax-ymin);
        x1=sx*x1+umin-sx*xmin;
        y1=sy*y1+vmin-sy*ymin;
        x2=sx*x2+umin-sx*xmin;
        y2=sy*y2+vmin-sy*ymin;
    }

    glBegin(GL_LINES);
        glVertex2f(x1,y1);
        glVertex2f(x2,y2);
    glEnd();
    glFlush();
}

void display()
{
    glClearColor(1,1,1,1);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1,0,0);
    glBegin(GL_LINE_LOOP);
        glVertex2f(xmin,ymin);
        glVertex2f(xmin,ymax);
        glVertex2f(xmax,ymax);
        glVertex2f(xmax,ymin);
    glEnd();

    glColor3f(0,0,1);
    glBegin(GL_LINES);
        glVertex2f(x1,y1);
        glVertex2f(x2,y2);
    glEnd();

    glColor3f(0,1,0);
    glBegin(GL_LINE_LOOP);
        glVertex2f(umin,vmin);

```

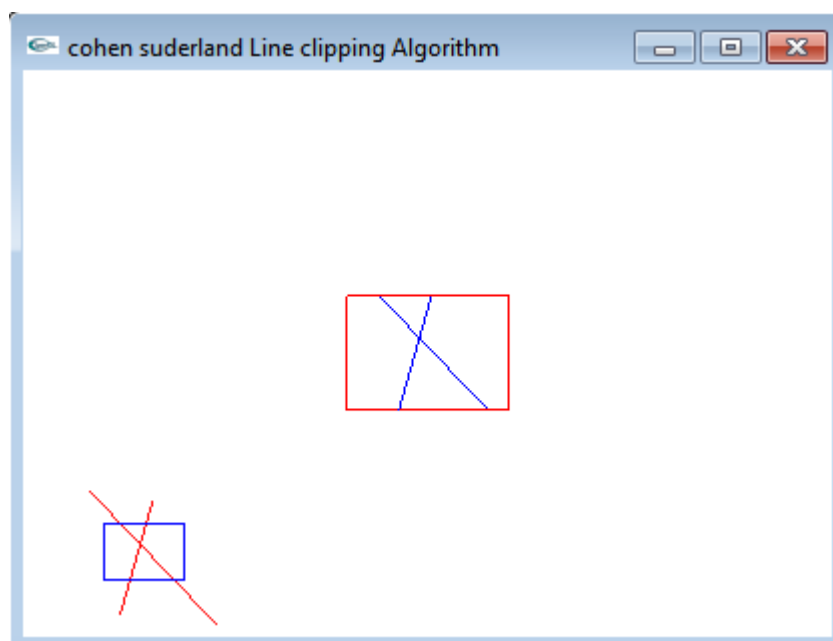
```

        glVertex2f(umin,vmax);
        glVertex2f(umax,vmax);
        glVertex2f(umax,vmin);
    glEnd();
    cohensutherland();
    glFlush();
}

void main()
{
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(500,500);
    glutCreateWindow("CLIPPING ALGORITHM");
    myinit();
    printf("Enter the clipping window points
xmin,xmax,ymin,ymax");
    scanf("%f%f%f%f",&xmin,&xmax,&ymin,&ymax);
    printf("Enter the display window points
umin,umax,vmin,vmax");
    scanf("%f%f%f%f",&umin,&umax,&vmin,&vmax);
    printf("\nEnter the first point x1,y1\n");
    scanf("%f%f",&x1,&y1);
    printf("\nEnter the second point x2,y2\n");
    scanf("%f%f",&x2,&y2);
    glutDisplayFunc(display);
    glutMainLoop();
}

```

5.2 Output



TEAPOT

- 6. To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene.**

Purpose: To demonstrate properties of the light source along with the properties of the surfaces of the solid object used in the 3D scene.

Procedure: To draw teapot using OpenGL function and apply properties of the light source along with the properties of the surfaces of the solid object used in the scene

Expected Output: A simple shaded scene consisting of a tea pot on a table is drawn.

6.1 Program

```
//Teapot
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<GL/glut.h>

void myinit()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-100,200,-100,200,-200,200);
    glMatrixMode(GL_MODELVIEW);
}

void drawtable()
{
    //table top
    glPushMatrix();
    glColor3f(0,1,0);
    glTranslatef(50,40,-50);
    glScalef(50,5,50);
    glutSolidCube(1);
    glPopMatrix();

    //1 st leg
    glPushMatrix();
    glColor3f(1,1,0);
    glTranslatef(30,20,-30);
    glScalef(5,35,5);
    glutSolidCube(1);
    glPopMatrix();

    //2nd leg
```

```
glPushMatrix();
glColor3f(1,1,0);
glTranslatef(70,20,-30);
glScalef(5,35,5);
glutSolidCube(1);
glPopMatrix();

// 3rd leg
glPushMatrix();
glColor3f(1,1,0);
glTranslatef(30,20,-70);
glScalef(5,35,5);
glutSolidCube(1);
glPopMatrix();

//4 th leg
glPushMatrix();
glColor3f(1,1,0);
glTranslatef(70,20,-70);
glScalef(5,35,5);
glutSolidCube(1);
glPopMatrix();

//floor
glPushMatrix();
glColor3f(1,0,1);
glTranslatef(50,0,-50);
glScalef(100,5,100);
glutSolidCube(1);
glPopMatrix();

//left wall
glPushMatrix();
glColor3f(1,0,0);
glRotatef(90,0,0,1);
glTranslatef(50,0,-50);
glScalef(100,5,100);
glutSolidCube(1);
glPopMatrix();

//backside wall
glPushMatrix();
glColor3f(1,0,0);
glTranslatef(50,50,-100);
glScalef(100,100,5);
glutSolidCube(1);
```

```

        glPopMatrix();

        //tea pot
        glPushMatrix();
        glTranslatef(50,50,-50);
        glRotatef(30,0,1,0);
        glutSolidTeapot(10);
        glPopMatrix();
    }
    void display()
    {
        glClearColor(0,0,0,1);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        GLfloat mat_ambient[]={.7,.7,.7,1};
        GLfloat mat_diffuse[]={.5,.5,.5,1};
        GLfloat mat_spec[]={1,1,1,1};
        GLfloat mat_shininess[]={50};

        glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat_spec);
        glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

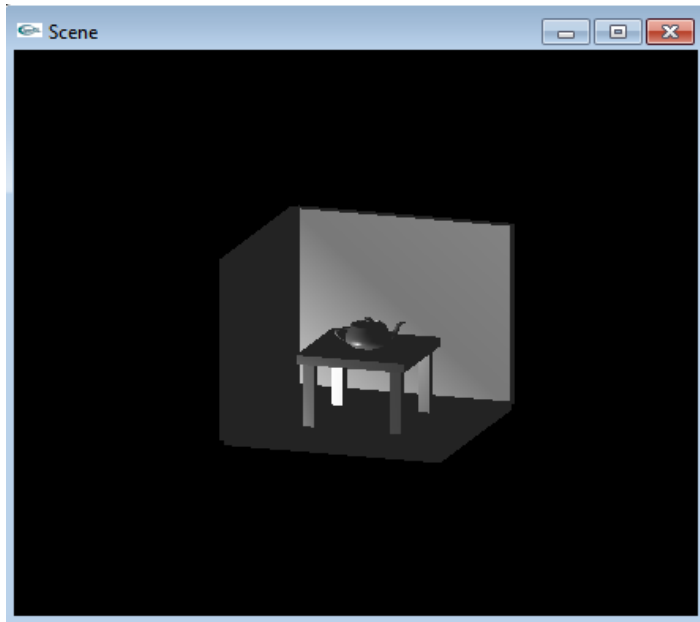
        GLfloat light_int[]={.7,.7,.7,1};
        GLfloat lightpos[]={100,100,100};
        glLightfv(GL_LIGHT0, GL_POSITION, lightpos);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, light_int);

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt(25,25,50,0,0,-25,0,1,0);
        glMatrixMode(GL_MODELVIEW);
        drawtable();
        glFlush();
    }
    void main()
    {
        glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(100,100);
        glutCreateWindow("Scene");
        myinit();
        glutDisplayFunc(display);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0 );
    }

```

```
glShadeModel(GL_SMOOTH);  
glEnable(GL_NORMALIZE);  
glEnable(GL_DEPTH_TEST);  
glutMainLoop();  
}
```

6.2 Output



SIERPINSKI GASKET

7. To Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user.

Purpose: To demonstrate recursively subdivide a tetrahedron to form 3D sierpinski gasket.

Procedure: To draw a tetrahedron to form 3D sierpinski gasket.

Input: n number of recursive steps to subdivide a tetrahedron to form 3D sierpinski gasket

Expected Output: A 3D sierpinski gasket is drawn by recursively subdividing a tetrahedron.

7.1 Program

```
//3D sierpinski gasket
#include<stdio.h>
#include<gl/glut.h>
int c;
float v[4][3] = {{0,0,1},{-1,-0.5,0},{0,1,0},{1,-0.5,0}};
void myinit()
{
    glMatrixMode(GL_PROJECTION_MATRIX);
    glLoadIdentity();
    glOrtho(-2,2,-2,2,-2,2);
    glMatrixMode(GL_MODELVIEW);
}

void disp_tri(float a[3],float b[3],float c[3])
{
    glBegin(GL_POLYGON);
        glVertex3fv(a);
        glVertex3fv(b);
        glVertex3fv(c);
    glEnd();
}

void div_tri(float *a,float *b,float *c,int n)
{
    float v1[3],v2[3],v3[3];
    int i;
    if(n>0)
    {
        for(i=0;i<3;i++)
        {
            v1[i] = (a[i] + b[i])/2;
            v2[i] = (a[i] + c[i])/2;
            v3[i] = (b[i] + c[i])/2;
        }
    }
}
```

```

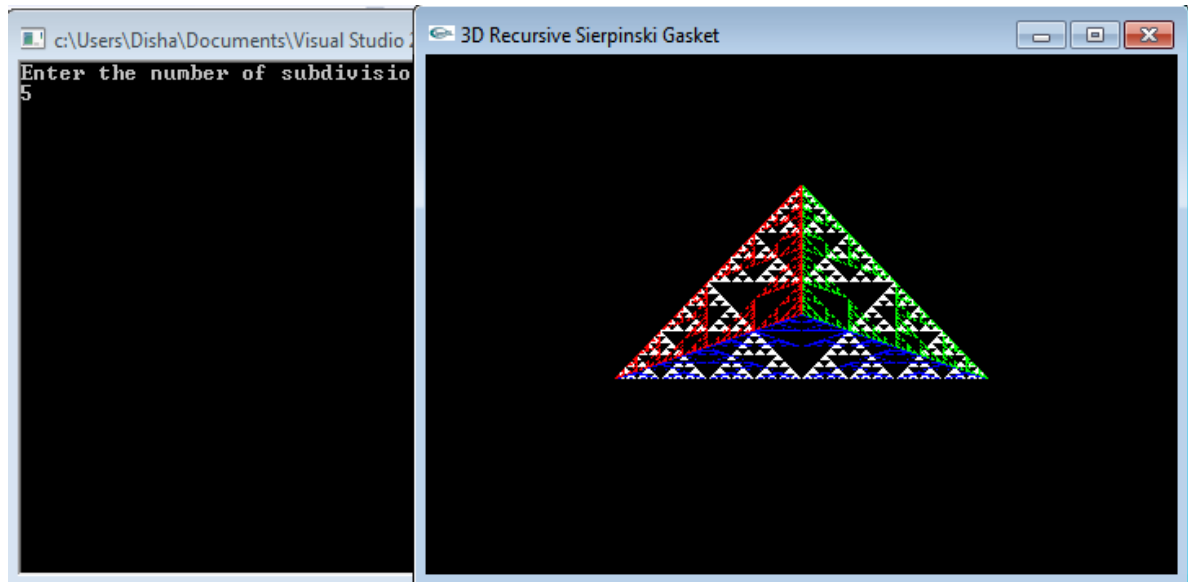
        div_tri(a,v1,v2,n-1);
        div_tri(v1,b,v3,n-1);
        div_tri(v2,v3,c,n-1);
    }
    else
        disp_tri(a,b,c);
}

void display()
{
    glClearColor(0,0,0,1);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1,1,1);
    div_tri(v[1],v[2],v[3],c);
    glColor3f(1,0,0);
    div_tri(v[0],v[1],v[2],c);
    glColor3f(0,1,0);
    div_tri(v[0],v[2],v[3],c);
    glColor3f(0,0,1);
    div_tri(v[0],v[1],v[3],c);
    glFlush();
}

void main()
{
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600,600);
    glutInitWindowPosition(300,150);
    printf("Enter the number of subdivisions:\n");
    scanf("%d",&c);
    glutCreateWindow("3D Recursive Sierpinski Gasket");
    myinit();
    glutDisplayFunc(display);
    glutMainLoop();
}

```

7.2 Output



BEZIER CURVE

8. Develop a menu driven program to animate a flag using Bezier Curve algorithm.

Purpose: To demonstrate Bezier Curve algorithm by animating a flag.

Procedure: To draw a Bezier Curve .

Input: Color is selected from the menu options

Expected Output: A flag is drawn in given color using Bezier Curve algorithm.

8.1 Program

```
// Bezier Curve
#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
#define PI 3.1416
GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = 0.0, xwcMax = 130.0;
GLfloat ywcMin = 0.0, ywcMax = 130.0;

typedef struct wcPt3D
{
    GLfloat x, y, z;
};

void bino(GLint n, GLint *C)
{
    GLint k, j;
    for(k=0; k<=n; k++)
    {
        C[k]=1;
        for(j=n; j>=k+1; j--)
            C[k]*=j;
        for(j=n-k; j>=2; j--)
            C[k]/=j;
    }
}

void computeBezPt(GLfloat u, wcPt3D *bezPt, GLint nCtrlPts,
wcPt3D *ctrlPts, GLint *C)
{
    GLint k, n=nCtrlPts-1;
    GLfloat bezBlendFcn;
    bezPt ->x =bezPt ->y = bezPt->z=0.0;
    for(k=0; k< nCtrlPts; k++)
```



```

        {
            bezBlendFcn = C[k] * pow(u, k) * pow( 1-u, n-k);
            bezPt ->x += ctrlPts[k].x * bezBlendFcn;
            bezPt ->y += ctrlPts[k].y * bezBlendFcn;
            bezPt ->z += ctrlPts[k].z * bezBlendFcn;
        }
    }

void bezier(wcPt3D *ctrlPts, GLint nCtrlPts, GLint
nBezCurvePts)
{
    wcPt3D bezCurvePt;
    GLfloat u;
    GLint *C, k;
    C= new GLint[nCtrlPts];
    bino(nCtrlPts-1, C);
    glBegin(GL_LINE_STRIP);
    for(k=0; k<=nBezCurvePts; k++)
    {
        u=GLfloat(k)/GLfloat(nBezCurvePts);
        computeBezPt(u, &bezCurvePt, nCtrlPts, ctrlPts, C);
        glVertex2f(bezCurvePt.x, bezCurvePt.y);
    }
    glEnd();
    delete[]C;
}

void displayFcn()
{
    GLint nCtrlPts = 4, nBezCurvePts =20;
    static float theta = 0;
    wcPt3D ctrlPts[4] = {
        {20, 100, 0},
        {30, 110, 0},
        {50, 90, 0},
        {60, 100, 0}};
    ctrlPts[1].x +=10*sin(theta * PI/180.0);
    ctrlPts[1].y +=5*sin(theta * PI/180.0);
    ctrlPts[2].x -= 10*sin((theta+30) * PI/180.0);
    ctrlPts[2].y -= 10*sin((theta+30) * PI/180.0);
    ctrlPts[3].x-= 4*sin((theta) * PI/180.0);
    ctrlPts[3].y += sin((theta-30) * PI/180.0);
    theta+=0.1;
    glClear(GL_COLOR_BUFFER_BIT);
    glPointSize(5);
    glPushMatrix();

```

```

        glLineWidth(5);

        for(int i=0;i<24;i++)
        {
            glTranslatef(0, -0.8, 0);
            bezier(ctrlPts, nCtrlPts, nBezCurvePts);
        }

        glPopMatrix();
        glLineWidth(5);
        glBegin(GL_LINES);
        glVertex2f(20,100);
        glVertex2f(20,40);
        glEnd();
        glFlush();
        glutPostRedisplay();
        glutSwapBuffers();
    }

void winReshapeFun(GLint newWidth, GLint newHeight)
{
    glViewport(0, 0, newWidth, newHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);
    glClear(GL_COLOR_BUFFER_BIT);
}

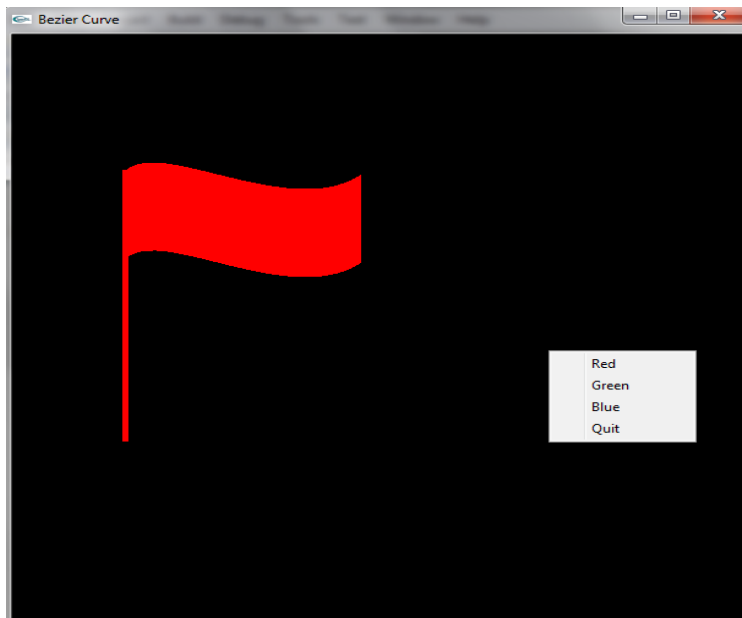
void d_menu(int op)
{
    if(op==1)
        glColor3f(1.0,0.0,0.0);
    else if(op==2)
        glColor3f(0.0,1.0,0.0);
    else if(op==3)
        glColor3f(0.0,0.0,1.0);
    else if(op==4)
        exit(0);
    glutPostRedisplay();
}

void main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);

```

```
glutInitWindowPosition(50, 50);  
glutInitWindowSize(winWidth, winHeight);  
glutCreateWindow("Bezier Curve");  
  
glutCreateMenu(d_menu);  
glutAddMenuEntry("Red",1);  
glutAddMenuEntry("Green",2);  
glutAddMenuEntry("Blue",3);  
glutAddMenuEntry("Quit",4);  
glutAttachMenu(GLUT_RIGHT_BUTTON);  
glutDisplayFunc(displayFcn);  
glutReshapeFunc(winReshapeFun);  
glutMainLoop();  
}
```

8.2 Output



SCAN LINE ALGORITHM

9. Develop a menu driven program to fill the polygon using scan line algorithm.

Purpose: To demonstrate scan line algorithm.

Procedure: To draw a filled polygon using *scan line algorithm*.

Input: To give color of choice using menu option to fill the polygon

Expected Output: Polygon is drawn with color beginning filled.

9.1 Program

```
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
float x1,x2,x3,x4,y1,y2,y3,y4;
void d_menu(int op)
{
    if(op==1)
        glColor3f(1.0,0.0,0.0);
    else if(op==2)
        glColor3f(0.0,1.0,0.0);
    else if(op==3)
        glColor3f(0.0,0.0,1.0);
    else if(op==4)
        exit(0);
    glutPostRedisplay();
}
void edgedetect(float x1,float y1,float x2,float y2,int *le,
int *re)
{
    float mx,x,temp;
    int i;
    if((y2-y1)<0)
    {
        temp=y1;y1=y2;y2=temp;
        temp=x1;x1=x2;x2=temp;
    }
    if((y2-y1)!=0)
        mx=(x2-x1)/(y2-y1);
    else
        mx=x2-x1;
        x=x1;
        for(i=y1;i<=y2;i++)
        {
            if(x<(float)le[i])
                le[i]=(int)x;
```

```

        if(x>(float)re[i])
            re[i]=(int)x;
        x+=mx;
    }
}

void draw_pixel(int x,int y)
{
    glBegin(GL_POINTS);
    glVertex2i(x,y);
    glEnd();
    glFlush();
}

void scanfill(float x1,float y1,float x2,float y2,float
x3,float y3,float x4,float y4)
{
    int le[500],re[500];
    int i,y;
    for(i=0;i<500;i++)
    {
        le[i]=500;
        re[i]=0;
    }
    edgedetect(x1,y1,x2,y2,le,re);
    edgedetect(x2,y2,x3,y3,le,re);
    edgedetect(x3,y3,x4,y4,le,re);
    edgedetect(x4,y4,x1,y1,le,re);
    for(y=0;y<500;y++)
    {
        if(le[y]<=re[y])
            for(i=(int)le[y];i<(int)re[y];i++)
                draw_pixel(i,y);
    }
}

void display()
{
    x1=200.0;y1=200.0;x2=100.0;y2=300.0;
    x3=200.0;y3=400.0;x4=300.0;y4=300.0;
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_LINE_LOOP);
    glVertex2f(x1,y1);
    glVertex2f(x2,y2);
    glVertex2f(x3,y3);

```

```

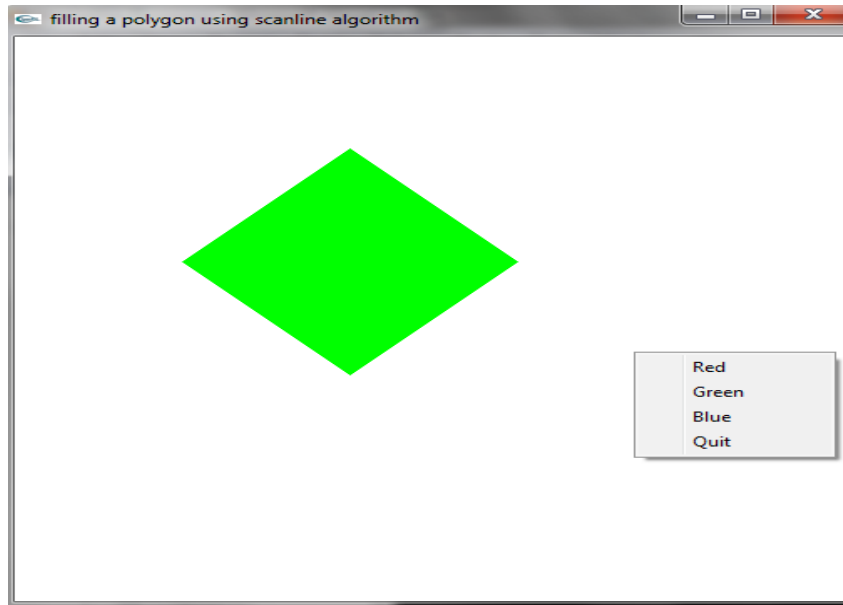
        glVertex2f(x4,y4);
    glEnd();
    scanfill(x1,y1,x2,y2,x3,y3,x4,y4);
    glFlush();
}

void myinit()
{
    glClearColor(1.0,1.0,1.0,1.0);
    glColor3f(1.0,0.0,1.0);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0,499.0,0.0,499.0);
}

void main(int argc,char **argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Filling a polygon using scanline
algorithm");
    glutCreateMenu(d_menu);
    glutAddMenuEntry("Red",1);
    glutAddMenuEntry("Green",2);
    glutAddMenuEntry("Blue",3);
    glutAddMenuEntry("Quit",4);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    myinit();
    glutDisplayFunc(display);
    glutMainLoop();
}

```

9.2 Output



Appendix-A

This section lists the viva questions from each module.

Module 1- Overview: Computer Graphics and OpenGL

1. Define Computer graphics?
2. What is scan conversion?
3. What is rasterization?
4. Name any four input and output devices?
5. Write the two techniques for producing color displays with a CRT?
6. What is vertical and horizontal retrace of the electron beam?
7. Describe video controller?
8. Define Random And Raster Scan Displays?
9. What is persistence?
10. What is Aspect ratio?
11. Define pixel?
12. What is frame buffer?
13. What is resolution?
14. Write the difference between vector and raster graphics?
15. What are the advantage and disadvantages of DDA Algorithm?

Module 2 - Fill area Primitives and 2D Geometric Transformations and 2D viewing

16. List the geometric primitives.
17. Which function is use to display geometric primitives?
18. What are the various attributes of a point and line?
19. Explain the following OpenGL functions:


```
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(300,150);
glutCreateWindow("Points");
glutDisplayFunc(displ);
glutMainLoop();
```
20. Why is it necessary to have glutMainLoop() function?
21. Why is glOrtho2D() function used? What are its parameters?
22. Explain the following OpenGL functions:


```
glClearColor(1,1,1,1);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1,0,0);
glVertex2f(25,25);
```
23. Explain the following OpenGL functions:


```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0,100,0,100);
glMatrixMode(GL_MODELVIEW);
```

Module 3 - Clipping, 3D Geometric Transformations, Color and Illumination Models

24. What is Transformation?
25. What is Translation? Which is the OpenGL function used for translation?
26. What is rotation? Which is the OpenGL function used for rotation?
27. What is scaling? Which is the OpenGL function used for scaling?
28. Distinguish between uniform scaling and differential scaling?
29. What is reflection?
30. What is shearing?
31. Define clipping?
32. What are the types of clipping?
33. Distinguish between window & view port?
34. What is fixed point scaling?
35. How to create and execute display list?
36. What is the need of homogeneous coordinates?

Module 4 - 3D Viewing and Visible Surface Detection

37. Define projection?
38. What is viewport?
39. Define orthogonal projection?
40. What you mean by parallel projection?
41. What do you mean by Perspective projection?
42. What is Projection reference point?
43. What is anti aliasing?
44. What are viewing parameters.
45. Explain Depth buffer algorithm.
46. Explain the following OpenGL visibility detection Functions
 - i). glCullFace() ii). glClearDepth()
 - iii). glDepthRange() iv). glFogi()
 - v). glDepthFunc()
47. Explain the following OpenGL functions
 - a). gluPrespective() b). glFrustrum()
 - c). glLookAt() d). GLViewport()

Module 5 - Input and interaction, Curves and Computer Animation

48. How to create and execute menus.
49. What is double buffer?
50. What are the callback functions?
51. Explain the following OpenGL functions:
 - `glutPostRedisplay();`
 - `glutMouseFunc(mouse);`
 - `glutIdleFunc(idle);`
 - `glutKeyboardFunc(keyboard);`
 - `glutReshapeFunc(reshape);`
52. What are the properties of Bézier Curves?
53. List the input devices.
54. What are different input modes?