Table of Contents

INTRODUCTION

What Is OpenGL?

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. With OpenGL, you must build up your desired model from a small set of *geometric primitives* - points, lines, and polygons. 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.

Construct shapes from geometric primitives, thereby creating mathematical descriptions of objects. (OpenGL considers points, lines, polygons, images, and bitmaps to be primitives.)

OpenGL programs can work across a network even if the client and server are different kinds of computers. If an OpenGL program isn't running across a network, then there's only one computer, and it is both the client and the server.

Advantages of OpenGL

1. Uniform approach to writing graphics applications.
2. Some code can be compiled and run on a variety of graphics environments.
3. OpenGL is portable
4. Cross platform (Windows, Linux, Mac, even some hand held devices).

Software's required:

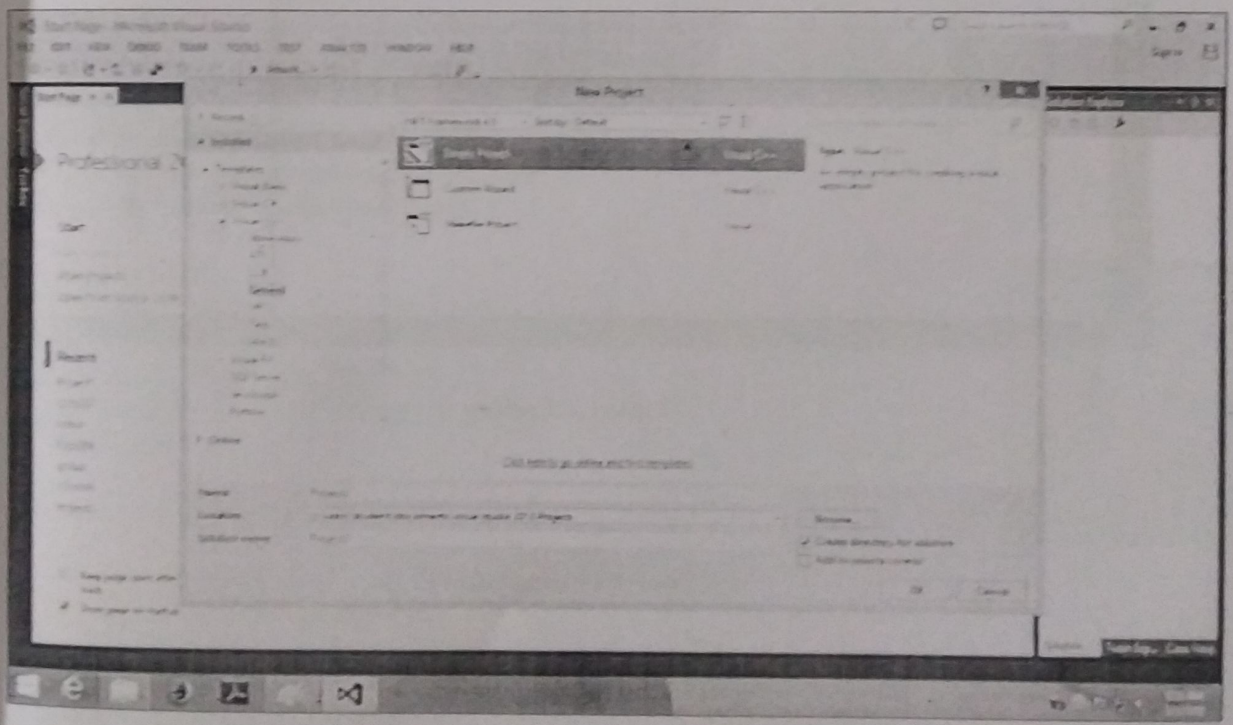
1. Visual Studio Express edition
2. The important files and libraries for running the OpenGL application are:
    - gl.h glu.h opengl32.libglu32.lib

PROJECT PROCEDURE

PROCEDURE TO START A NEW PROJECT

1. Click *start*
2. Choose All Programs
3. Select Microsoft Visual Studio 2013 Professional edition
4. Choose File → New → Project. You get the following Dialogbox

6. From Project type choose General and select Empty Project template. Give suitable project name and click ok button.
7. From the Project menu Choose Add new item. You get the following Dialogbox.
8. From categories code, choose c++ File template. Give suitable name and then clickAdd. You get the following Screen.
9. To compile the Program Use Function keyF7
10. To execute the programF5



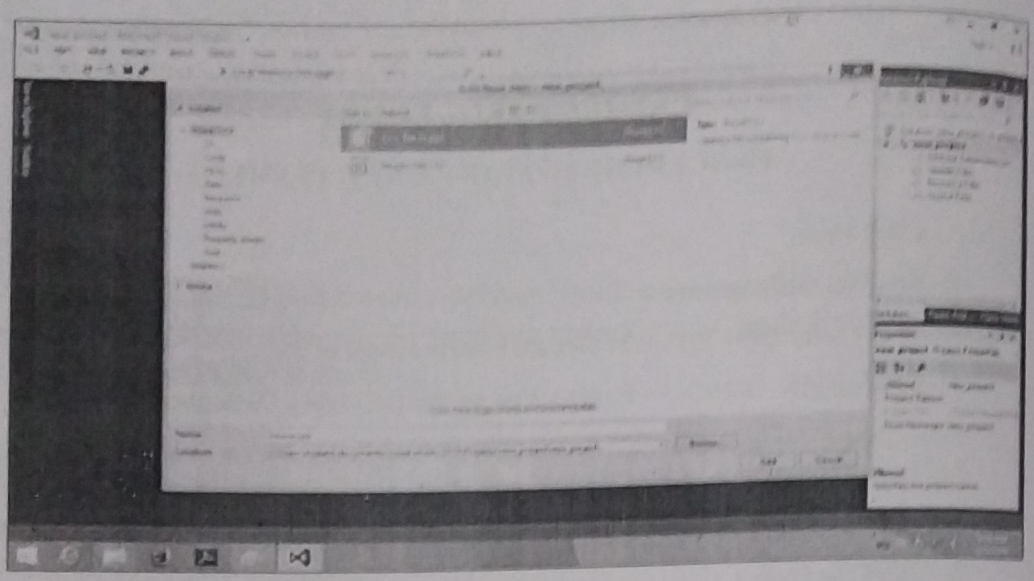


Figure 1. Add New Item

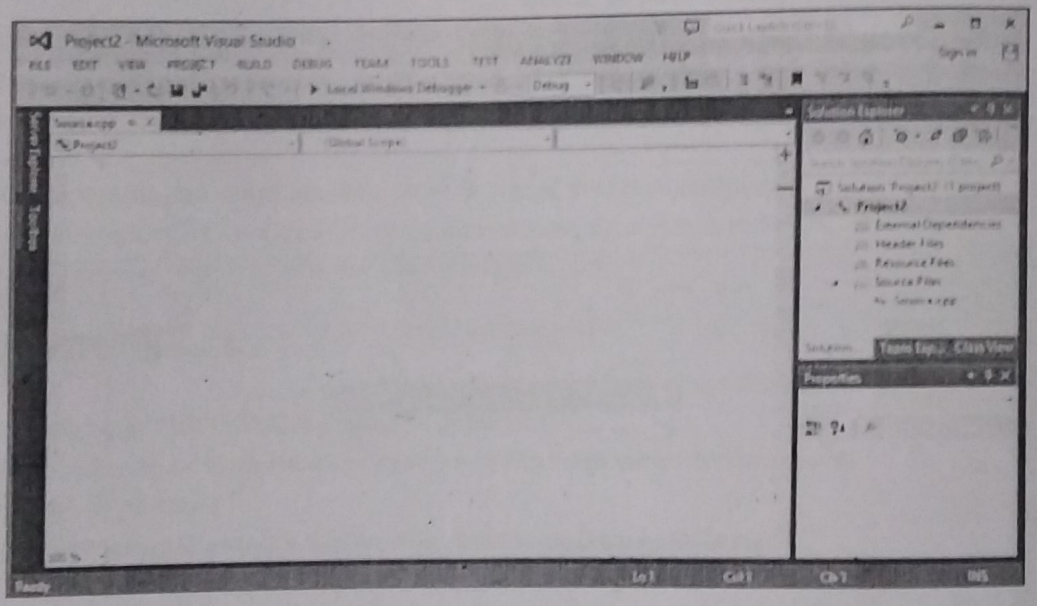


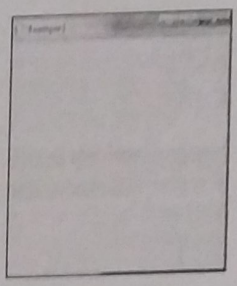
Figure 2. Code sp

SAMPLE PROGRAMS

1. Creating a Window

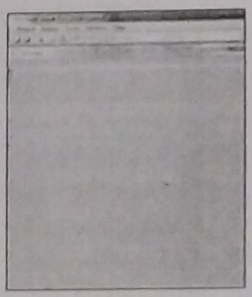
> *Note: By default, the window has a size of (300, 300) pixels, and its position is up to the window manager to choose.*

```
#include<glut.h>
void display () /* callback function which is called when OpenGL needs to update the display */
{
glClearColor (1.0,1.0,0.0,0.0);     /*defaultcolor–black..... Now set to YELLOW*/
glClear (GL_COLOR_BUFFER_BIT);          /*Clear the window-set the color of  pixels in
buffer*/
glFlush();  /* Force update of screen */
}

void main (int argc, char **argv)
{
glutInit (&argc, argv);                    /* Initialise OpenGL
*/
glutCreateWindow ("Example1");          /* Create the window */
glutDisplayFunc (display);          /* Register the "display" function
*/ glutMainLoop ();                 /* Enter the OpenGL main loop */
}
```



Make necessary change in program to get the following output

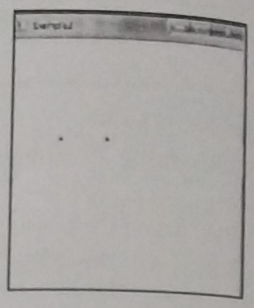
Change the window size to 500, 500
Change the window position to 100, 100
Change the window color to CYAN



```
#include<glut.h>
void display (void)
{                                   }
    glClearColor (0.0,1.0,1.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}
void main (int argc, char **argv)
{
    glutInit (&argc, argv);          /* Initialise OpenGL */
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow ("Activity1");          /* Create the window */
    glutDisplayFunc (display);               /* Register the "display" function */
    glutMainLoop ();                         /* Enter the OpenGL main loop *
```

2. Drawing pixels/points

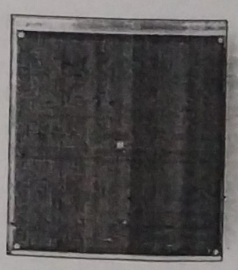
```
#include<glut.h>
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_POINTS);
      glVertex2i(100,300);
      glVertex2i(201,300);
glEnd();
glFlush();
}

void myinit()
{
      glClearColor(1.0,1.0,1.0,1.0);          // set the window color to white
      glColor3f(1.0,0.0,0.0);                 // set the point color to red (RGB)
      glPointSize(5.0);                       // set the pixel size
      gluOrtho2D(0.0,500.0,0.0,500.0);        // coordinates to be used with the
                                              //viewport(left,right,bottom,top)
}
void main(int argc, char** argv)
{
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB); // sets the initial display mode, GLUT single-default
      glutInitWindowSize(300,300);
      glutInitWindowPosition(0,0);
      glutCreateWindow("Example2");
      glutDisplayFunc(display);
      myinit();
      glutMainLoop();
}
```



Make necessary change in program to get the following output

Change the window color to BLUE
Change the point color to CYAN
Change the point width to 10
Draw FIVE points: at 4 corners of the window and one more at the Centre of the window.

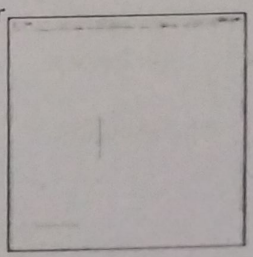
```
#include<GL/glut.h>
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POINTS);
          glVertex2i(10,10);
```



```
        glVertex2i(250,250);
        glVertex2i(10,490);
        glVertex2i(490,490);
        glVertex2i(490,10);
    glEnd();
    glFlush();
}
voidmyinit()
{
    glClearColor(0.0,0.0,1.0,0.0);      // set the window color to blue
    glColor3f(0.0,1.0,1.0);             // set the point color to cyan (RGB)
    glPointSize(10.0);
    gluOrtho2D(0.0,500.0,0.0,500.0);    // coordinates to be used with the viewport left, right,
                                           bottom, top)
}
void main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Activity2");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

3. Drawing lines

```
#include<glut.h>
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,0.0,0.0);         //draw the line with red color
    glLineWidth(3.0);               // Thickness of line
    glBegin(GL_LINES);
        glVertex2d(50,50);          // to draw horizontal line in red color
        glVertex2d(150,50);
        glColor3f(0.0,0.0,1.0);     //draw the line with blue color
        glVertex2d(200,200);        // to draw vertical line in blue color
        glVertex2d(200,300);
    glEnd();
    glFlush();
}
```



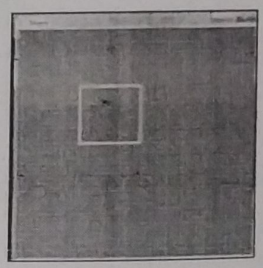
```
void myinit()
{
    glClearColor(1.0,1.0,1.0,1.0);
    glColor3f(1.0,0.0,0.0);
    glPointSize(1.0);
    gluOrtho2D(0.0,500.0,0.0,500.0);
}
void main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("LINE");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

Make necessary change in program to get the following output

Change the window color to MAGENTA

Change the line color to YELLOW

Change the line width to width to 4 Draw a square using 4lines

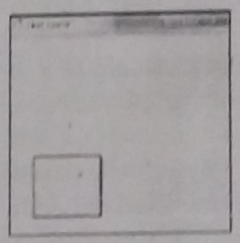


```
#include<glut.h>
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,1.0,0.0);
    glLineWidth(4.0);
    glBegin(GL_LINES);
            glVertex2d (50, 100);
            glVertex2d (100,100);
            glVertex2d (100,100);
            glVertex2d (100,150);
            glVertex2d (100,150);
            glVertex2d (50,150);
            glVertex2d (50,150);
            glVertex2d (50,100);
    glEnd();
    glFlush();
}
void myinit()
{
```

```
    glClearColor(1.0,0.0,1.0,1.0);
    gluOrtho2D(0.0,200.0,0.0,200.0);
}
void main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(10,100);
    glutCreateWindow("Square");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

4. Drawing a square usingLINE_LOOP

```
#include<GL/glut.h>
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 1.0);
    glLineWidth(3.0);
    glBegin(GL_LINE_LOOP);          // If you put GL_LINE_LOOP, it is only boundary.
        glVertex2f(50, 50);
        glVertex2f(200, 50);
        glVertex2f(200, 200);
        glVertex2f(50, 200);
    glEnd();
    glFlush();
}
void myinit()
{
    glClearColor(1.0,1.0,0.0,1.0);
    gluOrtho2D(0.0,499.0,0.0,499.0);
}

void main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(300,300);
    glutInitWindowPosition(0,0);
    glutCreateWindow("LINE LOOP");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

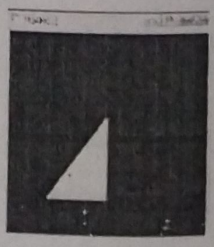


Make necessary change in program to get the following output

Change the window color to PURPLE
Change the line color to GREEN
Change the line width to width to 3
Draw a square using GL_POLYGON

```
#include<glut.h>
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0, 1.0, 0.0);              // set line color to green
        glLineWidth(3.0);
        glBegin(GL_POLYGON);
                glVertex2f(50, 50);
                glVertex2f(200, 50);
                glVertex2f(200, 200);
                glVertex2f(50, 200);
        glEnd();
        glFlush();
}
void myinit()
{
        glClearColor(0.5,0.0,0.5,1.0);         // set window color to purple
        gluOrtho2D(0.0,499.0,0.0,499.0);
}
void main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(300,300);
        glutInitWindowPosition(0,0);
        glutCreateWindow("POLYGON");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

5. Drawing a right angled triangle using GL_TRINGLES

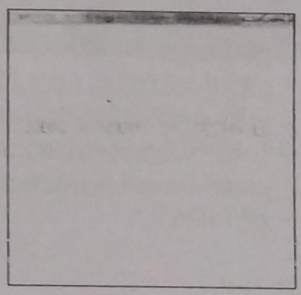


```
#include<glut.h>
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0, 1.0, 0.0);
        glLineWidth(3.0);
        glBegin(GL_TRIANGLES);
                glVertex2i(100,100);
```

```
            glVertex2i(250,100);
            glVertex2i(250,300);
        glEnd();
        glFlush();
}

void myinit()
{
        glClearColor(0.0,0.0,0.0,0.0);
        gluOrtho2D(0.0,499.0,0.0,499.0);
}
void main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(300,300);
        glutInitWindowPosition(0,0);
        glutCreateWindow("TRIANGLE");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

6. Writing Text

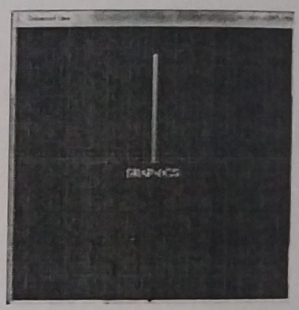
```
#include<glut.h>
void output(GLfloat x,GLfloat y,char *text)
{
        char*p;
        glPushMatrix();
        glTranslatef(x,y,0);
        glScaled(0.2,0.2,0);
        for(p=text;*p;p++)
        glutStrokeCharacter(GLUT_STROKE_ROMAN,*p);
        glPopMatrix();
}
void display
{
        glClear(GL_COLOR_BUFFER_BIT);
        output(10,300,"GLOBAL ACADEMY OF TECHNOLOGY");
        glFlush();
}
void myinit()
{
        glClearColor(1.0,1.0,1.0,1.0);
        glColor3f(1.0,0.0,0.0);
        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("GAT");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

7. Drawing colored line and writing Text

```
#include<glut.h>
#include<string.h>
char*str= "GRAPHICS";
void display()
{
        int i;
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0,0.0,0.0);
        glLineWidth(10.0);
        glBegin(GL_LINES);
                glVertex2f(0.0,0.0);
                glColor3f(0.0,1.0,0.0);
                glVertex2f(0.0,0.8);
        glEnd();
        glColor3f(0.0,1.0,1.0);
        glRasterPos2f(-0.2,-0.1); //font type character to be displayed
        for(i=0:i<strlen(str):i++)
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18.str[i]);
        glFlush():
}
void myinit()
{
        glClearColor(0.0.0.0.0.0.0.0);
        gluOrtho2D(-1.0,1.0,-1.0,1.0);
}

void main(int argc, char **argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("Coloured Line");
        myinit();
        glutDisplayFunc(display);
```



```
        glutMainLoop();
}
```

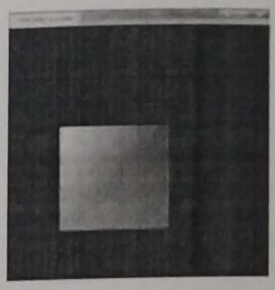
Make necessary change in program to get the following output

Change the window color to BLACK
Set the font to GLUT_STROKE_MONO_ROMAN
Display "GRAPICS IS FUN!" in YELLOW color
Display "REALLY FUN!" in RED color in the next line

```
#include<glut.h>
void output(GLfloat x,GLfloat y,char *text)
{
        char* p;
        glPushMatrix();
        glTranslatef(x,y,0);
        glScaled(0.2,0.2,0);
        for(p=text;*p;p++)
        glutStrokeCharacter( GLUT_STROKE_MONO_ROMAN,*p);
        glPopMatrix();
}
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        output(70,300,"GRAPHICS IS FUN!");
        glColor3f(1.0,0.0,0.0);
        output(120,250,"REALLY FUN!");
        glFlush();
}
void myinit
{
        glClearColor(0.0,0.0,0.0,0.0);
        glColor3f(1.0,1.0,0.0);
        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("STROKE TEXT");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

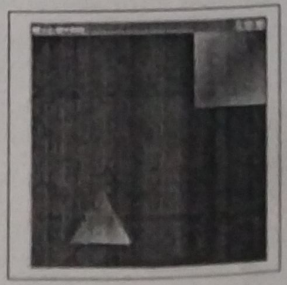
8. Drawing a colored square

```
#include<glut.h>
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0, 1.0, 0.0); // Green
        glBegin(GL_POLYGON);
                glVertex2f(100, 100);
                glColor3f(1.0,0.0,0.0); // Red
                glVertex2f(300, 100);
                glColor3f(0.0,0.0,1.0); // Blue
                glVertex2f(300, 300);
                glColor3f(1.0,1.0,0.0); // Yellow
                glVertex2f(100, 300);
        glEnd();
        glFlush();
}
void myinit()
{
        glClearColor(0.0,0.0,0.0,1.0);
        glColor3f(1.0,0.0,0.0);// Red
        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("COLORED SQUARE");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```



9. Creating 2 view ports

```
#include<glut.h>
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glViewport (5,-150,400,400);
        glBegin(GL_POLYGON);
                glColor3f(1.0,0.0,0.0);
                glVertex2f(90,250);
                glColor3f(0.0,1.0,0.0);
                glVertex2f(250,250);
                glColor3f(0.0,0.0,1.0);
```



```
        glVertex2f(175,400);
    glEnd();
    glViewport (300,300,400,400);
    glBegin(GL_POLYGON);
        glColor3f(1.0,0.0,0.0);
        glVertex2f(50,50);
        glColor3f(0.0,1.0,0.0);
        glVertex2f(250,50);
        glColor3f(0.0,0.0,1.0);
        glVertex2f(250,250);
        glColor3f(0.0,1.0,1.0);
    glVertex2f(50,250);
    glEnd();
    glFlush();
}
void myinit()
{
    glClearColor(0.0,0.0,0.0,1.0);
    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("2 VIEW PORTS");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

VTU SYLLABUS PROGRAMS

Program 1

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

Bresenham's line algorithm is named after Jack Elton Bresenham who developed it in 1962 at IBM. It is commonly used to draw line primitives in a bitmap image (e.g. on a computer screen), as it uses only integer addition, subtraction and bit shifting, all of which are very cheap operations in standard computer architectures. It is an incremental error algorithm.

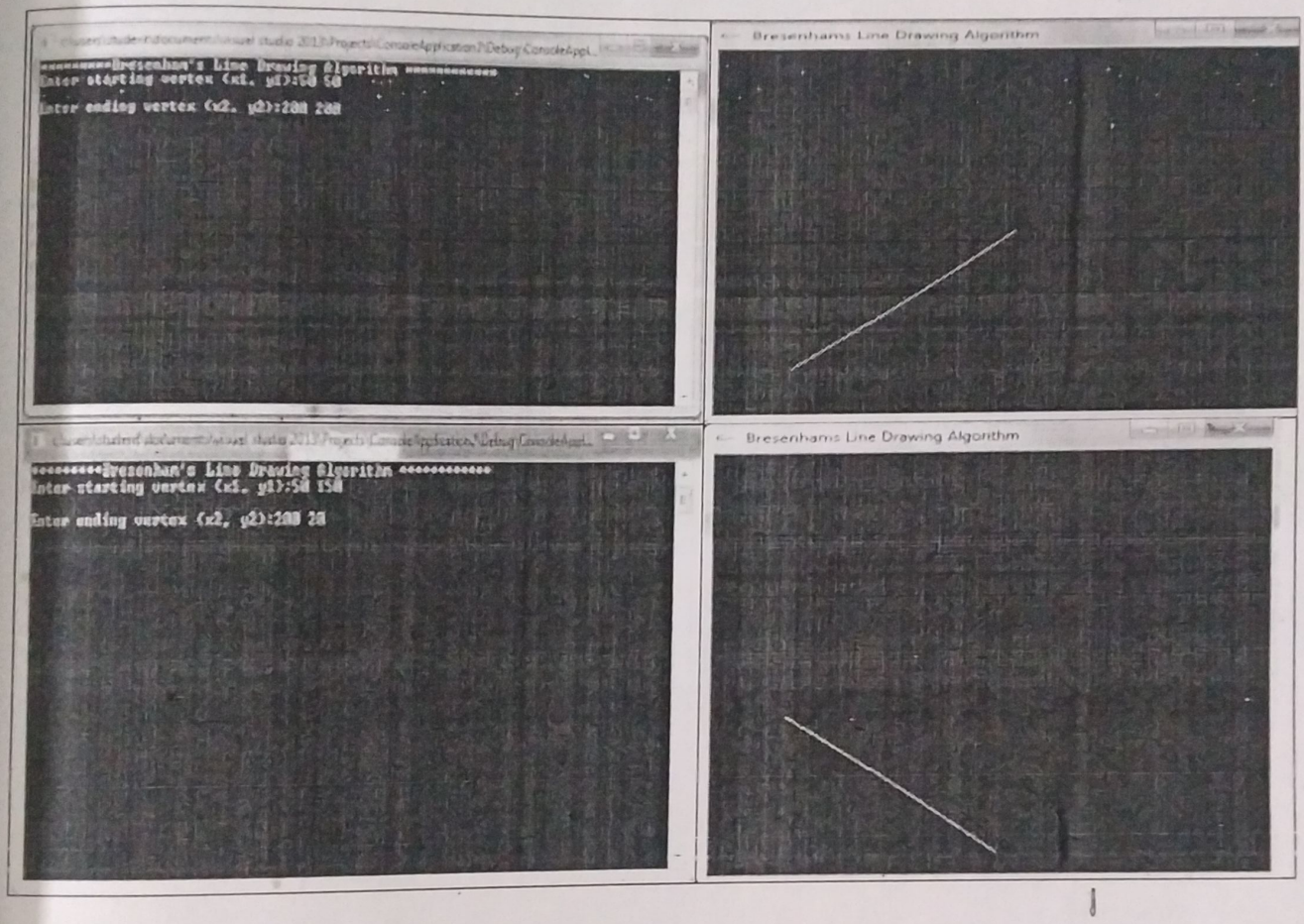
```
#include glut.h>
#include<math.h>
#include<stdio.h>

GLint xOne, yOne, xTwo, yTwo;

void resize(int, int);
voidsetPixel(GLint, GLint);
voidHLine(GLint, GLint, GLint); void VLine(GLint, GLint, GLint);
voidlineBres(GLint, GLint, GLint, GLint, GLfloat);
void display();

void main(int argc, char**argv)
{
printf("*********Bresenham's Line Drawing Algorithm ***********");
printf("\nEnter starting vertex (x1, y1):");
scanf_s("%d%d", &xOne, &yOne);
printf("\nEnter ending vertex (x2, y2):");
scanf_s("%d%d", &xTwo, &yTwo);

glutInit(&argc, argv);                                  //initialize GLUT
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);   //initialize display mode
glutInitWindowSize(400, 400);                           //set display-window width & height
glutInitWindowPosition(800, 50);                        //set display-window upper-left position

//create display-window with a title
glutCreateWindow("Bresenhams Line Drawing Algorithm");
glutDisplayFunc(display);                       //call graphics to be displayed on the window
glutReshapeFunc(resize);                        //calls whenever frame buffer window is resized
glutMainLoop();                                 //display everything and wait
}
void resize(int w, int h)
{
        //set projection paramaters
                glMatrixMode(GL_PROJECTION);
```

```
        glLoadIdentity();
        gluOrtho2D(0.0, w, 0.0, h);
        glViewport(0.0, 0.0, w, h);
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    GLfloatm;
    GLint t;
    if(xOne == xTwo)
    {
        if(yOne > yTwo)
        {
            t = yOne; yOne = yTwo; yTwo =t;
        }
        VLine(xOne, yOne, yTwo);  //verticalline
    }

    else if (yOne == yTwo) // horizontal line
    {
        HLine(xOne, yOne, xTwo);
    }
    else
    {
        m = (float)(yTwo - yOne) / (xTwo - xOne);                    //compute slope
        //call required function based on value of slope
        if (fabs(m) < 1)
            lineBres(xOne, yOne, xTwo, yTwo, m); // slope < one
        else
            lineBres(yOne, xOne, yTwo, xTwo, m); // slope >= one

    }
}
void lineBres(GLint x0, GLint y0, GLint xEnd, GLint yEnd, GLfloat m)
{
    GLint dx = abs(xEnd - x0);
    GLint dy = abs(yEnd - y0);
    GLint p = 2 * dy - dx;
    GLint twoDy = 2 * dy;
    GLint twoDyMinusDx = 2 * (dy - dx);
    GLint x = x0, y = y0;
    // determine which point to use as start position
    if (x0 > xEnd)
    {
        x = xEnd;
```

```
            y = yEnd;
            xEnd = x0;
        }
        else
        {
            x = x0;
            y = y0;
        }
        setPixel(x, y);
        while (x<xEnd)
        {
            x++;
            if (p<0)
                p += twoDy;
            else
            {
                if (m<0)
                    y--;
                else
                    y++;
                p += twoDyMinusDx;
            }
            setPixel(x, y);
        }
}
void VLine(GLint x1, GLint y1, GLint y2) //vertical line
{
        GLint x,y;
        x = x1;
        glBegin(GL_POINTS);
        for (y = y1; y <= y2; y++)
            glVertex2d(x, y);
        glEnd();
        glFlush();
}
void HLine(GLint x1, GLint y1, GLint x2) //horizontal line
{
        GLint x,y;
        y = y1;
        glBegin(GL_POINTS);
            for (x = x1; x <= x2; x++)
            glVertex2d(x, y);
        glEnd();
        glFlush();
}
void setPixel(GLint xCoordinate, GLint yCoordinate) //to plot point on the screen
```

```
{
    glBegin(GL_POINTS);
    glVertex2i(xCoordinate, yCoordinate);
    glEnd();
    glFlush(); //executes all OpenGL functions as quickly as possible
}
```

OUTPUT



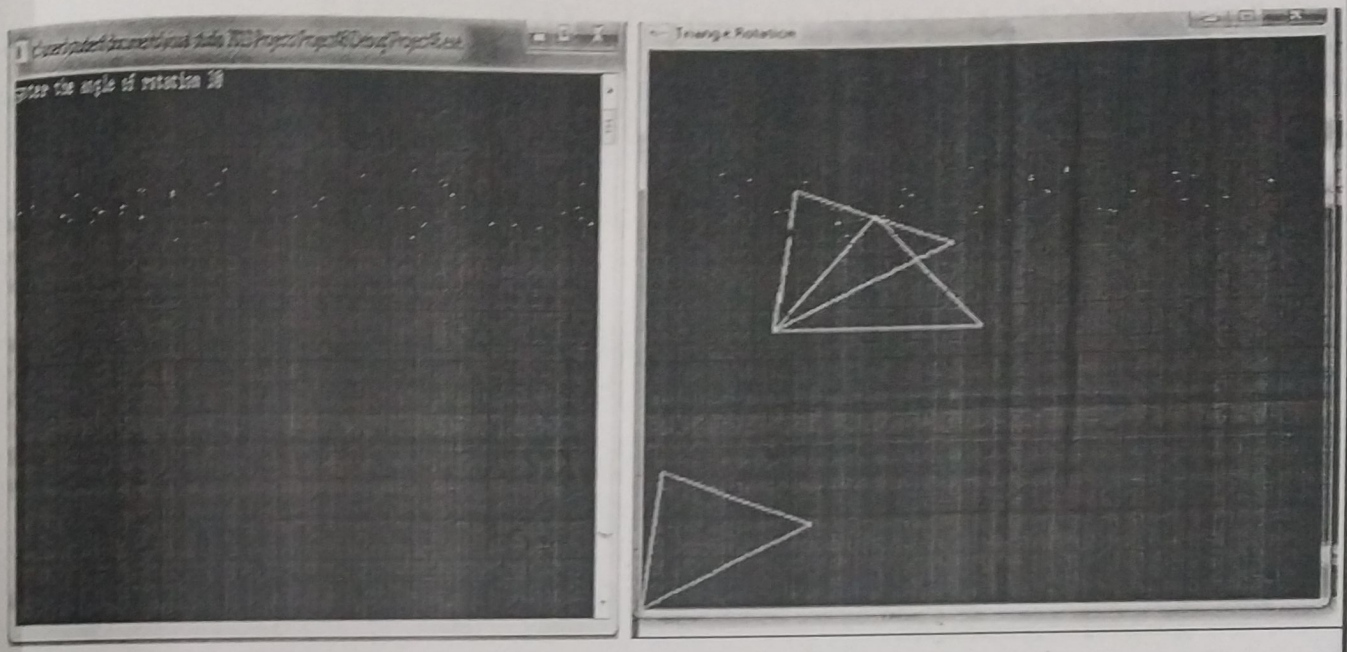
Program 2

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

```
#include<windows.h>
#include<glut.h>
#include<iostream>
using namespace std;
GLfloatangle;
voidtriangle()
{
        glBegin(GL_LINE_LOOP);
        glVertex2i(100, 250);
        glVertex2i(175,350);
        glVertex2i(250, 250);
        glEnd();
        glFlush();
}

void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glClearColor(1, 1, 1, 1);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        triangle();
        glRotatef(angle, 0.0, 0.0, 1.0);
        glTranslatef(-100, -250, 0.0);
        triangle();
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glTranslatef(100, 250, 0.0);
        glRotatef(angle, 0.0, 0.0, 1.0);
        glTranslatef(-100, -250, 0.0);
        triangle();
}
void main(int argc, char**argv)
{
        cout<< "Enter the angle of rotation";
        cin>> angle;
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(1000, 1000);
        glutCreateWindow("Triangle Rotation");
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0, 500, 0, 500);
```

```
    glutDisplayFunc(display);
    glutMainLoop();
}
```

OUTPUT:



Program 3

Draw a Colour Cube and spin it using OpenGL transformation matrices.

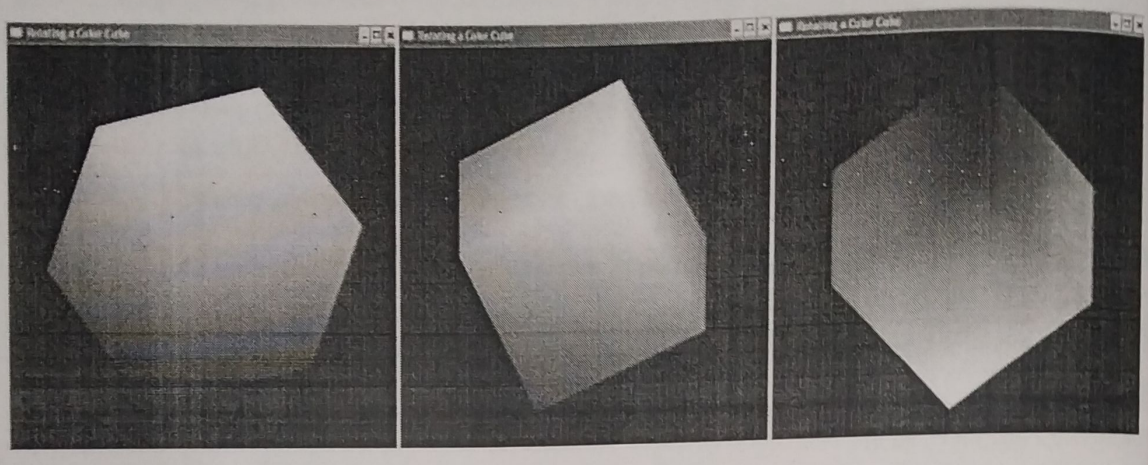
```
#include<glut.h>

GLfloat vertices[8][3] = { {-1.0,-1.0,1.0},{1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0},
{-1.0,-1.0,-1.0}, {1.0,-1.0,-1.0},{1.0,1.0,-1.0},{-1.0,1.0,-1.0} };

GLfloat colors[8][3] = { {0.0,0.0,1.0}, {1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0},
{0.0,0.0,0.0},{1.0,0.0,0.0},{1.0,1.0,0.0},{0.0,1.0,0.0} };

GLfloat theta[] = {0.0,0.0,0.0};
GLint axis =2;                    // default z-axisrotation

voidpolygon(int a, int b, int c , int d) // draw a polygon via list of vertices
{
        glBegin(GL_POLYGON);
        glColor3fv(colors[a]);
        glVertex3fv(vertices[a]);
        glColor3fv(colors[b]);
        glVertex3fv(vertices[b]);
        glColor3fv(colors[c]);
        glVertex3fv(vertices[c]);
        glColor3fv(colors[d]);
        glVertex3fv(vertices[d]);
        glEnd();
}

voidcolorcube(void) // map vertices to faces
{
polygon(0,3,2,1); // front face – counter clockwise
polygon(4,5,6,7); // back face – clockwise
polygon(2,3,7,6); // front face – counter clockwise
polygon(1,5,4,0); // back face – clockwise
polygon(1,2,6,5); // front face – counter clockwise
polygon(0,4,7,3); // back face – clockwise
}

voiddisplay(void)  // display callback. clear frame buffer and z buffer,rotate cube and draw, swapbuffers
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glRotatef(theta[0], 1.0, 0.0,0.0);
glRotatef(theta[1], 0.0, 1.0,0.0);
glRotatef(theta[2], 0.0, 0.0, 1.0);
colorcube();
```

```
glFlush():
glutSwapBuffers();
}

voidspinCube()                    // Idlecallback, spin the cube 0.05 degrees about selectedaxis
{
theta[axis]+= 0.05; // Controls the speed
if( theta[axis] > 360.0 ) theta[axis]-= 360.0;
glutPostRedisplay();
}

voidmouse(int btn, int state, int x,inty)// mouse callback, selects an axis about which to rotate
{
if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;
if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
}

voidmyReshape(int w, int h)
{
glViewport(0, 0, w, h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
if(w <= h)
glOrtho(-2.0, 2.0, -2.0 *(GLfloat) h/(GLfloat)w,2.0*(GLfloat) h /(GLfloat) w,-10.0, 10.0);
else
glOrtho(-2.0 *(GLfloat)w /(GLfloat) h, 2.0 *(GLfloat) w /(GLfloat) h,-2.0, 2.0,-10.0, 10.0);
glMatrixMode(GL_MODELVIEW);
}
voidmain(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);//need both double buffering and z
buffer glutInitWindowSize(500, 500);
glutCreateWindow("Rotating a Color Cube");
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
glutIdleFunc(spinCube);
glutMouseFunc(mouse);
glEnable(GL_DEPTH_TEST); // Enable hidden--surface—removal
glutMainLoop();
}
```

```
glFlush();
glutSwapBuffers();
}

voidspinCube()                    // Idlecallback, spin the cube 0.05 degrees about selectedaxis
{
theta[axis]+= 0.05; // Controls the speed
if( theta[axis] > 360.0 ) theta[axis]-= 360.0;
glutPostRedisplay();
}

voidmouse(int btn, int state, int x,inty)// mouse callback, selects an axis about which to rotate
{
if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;
if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
}

voidmyReshape(int w, int h)
{
glViewport(0, 0, w, h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
if(w <= h)
glOrtho(-2.0, 2.0, -2.0 *(GLfloat) h/(GLfloat)w,2.0*(GLfloat) h /(GLfloat) w,-10.0, 10.0);
else
glOrtho(-2.0 *(GLfloat)w /(GLfloat) h, 2.0 *(GLfloat) w /(GLfloat) h,-2.0, 2.0,-10.0, 10.0);
glMatrixMode(GL_MODELVIEW);
}
voidmain(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);//need both double buffering and z
buffer glutInitWindowSize(500, 500);
glutCreateWindow("Rotating a Color Cube");
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
glutIdleFunc(spinCube);
glutMouseFunc(mouse);
glEnable(GL_DEPTH_TEST); // Enable hidden--surface—removal
glutMainLoop();
}
```

OUTPUT

Rotation through Zaxis Rotation through Xaxis Rotation through Y axis



Program 4

Draw a color cube and allow the user to move the camera suitably to experiment.

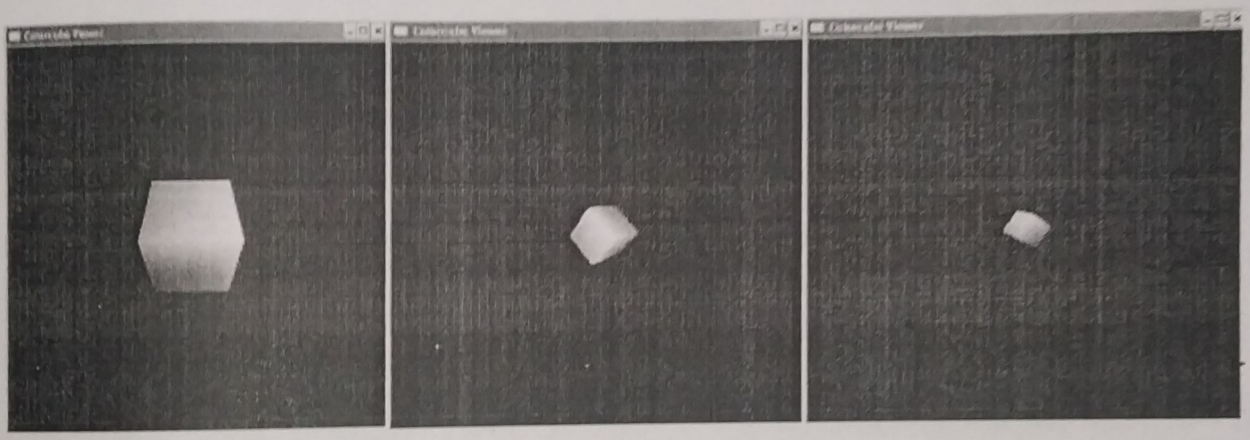
```
#include <GL/glut.h>

GLfloat vertices[8][3]={{-1.0,-1.0,1.0},{1.0,-1.0,1.0}, {1.0,1.0,1.0},{-1.0,1.0,1.0},
{-1.0,-1.0,-1.0}, {1.0,-1.0,-1.0}, {1.0,1.0,-1.0}, {-1.0,1.0,-1.0}};
GLfloat colors[8][3]= {{0.0,0.0,1.0}, {1.0,0.0,1.0}, {1.0,1.0,1.0},{0.0,1.0,1.0},
{0.0,0.0,0.0},{1.0,0.0,0.0}, {1.0,1.0,0.0}, {0.0,1.0,0.0}};

GLfloat theta[] =
{0.0,0.0,0.0}; GLint axis =
2;
GLdouble viewer[]= {0.0, 0.0, 5.0}; /* initial viewer location */

voidpolygon(int a, int b, int c , int d)
{
        glBegin(GL_POLYGON);
                glColor3fv(colors[a]);
                glVertex3fv(vertices[a]);
                glColor3fv(colors[b]);
                glVertex3fv(vertices[b]);
                glColor3fv(colors[c]);
                glVertex3fv(vertices[c]);
                glColor3fv(colors[d]);
                glVertex3fv(vertices[d]);
        glEnd();
}
voidcolorcube()
{
        polygon(0,3,2,1); // front face – counter clockwise
        polygon(4,5,6,7); // back face –clockwise

        polygon(2,3,7,6); // front face – counter clockwise
        polygon(1,5,4,0); // back face –clockwise

        polygon(1,2,6,5); // front face – counter clockwise
        polygon(0,4,7,3); // back face –clockwise

}

voiddisplay(void)
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        // Update viewer position in modelview matrix
```

```
        glLoadIdentity();
        gluLookAt(viewer[0],viewer[1],viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
        glRotatef(theta[0], 1.0, 0.0,0.0);
        glRotatef(theta[1], 0.0, 1.0,0.0);
        glRotatef(theta[2], 0.0, 0.0,
        1.0); colorcube();
        glFlush();
        glutSwapBuffer
        s();
}
voidmouse(int btn, int state, int x, int y)
{
        if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        axis =0;
        if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
        axis =1;
        if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
        axis =2;
        theta[axis]+= 2.0;

      if( theta[axis] > 360.0 ) theta[axis]-= 360.0;
      display();
}
voidkeys(unsigned char key, int x, int y)
{
      if(key == 'x') viewer[0]-= 1.0;
      if(key == 'X') viewer[0]+= 1.0;
      if(key == 'y') viewer[1]-= 1.0;
      if(key == 'Y') viewer[1]+= 1.0;
      if(key == 'z') viewer[2]-= 1.0;
      if(key == 'Z') viewer[2]+= 1.0;
      display();
}

voidmyReshape(int w, int h)
{
      glViewport(0, 0, w, h);
      /* Use a perspective view */
      glMatrixMode(GL_PROJECTIO
      N); glLoadIdentity();
      if(w<=h)
      glFrustum(-2.0, 2.0, -2.0 *(GLfloat) h/(GLfloat) w,2.0*(GLfloat) h/(GLfloat) w, 2.0, 20.0);
      else
      glFrustum(-2.0, 2.0, -2.0 *(GLfloat)w/(GLfloat) h,2.0* (GLfloat) w /(GLfloat) h, 2.0, 20.0);
      glMatrixMode(GL_MODELVIEW);
}
```

```
voidmain(int argc, char **argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |
        GLUT_DEPTH); glutInitWindowSize(500, 500);
        glutCreateWindow("Colorcube
        Viewer");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glutMouseFunc(mouse);
        glutKeyboardFunc(keys);
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();
}
```

OUTPUT

Rotation throughXaxis **Perspective view withKey'x'** **Perspective view with Key'y'**



Program 5

Clip a lines using Cohen-Sutherland algorithm.

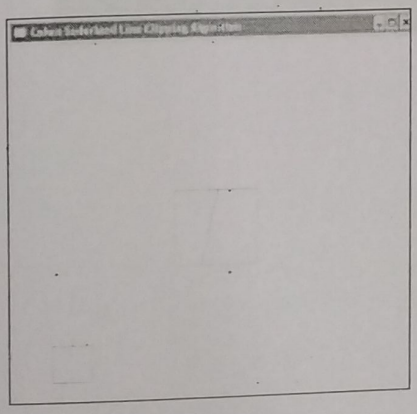
```
#include<glut.h>
#define outcode int

doublexmin=50,ymin=50, xmax=100,ymax=100; // Window boundaries
double xvmin=200,yvmin=200,xvmax=300,yvmax=300; // Viewport boundaries
double m;
//bit codes for the right, left, top, & bottom
const int LEFT = 1;
const int RIGHT = 2;
const int BOTTOM = 4;
const int TOP = 8;

outcode ComputeOutCode (double x, double y)
{
        outcode code = 0;
        if(y>ymax)      // above the clip window
        code |=TOP;
        else if(y<ymin) // below the clip window
        code |= BOTTOM;
        if(x>xmax)      // to the right of clip window
        code |= RIGHT;
        else if(x<xmin) // to the left of clipwindow
        code |= LEFT;
        return code;
}

void CohenSutherlandLineClipAndDraw(double x0, double y0,double x1, double y1)
{
        outcode outcode0, outcode1, outcodeOut;
        bool accept = false, done = false;
        outcode0 = ComputeOutCode (x0, y0);
        outcode1 = ComputeOutCode (x1, y1);
        m=(y1-y0)/(x1-x0);
        do
        {
                if(!(outcode0 | outcode1)) // logical OR is 0000, Trivially accept & exit
                {
                        accept = true;
                        done = true;
                }
        else if (outcode0&outcode1) //logicalAND is not 0000 trivially reject and exit
                done=true;
```

```
else
{
// failed both tests, so calculate the line segment to clip from an outside point
// to an intersection with clip edge

doublex, y;
outcodeOut = outcode0? outcode0: outcode1;

if(outcodeOut&TOP)   //point is above the cliprectangle
{
        x = x0 + (1/m) * (ymax-y0);
        y = ymax;
}

else if(outcodeOut&BOTTOM)          //point is below the cliprectangle
{
        x = x0 + (1/m) * (ymin-y0);
        y = ymin;
}
elseif(outcodeOut&RIGHT)     //point is to right of cliprectangle
{
        y = y0 +m*(xmax-x0);
        x = xmax;
}

else     //point is to the left of cliprectangle
{
        y = y0 +m*(xmin-x0);
        x = xmin;
}
// Now we move outside point to intersection point to clip and get ready for nextpass.

if( outcodeOut == outcode0)
{
        x0 = x;
        y0 = y;
        outcode0 = ComputeOutCode (x0, y0);
}
else
{
        x1 = x;
        y1 = y;
        outcode1= ComputeOutCode (x1, y1);
}
}
} while (!done);
```

```
if (accept)
{
        doublesx=(xvmax-xvmin)/(xmax-xmin);
        double sy=(yvmax-yvmin)/(ymax-ymin);

        doublevx0=xvmin+(x0-xmin)*sx;
        double vy0=yvmin+(y0-ymin)*sy;
        double vx1=xvmin+(x1-xmin)*sx;
        double vy1=yvmin+(y1-ymin)*sy;

        glColor3f(1.0, 0.0, 0.0); // new view port in red color
        glBegin(GL_LINE_LOOP);
                glVertex2f(xvmin, yvmin);
                glVertex2f(xvmax, yvmin);
                glVertex2f(xvmax, yvmax);
                glVertex2f(xvmin, yvmax);
        glEnd();
        glColor3f(0.0,0.0,1.0); // clipped line in blue color
        glBegin(GL_LINES);
                glVertex2d (vx0, vy0);
                glVertex2d (vx1, vy1);
        glEnd();
        }
}

voiddisplay()
{
        doublex0=60,y0=20,x1=80,y1=120;
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0,0.0,0.0);
        glBegin(GL_LINES);
                glVertex2d (x0, y0);
                glVertex2d (x1, y1);
        glEnd();
        glColor3f(0.0, 0.0, 1.0);
        glBegin(GL_LINE_LOOP);
                glVertex2f(xmin, ymin);
                glVertex2f(xmax, ymin);
                glVertex2f(xmax, ymax);
                glVertex2f(xmin, ymax);
        glEnd();
        CohenSutherlandLineClipAndDraw(x0,y0,x1,y1);
        glFlush();
}

voidmyinit()
```

```
{
        glClearColor(1.0,1.0,1.0,1.0);
        glColor3f(1.0,0.0,0.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0,499.0,0.0,499.0);
}
voidmain(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutCreateWindow("Cohen Suderland Line Clipping Algorithm");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

OUTPUT



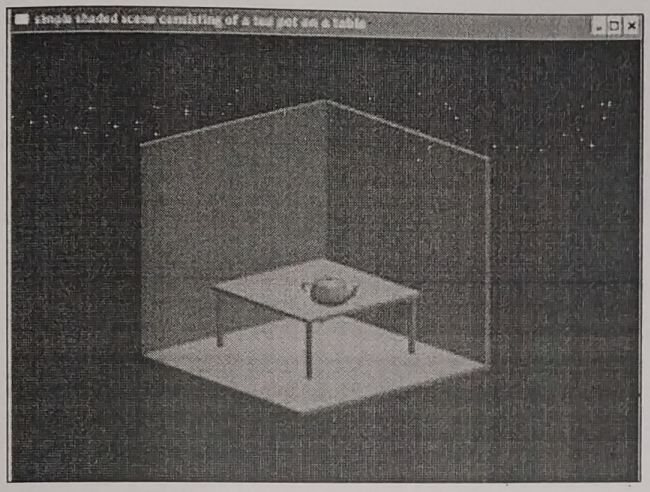
Program 6

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.

```
#include<glut.h>
void obj(double tx, double ty, double tz, double sx, double sy, double sz)
{
        glRotated(50,0,1,0);
        glRotated(10,-1,0,0);
        glRotated(11.7,0,0,-1);
        glTranslated(tx,ty,tz);
        glScaled(sx,sy,sz);
        glutSolidCube(1);
        glLoadIdentity();
}
void display()
{
        glViewport(0,0,700,700);
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        obj(0,0,0.5,1,1,0.04); // three walls
        obj(0,-0.5,0,1,0.04,1);
        obj(-0.5,0,0,0.04,1,1);
        obj(0,-0.3,0,0.02,0.2,0.02); // four table legs
        obj(0,-0.3,-0.4,0.02,0.2,0.02);
        obj(0.4,-0.3,0,0.02,0.2,0.02);
        obj(0.4,-0.3,-0.4,0.02,0.2,0.02);
        obj(0.2,-0.18,-0.2,0.6,0.02,0.6); // table top
        glRotated(50,0,1,0);
        glRotated(10,-1,0,0);
        glRotated(11.7,0,0,-1);
        glTranslated(0.3,-0.1,-0.3);
        glutSolidTeapot(0.09);
        glFlush();
        glLoadIdentity();
}
void main()
{
        float ambient[]={1,1,1,1};
        float light_pos[]={27,80,2,3};
        glutInitWindowSize(700,700);
        glutCreateWindow("scene");
        glutDisplayFunc(display);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
```

```
    glMaterialfv(GL_FRONT,GL_AMBIENT,ambient);
    glLightfv(GL_LIGHT0,GL_POSITION,light_pos);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

OUTPUT:



Program 7

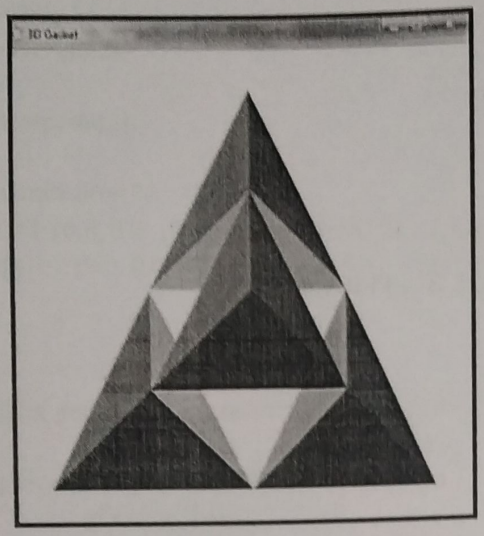
Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user.

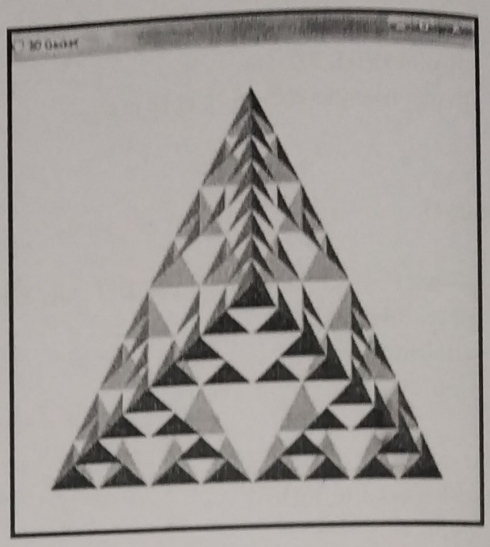
```
#include<stdio.h>
#include<GL/glut.h>

typedef floatpoint[3];

/* initial tetrahedron */
point v[4]={ {0.0, 0.0, 10.0}, {0.0, 10.0, -10.0}, {-10.0, -10.0, -10.0}, {10.0, -10.0, -10.0} };
// point v[4]={ {0.0, 0.0, 0.0}, {10.0, 0.0, 0.0}, {5.0, 10.0, 0.0}, {5.0, 5.0, 10.0} };

intn;

voidtriangle( point a, point b,pointc)      // display onetriangle
{
        glBegin(GL_POLYGON);
                glVertex3fv(a);
                glVertex3fv(b);
                glVertex3fv(c);
        glEnd();
}

voiddivide_triangle(point a, point b, point c, int m) // triangle subdivision
{
        point p1, p2, p3;
        int i;
        if(m>0)
        {
                for(i=0; i<3;i++)
                {
                        p1[i]=(a[i]+b[i])/2;
                        p2[i]=(a[i]+c[i])/2;
                        p3[i]=(b[i]+c[i])/2;
                }
                divide_triangle(a, p1, p2, m-1);
                divide_triangle(c, p2, p3, m-1);
                divide_triangle(b, p3, p1, m-1);

        }
        else
        triangle(a,b,c); // draw triangle at end ofrecursion
}
voidtetrahedron(intm)  // Apply triangle subdivision to faces oftetrahedron
```

```
{
        glColor3f(1.0,0.0,0.0);
        divide_triangle(v[0], v[1], v[2], m);
        glColor3f(0.0,1.0,0.0);
        divide_triangle(v[3], v[2], v[1], m);
        glColor3f(0.0,0.0,1.0);
        divide_triangle(v[0], v[3], v[1], m);
        glColor3f(0.0,0.0,0.0);
        divide_triangle(v[0], v[2], v[3], m);
}

voiddisplay()
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        tetrahedron(n);
        glFlush();
}

voidmyReshape(int w, int h)
{
        glViewport(0, 0, w, h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
if(w <= h)
glOrtho(-12.0, 12.0, -12.0 * (GLfloat) h / (GLfloat) w, 12.0 * (GLfloat) h / (GLfloat) w, - 12.0,12.0);
else
glOrtho(-12.0 * (GLfloat) w / (GLfloat) h, 12.0 * (GLfloat) w / (GLfloat) h, -12.0, 12.0, - 12.0,12.0);
        glMatrixMode(GL_MODELVIEW);
        glutPostRedisplay();
}

voidmain(int argc, char **argv)
{
        printf(" No. of Divisions ? ");
        scanf_s("%d",&n);
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_DEPTH);          // to view the behind triangle
        glutInitWindowSize(500, 500);
        glutCreateWindow("3D Gasket");
        glClearColor (1.0, 1.0, 1.0, 1.0);
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();
}
```

OUTPUT

No. of divisions? 1No. of divisions?3





Program 8

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

```
#include<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; static int window;
static int menu_id = 2;
static int submenu_id =
1; static int value = 0;

typedef structwcPt3D
{
        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;
        }
}

voidcomputeBezPt(GLfloat u, wcPt3D *bezPt, GLint nCtrlPts, wcPt3D *ctrlPts, GLint *C)
{
        GLint k, n = nCtrlPts
        - 1; GLfloat
        bezBlendFcn;
```

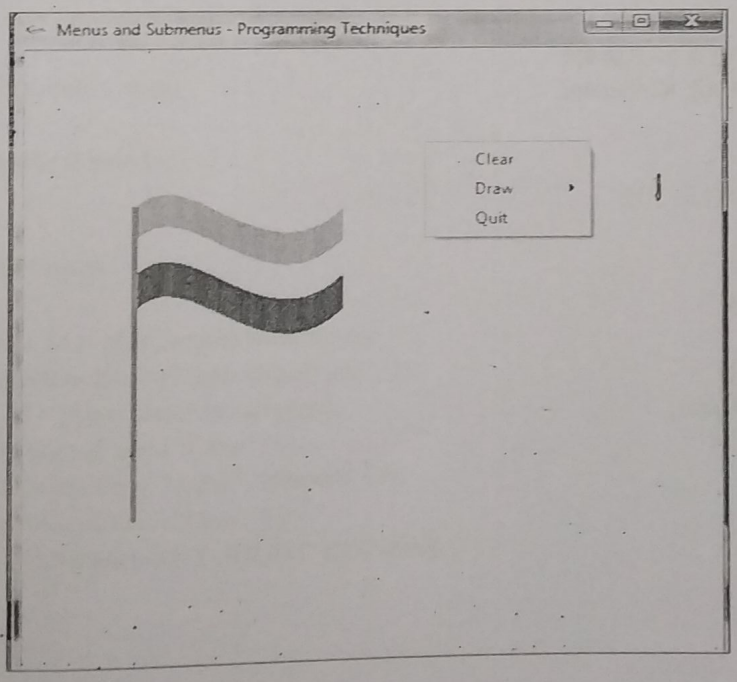
```
        bezPt->x = bezPt->y = bezPt->z =
        0.0; for (k = 0; k<nCtrlPts; k++)
        {
                bezBlendFcn = C[k] * pow(u, k) * pow(1 - u, n -
                k); bezPt->x += ctrlPts[k].x *bezBlendFcn;
                bezPt->y += ctrlPts[k].y
        *bezBlendFcn;
                bezPt->z += ctrlPts[k].z
        *bezBlendFcn;
        }
}
voidbezier(wcPt3D *ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)
{
        wcPt3DbezCurve
        Pt; GLfloat u;
        GLint *C, k;
        C = new
        GLint[nCtrlPts];
        bino(nCtrlPts - 1, C);
        glBegin(GL_LINE_STR
        IP);
        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;
}

voiddisplayFunc()
{
        GLint nCtrlPts = 4, nBezCurvePts =
        20; static float theta =0;
        wcPt3DctrlPts[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);
glColor3f(1.0, 1.0, 1.0);
glPointSize(5
);
glPushMatrix
();
glLineWidth(
5);
glColor3f(255 / 255, 153 / 255.0, 51 / 255.0);          //Indian flag: Orange color
code for (int i = 0; i<8; i++)
{
        glTranslatef(0, -0.8, 0);
        bezier(ctrlPts, nCtrlPts,nBezCurvePts);
}
glColor3f(1, 1, 1); //Indian flag: white colorcode
for(int i = 0; i<8; i++)
{
        glTranslatef(0, -0.8, 0);
        bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glColor3f(19 / 255.0, 136 / 255.0, 8 / 255.0); //Indian flag: green color code
for (int i = 0; i<8; i++)
{
        glTranslatef(0, -0.8, 0);
        bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glPopMatrix();
glColor3f(0.7, 0.5, 0.3);
glLineWidth(5);
glBegin(GL_LINES);
        glVertex2f(20, 100);
        glVertex2f(20, 40);
glEnd();
glFlush();
glutPostRedisplay();
glutSwapBuffers();
}
voidwinReshapeFun(GLint newWidth, GLint newHeight)
{
        glViewport(0, 0, newWidth, newHeight);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
```

```
        gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);
        glClear(GL_COLOR_BUFFER_BIT);
}

voiddisplay(void)
{
        glClear(GL_COLOR_BUFFER_BIT);
        if (value == 1)
        {
                glutPostRedisplay();
        }
        else if (value == 2)
        {
                glPushMatrix();
                glColor3d(1.0, 0.0, 0.0);
                glutDisplayFunc(displayFunc);
                //glutWireSphere(0.5, 50, 50);
                glPopMatrix();
        }
        glFlush();
}

voidmenu(int num)
{
        if(num == 0)
        {
                glutDestroyWindow(window);
                exit(0);
        }
        else
        {
                value = num;
        }
        glutPostRedisplay();
}

voidcreateMenu(void)
{
        submenu_id = glutCreateMenu(menu);
        glutAddMenuEntry("draw a flag", 2);
        menu_id = glutCreateMenu(menu);
        glutAddMenuEntry("Clear", 1);
        glutAddSubMenu("Draw", submenu_id);
        glutAddMenuEntry("Quit", 0);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

```
voidmyinit()
{
        glViewport(0, 0, 500, 500);
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glColor3f(1.0, 0.0, 0.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0, 499.0, 0.0, 499.0);
}

intmain(int argc, char **argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(100, 100);
        window = glutCreateWindow("Menus and Submenus - Programming Techniques");
        createMenu();
        glClearColor(0.0, 0.0, 0.0, 0.0);
        glutDisplayFunc(display);
        //glutDisplayFunc(displayFunc);
        glutReshapeFunc(winReshapeFun);
        myinit();
        glutMainLoop();
}
```

OUTPUT



Program 9

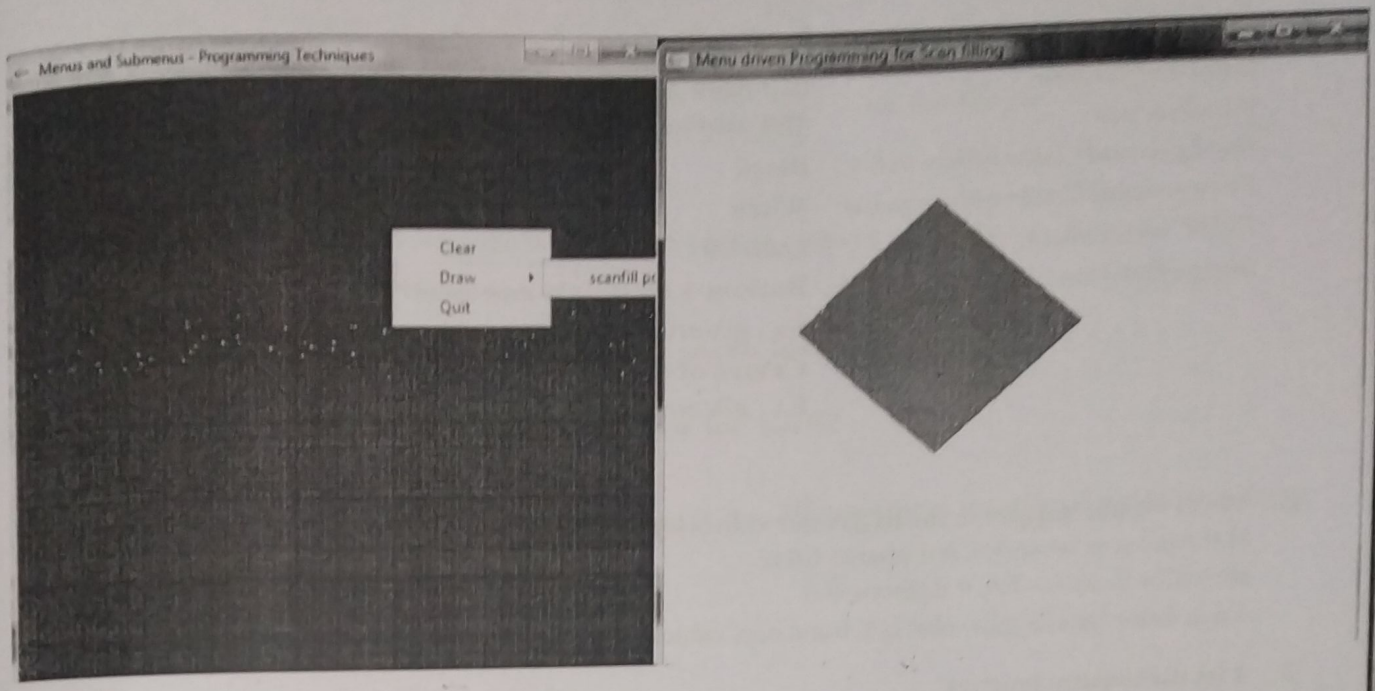
Develop a menu driven program to fill the polygon using scan line algorithm.

```
#include <windows.h>
#include <glut.h>
#include<stdio.h>

static int window;
static int menu_id;
static int submenu_id;
static int value = 0;

float x1 = 200.0, y1 = 200.0, x2 = 100.0, y2 = 300.0, x3 = 200.0, y3 = 400.0, x4 = 300.0, y4=300.0;

void draw_pixel(int x, int y)
{
        glColor3f(1.0, 0.0, 0.0);
        //Sleep(50);      // To set the delay time
        glBegin(GL_POINTS);
                glVertex2i(x, y);
        glEnd();
        glFlush();
}
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 scanfill(float x1, float y1, float x2, float y2, float x3, float y3, float x4, float y4)
{
        int le[500], re[500], 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 menu(int num)
{
        if (num == 0)
        {
                glutDestroyWindow(window);
                exit(0);
        }
        else
        {
                value = num;
        }
        glutPostRedisplay();
}

void createMenu(void)
{
        submenu_id = glutCreateMenu(menu);
        glutAddMenuEntry("scanfill polygon", 2);
        menu_id = glutCreateMenu(menu);
        glutAddMenuEntry("Clear", 1);
        glutAddSubMenu("Draw", submenu_id);
        glutAddMenuEntry("Quit", 0);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

```
void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT);
        if (value == 1)
        {
                glutPostRedisplay();
        }
        else if (value == 2)
        {
                glClear(GL_COLOR_BUFFER_BIT);
                glColor3f(0.0, 0.0, 1.0);
                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, 0.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0, 499.0, 0.0, 499.0);
}

int main(int argc, char **argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(100, 100);
        window = glutCreateWindow("Menu driven Programming for Scan filling ");
        createMenu();
        glClearColor(0.0, 0.0, 0.0, 0.0);
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
        return EXIT_SUCCESS;
}
```

OUTPUT


Menus and Submenus - Programming Techniques
Clear
Draw
Quit
Menu driven Programming for Scan filling

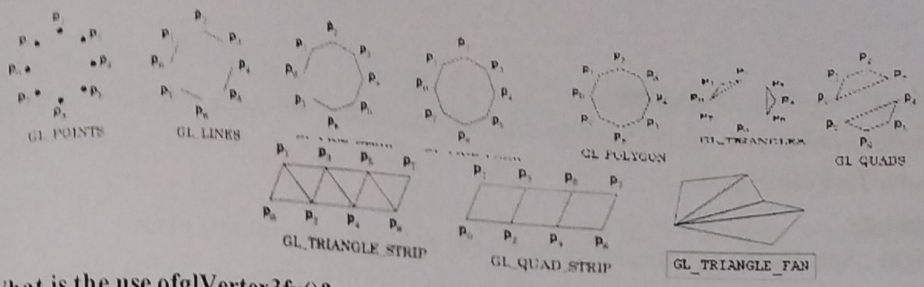
Viva Questions

1. **Specify the default values for thefollowing**

WindowPosition:	0, 0 from Top-LeftCorner
WindowSize:	300,300 from Top-LeftCorner
Background Color ofthewindow:	Black
Foreground Color ofthewindow:	White
RGBColorvalues:	Each1.0
ImagePosition:	**Bottom-Left if only one quadrant is used.** Ex : glVertex3i(70, 80), glVertex3f(7.5, 8.0) **Centre of the Screen if 4 quadrants are used.** Ex : glVertex3f(-5.0, 3.0, 0.0)

2. **What would happen to the RGB color values set to 3.0 OR-7.0?**
   If the color is set to 3.0, it is reset to 1.0 If
   the color is set to -7.0, it is reset to 0.0
   Each color takes a min value as 0.0 and max value as 1.0

3. **List differentprimitives.**
   - **GL_POINTS: Each** vertex is displayed at a size of at least one pixel.
   - **GL_LINES: Successive** pairs of vertices would be connected as aline.
   - **GL_LINE_STRIP: Connects** the successive vertices using line segments. However the final vertex would not be connected to the initialvertex.
   - **GL_LINE_LOOP: Connects** the successive vertices using line segments to form a closedpath.
   - **GL_POLYGON:** Connects the successive vertices using line segments to forma closed path. The interior is filled according to the state of the relevantattributes.
   - **GL_QUADS:** Special case of polygon where successive group of 4 vertices are interpreted asquadrilaterals.
   - **GL_TRIANGLES:** Special case of polygon where successive group of three vertices would be interpretedas atriangle.
   - **GL_TRIANGLE_STRIP:** Each additional vertex is combined with the previous two vertices to define a newtriangle.
   - **GL_QUAD_STRIP: W**ecombine two new vertices with the previous two verticesto design a newquadrilateral.
   - **GL_TRIANGLE_FAN:** Based on one fixed point. The next two points determine the first triangle. The subsequent triangles are formed from one new point, the previous point and the first (fixed)point.



4. **What is the use ofglVertex3fv()?**

A Vertex is used to define geometric primitives.

In OpenGL a vertex can be represented as **glVertex***() where * can be nt OR ntv

n : no. of dimensions (2, 3 , 4)

t : data types (int, float,double)

v : specifies that variables are specified through a pointer to an array.

5. **How to set the background color of the window toCYAN?**

glClearColor(0.0, 1.0,1.0,1.0);

The first 3 arguments represent RGB values and the 4th argument represents alpha used for creating ***fog effects*** (combining the images).

0.0 – transparent (passes all kinds of light)

1.0 – opaque(does not pass any light)

6. **How to set the image color toMAGENTA?**

glColor3f(1.0,0.0,1.0)

7. **What is the use ofglEnable(GL_DEPTH_TEST)?**

This function enables hidden surface removal so that the hidden surface will not be seen.

8. **What is the use ofglutMainLoop()?**

- Causes the program to begin an event processingloop.
- If there are no events to process, then the program would enter the wait statewith the output on thescreen.
- It is similar to getch() in Cprogram.

9. **Which function is used to make the hidden surface to beviewed?**

glClear(GL_DEPTH_BUFFER_BIT);

10. **Why do we needglLoadIdentity()?**

- It replaces the current matrix with the identitymatrix.
- Serves to "reset" the coordinate system to unity before any matrix manipulations are performed.
- Use glLoadIdentity to clear a matrix stack rather than loading yourown.

11. **What are glOrtho() and gluOrtho2D() functions?**

These functions are used for parallelprojections.

glOrtho() --Syntax:

glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble nearVal, GLdouble farVal)

Parameters

left, right : Specify the coordinates for the left and right vertical clipping planes.

bottom, top : Specify the coordinates for the bottom and top horizontal clipping planes.

nearVal, farVal : Specify the distances to the nearer and farther depth clipping planes. These values are negative if the plane is to be behind the viewer.

gluOrtho2D()

gluOrtho2D sets up a two-dimensional orthographic viewing region. This is equivalent to calling glOrtho with near = -1 and far = 1 .

Syntax :

glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)

Parameters

left, right : Specify the coordinates for the left and right vertical clipping planes.

bottom, top : Specify the coordinates for the bottom and top horizontal clippingplanes.

12. What are the functions for viewing perspectiveprojections?

- glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top,GLdouble nearVal, GLdoublefarVal)

Parameters

left, right : Specify the coordinates for the left and right vertical clippingplanes.

bottom, top : Specify the coordinates for the bottom and top horizontalclippingplanes.

nearVal, farVal : Specify the distances to the near and far depth clipping planes.

Both distances must be positive.

- gluPerspective(GLdouble fov, GLdouble aspect, GLdouble znear, GLdoublezfar)

Parameters

fovy : Specifies the field of view angle, in degrees, in the y direction.

aspect : Specifies the aspect ratio that determines the field of view in the x direction. Theaspect ratio is the ratio of x (width) to y (height).

zNear : Specifies the distance from the viewer to the near clipping plane(always positive).

zFar : Specifies the distance from the viewer to the far clipping plane (always positive).

13. What are window callbackfunctions?

Window callbacks indicate when to redisplay or reshape a window, when the visibility of the window changes, and when input is available for the window.

14. What is the use ofglFlush()?

glFlush empties all of the buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

15. In Sierpinski gasket, if the no. of divisions=2, how many triangles areformed?

Sierpinski gasket is a tetrahedran, hence there are 4 triangles.

If no. of divisions n=0, the no. of triangles=4, If n=1, the no. of triangles=4*3=12

If n=2, the no. of triangles=4*3*3=4*3^2 Hence if n is the no. of divisions, triangles formed = 4 * 3^n

16. Name the rule which is followed for rotation of acube.

Right-handrule: The front faces are rotated in anti-clockwisedirection.
The back faces are rotated in clock-wise direction.

17. Differentiate between

a. GL_LINE_LOOP andGL_POLYGON
b. GL_QUADS andGL_POLYGON
c. GL_LINES andGL_LINE_LOOP

a. **GL_LINE_LOOP** : forms a closed path but not a solid(filled)polygon
GL_POLYGON: similar to GL_LINE_LOOP but the interior is filled according to the state of the relevant attributes.
b. **GL_QUADS** : successive group of 4 vertices are interpreted asquadrilaterals.
GL_POLYGON: any no. of vertices can be connected.
c. **GL_LINES:** Successive pairs of vertices would be connected as a line.
GL_LINE_LOOP : Connects the successive vertices using line segments to form a closedpath.

18. What is the use of GLUT_DOUBLE?

It is a bit mask to select a double buffered window. This overrides GLUT_SINGLE if it is also specified.

19. How do we swap thebuffers?

glClear(GL_DEPTH_BUFFER_BIT);glutSwapBuffers();

20. Name the type of graphic function of the following:
glutMouseFunc(mouse),glutKeyboardFunc(keys)

Input functions.

21. What is the rotation matrix in 2D about

(i) Origin **(ii) fixed (pivot)point**

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & m \\ \sin\theta & \cos\theta & n \\ 0 & 0 & 1 \end{pmatrix}$$

where
m =x-xcos(θ)+ysin(θ); n= y-xsin(θ)-ycos(θ);

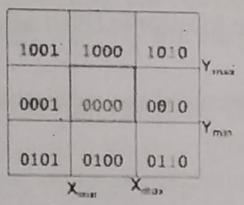
22. In Cohen-Sutherland line clipping algorithm, what is the condition for trivial accept and trivial reject?

Trivial accept:

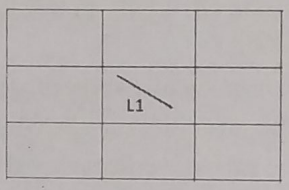
a. Both end points have to be in the region with code**0000.**
b. Trivial accept happens only if **code1 | code2 = 0000**

Trivial reject:

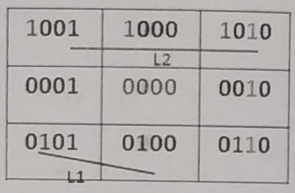
a. Codes of both end points will have **1** in the same bitposition.
b. Trivial reject happens only if **code1 & code2 != 0000**



Outcodes



Trivial accept



Trivial reject