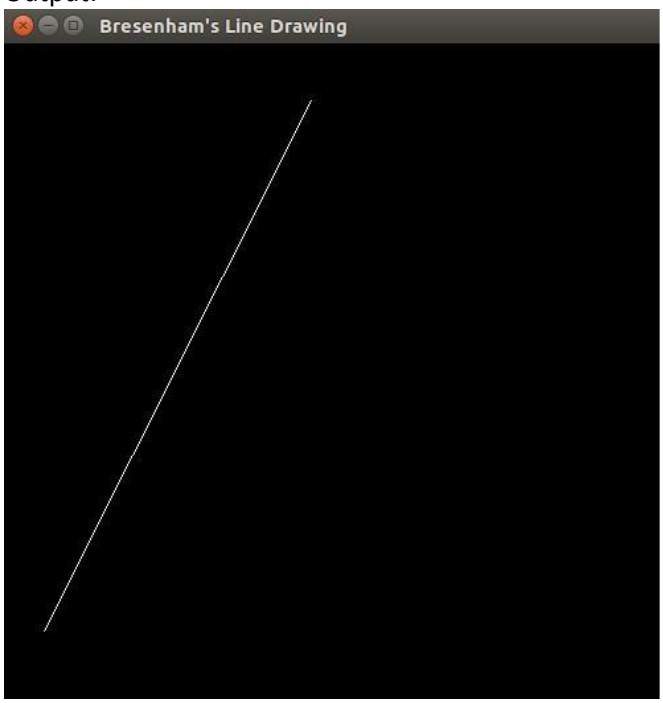
//Program 1: Implement Brenham's line drawing algorithm for all types of slope.

```
#include <GL/glut.h>
#include <stdio.h>
void myInit() {
	glClear(GL_COLOR_BUFFER_BIT);
	glClearColor(0.0, 0.0, 0.0, 0.0);
	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(0, 500, 0, 500);
}
void draw_pixel(int x, int y) {
	glBegin(GL_POINTS);
	glColor3f(0.0, 0.0, 1.0);
	glVertex2i(x, y);
	glEnd();
}
void draw_line(int x1, int x2, int y1, int y2)
{
	int dx, dy, i, e;
	int incx, incy, inc1, inc2;
	int x, y;
	dx = x2 - x1;
	dy = y2 - y1;
	incx = 1;
	 if (x2 < x1) incx = -1;
	incy = 1;
	if (y2 < y1) incy = -1;
	x = x1; y = y1;
	if (dx < 0) dx = -dx;
	if (dy < 0) dy = -dy;
	if (dx > dy) {

		draw_pixel(x, y);

		e = 2 * dy - dx;

		inc1 = 2 * (dy - dx);

		inc2 = 2 * dy;

		for (i = 0; i<dx; i++) {

			if (e >= 0) {

				y += incy;

				e += inc1;

			}

			else
```

```
                    e += inc2;

                x += incx;

                draw_pixel(x, y);

            }

        }
        else {

            draw_pixel(x, y);
            e = 2 * dx - dy;

            inc1 = 2 * (dx - dy);

            inc2 = 2 * dx;

            for (i = 0; i<dy; i++) {

                if (e >= 0) {

                    x += incx;

                    e += inc1;

                }

                else

                    e += inc2;

                y += incy;

                draw_pixel(x, y);

            }

        }

}

void myDisplay() {
        int x1, y1, x2, y2;

        x1 = 280; y1 = 340; x2 = 390; y2 = 480;
        draw_line(x1, x2, y1, y2);

        glFlush();

}
```

```
int main(int argc, char **argv) {

        glutInit(&argc, argv);

        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(500, 500);

        glutInitWindowPosition(0, 0);

        glutCreateWindow("Bresenham's Line Drawing");

        myInit();

        glutDisplayFunc(myDisplay);

        glutMainLoop();

        return 0;

}
```

Compile: gcc filename.c –lGL –lGLU -lglut
Run: ./a.out
Output:



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

```
#include<stdio.h>
#include<GL/glut.h>
int ch;
float angle,xf=50.0,yf=60.0;
void triangle()
{
glBegin(GL_TRIANGLES);
glVertex2i(250,250);
glVertex2i(400,250);
glVertex2i(325,270);
glEnd();
glFlush();
}
void rotate()
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,0.0,0.0);
glClearColor(0.0,0.0,0.0,1.0);
printf("Enter the choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("Enter the angle");
        scanf("%f",&angle);
        triangle();
        glRotatef(angle,0,0,1);
        triangle();
        break;
case 2:
        printf("Enter the angle");
        scanf("%f",&angle);
        printf("Enter the value for xf and yf");
        scanf("%f",&xf);
        scanf("%f",&yf);
        triangle();
        glTranslatef(xf,yf,0);
        glRotatef(angle,0,0,1);
        glTranslatef(-xf,-yf,0);
        triangle();
        break;
}
}

int main(int argc, char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(1000,500);
glutCreateWindow(argv[0]);
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0,800,0,500);
glutDisplayFunc(rotate);
glutMainLoop();
```

```
}
```

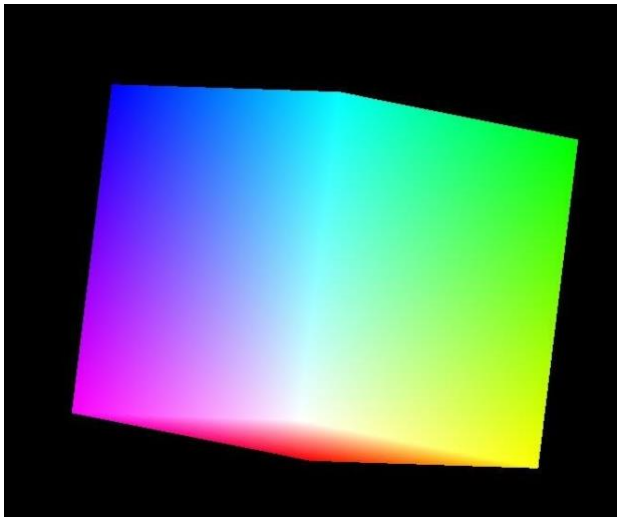
Output:

//Program 3: Draw a colour cube and spin it using OpenGL transformation matrices.

```
#include <stdlib.h>
#include <GL/glut.h>
GLfloat vertices[][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}};
GLfloat colors[][3] = {{1,0,0},{1,1,0},{0,1,0},{0,0,1},
{1,0,1},{1,1,1},{0,1,1},{0.5,0.5,0.5}};
void polygon(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();
}
void colorcube(void)
{
polygon(0,3,2,1);
polygon(0,4,7,3);
polygon(5,4,0,1);
polygon(2,3,7,6);
polygon(1,2,6,5);
polygon(4,5,6,7);
}
GLfloat theta[] = {0.0,0.0,0.0};
GLint axis = 2;
void display(void)
{
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();
glutSwapBuffers();
}
void spinCube()
{
theta[axis] += 1.0;
if( theta[axis] > 360.0 )
theta[axis] -= 360.0;
glutPostRedisplay();
}

void mouse(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;
}
void myReshape(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);
}
void main(int argc, char *argv[])
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
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:

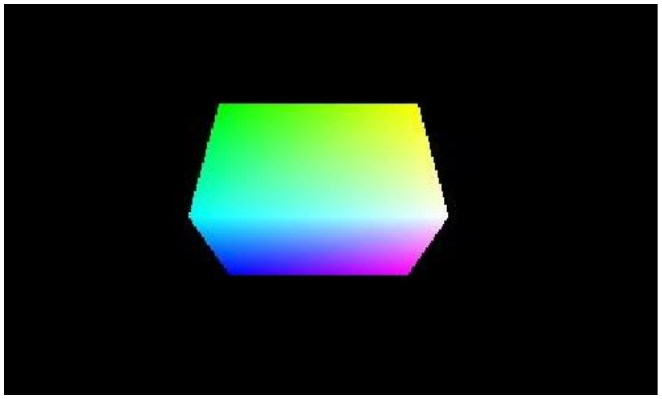


//Program 4: Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.

```
#include <stdlib.h>
#include <GL/glut.h>
GLfloat vertices[][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}};
GLfloat colors[][3] = {{1,0,0},{1,1,0},{0,1,0},{0,0,1},
{1,0,1},{1,1,1},{0,1,1},{0.5,0.5,0.5}};
void polygon(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();
}
void colorcube(void)
{
polygon(0,3,2,1);
polygon(0,4,7,3);
polygon(5,4,0,1);
polygon(2,3,7,6);
polygon(1,2,6,5);
polygon(4,5,6,7);
}
GLfloat theta[] = {0.0,0.0,0.0};
GLint axis = 2;
GLdouble viewer[]= {0.0, 0.0, 5.0}; /* initial viewer location */
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
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();
glutSwapBuffers();
}
void mouse(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();
}
void keys(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();
}
void myReshape(int w, int h)
{
glViewport(0, 0, w, h);
glMatrixMode(GL_PROJECTION);
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);
}
void main(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:



//Program 5: Clip a lines using Cohen-Sutherland algorithm.

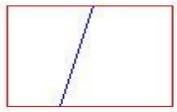
```
#include<stdio.h>
#include<GL/glut.h>
#include<stdbool.h>

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

//used to compute bit codes of a point
outcode ComputeOutCode (double x, double y);

//Cohen-Sutherland clipping algorithm clips a line from
//P0 = (x0, y0) to P1 = (x1, y1) against a rectangle with
//diagonal from (xmin, ymin) to (xmax, ymax).
void CohenSutherlandLineClipAndDraw (double x0, double y0,double x1,
double y1)
{
      //Outcodes for P0, P1, and whatever point lies outside the
clip rectangle
      outcode outcode0, outcode1, outcodeOut;
      bool accept = false, done = false;

      //compute outcodes
      outcode0 = ComputeOutCode (x0, y0);
      outcode1 = ComputeOutCode (x1, y1);

      do{
            if (!(outcode0 | outcode1))      //logical or is 0
Trivially accept & exit
            {
                  accept = true;
                  done = true;
            }
```

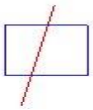
```
           else if (outcode0 & outcode1)   //logical and is not 0.
Trivially reject and exit
                 done = true;
           else
           {
                 //failed both tests, so calculate the line segment
to clip
                 //from an outside point to an intersection with clip
edge
                 double x, y;

           //At least one endpoint is outside the clip rectangle;
pick it.
                 outcodeOut = outcode0? outcode0: outcode1;

                 //Now find the intersection point;
     //use formulas y = y0 + slope * (x - x0), x = x0 + (1/slope)*
(y - y0)
                 if (outcodeOut & TOP)     //point is above the clip
rectangle
                 {
                       x = x0 + (x1 - x0) * (ymax - y0)/(y1 - y0);
                       y = ymax;
                 }
                 else if (outcodeOut & BOTTOM)  //point is below the
clip rectangle
                 {
                       x = x0 + (x1 - x0) * (ymin - y0)/(y1 - y0);
                       y = ymin;
                 }
                 else if (outcodeOut & RIGHT)   //point is to the
right of clip rectangle
                 {
                       y = y0 + (y1 - y0) * (xmax - x0)/(x1 - x0);
                       x = xmax;
                 }
                 else                           //point is to the
left of clip rectangle
                 {
                       y = y0 + (y1 - y0) * (xmin - x0)/(x1 - x0);
                       x = xmin;
                 }

                 //Now we move outside point to intersection point to
clip
                 //and get ready for next pass.
                 if (outcodeOut == outcode0)
                 {
                       x0 = x;
                       y0 = y;
                       outcode0 = ComputeOutCode (x0, y0);
                 }
                 else
                 {
                       x1 = x;
                       y1 = y;
```

```
                        outcode1 = ComputeOutCode (x1, y1);
                  }
            }
      }while (!done);

      if (accept)
      {                 // Window to viewport mappings
            double sx=(xvmax-xvmin)/(xmax-xmin); // Scale parameters
            double sy=(yvmax-yvmin)/(ymax-ymin);
            double vx0=xvmin+(x0-xmin)*sx;
            double vy0=yvmin+(y0-ymin)*sy;
            double vx1=xvmin+(x1-xmin)*sx;
            double vy1=yvmin+(y1-ymin)*sy;
                  //draw a red colored viewport
            glColor3f(1.0, 0.0, 0.0);
            glBegin(GL_LINE_LOOP);
                  glVertex2f(xvmin, yvmin);
                  glVertex2f(xvmax, yvmin);
                  glVertex2f(xvmax, yvmax);
                  glVertex2f(xvmin, yvmax);
            glEnd();
            glColor3f(0.0,0.0,1.0); // draw blue colored clipped line
            glBegin(GL_LINES);
                  glVertex2d (vx0, vy0);
                  glVertex2d (vx1, vy1);
            glEnd();
      }
}

//Compute the bit code for a point (x, y) using the clip rectangle
//bounded diagonally by (xmin, ymin), and (xmax, ymax)
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 clip window
            code |= LEFT;
      return code;
}

void display()
{


double x0=60,y0=20,x1=80,y1=120;
glClear(GL_COLOR_BUFFER_BIT);
//draw the line with red color
glColor3f(1.0,0.0,0.0);
//bres(120,20,340,250);
glBegin(GL_LINES);
                  glVertex2d (x0, y0);
```

```
                glVertex2d (x1, y1);
            glEnd();

//draw a blue colored window
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();
}
void myinit()
{
      glClearColor(1.0,1.0,1.0,1.0);
      glColor3f(1.0,0.0,0.0);
      glPointSize(1.0);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(0.0,499.0,0.0,499.0);
}
void main(int argc, char** argv)
{
      //int x1, x2, y1, y2;
      //printf("Enter End points:");
      //scanf("%d%d%d%d", &x1,&x2,&y1,&y2);

      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
      glutInitWindowSize(500,500);
      glutInitWindowPosition(0,0);
      glutCreateWindow("Cohen Suderland Line Clipping Algorithm");
      glutDisplayFunc(display);
      myinit();
      glutMainLoop();  }
```

Output:



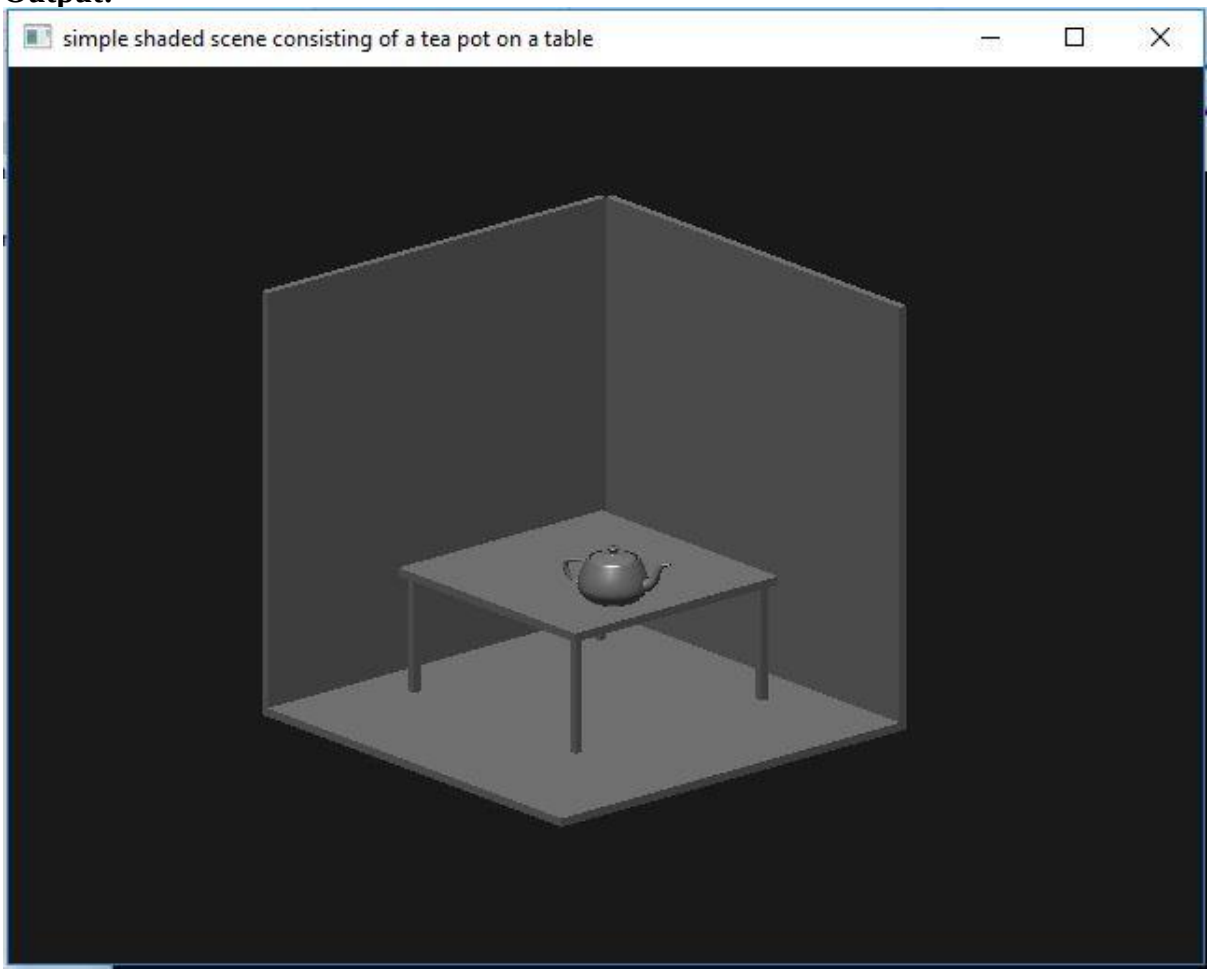


//Program 6: To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene.

```
#include<stdlib.h>
#include<stdio.h>
#include<GL/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(12, 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, 0.01);//to Draw 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);//to draw legs of table
      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);//To draw table surface
      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 myinit()
{
      glClearColor(1.0, 1.0, 1.0, 1.0);
      glPointSize(1.0);
      glMatrixMode(GL_PROJECTION);
```

```
	glLoadIdentity();
	gluOrtho2D(0.0, 499.0, 0.0, 499.0);
}
void main(int argc, char **argv)
{
	float ambient[] = { 1, 1, 1, 1 };
	float light_pos[] = { 27, 80, 2, 3 };
	glutInit(&argc, argv);
	glutCreateWindow("simple shaded scene consisting of tea pot on
a table");
	glutInitWindowSize(700, 700);
	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:



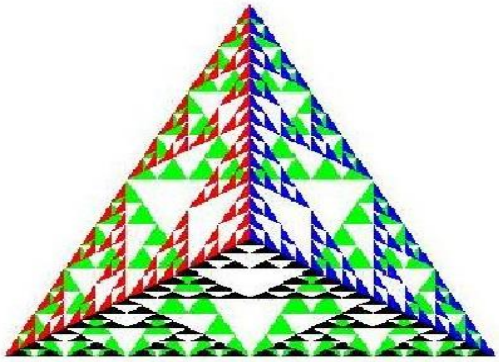
//Program 7: Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user.

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

typedef float point[3];

/* initial tetrahedron */

point v[]={{0.0, 0.0, 1.0}, {0.0, 0.942809, -0.33333},
          {-0.816497, -0.471405, -0.333333}, {0.816497, -0.471405, -
0.333333}};

//static GLfloat theta[] = {0.0,0.0,0.0};

int n;

void triangle( point a, point b, point c)

/* display one triangle using a line loop for wire frame, a single
normal for constant shading, or three normals for interpolative
shading */
{
    glBegin(GL_POLYGON);
       //glNormal3fv(a);
       glVertex3fv(a);
       glVertex3fv(b);
       glVertex3fv(c);
    glEnd();
}




void divide_triangle(point a, point b, point c, int m)
{

/* triangle subdivision using vertex numbers
righthand rule applied to create outward pointing faces */

    point v1, v2, v3;
    int j;
    if(m>0)
    {
        for(j=0; j<3; j++) v1[j]=(a[j]+b[j])/2;
        for(j=0; j<3; j++) v2[j]=(a[j]+c[j])/2;
        for(j=0; j<3; j++) v3[j]=(b[j]+c[j])/2;
        divide_triangle(a, v1, v2, m-1);
        divide_triangle(c, v2, v3, m-1);
        divide_triangle(b, v3, v1, m-1);
    }
    else(triangle(a,b,c)); /* draw triangle at end of recursion */
}
```

```
void tetrahedron( int m)
{

/* Apply triangle subdivision to faces of tetrahedron */

      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);
}

void display(void)
{


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


void myReshape(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);
    glutPostRedisplay();
}


void main(int argc, char **argv)
{
    //n=atoi(argv[1]);
      printf(" No. of Divisions ? ");
      scanf("%d",&n);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("3D Gasket");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
```

```
      glEnable(GL_DEPTH_TEST);
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glutMainLoop();
}
```

Output:



//Program 8: Develop a menu driven program to animate a flag using Bezier Curve algorithm

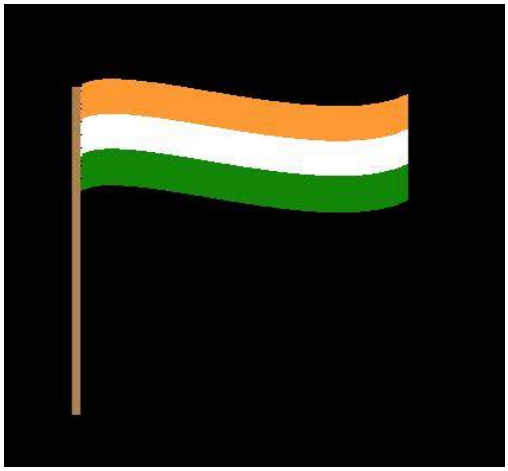
//Note: This is C++ Program, Save the file with Extension cpp.

```
#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
#define p1 3.1416
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 bezBlendFunc;
	bezPt->x = bezPt->y = bezPt->z = 0.0;
	for (k = 0; k<nctrlPts; k++)
	{
		bezBlendFunc = c[k] * pow(u, k)*pow(1 - u, n - k);
		bezPt->x += ctrlPts[k].x*bezBlendFunc;
		bezPt->y += ctrlPts[k].y*bezBlendFunc;
		bezPt->z += ctrlPts[k].z*bezBlendFunc;
	}
}
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 displayFunc()
{
	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*p1 / 180.0);
	ctrlPts[1].y += 5 * sin(theta*p1 / 180.0);
	ctrlPts[2].x -= 10 * sin((theta + 30)*p1 / 180.0);
	ctrlPts[2].y -= 10 * sin((theta + 30)*p1 / 180.0);
	ctrlPts[3].x -= 4 * sin((theta)*p1 / 180.0);
	ctrlPts[3].y += sin((theta - 30)*p1 / 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);
	for (int i = 0; i<8; i++)
	{
		glTranslatef(0, -0.8, 0);
		bezier(ctrlPts, nctrlPts, nBezcurvePts);
	}
	glColor3f(1, 1, 1);
	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);
	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();
}

void winReshapeFunc(GLint newWidth, GLint newHeight)
{
	glViewport(0, 0, newWidth, newHeight);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0.0,130.0,0.0,130.0);
	glClear(GL_COLOR_BUFFER_BIT);
}
void fillMenu(int option)
{
	if (option == 1)
		glutDisplayFunc(displayFunc);
	if (option == 2)
		exit(0);


}

int main(int argc, char **argv)
{

	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
	glutInitWindowPosition(50, 50);
	glutInitWindowSize(600, 600);
	glutCreateWindow("BEZIER CURVE");
	glutCreateMenu(fillMenu);
	glutAddMenuEntry("Animate Flag", 1);
	glutAddMenuEntry("Stop", 2);
	glutAttachMenu(GLUT_RIGHT_BUTTON);
	glutReshapeFunc(winReshapeFunc);
	glutMainLoop();
	return 0;
}
```

Output:



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

```
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>
int fillFlag=0;
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)
{
glColor3f(1.0,1.0,0.0);
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
}
```

```
void scanfill(float x1,float y1,float x2,float y2,float x3,float
y3,float x4,float y4)
{
int i,y;
int le[500],re[500];
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++)
{
for(i=(int)le[y];i<(int)re[y];i++)
draw_pixel(i,y);
}
}
void display()
{
float x1,x2,x3,x4,y1,y2,y3,y4;
x1=200.0;y1=200.0;x2=100.0;y2=300.0;x3=200.0;y3=400.0;x4=300.0;y4=30
0.0;
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();
if(fillFlag==1)
scanfill(x1,y1,x2,y2,x3,y3,x4,y4);
glFlush();
}

void init()
{
glClearColor(0.0,0.0,0.0,1.0);
glColor3f(1.0,0.0,0.0);
glPointSize(1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,499.0,0.0,499.0);
}
void fillMenu(int option)
{
if(option==1)
fillFlag=1;
if(option==2)
fillFlag=2;
display();
}
```

```
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 Scan-line Algorithm");
init();
glutDisplayFunc(display);
glutCreateMenu(fillMenu);
glutAddMenuEntry("Fill Polygon",1);
glutAddMenuEntry("Empty Polygon",2);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glutMainLoop();
}
```

Output: