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 colour 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

**Program 1: Implement Brenham’s line drawing algorithm for all types Of slope**

PROGRAM

# include<gl/glut.h>

# include <stdio.h>

int x1, y1, x2, y2;

void myInit()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(0.0, 0.0, 0.0, 1.0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, 500, 0, 500);

}

void draw\_pixel(int x, int y)

{

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

}

void draw\_line(int x1, int x2, int y1, int y2)

{

int dx, dy, i, e;

intincx, incy, inc1, inc2;

intx,y;

dx = x2-x1;

dy = y2-y1;

if (dx < 0) dx = -dx;

if (dy< 0) dy = -dy;

incx = 1;

if (x2 < x1)

incx = -1;

incy = 1;

if (y2 < y1) incy = -1;

x = x1; y = y1;

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() {

draw\_line(x1, x2, y1, y2);

glFlush();

}

void main(intargc, char \*\*argv) {

printf( "Enter (x1, y1, x2, y2)\n");

scanf("%d %d %d %d", &x1, &y1, &x2, &y2);

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

glutCreateWindow("Bresenham's Line Drawing");

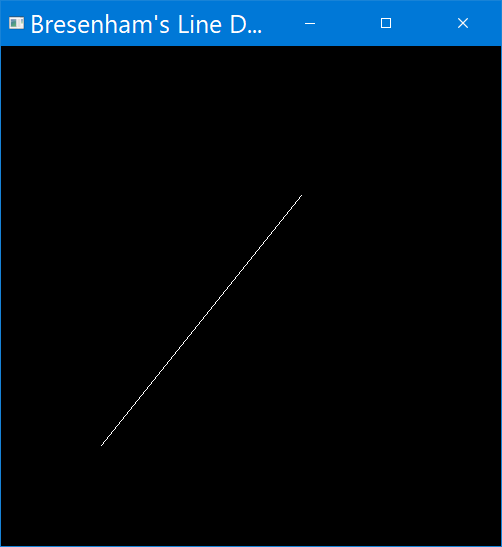
myInit();

glutDisplayFunc(myDisplay);

glutMainLoop();

}

OUTPUT



**Program-2 To create and rotate a triangle about the origin and a fixed point**

PROGRAM

#include<GL/glut.h>

#include<stdio.h>

int x, y;

int rFlag = 0;

void draw\_pixel(float x1, float y1)

{ glColor3f(0.0, 0.0, 1.0);

glPointSize(5.0);

glBegin(GL\_POINTS);

glVertex2f(x1, y1);

glEnd(); }

void triangle()

{ glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON);

glVertex2f(100, 100);

glVertex2f(250, 400);

glVertex2f(400, 100);

glEnd(); }

float th = 0.0;

float trX = 0.0,

trY = 0.0;

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

if (rFlag == 1) //Rotate Around origin

{ trX=0.0;

trY=0.0; th+=0.1;

draw\_pixel(0.0,0.0); }

if(rFlag==2) //Rotate Around Fixed Point

{ trX=x; trY=y; th+=0.1;

draw\_pixel(x,y); }

glTranslatef(trX,trY,0.0);

glRotatef(th,0.0,0.0,1.0);

glTranslatef(-trX,-trY,0.0);

triangle();

glutPostRedisplay();

glutSwapBuffers(); }

void myInit()

{ glClearColor(0.0, 0.0, 0.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-500.0, 500.0, -500.0, 500.0);

glMatrixMode(GL\_MODELVIEW); }

void rotateMenu(int option)

{ if (option == 1)

rFlag = 1;

if (option == 2)

rFlag = 2;

if (option == 3)

rFlag = 3; }

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

{ printf( "Enter Fixed Points (x,y) for Roration: \n");

scanf("%d %d", &x, &y); glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB); glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

glutCreateWindow("Create and Rotate Triangle");

myInit(); glutDisplayFunc(display);

glutCreateMenu(rotateMenu);

glutAddMenuEntry("Rotate around ORIGIN",1);

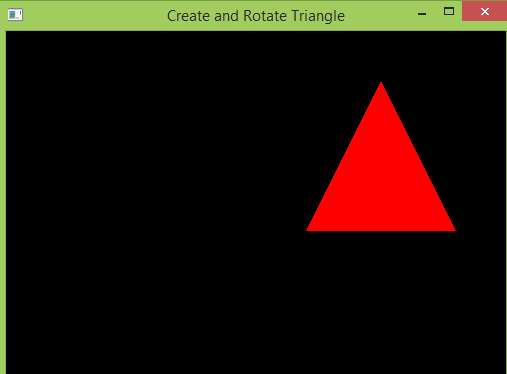
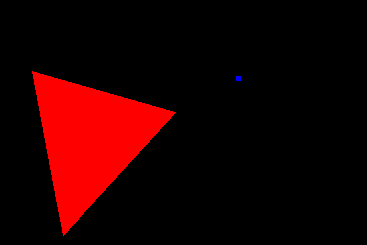
glutAddMenuEntry("Rotate around FIXED POINT",2);

glutAddMenuEntry("Stop Rotation",3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop(); }

OUTPUT

 ****

**LAB PROGRAM -3**

**Program to draw a color cube and spin it using OpenGL transformation matrices.**

DESCRIPTION

Draw the cube from list of indices into the array vertices and use corresponding color to the first index. Define display function to rotate the cube, draw and swap.Define the spincube function to spin cube 2 degrees about selected axis, mouse callback to select axis about which to rotate and reshape function to maintain aspect ratio.

PROGRAM

#include <stdlib.h>

#include <GL/glut.h>

GLfloatvertices[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};

GLfloatnormals[] = {-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};

GLfloatcolors[] = {0.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,0.0, 0.0,1.0,0.0, 0.0,0.0,1.0,

1.0,0.0,1.0, 1.0,1.0,1.0, 0.0,1.0,1.0};

GLubytecubeIndices[]={0,3,2,1,2,3,7,6,0,4,7,3,1,2,6,5,4,5,6,7,0,1,5,4};

static GLfloattheta[] = {0.0,0.0,0.0};

static GLint axis = 2;

void display(void)

{/\* display callback, clear frame buffer and z buffer, rotate cube and draw, swap buffers \*/

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);

glDrawElements(GL\_QUADS, 24, GL\_UNSIGNED\_BYTE, cubeIndices);

glBegin(GL\_LINES);

glVertex3f(0.0,0.0,0.0);

glVertex3f(1.0,1.0,1.0);

glEnd();

glFlush();

glutSwapBuffers();

}

void spinCube()

{/\* Idle callback, spin cube 2 degrees about selected axis \*/

theta[axis] += 0.5;

if( theta[axis] > 360.0 ) theta[axis] -= 360.0;

glutPostRedisplay();

}

void mouse(intbtn, int state, int x, int y)

{/\* 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;

}

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);

}

voidmain(intargc, char \*\*argv)

{/\* need both double buffering and z buffer \*/

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutCreateWindow("colorcube");

glutReshapeFunc(myReshape);

glutDisplayFunc(display);

glutIdleFunc(spinCube);

glutMouseFunc(mouse);

glEnable(GL\_DEPTH\_TEST); /\* Enable hidden--surface--removal \*/

glEnableClientState(GL\_COLOR\_ARRAY);

glEnableClientState(GL\_NORMAL\_ARRAY);

glEnableClientState(GL\_VERTEX\_ARRAY);

glVertexPointer(3, GL\_FLOAT, 0, vertices);

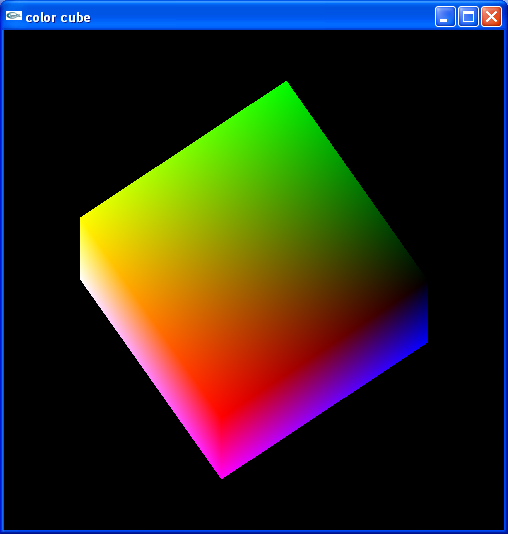
glColorPointer(3,GL\_FLOAT, 0, colors);

glNormalPointer(GL\_FLOAT,0, normals);

glColor3f(1.0,1.0,1.0);

glutMainLoop();}

OUTPUT

****

**LAB PROGRAM -4**

**Program to draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing. Use OpenGL functions.**

PROGRAM

#include <GL/glut.h>

float v[]={-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[]={0,0,0, 1,0,0, 1,1,0, 0,1,0, 0,0,1, 1,0,1, 1,1,1, 0,1,1,};

GLubytelist[]={0,1,2,3, 2,3,7,6, 4,5,6,7, 4,5,1,0, 5,6,2,1, 0,3,7,4};

intgx=0,gy=0,gz=1;

static GLfloattheta[] = {0.0,0.0,0.0};

static GLint axis = 2;

static GLdouble viewer[]= {0.0, 0.0, 5.0};

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);

glDrawElements(GL\_QUADS,24,GL\_UNSIGNED\_BYTE,list);

glFlush();

}

void mouse(intbtn, 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 main()

{ glutInitWindowSize(700, 700);

glutCreateWindow("Colorcube Viewer");

glMatrixMode(GL\_PROJECTION);

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

glMatrixMode(GL\_MODELVIEW);

glutDisplayFunc(display);

glEnableClientState(GL\_VERTEX\_ARRAY);

glEnableClientState(GL\_COLOR\_ARRAY);

glVertexPointer(3,GL\_FLOAT,0,v);

glColorPointer(3,GL\_FLOAT,0,c);

glutMouseFunc(mouse);

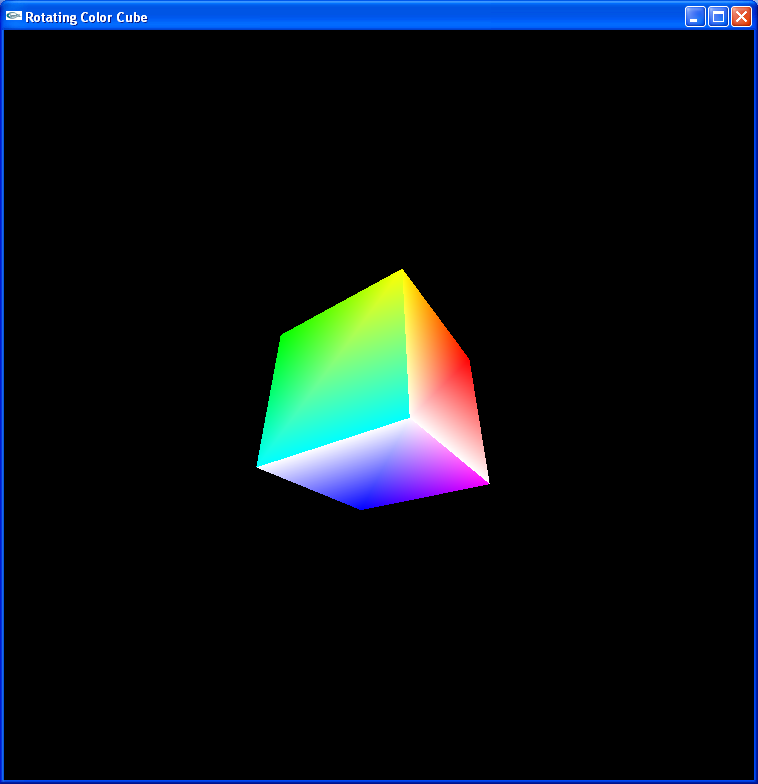
glutKeyboardFunc(keys);

glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

}

OUTPUT



**LAB PROGRAM -5**

**Program to implement the Cohen-Sutherland line-clipping algorithm. Make provision to specify the input line, window for clipping and view port for displaying the clipped image.**

PROGRAM

#include <stdio.h>

#include <GL\glut.h>

double xmin=50,ymin=50, xmax=100,ymax=100;

double xvmin=200,yvmin=200,xvmax=300,yvmax=300;

const int RIGHT = 8; const int LEFT = 2;

const int TOP = 4; const int BOTTOM = 1;

int ComputeOutCode (double x, double y)

{ int code = 0; if (y > ymax)

code |= TOP;

//above the clip window

else if (y < ymin)

code |= BOTTOM;

//below the clip window

if (x > xmax)

code |= RIGHT;

//to the right of clip window

else if (x < xmin)

code |= LEFT;

//to the left of clip window

return code; }

void CohenSutherland(double x0, double y0,double x1, double y1)

{ int outcode0, outcode1, outcodeOut;

int accept = 0, done = 0;

outcode0 = ComputeOutCode (x0, y0);

outcode1 = ComputeOutCode (x1, y1);

do{ if (!(outcode0 | outcode1))

{ accept = 1; done = 1; }

else if (outcode0 & outcode1) done = 1;

else { double x, y;

outcodeOut = outcode0? outcode0: outcode1;

if (outcodeOut & TOP)

{ x = x0 + (x1 - x0) \* (ymax - y0) / (y1 - y0);

y = ymax; }

else if (outcodeOut & BOTTOM)

{ x = x0 + (x1 - x0) \* (ymin - y0) / (y1 - y0);

y = ymin; }

else if (outcodeOut & RIGHT)

{ y = y0 + (y1 - y0) \* (xmax - x0) / (x1 - x0);

x = xmax; }

else

{ y = y0 + (y1 - y0) \* (xmin - x0) / (x1 - x0);

x = xmin; }

if (outcodeOut == outcode0)

{ x0 = x; y0 = y;

outcode0 = ComputeOutCode(x0, y0); }

else { x1 = x; y1 = y;

outcode1 = ComputeOutCode(x1, y1); }

}

} while (!done);

if (accept)

{ double sx = (xvmax - xvmin) / (xmax - xmin);

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;

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xvmin, yvmin);

glVertex2f(xvmax, yvmin);

glVertex2f(xvmax, yvmax);

glVertex2f(xvmin, yvmax);

glEnd(); glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_LINES);

glVertex2d(vx0, vy0);

glVertex2d(vx1, vy1);

glEnd(); }

}

void display()

{

double x0 = 60, y0 = 20, x1 = 80, y1 = 120;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_LINES); glVertex2d(x0, y0);

glVertex2d(x1, y1); glEnd();

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xmin, ymin);

glVertex2f(xmax, ymin);

glVertex2f(xmax, ymax);

glVertex2f(xmin, ymax);

glEnd();

CohenSutherland(x0, y0, x1, y1);

glFlush();

}

void myinit()

{ glClearColor(0.0, 0.0, 0.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, 500.0, 0.0, 500.0);

glMatrixMode(GL\_MODELVIEW); }

void main(int argc, char\*\* argv)

{ glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

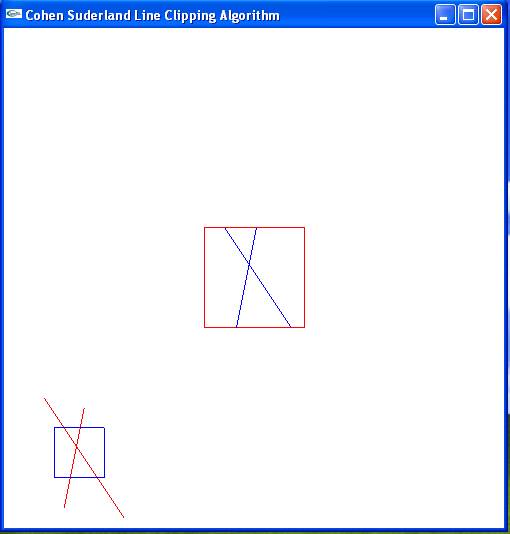
glutCreateWindow("Cohen Suderland Line Clipping Algorithm");

myinit();

glutDisplayFunc(display);

glutMainLoop(); }

OUTPUT



**LAB PROGRAM -6**

**Program, using OpenGL functions, 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 properties of the surfaces of the solid object used in the scene.**

PROGRAM

#include<GL/glut.h>

void obj(double tx, doublety, doubletz, doublesx, doublesy, doublesz)

{ 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);

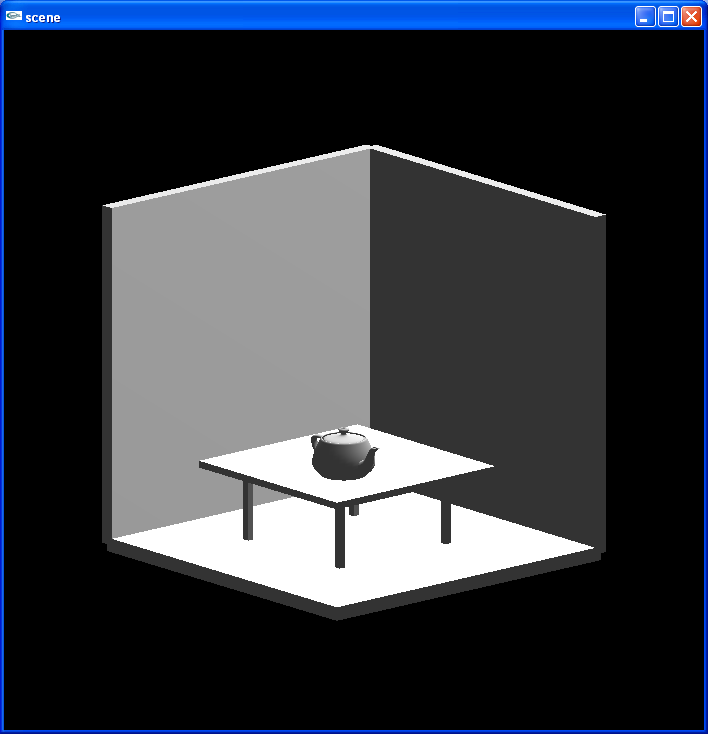
glLightfv(GL\_LIGHT0,GL\_POSITION,light\_pos);

glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

}

OUTPUT



**LAB PROGRAM -7**

**Program to recursively subdivide a tetrahedron to from 3D Sierpinski gasket. The number of recursive steps is to be specified by the user.**

PROGRAM

#include <GL/glut.h>

/\* initial triangle \*/

typedef GLfloat point[3];

point v[]={{30.0, 50.0, 100.0}, {0.0, 450.0, -150.0},

{-350.0, -400.0, -150.0}, {350., -400., -150.0}};

int n; /\* number of recursive steps \*/

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

{ glBegin(GL\_TRIANGLES); /\* display one triangle \*/

glVertex3fv(a);

glVertex3fv(b);

glVertex3fv(c);

glEnd();

}

void tetra(point a, point b, point c, point d)

{ glColor3f(1.0,0.0,0.0);

triangle(a, b ,c);

glColor3f(0.0,1.0,0.0);

triangle(a, c, d);

glColor3f(0.0,0.0,1.0);

triangle(a, d, b);

glColor3f(0.0,0.0,0.0);

triangle(b, d, c);

}

void divide\_tetra(point a, point b, point c, point d, int m)

{

/\* tretra subdivision using vertex numbers \*/

GLfloatmid[6][3];

int j;

if(m > 0)

{

for(j=0; j<3; j++) mid[0][j] = (a[j] + b[j]) / 2;

for(j=0; j<3; j++) mid[1][j] = (a[j] + c[j]) / 2;

for(j=0; j<3; j++) mid[2][j] = (a[j] + d[j]) / 2;

for(j=0; j<3; j++) mid[3][j] = (b[j] + c[j]) / 2;

for(j=0; j<3; j++) mid[4][j] = (c[j] + d[j]) / 2;

for(j=0; j<3; j++) mid[5][j] = (b[j] + d[j]) / 2;

divide\_tetra(a, mid[0], mid[1], mid[2], m-1);

divide\_tetra(mid[0], b, mid[3], mid[5], m-1);

divide\_tetra(mid[1], mid[3], c, mid[4], m-1);

divide\_tetra(mid[2], mid[5], mid[4],d, m-1);

}

else tetra(a, b, c, d);}

void display()

{ glEnable(GL\_DEPTH\_TEST);

glClearColor (1.0, 1.0, 1.0,1.0);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

divide\_tetra(v[0], v[1], v[2], v[3], n);

glFlush();}

void myinit()

{ glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-499.0, 499.0, -499.0, 499.0,-499.0,499.0);

glMatrixMode(GL\_MODELVIEW); }

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

{

printf("\n Enter the number of division:\n");

scanf("%d",&n);

glutInit(&, argv);

glutInitDisplayMode(GLUT\_SINGLE| GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutCreateWindow("3D gasket");

glEnable(GL\_DEPTH\_TEST);

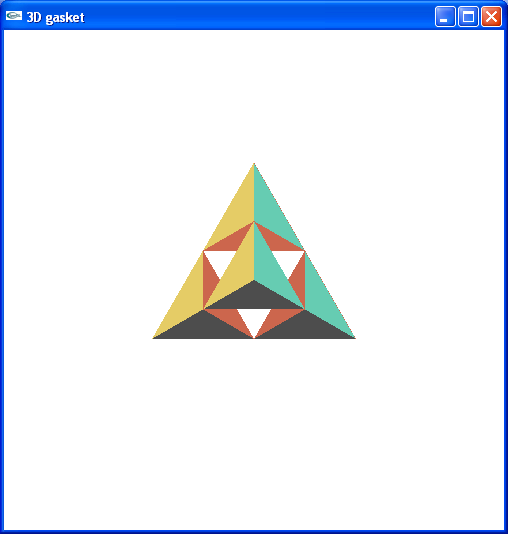
myinit();

glutDisplayFunc(display);

glutMainLoop();

}

OUTPUT

****

**LAB PROGRAM -8**

**Program to develop a menu driven program to animate a flag using Bezier curve algorithm.**

PROGRAM

#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()

{

GLintnCtrlPts = 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);

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(inti=0;i<8;i++)

{

glTranslatef(0, -0.8, 0);

bezier(ctrlPts, nCtrlPts, nBezCurvePts);

}

glColor3f(1, 1, 1); //Indian flag: white color code

for(inti=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(inti=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 winReshapeFun(GLintnewWidth, GLintnewHeight)

{

glViewport(0, 0, newWidth, newHeight);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);

glClear(GL\_COLOR\_BUFFER\_BIT);

}

void main(int argc, char \*\*argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowPosition(50, 50);

glutInitWindowSize(winWidth, winHeight);

glutCreateWindow("Bezier Curve");

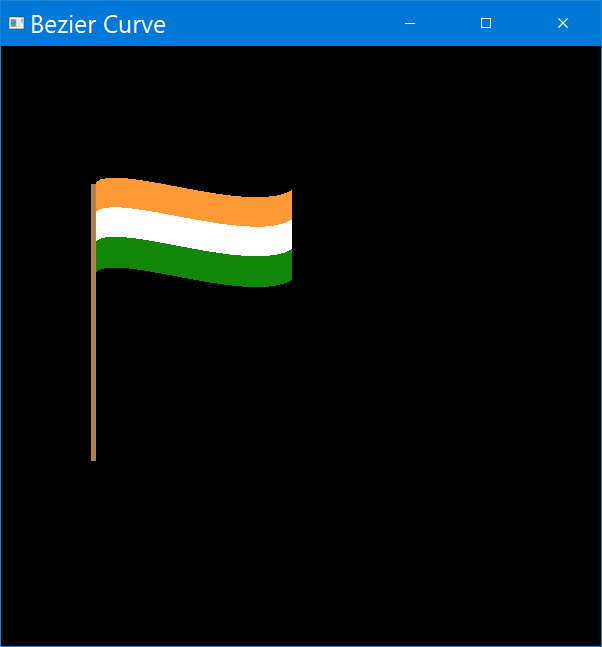
glutDisplayFunc(displayFcn);

glutReshapeFunc(winReshapeFun);

glutMainLoop();

}

OUTPUT

**LAB PROGRAM -9**

**Program to fill any given polygon using scan-line area filling algorithm. (Use appropriate data structures.)**

PROGRAM

#include <GL/glut.h>

#include <stdlib.h>

float x1,x2,x3,x4,y1,y2,y3,y4;

void edgedetect(float x1,float y1,float x2,float y2,int \*le,int \*re)

{ float mx,x,temp;

inti;

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(intx,int y)

{

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 le[500],re[500];

inti,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()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

}

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 select(int ch)

{

switch(ch)

{

case 1:

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

glColor3f(1.0, 0.0, 0.0);

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();

glutPostRedisplay();

break;

case 2:

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

glColor3f(0.0, 1.0, 0.0);

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();

glutPostRedisplay();

break;

case 3:

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

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();

glutPostRedisplay();

break;

}

}

void main(int argc, char\*\* argv)

{ glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(500,500);

glutInitWindowPosition(0,0);

glutCreateWindow("scan");

myinit();

glutDisplayFunc(display);

glutCreateMenu(select);

glutAddMenuEntry("Red",1);

glutAddMenuEntry("Green",2);

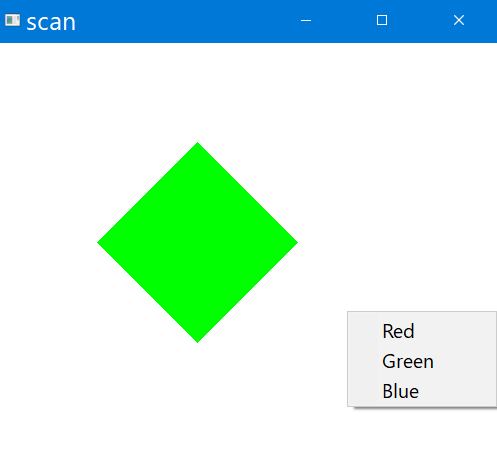
glutAddMenuEntry("Blue",3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

}

**OUTPUT**

****

s