**EXPERIMENT - 01**

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

#include<GL/glut.h>

#include<stdio.h>

intx1,y1,x2,y2;

voiddraw\_pixel(intx,inty)

{

glColor3f(1.0,0.0,0.0);glBegin(GL\_POINTS);glVertex2i(x,y);glEnd();

}

voidbrenhams\_line\_draw(intx1,inty1,intx2,inty2)

{

intdx=x2-x1,dy=y2-y1;intp=2\*dy\*dx;inttwoDy=2\*dy;inttwoDyMinusDx=2\*(dy-dx);// paranthesis are requiredintx=x1,y=y1;

if(dx<0)

{

x=x2;y=y2;

x2=x1;

}

draw\_pixel(x,y);

while(x<x2)

{

x++;

if(p<0)

p+=twoDy;else

{

y++;

p+=twoDyMinusDx;

}

draw\_pixel(x,y);

}

}

voidmyInit()

{

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);

}

voiddisplay()

{

glClear(GL\_COLOR\_BUFFER\_BIT);brenhams\_line\_draw(x1,y1,x2,y2);

glFlush();

}

voidmain(intargc,char\*\*argv)

{

printf("Enter Start Points (x1,y1)\n");scanf("%d %d",&x1,&y1); printf("Enter End Points (x2,y2)\n");scanf("%d %d",&x2,&y2);

glutInit(&argc,argv);glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

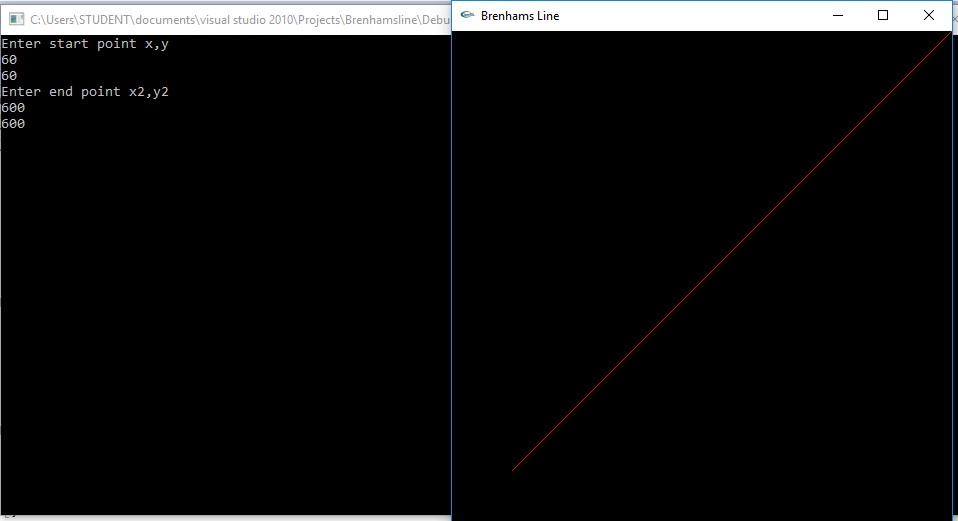
glutInitWindowSize(500,500);glutInitWindowPosition(0,0);

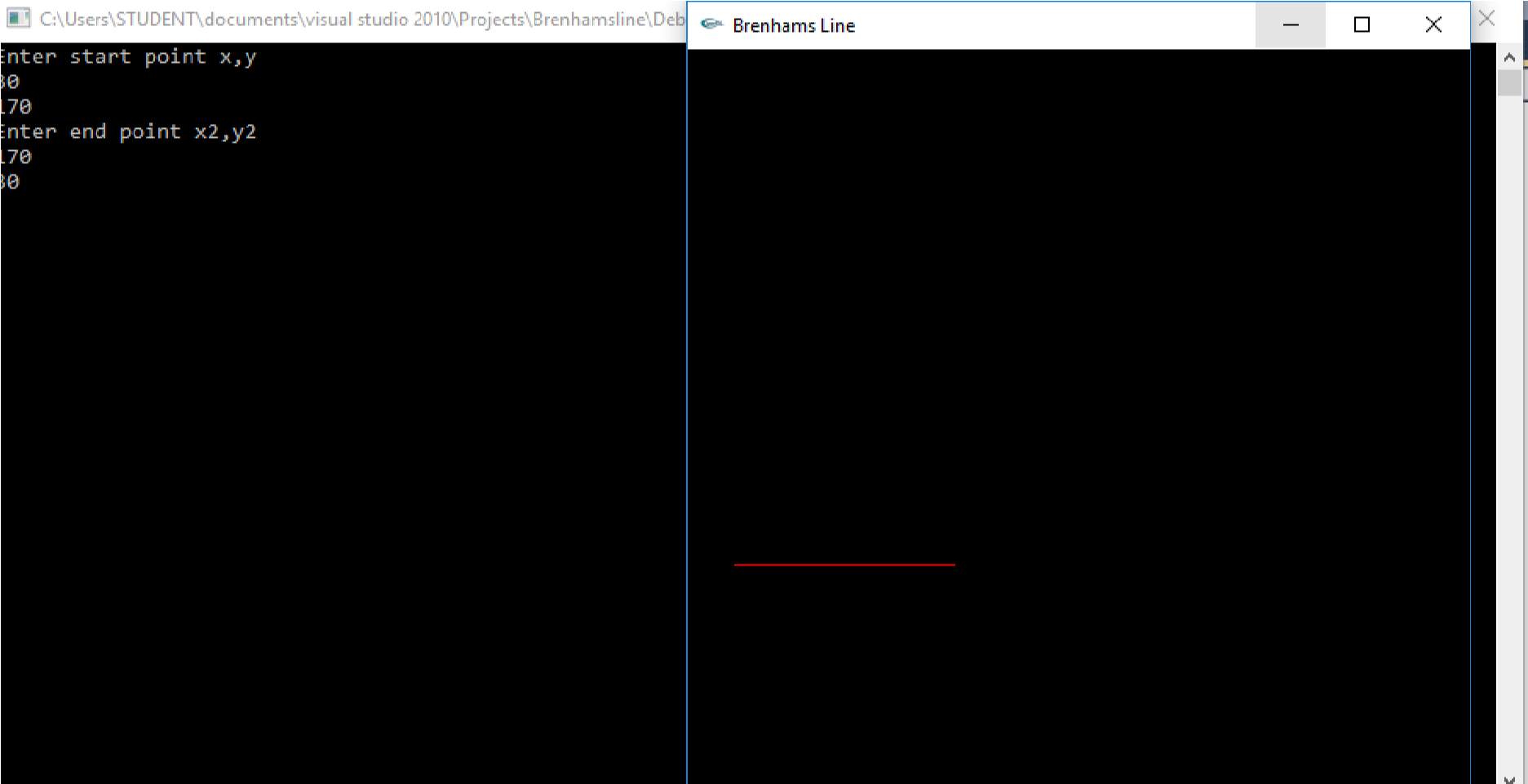
glutCreateWindow("Bresenham's Line Drawing");myInit();glutDisplayFunc(display);

glutMainLoop();

}

**OUTPUT:**

****

****

**EXPERIMENT - 02**

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

#include<GL/glut.h>

#include<stdio.h>

intx,y;

intrFlag=0;

voiddraw\_pixel(floatx1,floaty1)

{

glColor3f(0.0,0.0,1.0);glPointSize(5.0);glBegin(GL\_POINTS);glVertex2f(x1,y1);glEnd();

}

voidtriangle()

{

glColor3f(1.0,0.0,0.0);glBegin(GL\_POLYGON);glVertex2f(100,100);glVertex2f(250,400);glVertex2f(400,100);glEnd();

}

floatth=0.0;floattrX=0.0,trY=0.0;

voiddisplay()

{

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();

}

voidmyInit()

{

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);

}

voidrotateMenu(intoption)

{

if(option==1)rFlag=1;if(option==2)rFlag=2;if(option==3)

rFlag=3;

}

voidmain(intargc,char\*\*argv)

{

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

scanf("%d %d",&x,&y);

glutInit(&argc,argv);

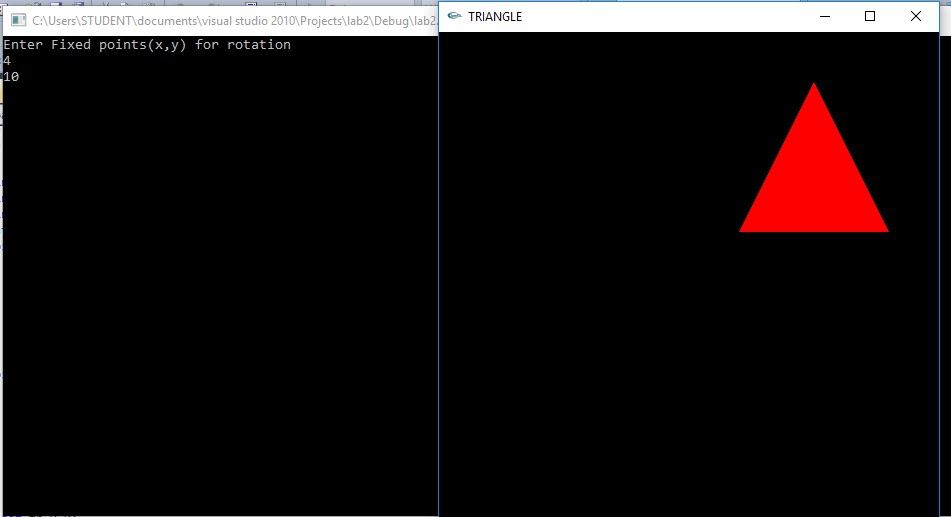
glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB);

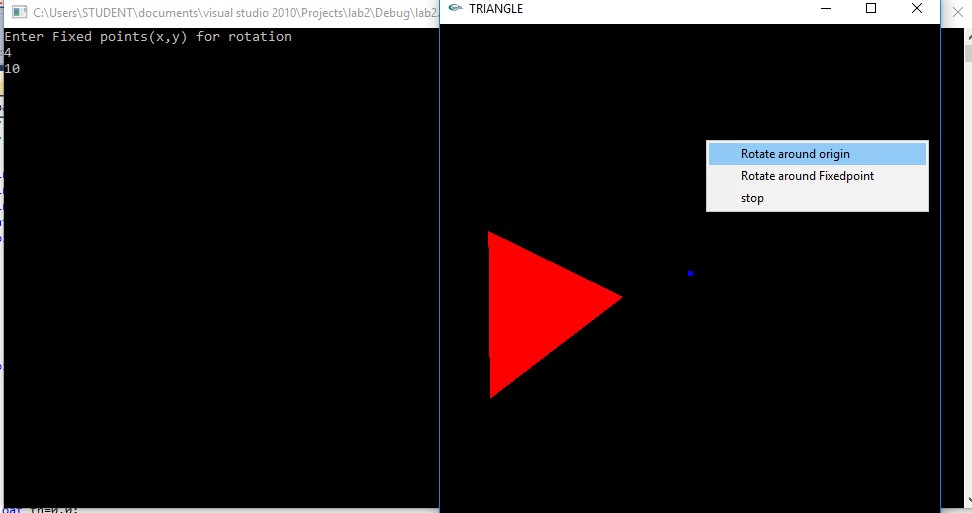
glutInitWindowSize(500,500);glutInitWindowPosition(0,0);glutCreateWindow("Create and Rotate Triangle");myInit();glutDisplayFunc(display);glutCreateMenu(rotateMenu);glutAddMenuEntry("Rotate around ORIGIN",1);glutAddMenuEntry("Rotate around FIXED POINT",2);glutAddMenuEntry("Stop Rotation",3);glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

}

**OUTPUT**:





Rotation around origin

**EXPERIMENT - 03**

**3.Program to draw a color 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] += 0.5;

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[])

{

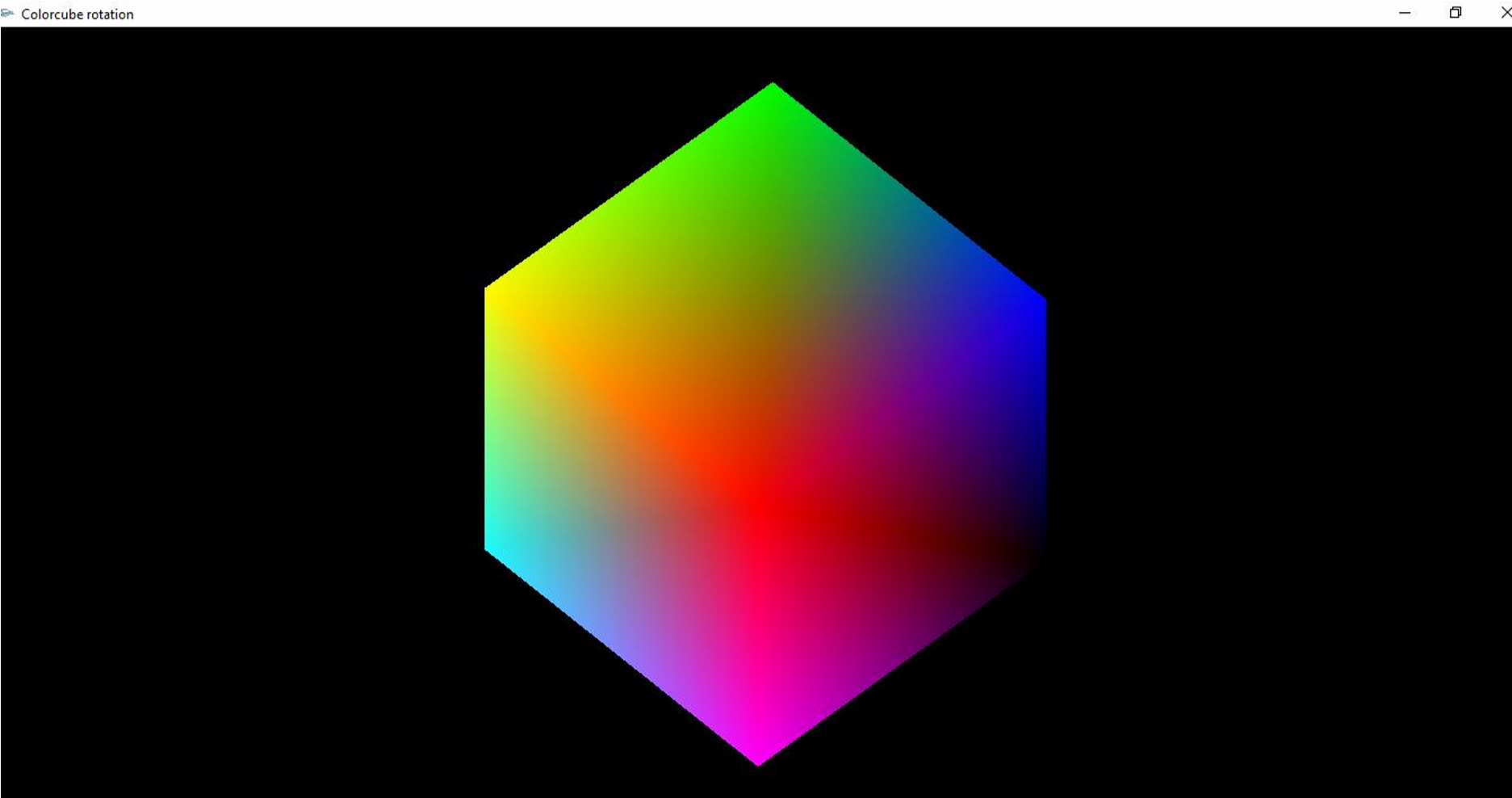
glutInit(&argc, argv);

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 :**

****

**EXPERIMENT - 04**

**4. Program to 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)

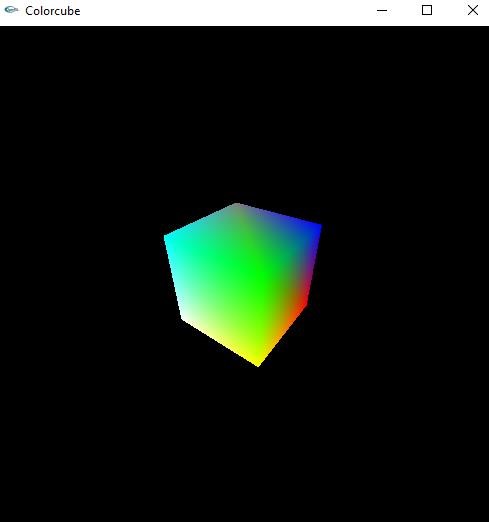
{

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 :**



**EXPERIMENT - 05**

**5. Program to clip a lines using Cohen-Sutherland line-clipping algorithm.**

#include<stdio.h>

#include<GL\glut.h>

doublexmin=50,ymin=50,xmax=100,ymax=100;doublexvmin=200,yvmin=200,xvmax=300,yvmax=300;constintRIGHT=8;constintLEFT=2;constintTOP=4;

constintBOTTOM=1;

intComputeOutCode(doublex,doubley)

{

intcode=0;

if(y>ymax)//above the clip windowcode|=TOP;elseif(y<ymin)//below the clip windowcode|=BOTTOM;

if(x>xmax)//to the right of clip windowcode|=RIGHT;elseif(x<xmin)//to the left of clip windowcode|=LEFT;

returncode;

}

voidCohenSutherland(doublex0,doubley0,doublex1,doubley1)

{

intoutcode0,outcode1,outcodeOut;boolaccept=false,done=false; outcode0=ComputeOutCode(x0,y0);

outcode1=ComputeOutCode(x1,y1);do{if(!(outcode0|outcode1))

{

accept=true;

done=true;

}

elseif(outcode0&outcode1)done=true;else{doublex,y;

outcodeOut=outcode0?outcode0:outcode1;

if(outcodeOut&TOP)

{

1. =x0+(x1-x0)\*(ymax-y0)/(y1-y0);
2. =ymax;

}

elseif(outcodeOut&BOTTOM)

{

1. =x0+(x1-x0)\*(ymin-y0)/(y1-y0);y=ymin;

}

elseif(outcodeOut&RIGHT)

{

1. =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)

{

doublesx=(xvmax-xvmin)/(xmax-xmin);doublesy=(yvmax-yvmin)/(ymax-ymin);doublevx0=xvmin+(x0-xmin)\*sx;doublevy0=yvmin+(y0-ymin)\*sy;doublevx1=xvmin+(x1-xmin)\*sx;

doublevy1=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();

}

}

voiddisplay()

{

doublex0=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();

**}**

voidmyinit()

{

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);

}

voidmain(intargc,char \***\***argv)

{

glutInit(&argc,argv);

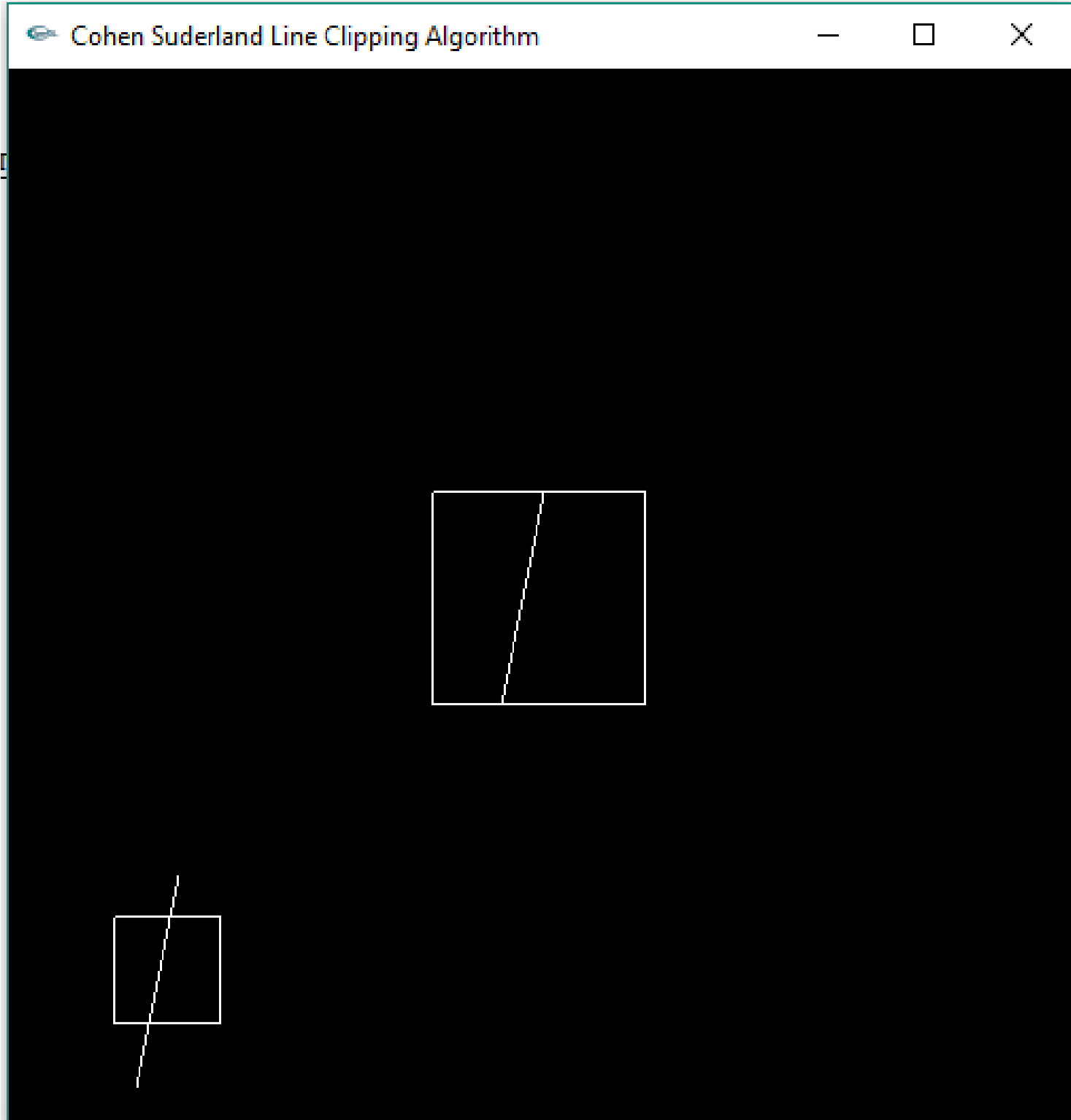
glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);glutInitWindowSize(500,500);glutInitWindowPosition(0,0);glutCreateWindow("Cohen Suderland Line Clipping Algorithm");myinit();

glutDisplayFunc(display);

glutMainLoop();

}

**OUTPUT :**



**EXPERIMENT - 06**

**6. Program 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.**

#include<GL/glut.h>

void teapot(GLfloat x, GLfloat y, GLfloat z)

{

glPushMatrix(); glTranslatef(x,y,z); glutSolidTeapot(0.1);

glPopMatrix();

}

void tableTop(GLfloat x,GLfloat y,GLfloat z)

{

glPushMatrix(); glTranslatef(x,y,z); glScalef(0.6,0.02,0.5); glutSolidCube(1.0);

glPopMatrix();

}

void tableLeg(GLfloat x,GLfloat y,GLfloat z)

{

glPushMatrix(); glTranslatef(x,y,z); glScalef(0.02,0.3,0.02); glutSolidCube(1.0);

glPopMatrix();

}

void wall(GLfloat x,GLfloat y,GLfloat z)

{

glPushMatrix(); glTranslatef(x,y,z); glScalef(1.0,1.0,0.02); glutSolidCube(1.0);

glPopMatrix();

}

void light()

{

GLfloat mat\_ambient[]={1.0,1.0,1.0,1.0};

GLfloat mat\_diffuse[]={0.5,0.5,0.5,1.0};

GLfloat mat\_specular[]={1.0,1.0,1.0,1.0}; GLfloat mat\_shininess[]={50.0f};

glMaterialfv(GL\_FRONT,GL\_AMBIENT,mat\_ambient); glMaterialfv(GL\_FRONT,GL\_DIFFUSE,mat\_diffuse); glMaterialfv(GL\_FRONT,GL\_SPECULAR,mat\_specular); glMaterialfv(GL\_FRONT,GL\_SHININESS,mat\_shininess);

GLfloat light\_position[]={2.0,6.0,3.0,1.0}; GLfloat lightIntensity[]={0.7,0.7,0.7,1.0}; glLightfv(GL\_LIGHT0,GL\_POSITION,light\_position);

glLightfv(GL\_LIGHT0,GL\_DIFFUSE,lightIntensity);

}

void display()

{

GLfloat teapotP=-0.07,tabletopP=-0.15,tablelegP=0.2,wallP=0.5;

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

gluLookAt(-2.0,2.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);

light(); //Adding light source to your project

teapot(0.0,teapotP,0.0); //Create teapot

tableTop(0.0,tabletopP,0.0); //Create table’s top tableLeg(tablelegP,-0.3,tablelegP); //Create 1st leg tableLeg(-tablelegP,-0.3,tablelegP); //Create 2nd leg tableLeg(-tablelegP,-0.3,-tablelegP); //Create 3rd leg

tableLeg(tablelegP,-0.3,-tablelegP); //Create 4th leg

wall(0.0,0.0,-wallP); //Create 1st wall glRotatef(90.0,1.0,0.0,0.0); wall(0.0,0.0,wallP); //Create 2nd wall glRotatef(90.0,0.0,1.0,0.0); wall(0.0,0.0,wallP); //Create 3rd wall glFlush();

}

void myinit()

{

glClearColor(0.0,0.0,0.0,1.0);

glMatrixMode(GL\_PROJECTION); glLoadIdentity(); glOrtho(-1.0,1.0,-1.0,1.0,-1.0,10.0);

glMatrixMode(GL\_MODELVIEW);

}

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

{

glutInit(&argc,argv);

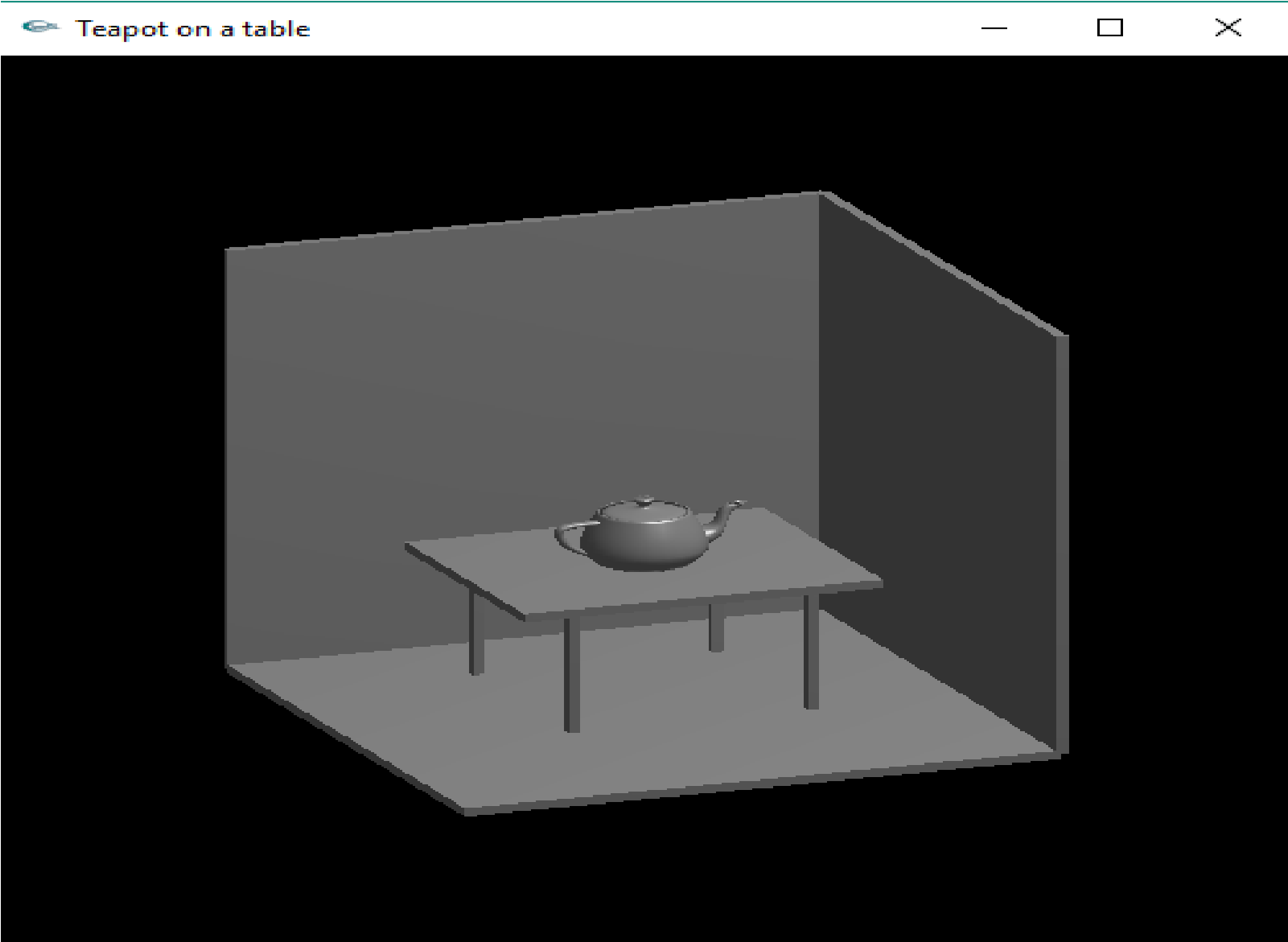
glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB|GLUT\_DEPTH); glutInitWindowSize(500,500); glutInitWindowPosition(0,0); glutCreateWindow("Teapot on a table"); myinit();

glutDisplayFunc(display); glEnable(GL\_LIGHTING); glEnable(GL\_LIGHT0); glShadeModel(GL\_SMOOTH); glEnable(GL\_NORMALIZE); glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

}

**OUTPUT**

**:**

**EXPERIMENT - 07**

**7. Program to recursively subdivide a tetrahedron to from 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] **;** point v[]=**{{0.0, 0.0, 0.0}, {0.0, 1.0, -1.0},{-1.0, -1.0, -1.0},**

**{1.0, -1.0, -1.0}};**  int n;

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

{

glBegin(GL\_POLYGON);

glVertex3fv(a); glVertex3fv(b); glVertex3fv(c);

glEnd();

}

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

{

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)

{

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);

}

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

{

printf(" Enter the Number of Divisions ? "); scanf("%d",&n); glutInit(&argc, argv);

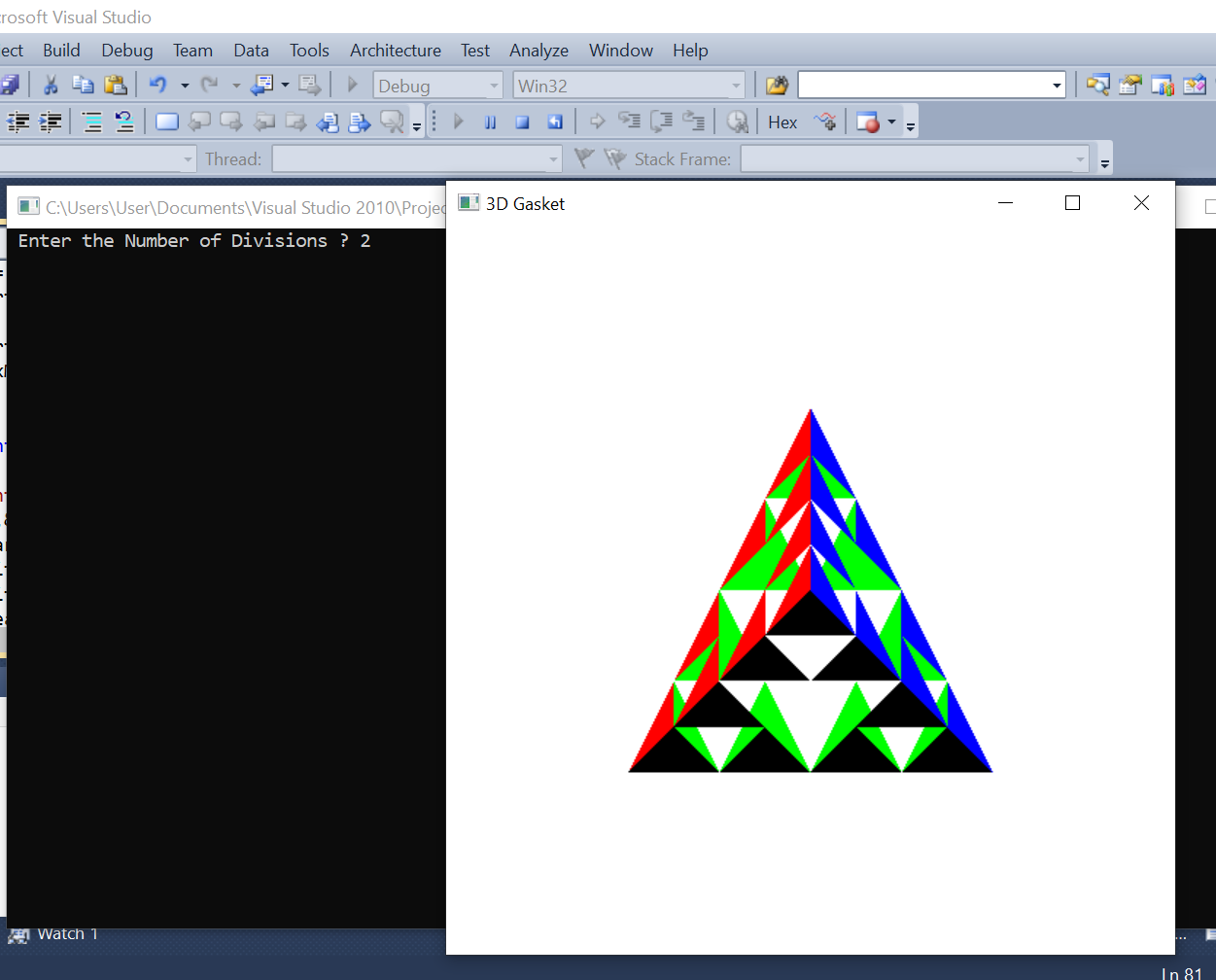
glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH); glutInitWindowSize(500, 500); glutCreateWindow("3D Gasket"); glutReshapeFunc(myReshape); glutDisplayFunc(display);

glEnable(GL\_DEPTH\_TEST);

glClearColor (1.0, 1.0, 1.0, 1.0); glutMainLoop();

}

**OUTPUT :**



**EXPERIMENT - 08**

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

#include<GL/glut.h>

#include<stdio.h>

#include<math.h> #define PI 3.1416

typedef struct point

{

GLfloat x, y, z;

};

void bino(int n, int \*C)

{

int 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(float u, point \*pt1, int cPt, point \*pt2, int \*C)

{

int k, n=cPt-1; float bFcn; pt1 ->x =pt1 ->y = pt1->z=0.0; for(k=0; k< cPt; k++)

{

bFcn = C[k] \* pow(u, k) \* pow( 1-u, n-k); pt1 ->x += pt2[k].x \* bFcn; pt1 ->y += pt2[k].y \* bFcn;

pt1 ->z += pt2[k].z \* bFcn;

}

}

void bezier(point \*pt1, int cPt, int bPt)

{

point bcPt; float u; int \*C, k; C= new int[cPt]; bino(cPt-1, C); glBegin(GL\_LINE\_STRIP); for(k=0; k<=bPt; k++)

{

u=float(k)/float(bPt); computeBezPt(u, &bcPt, cPt, pt1, C); glVertex2f(bcPt.x, bcPt.y);

}

glEnd();

delete[]C;

}

float theta = 0;

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT); int nCtrlPts = 4, nBCPts =20; point ctrlPts[4] = {{100, 400, 0}, {150, 450, 0}, {250, 350, 0},

{300, 400, 0}}; ctrlPts[1].x +=50\*sin(theta \* PI/180.0); ctrlPts[1].y +=25\*sin(theta \* PI/180.0); ctrlPts[2].x -= 50\*sin((theta+30) \* PI/180.0); ctrlPts[2].y -= 50\*sin((theta+30) \* PI/180.0); ctrlPts[3].x -= 25\*sin((theta) \* PI/180.0); ctrlPts[3].y += sin((theta-30) \* PI/180.0); theta+=0.2;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 1.0, 1.0); glPointSize(5); glPushMatrix(); glLineWidth(5);

glColor3f(1, 0.4, 0.2); //Indian flag: Orange color code for(int i=0;i<50;i++)

{

glTranslatef(0, -0.8, 0); bezier(ctrlPts, nCtrlPts, nBCPts);

}

glColor3f(1, 1, 1); //Indian flag: white color code for(int i=0;i<50;i++)

{

glTranslatef(0, -0.8, 0); bezier(ctrlPts, nCtrlPts, nBCPts);

}

glColor3f(0, 1, 0); //Indian flag: green color code for(int i=0;i<50;i++)

{

glTranslatef(0, -0.8, 0);

bezier(ctrlPts, nCtrlPts, nBCPts);

}

glPopMatrix(); glColor3f(0.7, 0.5,0.3); glLineWidth(5); glBegin(GL\_LINES); glVertex2f(100,400); glVertex2f(100,40); glEnd();

glutPostRedisplay();

glutSwapBuffers();

}

void init()

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0,500,0,500);

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowPosition(0, 0); glutInitWindowSize(500,500);

glutCreateWindow("Bezier Curve");

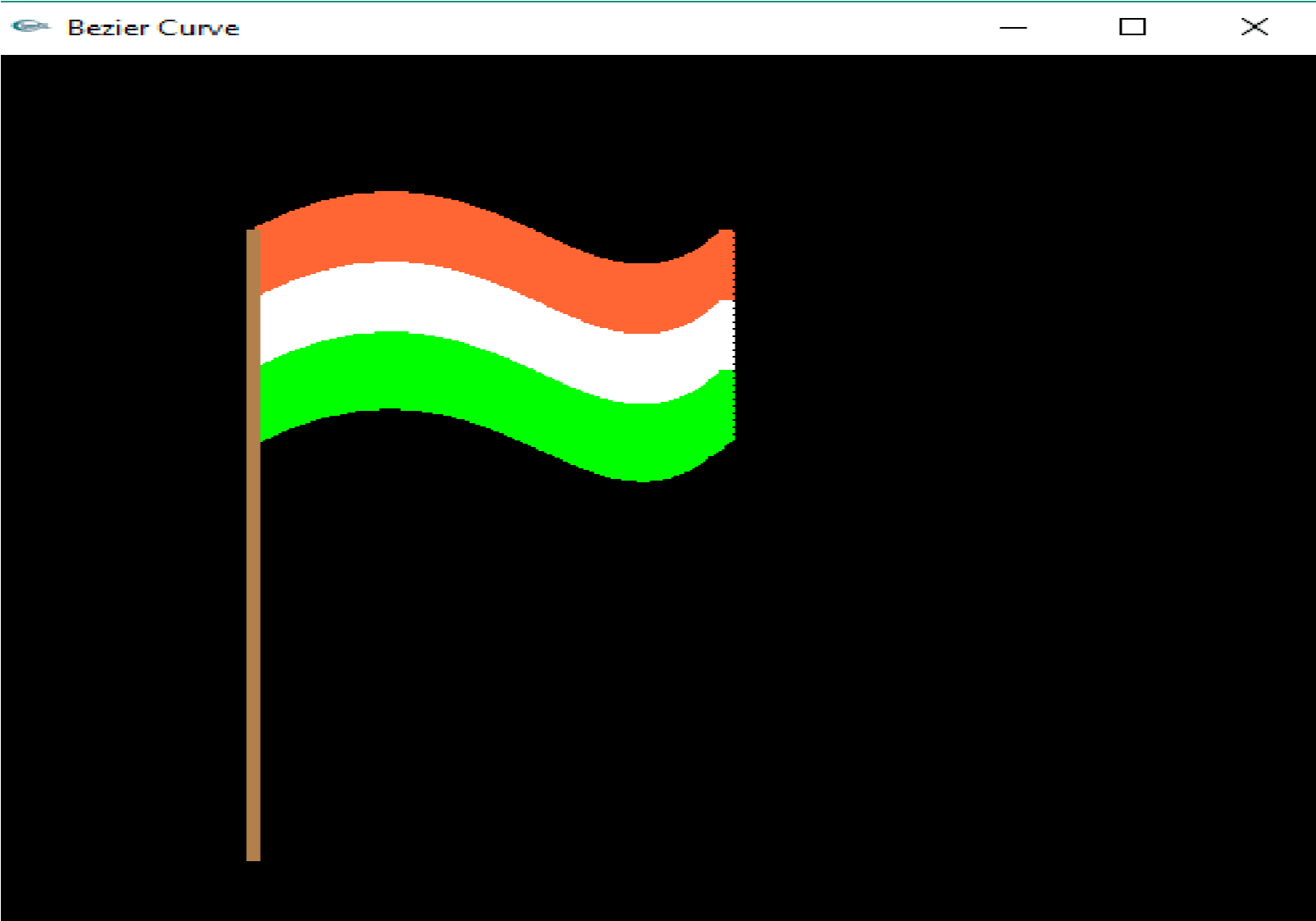
init();

glutDisplayFunc(display);

glutMainLoop();

}

**OUTPUT :**



**EXPERIMENT - 09**

**9. Develop a menu driven program to fill any given polygon using scanline area filling algorithm.**

#include<stdlib.h>

#include<stdio.h>#include<GL/glut.h>floatx1,x2,x3,x4,y1,y2,y3,y4;

intfillFlag=0;

voidedgedetect(floatx1,floaty1,floatx2,floaty2,int\*le,int\*re)

{

floatmx,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);

elsemx=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;

}

}

voiddraw\_pixel(intx,inty)

{

glColor3f(1.0,1.0,0.0);glBegin(GL\_POINTS);glVertex2i(x,y);

glEnd();

}

voidscanfill(floatx1,floaty1,floatx2,floaty2,floatx3,floaty3,floatx4,floaty4)

{

intle[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++)

{

for(i=(int)le[y];i<(int)re[y];i++)

draw\_pixel(i,y);

}

}

voiddisplay()

{

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

;

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

}

voidinit()

{

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);

}

voidfillMenu(intoption)

{

if(option==1)fillFlag=1;if(option==2)fillFlag=2;

display();

}

voidmain(intargc,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 :**

