1. Write a program to create a house like figure and perform the following operations.
2. Rotate it about a given fixed point using OpenGL transformation functions.
3. ii. Reflect it about an axis y=mx+c using OpenGL transformation functions.

#include<gl/glut.h>

#include <math.h>

#include<stdio.h>

//RIGHT CLICK TO SHOW REFLECTED HOUSE

float house[11][2] = { { 100,200 },{ 200,250 },{ 300,200 },{ 100,200 },{ 100,100 },{ 175,100 },{ 175,150 },{ 225,150 },{ 225,100 },{ 300,100 },{ 300,200 } };

int angle;

float m, c, theta;

void display()

{

glClearColor(1, 1, 1, 0);

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-450, 450, -450, 450);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

//NORMAL HOUSE

glColor3f(1, 0, 0);

glBegin(GL\_LINE\_LOOP);

for (int i = 0; i < 11; i++)

glVertex2fv(house[i]);

glEnd();

glFlush();

//ROTATED HOUSE

glPushMatrix();

glTranslatef(100, 100, 0);

glRotatef(angle, 0, 0, 1);

glTranslatef(-100, -100, 0);

glColor3f(1, 1, 0);

glBegin(GL\_LINE\_LOOP);

for (int i = 0; i < 11; i++)

glVertex2fv(house[i]);

glEnd();

glPopMatrix();

glFlush();

}

void display2()

{

glClearColor(1, 1, 1, 0);

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-450, 450, -450, 450);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

//normal house

glColor3f(1, 0, 0);

glBegin(GL\_LINE\_LOOP);

for (int i = 0; i < 11; i++)

glVertex2fv(house[i]);

glEnd();

glFlush();

// line

float x1 = 0, x2 = 500;

float y1 = m \* x1 + c;

float y2 = m \* x2 + c;

glColor3f(1, 1, 0);

glBegin(GL\_LINES);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glEnd();

glFlush();

//Reflected

glPushMatrix();

glTranslatef(0, c, 0);

theta = atan(m);

theta = theta \* 180 / 3.14;

glRotatef(theta, 0, 0, 1);

glScalef(1, -1, 1);

glRotatef(-theta, 0, 0, 1);

glTranslatef(0, -c, 0);

glBegin(GL\_LINE\_LOOP);

for (int i = 0; i < 11; i++)

glVertex2fv(house[i]);

glEnd();

glPopMatrix();

glFlush();

}

void myInit() {

glClearColor(1.0, 1.0, 1.0, 1.0);

glColor3f(1.0, 0.0, 0.0);

glLineWidth(2.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-450, 450, -450, 450);

}

void mouse(int btn, int state, int x, int y) {

if (btn == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) {

display();

}

else if (btn == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN) {

display2();

}

}

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

{

printf("Enter the rotation angle\n");

scanf("%d", &angle);

printf("Enter c and m value for line y=mx+c\n");

scanf("%f %f", &c, &m);

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(900, 900);

glutInitWindowPosition(100, 100);

glutCreateWindow("House Rotation");

glutDisplayFunc(display);

glutMouseFunc(mouse);

myInit();

glutMainLoop();

}

**Output and Observation:**

enter the rotation angle

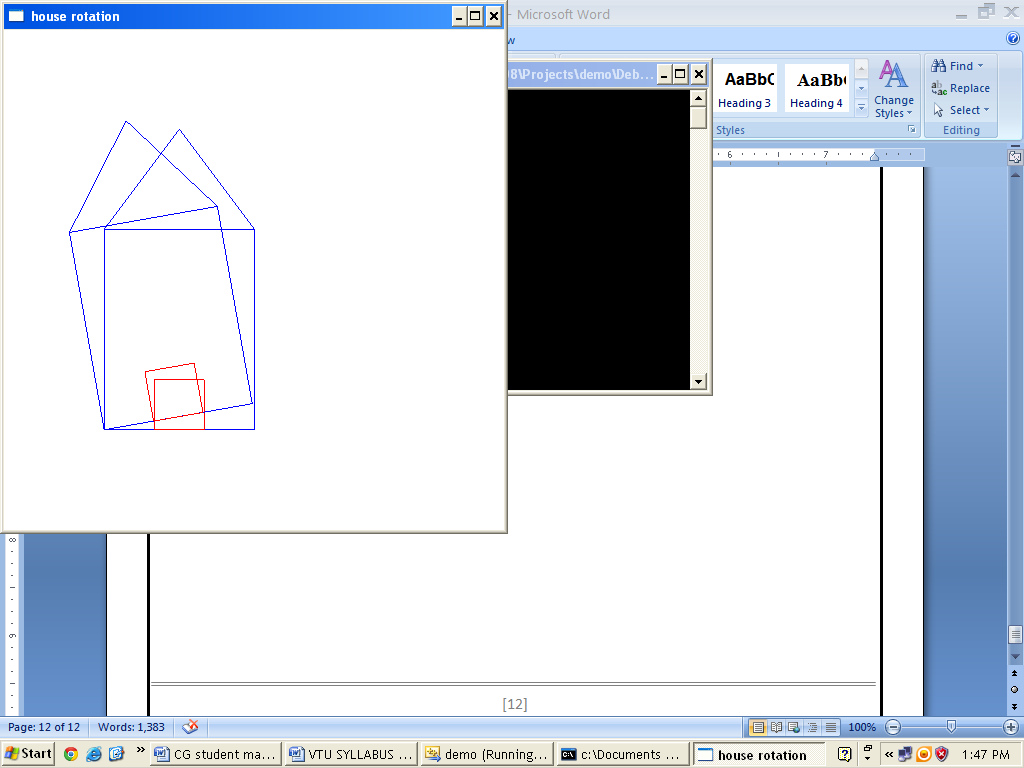
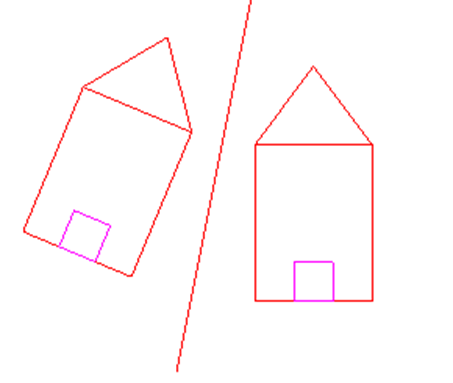
30

Enter the pivot point position

100 100

Enter the rotation angle enter the value of m(y=mx+c):5

10 Enter the value of c:10

**Program 8**

Write a program to implement the Cohen-Sutherland line clipping algorithm. Make provision to specify the input for multiple lines, window for clipping and viewport for displaying the clipped image.

#include<stdio.h>

#include<stdlib.h>

#include<gl/glut.h>

#define outcode int

#define true 1

#define false 0

double xmin, ymin, xmax, ymax;

double xvmin, yvmin, xvmax, yvmax;

const int RIGHT = 4;

const int LEFT = 8;

const int TOP = 1;

const int BOTTOM = 2;

int n;

struct line\_segment {

int x1;

int y1;

int x2;

int y2;

};

struct line\_segment ls[10];

outcode computeoutcode(double x, double y)

{

outcode code = 0;

if (y > ymax)

code |= TOP;

else if (y < ymin)

code |= BOTTOM;

if (x > xmax)

code |= RIGHT;

else if (x < xmin)

code |= LEFT;

return code;

}

void cohensuther(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);

do

{

if (!(outcode0 | outcode1))

{

accept = true;

done = true;

}

else if (outcode0 & outcode1)

done = true;

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, 0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xvmin, yvmin);

glVertex2f(xvmax, yvmin);

glVertex2f(xvmax, yvmax);

glVertex2f(xvmin, yvmax);

glEnd();

glColor3f(0, 0, 1);

glBegin(GL\_LINES);

glVertex2d(vx0, vy0);

glVertex2d(vx1, vy1);

glEnd();

}

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0, 0, 1);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xmin, ymin);

glVertex2f(xmax, ymin);

glVertex2f(xmax, ymax);

glVertex2f(xmin, ymax);

glEnd();

for (int i = 0; i < n; i++)

{

glBegin(GL\_LINES);

glVertex2d(ls[i].x1, ls[i].y1);

glVertex2d(ls[i].x2, ls[i].y2);

glEnd();

}

for (int i = 0; i < n; i++)

cohensuther(ls[i].x1, ls[i].y1, ls[i].x2, ls[i].y2);

glFlush();

}

void myinit()

{

glClearColor(1, 1, 1, 1);

glColor3f(1, 0, 0);

glPointSize(1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, 500, 0, 500);

}

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

{

printf("Enter window coordinates (xmin ymin xmax ymax): \n");

scanf("%lf%lf%lf%lf", &xmin, &ymin, &xmax, &ymax);

printf("Enter viewport coordinates (xvmin yvmin xvmax yvmax) :\n");

scanf("%lf%lf%lf%lf", &xvmin, &yvmin, &xvmax, &yvmax);

printf("Enter no. of lines:\n");

scanf("%d", &n);

for (int i = 0; i < n; i++)

{

printf("Enter line endpoints (x1 y1 x2 y2):\n");

scanf("%d%d%d%d", &ls[i].x1, &ls[i].y1, &ls[i].x2, &ls[i].y2);

}

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

glutCreateWindow("clip");

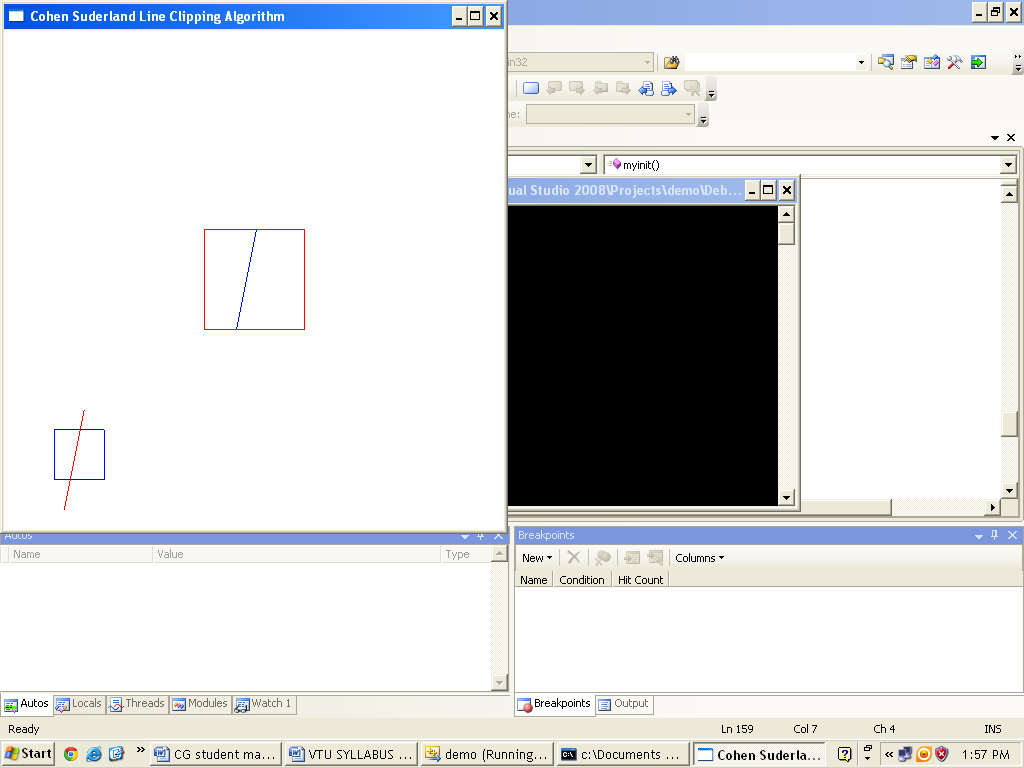
myinit();

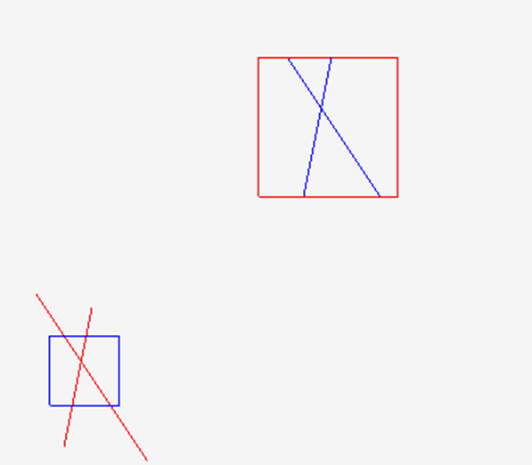
glutDisplayFunc(display);

glutMainLoop();

}

**Output and Observation:**



****

**Program 9**

Write a program to implement the Liang-Barsky line clipping algorithm. Make provision to specify the input for multiple lines, window for clipping and viewport for displaying the clipped image.

#include <stdio.h>

#include <GL/glut.h>

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

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

int n;

struct line\_segment {

int x1;

int y1;

int x2;

int y2;

};

struct line\_segment ls[10];

int cliptest(double p, double q, double\* u1, double\* u2)

{

double r;

if (p) r = q / p; // to check whether p

if (p < 0.0) // potentially entry point, update te

{

if (r > \* u1)\* u1 = r;

if (r > \* u2) return(false); // line portion is outside

}

else

if (p > 0.0) // Potentially leaving point, update tl

{

if (r < \*u2)\* u2 = r;

if (r < \*u1) return(false); // line portion is outside

}

else

if (p == 0.0)

{

if (q < 0.0) return(false); // line parallel to edge but outside

}

return(true);

}

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

{

double dx = x1 - x0, dy = y1 - y0, u1 = 0.0, u2 = 1.0;

//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();

if (cliptest(-dx, x0 - xmin, &u1, &u2)) // inside test wrt left edge

if (cliptest(dx, xmax - x0, &u1, &u2)) // inside test wrt right edge

if (cliptest(-dy, y0 - ymin, &u1, &u2)) // inside test wrt bottom edge

if (cliptest(dy, ymax - y0, &u1, &u2)) // inside test wrt top edge

{

if (u2 < 1.0)

{

x1 = x0 + u2 \* dx;

y1 = y0 + u2 \* dy;

}

if (u1 > 0.0)

{

x0 = x0 + u1 \* dx;

y0 = y0 + u1 \* dy;

}

// 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;

glColor3f(0.0, 0.0, 1.0); // draw blue colored clipped line

glBegin(GL\_LINES);

glVertex2d(vx0, vy0);

glVertex2d(vx1, vy1);

glEnd();

}

}// end of line clipping

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

//draw the line with red color

glColor3f(1.0, 0.0, 0.0);

for (int i = 0; i < n; i++)

{

glBegin(GL\_LINES);

glVertex2d(ls[i].x1, ls[i].y1);

glVertex2d(ls[i].x2, ls[i].y2);

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();

for (int i = 0; i < n; i++)

LiangBarskyLineClipAndDraw(ls[i].x1, ls[i].y1, ls[i].x2, ls[i].y2);

glFlush();

}

void myinit()

{

glClearColor(1.0, 1.0, 1.0, 1.0);

glColor3f(1.0, 0.0, 0.0);

glLineWidth(2.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, 499.0, 0.0, 499.0);

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

printf("Enter window coordinates: (xmin ymin xmax ymax) \n");

scanf("%lf%lf%lf%lf", &xmin, &ymin, &xmax, &ymax);

printf("Enter viewport coordinates: (xvmin yvmin xvmax yvmax) \n");

scanf("%lf%lf%lf%lf", &xvmin, &yvmin, &xvmax, &yvmax);

printf("Enter no. of lines:\n");

scanf("%d", &n);

for (int i = 0; i < n; i++)

{

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

scanf("%d%d%d%d", &ls[i].x1, &ls[i].y1, &ls[i].x2, &ls[i].y2);

}

glutCreateWindow("Liang Barsky Line Clipping Algorithm");

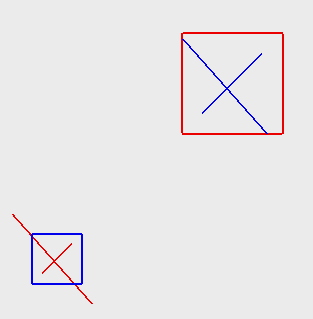
glutDisplayFunc(display);

myinit();

glutMainLoop();

}

**Output and Observation:**



**Program 10**

Write a program to implement the Cohen-Hodgeman polygon clipping algorithm. Make provision to specify the input polygon and window for clipping.

// C++ program for implementing Sutherlandï¿½Hodgman

// algorithm for polygon clipping

#include<iostream>

#include<GL/glut.h>

using namespace std;

int poly\_size, poly\_points[20][2], org\_poly\_size, org\_poly\_points[20][2], clipper\_size, clipper\_points[20][2];

const int MAX\_POINTS = 20;

// Returns x-value of point of intersection of two

// lines

void drawPoly(int p[][2], int n) {

glBegin(GL\_POLYGON);

for (int i = 0; i < n; i++)

glVertex2f(p[i][0], p[i][1]);

glEnd();

}

int x\_intersect(int x1, int y1, int x2, int y2,

int x3, int y3, int x4, int y4)

{

int num = (x1 \* y2 - y1 \* x2) \* (x3 - x4) -

(x1 - x2) \* (x3 \* y4 - y3 \* x4);

int den = (x1 - x2) \* (y3 - y4) - (y1 - y2) \* (x3 - x4);

return num / den;

}

// Returns y-value of point of intersectipn of

// two lines

int y\_intersect(int x1, int y1, int x2, int y2,

int x3, int y3, int x4, int y4)

{

int num = (x1 \* y2 - y1 \* x2) \* (y3 - y4) -

(y1 - y2) \* (x3 \* y4 - y3 \* x4);

int den = (x1 - x2) \* (y3 - y4) - (y1 - y2) \* (x3 - x4);

return num / den;

}

// This functions clips all the edges w.r.t one clip

// edge of clipping area

void clip(int poly\_points[][2], int& poly\_size,

int x1, int y1, int x2, int y2)

{

int new\_points[MAX\_POINTS][2], new\_poly\_size = 0;

// (ix,iy),(kx,ky) are the co-ordinate values of

// the points

for (int i = 0; i < poly\_size; i++)

{

// i and k form a line in polygon

int k = (i + 1) % poly\_size;

int ix = poly\_points[i][0], iy = poly\_points[i][1];

int kx = poly\_points[k][0], ky = poly\_points[k][1];

// Calculating position of first point

// w.r.t. clipper line

int i\_pos = (x2 - x1) \* (iy - y1) - (y2 - y1) \* (ix - x1);

// Calculating position of second point

// w.r.t. clipper line

int k\_pos = (x2 - x1) \* (ky - y1) - (y2 - y1) \* (kx - x1);

// Case 1 : When both points are inside

if (i\_pos >= 0 && k\_pos >= 0)

{

//Only second point is added

new\_points[new\_poly\_size][0] = kx;

new\_points[new\_poly\_size][1] = ky;

new\_poly\_size++;

}

// Case 2: When only first point is outside

else if (i\_pos < 0 && k\_pos >= 0)

{

// Point of intersection with edge

// and the second point is added

new\_points[new\_poly\_size][0] = x\_intersect(x1,

y1, x2, y2, ix, iy, kx, ky);

new\_points[new\_poly\_size][1] = y\_intersect(x1,

y1, x2, y2, ix, iy, kx, ky);

new\_poly\_size++;

new\_points[new\_poly\_size][0] = kx;

new\_points[new\_poly\_size][1] = ky;

new\_poly\_size++;

}

// Case 3: When only second point is outside

else if (i\_pos >= 0 && k\_pos < 0)

{

//Only point of intersection with edge is added

new\_points[new\_poly\_size][0] = x\_intersect(x1,

y1, x2, y2, ix, iy, kx, ky);

new\_points[new\_poly\_size][1] = y\_intersect(x1,

y1, x2, y2, ix, iy, kx, ky);

new\_poly\_size++;

}

// Case 4: When both points are outside

else

{

//No points are added

}

}

// Copying new points into original array

// and changing the no. of vertices

poly\_size = new\_poly\_size;

for (int i = 0; i < poly\_size; i++)

{

poly\_points[i][0] = new\_points[i][0];

poly\_points[i][1] = new\_points[i][1];

}

}

void init() {

glClearColor(0.0f, 0.0f, 0.0f, 0.0f);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(0.0, 500.0, 0.0, 500.0, 0.0, 500.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

}

// Implements Sutherlandï¿½Hodgman algorithm

void display()

{

init();

glColor3f(1.0f, 0.0f, 0.0f);

drawPoly(clipper\_points, clipper\_size);

glColor3f(0.0f, 1.0f, 0.0f);

drawPoly(org\_poly\_points, org\_poly\_size);

//i and k are two consecutive indexes

for (int i = 0; i < clipper\_size; i++)

{

int k = (i + 1) % clipper\_size;

// We pass the current array of vertices, it's size

// and the end points of the selected clipper line

clip(poly\_points, poly\_size, clipper\_points[i][0],

clipper\_points[i][1], clipper\_points[k][0],

clipper\_points[k][1]);

}

glColor3f(0.0f, 0.0f, 1.0f);

drawPoly(poly\_points, poly\_size);

glFlush();

}

//Driver code

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

{

printf("Enter no. of vertices: \n");

scanf\_s("%d", &poly\_size);

org\_poly\_size = poly\_size;

for (int i = 0; i < poly\_size; i++)

{

printf("Polygon Vertex:\n");

scanf\_s("%d%d", &poly\_points[i][0], &poly\_points[i][1]);

org\_poly\_points[i][0] = poly\_points[i][0];

org\_poly\_points[i][1] = poly\_points[i][1];

}

printf("Enter no. of vertices of clipping window:");

scanf\_s("%d", &clipper\_size);

for (int i = 0; i < clipper\_size; i++)

{

printf("Clip Vertex:\n");

scanf\_s("%d%d", &clipper\_points[i][0], &clipper\_points[i][1]);

}

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(400, 400);

glutInitWindowPosition(100, 100);

glutCreateWindow("Polygon Clipping!");

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

**Output and Observation:**

****

**Program 12**

Write a program to create a color cube and spin it using OpenGL transformations.

#include <stdlib.h>

#include <GL/glut.h>

#include<gl\GL.h>

#include<gl\GLU.h>

#include <time.h>

GLfloat vertices[] = { -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 normals[] = { -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[] = { 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 };

GLubyte cubeIndices[] = { 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 GLfloat theta[] = { 0.0,0.0,0.0 };

static GLfloat beta[] = { 0.0,0.0,0.0 };

static GLint axis = 2;

void delay(float secs)

{

float end = clock() / CLOCKS\_PER\_SEC + secs;

while ((clock() / CLOCKS\_PER\_SEC) < end);

}

void displaySingle(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();

}

void spinCube()

{

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

//sleep(50);

delay(0.01);

theta[axis] += 2.0;

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

glutPostRedisplay();

}

void mouse(int btn, 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);

}

void

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

{

//window 1

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(100, 100);

glutInitWindowSize(500, 500);

glutCreateWindow("colorcube");

glutReshapeFunc(myReshape);

glutDisplayFunc(displaySingle);

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();