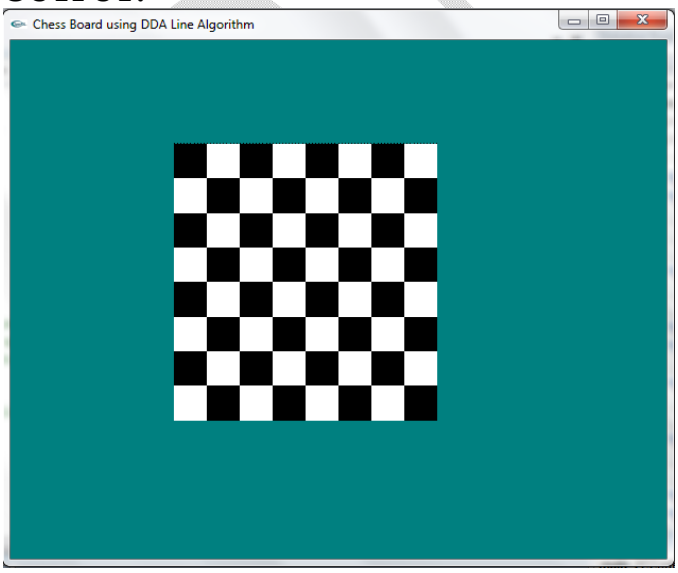
1. Write a program to create a chess board using DDA line algorithm.

```
#include<gl\glut.h>
#include<math.h>

GLint start_x=50,start_y=40,end_x=start_x+80,end_y=start_y+80;

void setPixel(GLint, GLint);
void init();
void display();
void lineDDA(GLint,GLint,GLint,GLint);
void fillRow(GLint,GLint,GLint,GLint,GLfloat);

void main(int argc, char** argv)
{
	glutInit(&argc, argv);			//initialize GLUT
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);	//initialize display mode
	glutInitWindowPosition(250,100);	//set display-window upper-left position
	glutInitWindowSize(600,500);		//set display-window width & height
	glutCreateWindow("Chess Board using DDA Algorithm");	//create display-window

	init();				//initialize window prpperties
	glutDisplayFunc(display);	//call graphics to be displayed on the window
	glutMainLoop();			//display everything and wait
}

inline int round(const float a)
{
	return int(a+0.5);
}

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

void init(void)
{
	glClearColor(0.0,0.5,0.5,0.0);
	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(0.0,200.0,0.0,150.0);
}
```

```
//DDA line drawing procedure
void lineDDA(GLint x0,GLint y0, GLint xe, GLint ye)
{
	GLint dx=xe-x0, dy=ye-y0, steps, k;
	GLfloat xinc, yinc, x=x0, y=y0;

	if(abs(dx)>abs(dy))
		steps=abs(dx);
	else
		steps=abs(dy);

	xinc=float(dx)/float(steps);
	yinc=float(dy)/float(steps);

	setPixel(round(x),round(y));

	for(k=0;k<steps;k++)
	{
		x+=xinc;
		y+=yinc;
		setPixel(round(x), round(y));
	}
}

//Function fills one row of chessbord with alternate black and white color
void fillRow(GLint x1,GLint y1,GLint x2,GLint y2,GLfloat c)
{
	while(x1<end_x)
	{
		glColor3f(c,c,c);
		glRecti(x1,y1,x2,y2);

		x1=x2;
		x2+=10;

		if(c==0.0)
			c=1.0;
		else
			c=0.0;
	}
}

void display(void)
{
	GLint i=0,a,b;
	a=start_x;
```

```
        b=start_y;

        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0,0.0,0.0);

        while(i<9)
        {
                lineDDA(a,start_y,a,end_y);
                a+=10;
                lineDDA(start_x,b,end_x,b);
                b+=10;
                i++;
        }
        GLint x1=start_x,y1=end_y,x2=start_x+10,y2=end_y-10;
        GLfloat cl=0.0;

        while(y1>start_y)
        {
                fillRow(x1,y1,x2,y2,cl);
                if(cl==0.0)
                        cl=1.0;
                else
                        cl=0.0;
                y1=y2;
                y2-=10;
        }
        glFlush();
}
```

OUTPUT:



2. Write a Program to implement Bresenham's line drawing algorithm with all values of slopes.

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

//Input vairables
GLint xOne, yOne, xTwo, yTwo;

//Funtion declaration
void resize(int, int);
void setPixel(GLint, GLint);
void lineBres_L1(GLint, GLint, GLint, GLint, GLfloat) ;
void lineBres_GE1(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("%d%d",&xOne, &yOne);

	printf("\nEnter ending vertex (x2, y2):");
	scanf("%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();
```

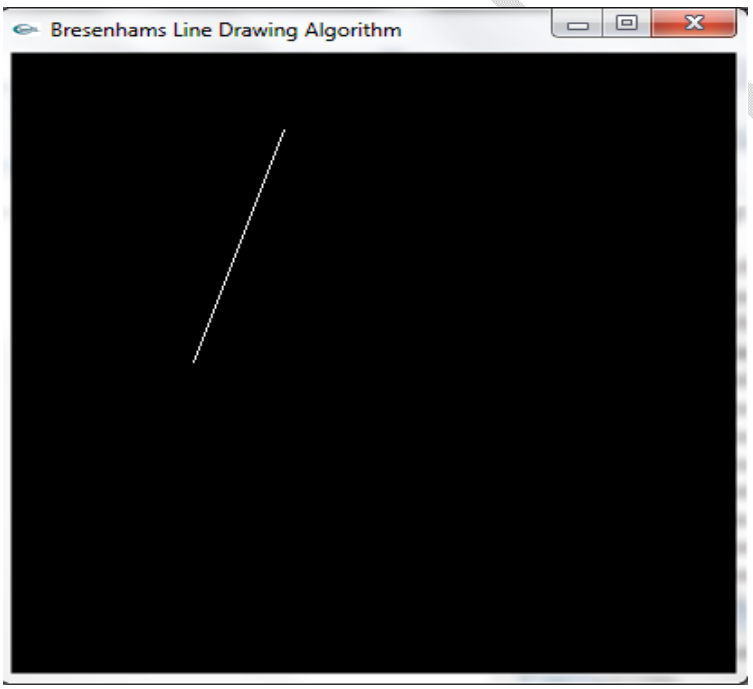
```
	gluOrtho2D(0.0,w,0.0,h);
	glViewport(0.0, 0.0, w, h);
}

void display()
{
	glClear(GL_COLOR_BUFFER_BIT);

	GLfloat m;

	m=(float)(yTwo-yOne)/(xTwo-xOne);	//compute slope

	//call required function based on value of slope
	if(fabs(m)>=1)
		lineBres_GE1(xOne,yOne,xTwo,yTwo,m);
	else
		lineBres_L1(xOne, yOne, xTwo,yTwo, m);

}

//Bresenham line-drawing procedure for |m| < 1.0
void lineBres_L1(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);
        }
}

//Bresenham line-drawing procedure for |m| >= 1.0
void lineBres_GE1(GLint x0, GLint y0, GLint xEnd, GLint yEnd, GLfloat m)
{
        GLint dx = abs(xEnd - x0);
        GLint dy = abs(yEnd - y0);

        GLint p=2*dx-dy;
        GLint twoDx = 2*dx;
        GLint twoDxMinusDy=2*(dx-dy);
        GLint x=x0,y=y0;

        // determine which point to use as start position
        if (y0 > yEnd)
        {
                x = xEnd;
                y = yEnd;
                yEnd = y0;
        }
        else
        {
                x = x0;
                y = y0;
        }
        setPixel(x,y);
        while(y<yEnd)
        {
                y++;
                if(p<0)
                        p+=twoDx;
                else
                {
                        if(m<0)
```

```
                        x--;
                else
                        x++;
                p+=twoDxMinusDy;
        }
        setPixel(x,y);
    }
}

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

OUTPUT:

********** Bresenham's Line Drawing Algoirhtm **********
Enter Starting vertex (x1, y1): 100 200
Enter Ending Vertex (x2, y2): 150 350



3. Write a Program to implement mid point circle generation algorithm.

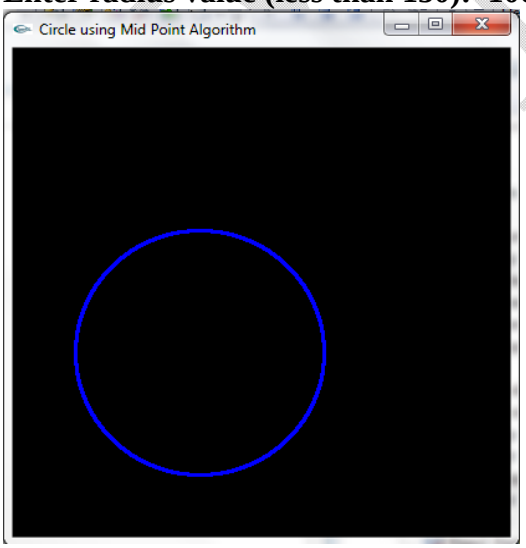
```
#include<GL/glut.h>
#include<stdio.h>

GLint radius;

class screenPoint
{
	GLint x,y;
	public:
			screenPoint()
			{
				x = 0;
				y = 0;
			}
			void setCoordinates(GLint xCoordinate, GLint yCoordinate)
			{
				x = xCoordinate;
				y = yCoordinate;
			}
			GLint getx () const
			{
				return x;
			}
			GLint gety () const
			{
				return y;
			}
			void incrementx ()
			{
				x++;
			}
			void decrementy ()
			{
				y--;
			}
};

void resize(int, int);
void display();
void setPixel(GLint, GLint);
void circleMidPoint(GLint, GLint, GLint);
void circlePlotPoints(GLint, GLint, screenPoint);

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

```
	printf("Enter Radius value (less than 150):\n");
	scanf("%d",&radius);

	glutInit(&argc, argv);			//initialize GLUT
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);	//initialize display mode
	glutInitWindowPosition(250,50);		//set display-window upper-left position
	glutInitWindowSize(400,400);		//set display-window width & height
	glutCreateWindow("Circle using Mid Point Algorithm");	//create display-window
	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 setPixel(GLint xCoordinate, GLint yCoordinate)
{
	glBegin(GL_POINTS);
	glVertex2i(xCoordinate, yCoordinate);
	glEnd();
	glFlush();	//process all OpenGL functions as quickly as possible
}

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 circleMidPoint(GLint xc, GLint yc, GLint raduis)
{
	screenPoint circlePoint;

	GLint p = 1-raduis;//initialize value for midpoint parameter
	circlePoint.setCoordinates(0,raduis);  //set coordinates for top point of circle

	circlePlotPoints(xc, yc, circlePoint); //plot the initial point in each quadrant

	//calculate the next point and plot in each octant
	while(circlePoint.getx() < circlePoint.gety())
	{
		circlePoint.incrementx();
		if(p<0)
			p += 2 * circlePoint.getx() + 1;
		else
		{
```

```
                circlePoint.decrementy ();
                p += 2 * (circlePoint.getx () - circlePoint.gety ()) + 1;
            }
            circlePlotPoints(xc, yc, circlePoint);
        }
}

void circlePlotPoints(GLint xc, GLint yc, screenPoint circPoint)
{
        setPixel(xc + circPoint.getx () , yc + circPoint.gety ());
        setPixel(xc - circPoint.getx () , yc + circPoint.gety ());
        setPixel(xc + circPoint.getx () , yc - circPoint.gety ());
        setPixel(xc - circPoint.getx () , yc - circPoint.gety ());
        setPixel(xc + circPoint.gety () , yc + circPoint.getx ());
        setPixel(xc - circPoint.gety () , yc + circPoint.getx ());
        setPixel(xc + circPoint.gety () , yc - circPoint.getx ());
        setPixel(xc - circPoint.gety () , yc - circPoint.getx ());
}

void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0,0.0,1.0);
        glPointSize(3.0);
        GLint xCenter=150;
        GLint yCenter=150;
        circleMidPoint(xCenter, yCenter, radius);
}
```

OUTPUT:

Enter radius value (less than 150): 100

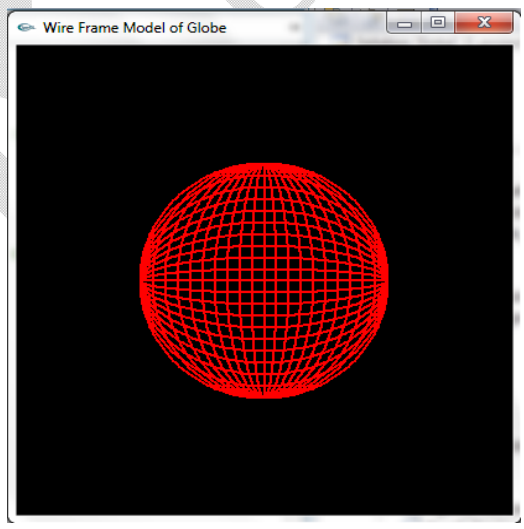


4. Write a Program to create a wire frame model of globe using equation of ellipse.

```
#include <GL/glut.h>
#include<math.h>
#define PI 3.1416

void setPixel(GLint, GLint);
void ellipse(GLint, GLint, GLint, GLint, GLfloat);
void display();
void resize(int, int);

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

    glutCreateWindow("Wire Frame Model of Globe");  //create display-window
    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);
    glColor3f(1.0,0.0,0.0);
    glPointSize(2.0);

    GLint xc=200;       // setting the center of globe
    GLint yc=200;

    GLint ry=100; // setting the radius_accross_y-axis of ellipse as constant

    //change angle (0 – 360) and radius_accross_x-axis (0 to 100 in steps of 10)
    // and draw ellipses
```

```
    for(GLint theta =1;theta<=360;theta++)
        for(GLint rx=100;rx>=0;rx-=10)
            ellipse(xc, yc, rx, ry, theta);

    GLint rx=100;   //setting the radius_accross_x-axis of ellipse as constant

    //change angle (0 – 360) and radius_accross_y-axis (0 to 100 in steps of 10)
    // and draw ellipses

    for(GLint theta =1;theta<=360;theta++)
        for(GLint ry=100;ry>=0;ry-=10)
            ellipse(xc, yc, rx, ry, theta);

    glFlush();
}

void ellipse(GLint xc, GLint yc, GLint rx, GLint ry, GLfloat theta)
{
    GLint x = xc + rx *cos(theta* PI/180.0);
    GLint y = yc + ry * sin(theta* PI/180.0);
    setPixel(x,y);
}

void setPixel(GLint xCoordinate, GLint yCoordinate)
{
    glBegin(GL_POINTS);
    glVertex2i(xCoordinate,yCoordinate);
    glEnd();
}
```

OUTPUT:



5. Write a program to create and fill two dimensional object by using boundary fill algorithm.

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

int ww = 600, wh = 500;
float fillCol[3] = {0.4,0.0,0.0};
float borderCol[3] = {0.0,0.0,0.0};

void setPixel(int, int, float[]);
void getPixel(int, int, float[]);
void resize(int, int);
void drawPolygon(int, int, int, int);
void boundaryFill4(int,int,float[],float[]);
void display();
void mouse(int, int, int, int);

int main(int argc, char** argv)
{
	glutInit(&argc,argv);
	glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize(ww,wh);
	glutInitWindowPosition(500, 50);
	glutCreateWindow("Bountry-Fill-Recursive");
	glutDisplayFunc(display);
	glutMouseFunc(mouse);

	//calls whenever frame buffer window is resized
	glutReshapeFunc(resize);

	glutMainLoop();
	return 0;
}

void setPixel(int pointx, int pointy, float f[3])
{
	glBegin(GL_POINTS);
	glColor3fv(f);
	glVertex2i(pointx,pointy);
	glEnd();
	glFlush();
}
```

```
void getPixel(int x, int y, float pixels[3])
{
	glReadPixels(x,y,1.0,1.0,GL_RGB,GL_FLOAT,pixels);
}

void resize(int w, int h)
{
	glMatrixMode(GL_PROJECTION);		//set projection paramaters
	glLoadIdentity();
	gluOrtho2D(0.0,w,0.0,h);
	glutReshapeWindow(ww, wh);
	glViewport(0.0, 0.0, w, h);
}

void drawPolygon(int x1, int y1, int x2, int y2)
{
	glColor3f(0.0, 0.0, 0.0);

	glBegin(GL_LINE_LOOP);
		glVertex2i(x1, y1);
		glVertex2i(x1, y2);
		glVertex2i(x2,y2);
		glVertex2i(x2,y1);
	glEnd();
	glFlush();
}

void display()
{
	glClearColor(1.0, 1.0, 1.0, 1.0);
	glClear(GL_COLOR_BUFFER_BIT);

	//If you modify following values, u should change condition in mouse() also.
	drawPolygon(150,250,200,300);
	glFlush();
}

void boundaryFill4(int x,int y,float fillColor[3],float borderColor[3])
{
	float interiorColor[3];
	getPixel(x,y,interiorColor);

	if(((interiorColor[0]!=borderColor[0] && (interiorColor[1])!=borderColor[1]
		&& (interiorColor[2])!=borderColor[2]) && (interiorColor[0]!=fillColor[0]
		&& (interiorColor[1])!=fillColor[1] && (interiorColor[2])!=fillColor[2])))
	{
		setPixel(x,y,fillColor);
```

```
                boundaryFill4(x+1,y,fillColor,borderColor);
                boundaryFill4(x-1,y,fillColor,borderColor);
                boundaryFill4(x,y+1,fillColor,borderColor);
                boundaryFill4(x,y-1,fillColor,borderColor);
        }
}

void mouse(int btn, int state, int x, int y)
{
        //This check is based on size of the polygon mentioned in display() function
        if(((x<150 || x>200) ||( y<200 || y>250)))
                printf("Invalid click !!\n");
        else
                if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
                {
                        int xi = x;
                        int yi = (wh-y);

                        boundaryFill4(xi,yi,fillCol,borderCol);
                }
}
```

OUTPUT:

A frame buffer window containing a polygon will be displayed as shown in Figure 1, and after clicking the mouse inside the polygon, it will be filled as shown in Figure 2. If you click the mouse any where outside the polygon, then the message "Invalid click!!" will be displayed on console window.

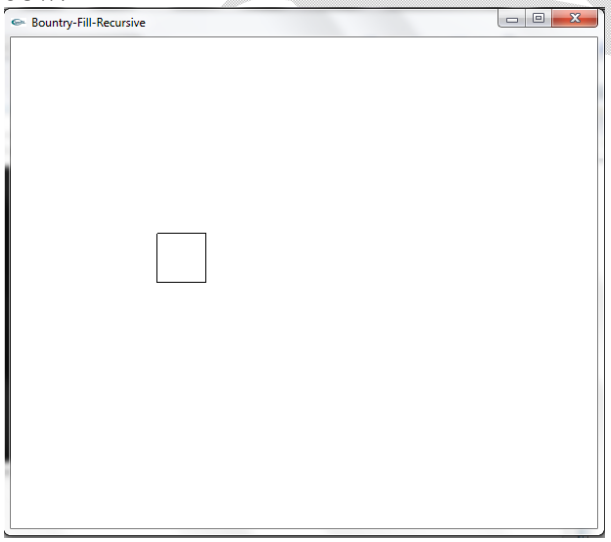


Figure 1. Original polygon

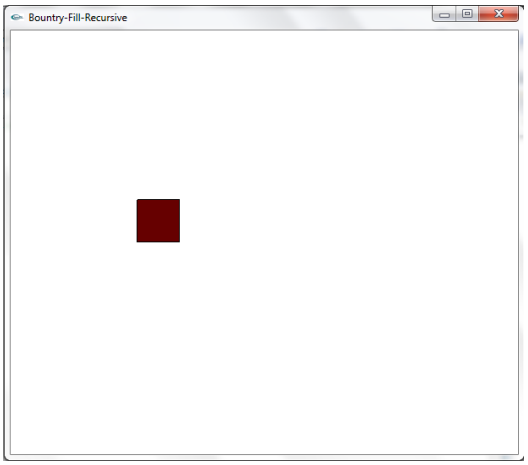


Figure 2. Polygon after filling

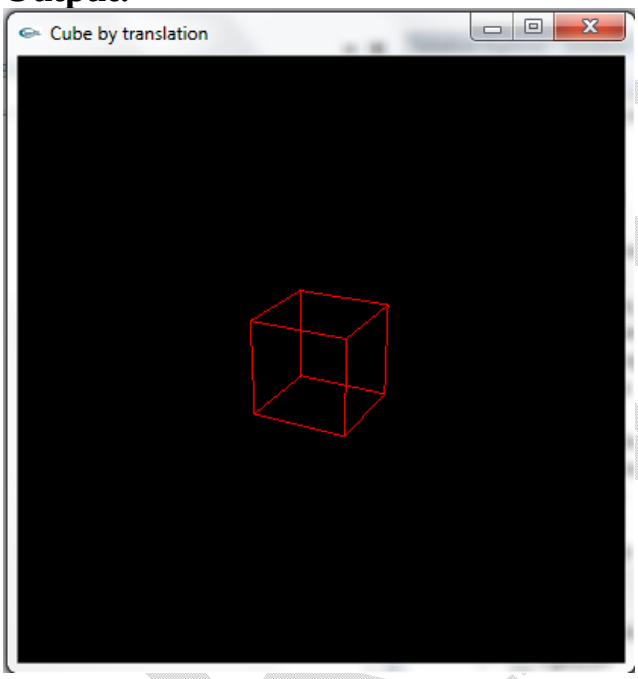
6. Write a program to create (without using built in function) a cube by implementing translation algorithm by translating along (i) X-axis (ii) Y- axis and (iii) XY plane.

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


typedef struct V
{
   float x, y, z;
};

void resize(int, int);
V translate(V point, V offset);
void display();

void main(int argc, char**argv)
{
   glutInit(&argc,argv);              //initialize GLUT
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);          //initialize display mode
   glutInitWindowSize(400,400);                  //set display-window width & height
   glutInitWindowPosition(50,50);          //set display-window upper-left position
   glutCreateWindow("Cube by translation");              //create display-window
   glutDisplayFunc(display);          //call graphics to be displayed on the window
   glutReshapeFunc(resize);
   glutMainLoop();                    //display everything and wait
}

void resize(int w, int h)
{
   glMatrixMode(GL_MODELVIEW);          //set projection paramaters
   glLoadIdentity();

   //change first three values to see the cube from different directions
   gluLookAt(3,3,6, 0,0, 0, 0, 1, 0);

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();             //reset to identity matrix
   gluPerspective(45,(float)w/h, 1, 100);
   glViewport(0,0,w,h);
}

V translate(V point, V offset)
{
   point.x +=offset.x;
```

```
    point.y +=offset.y;
    point.z +=offset.z;

    return point;
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,0.0,0.0);

    glMatrixMode(GL_MODELVIEW);

    V p0 = { -0.5, -0.5, -0.5};

    V amountX = {1, 0, 0};  //matrix values needed for translation
    V amountY ={0, 1, 0};
    V amountZ = {0, 0, 1};

    V p1 = translate(p0, amountX);
    V p2 = translate(p1, amountY);
    V p3 = translate(p0, amountY);

    glBegin(GL_LINE_LOOP); //draw one surface of cube
          glVertex3f(p0.x, p0.y, p0.z);
          glVertex3f(p1.x, p1.y, p1.z);
          glVertex3f(p2.x, p2.y, p2.z);
          glVertex3f(p3.x, p3.y, p3.z);
    glEnd();

    //calculate and draw second surface parallel to first surface using translation
    V p4 = translate(p0, amountZ);
    V p5 = translate(p1, amountZ);
    V p6 = translate(p2, amountZ);
    V p7 = translate(p3, amountZ);

    glBegin(GL_LINE_LOOP);
          glVertex3f(p4.x, p4.y, p4.z);
          glVertex3f(p5.x, p5.y, p5.z);
          glVertex3f(p6.x, p6.y, p6.z);
          glVertex3f(p7.x, p7.y, p7.z);
    glEnd();

    //draw remaining lines to join two surface, to create a cube
    glBegin(GL_LINES);
          glVertex3f(p0.x, p0.y, p0.z);
          glVertex3f(p4.x, p4.y, p4.z);
```

```
            glVertex3f(p1.x, p1.y, p1.z);
            glVertex3f(p5.x, p5.y, p5.z);

            glVertex3f(p2.x, p2.y, p2.z);
            glVertex3f(p6.x, p6.y, p6.z);

            glVertex3f(p3.x, p3.y, p3.z);
            glVertex3f(p7.x, p7.y, p7.z);
    glEnd();

    glFinish();
}
```

Output:



7. Write a Program to create (without using built-in function) and rotate (1. given angle, 2. around X and Y axis) a triangle by implementing rotation algorithm.

```
#include <GL/glut.h>
#include<math.h>
#include<stdio.h>
#define PI 3.1416

float angle=10;
float theta;
```

```
typedef struct Point
{
	float x, y, z;
};

void rotate_x(float);
void rotate_y(float);
void init();

Point p[3]={ {3.0, 0, -0.50}, {3.0, 0, -1.50}, {3.0, 1, -1.0}};

void drawTriangle(Point p[3])
{
	glColor3f(0.3, 0.6, 0.9);

	glLineWidth(2.0);

	glBegin(GL_LINES);
		glVertex3f(p[0].x, p[0].y, p[0].z);
		glVertex3f(p[1].x, p[1].y, p[1].z);
	glEnd();

	glColor3f(0.6, 0.9, 0.3);

	glBegin(GL_LINES);
		glVertex3f(p[1].x, p[1].y, p[1].z);
		glVertex3f(p[2].x, p[2].y, p[2].z);
	glEnd();

	glColor3f(0.9, 0.3, 0.6);

	glBegin(GL_LINES);
		glVertex3f(p[0].x, p[0].y, p[0].z);
		glVertex3f(p[2].x, p[2].y, p[2].z);
	glEnd();

	glFlush();
}

void display()
{
	init();
	int opt;
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(1.0,1.0,1.0);
	glMatrixMode(GL_MODELVIEW);
```

```
        glBegin(GL_LINES);
                glColor3f(1,0,0);
                glVertex3f(0,0,0);
                glVertex3f(7, 0, 0);

                glColor3f(0,1,0);
                glVertex3f(0,0,0);
                glVertex3f(0,3,0);

                glColor3f(0,0,1);
                glVertex3f(0,0,0);
                glVertex3f(0,0,3);
        glEnd();

        glRasterPos3f(7, 0, 0);
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, 'X');

        glRasterPos3f(0, 3, 0);
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, 'Y');

        glRasterPos3f(0, 0, 3);
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, 'Z');

        glFinish();

        drawTriangle(p);

        printf("************ Traingle Rotation ***************");
        printf("\n1. Rotate around x-axis \n 2. Rotate around y-axis \n");
        printf("Enter your option:");
        scanf("%d", & opt);
        printf("\n Enter value for theta: ");
        scanf("%f", &theta);

        switch(opt)
        {
                case 1: rotate_x(theta);
                        break;
                case 2: rotate_y(theta);
                        break;
        }
        glFlush();
}

void rotate_x(float theta)
{
```

```
        int i;
        Point new_p[3];

        for(i=0;i<3;i++)
        {
                new_p[i].x= p[i].x;
                new_p[i].y = p[i].y * cos(theta * PI/180.0) –
                                        p[i].z * sin(theta*PI/180.0);
                new_p[i].z = p[i].y * sin(theta * PI/180.0) +
                                        p[i].z * cos(theta*PI/180.0);
        }

        drawTriangle(new_p);
}

void rotate_y(float theta)
{
        int i;
        Point new_p[3];

        for(i=0;i<3;i++)
        {
                new_p[i].x = p[i].z * sin(theta * PI/180.0) +
                                        p[i].x * cos(theta*PI/180.0);
                new_p[i].y= p[i].y;
                new_p[i].z = p[i].z * cos(theta * PI/180.0) –
                                        p[i].x * sin(theta*PI/180.0);
        }
        drawTriangle(new_p);
}

void init(void)
{
        glMatrixMode(GL_MODELVIEW);             //set projection parameters
        glLoadIdentity();
        gluLookAt(10,1,1,  0,0, 0, 0, 1, 0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity(); //reset to identity matrix
        gluPerspective(45, 1, 1, 100);
}

void main(int argc, char**argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(400,400);
        glutInitWindowPosition(500,100);
```

```
        glutCreateWindow(" Triangle Rotation");
        init();
        glutDisplayFunc(display);
        glutMainLoop();
}
```

Output:

In this program, you can rotate a triangle either around X axis or around Y axis. Once you execute this program, you have choose the option as:

1. Rotation along x-axis
2. Rotation along y-axis

Enter your option:

Once you choose 1 or 2, you have to give the value of theta (the angle of rotation). The initial frame buffer window with original triangle is shown in Figure 1. Then the rotations are shown in Figure 2 and Figure 3.

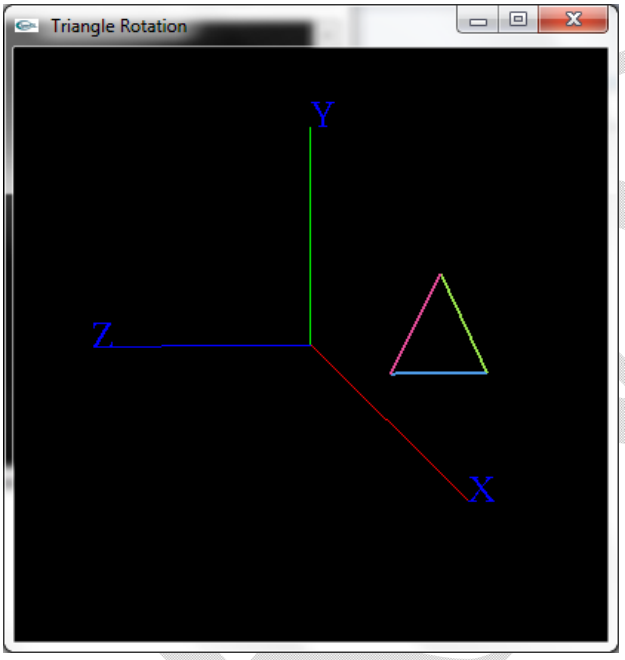


Figure 1. Original triangle

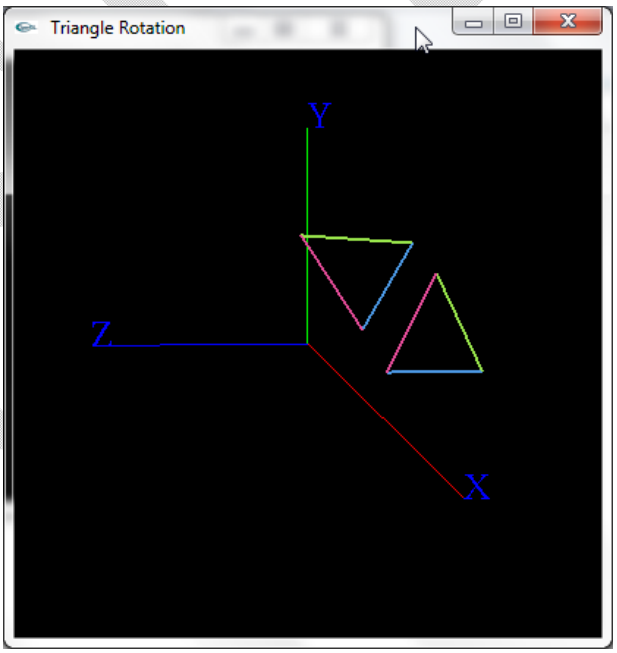


Figure 2. Rotation along X-axis with theta=60

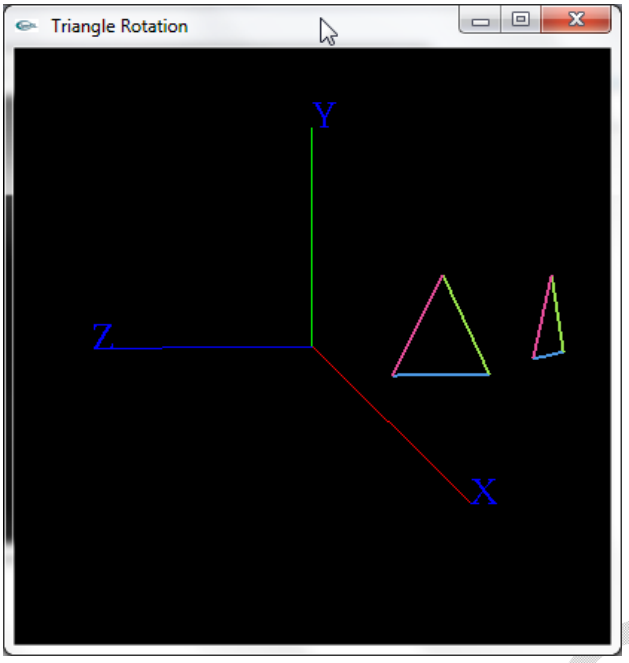


Figure 3. Rotation along Y-axis with theta =45

8. Write a Program to create (without using built-in function) a triangle by implementing scaling algorithm by zooming/un-zooming along (i) X axis (ii) Y axis (iii)XY plane.

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

void scaling(float , float);

typedef struct Point
{
	float x, y;
};

Point p[3]={ {50, 50}, {150, 50}, {100, 125}};


void drawTriangle(Point p[3])
{
	glColor3f(0.0, 0.0, 0.0);

	glBegin(GL_LINE_LOOP);
		glVertex2i(p[0].x, p[0].y);
		glVertex2i(p[1].x, p[1].y);
		glVertex2i(p[2].x, p[2].y);
```

```
        glEnd();

        glFlush();
}

void display()
{
        int opt;
        float sx, sy;

        glClearColor(1.0, 1.0, 1.0, 1.0);
        glClear(GL_COLOR_BUFFER_BIT);

        drawTriangle(p);

        printf("************ Traingle Scaling ***************");
        printf("\n1. Scale along x-axis \n 2. Scale along y-axis \n
                                        3. Scale along both x-axis and y-axis \n");
        printf("Enter your option:");
        scanf("%d", & opt);
        switch(opt)
        {
                case 1: printf("\n Enter value for sx: ");
                        scanf("%f", &sx);
                        scaling(sx, 0);
                        break;
                case 2: printf("\n Enter value for sy: ");
                        scanf("%f", &sy);
                        scaling(0, sy);
                        break;
                case 3: printf("\n Enter value for sx : ");
                        scanf("%f", &sx);
                        printf("\n Enter value for sy : ");
                        scanf("%f", &sy);
                        scaling(sx, sy);
                        break;
                default:  return;
        }
        glFlush();
}

void scaling(float sx, float sy)
{       int i;
        Point new_p[3];

        for(i=0;i<3;i++)
        {
```

```
		if(sx !=0)
			new_p[i].x = p[i].x * sx;
		else
			new_p[i].x=p[i].x;

		if(sy!=0)
			new_p[i].y = p[i].y * sy;
		else
			new_p[i].y=p[i].y;
	}

	drawTriangle(new_p);
}
void init(void)
{
	glClearColor(1.0,1.0,1.0,0.0);	//set display-window background color to white
	glMatrixMode(GL_PROJECTION);		//set projection paramaters
	gluOrtho2D(0.0,400.0,0.0,400.0);

}
void main(int argc, char**argv)
{
	glutInit(&argc,argv);			//initialize GLUT
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);	//initialize display mode
	glutInitWindowSize(400,400);		//set display-window width & height
	glutInitWindowPosition(500,100);	//set display-window upper-left position
	glutCreateWindow("Triangle Scaling");		//create display-window
	init();					//initialize OpenGL
	glutDisplayFunc(display);
	glutMainLoop();
}
```

Output:

After executing this program, you will get a frame buffer window as shown in Figure 1. A console window will display following:

********** Triangle Scaling ********

1. Scale along X-axis
2. Scale along Y-axis
3. Scale along both X and Y

Enter your option:

Now, you have to choose one option out of three. Scaling factor should be given as input later. Few of the sample outputs are shown in Figure 2, Figure 3, Figure 4.

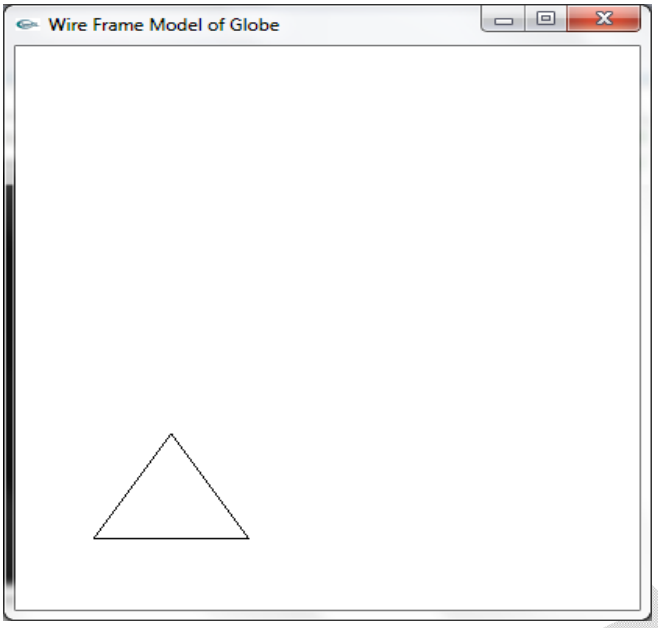


Figure 1: Original Triangle

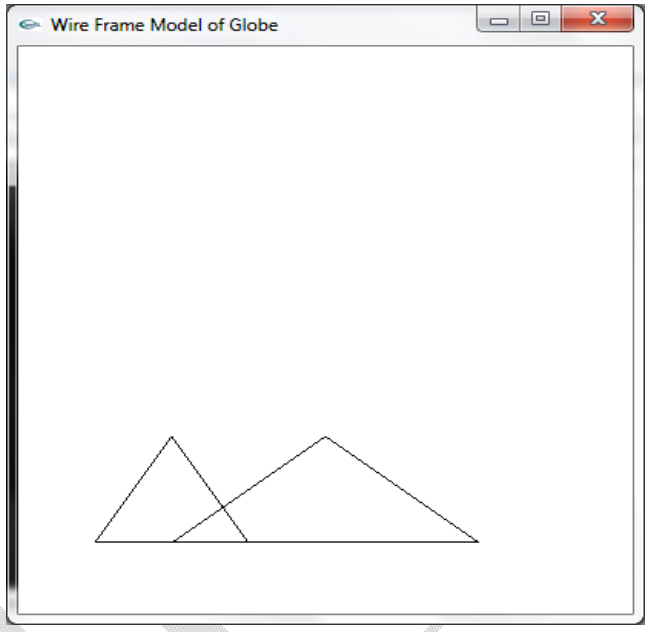


Figure 2: Scaling along X-axis (sx=2)

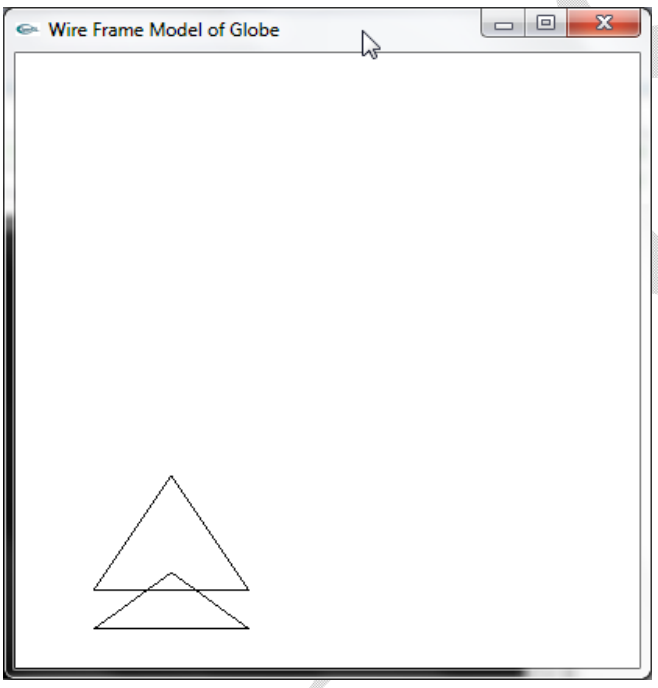


Figure 3: Scaling along Y-axis (sy=0.5)

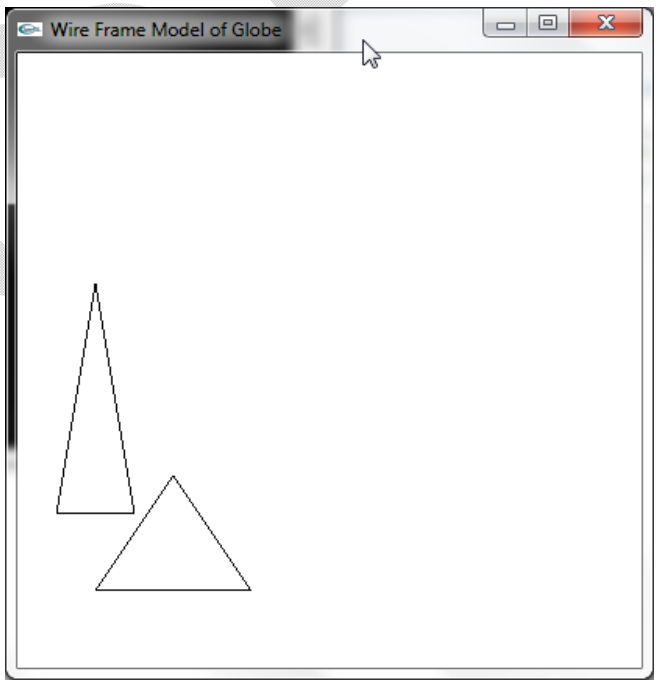


Figure 4: Scaling along both X & Y (sx=0.5, sy=2)

9. Write a program to create (without using built-in function) a cube and implement reflection algorithm (i) X axis (ii) Y axis

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

void Option();

typedef struct Point
{
	float x, y, z;
};
const float Center_x=2, Center_y=0, Center_z=-2;

Point cube_vertex[8]={
	{-0.5 + Center_x, 0.5 + Center_y, 0.5 + Center_z},
	{0.5 + Center_x, 0.5 + Center_y, 0.5 + Center_z},
	{0.5 + Center_x, -0.5 + Center_y, 0.5 + Center_z},
	{-0.5 + Center_x, -0.5 + Center_y, 0.5 + Center_z},

	{-0.5 + Center_x, 0.5 + Center_y,- 0.5 + Center_z},
	{0.5 + Center_x, 0.5 + Center_y, -0.5 + Center_z},
	{0.5 + Center_x, -0.5 + Center_y, -0.5 + Center_z},
	{-0.5 + Center_x, -0.5 + Center_y, -0.5 + Center_z}};

void init(void)
{
	glMatrixMode(GL_MODELVIEW);		//set projection paramaters
	glLoadIdentity();			//gluOrtho2D(0.0,400.0,0.0,400.0);

	//change first three values to see the cube from different directions
	gluLookAt(10,2,1, 0,0, 0, 0, 1, 0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();			//reset to identity matrix
	gluPerspective(45, 1, 1, 100);
}

void drawCube(Point vertex[8])
{
	glBegin(GL_LINE_LOOP);
		glVertex3f(vertex[0].x, vertex[0].y, vertex[0].z);
		glVertex3f(vertex[1].x, vertex[1].y, vertex[1].z);
		glVertex3f(vertex[2].x, vertex[2].y, vertex[2].z);
		glVertex3f(vertex[3].x, vertex[3].y, vertex[3].z);
	glEnd();
```

```
	glBegin(GL_LINE_LOOP);
		glVertex3f(vertex[4].x, vertex[4].y, vertex[4].z);
		glVertex3f(vertex[5].x, vertex[5].y, vertex[5].z);
		glVertex3f(vertex[6].x, vertex[6].y, vertex[6].z);
		glVertex3f(vertex[7].x, vertex[7].y, vertex[7].z);
	glEnd();

	glBegin(GL_LINES);
		glVertex3f(vertex[0].x, vertex[0].y, vertex[0].z);
		glVertex3f(vertex[4].x, vertex[4].y, vertex[4].z);

		glVertex3f(vertex[1].x, vertex[1].y, vertex[1].z);
		glVertex3f(vertex[5].x, vertex[5].y, vertex[5].z);

		glVertex3f(vertex[2].x, vertex[2].y, vertex[2].z);
		glVertex3f(vertex[6].x, vertex[6].y, vertex[6].z);

		glVertex3f(vertex[3].x, vertex[3].y, vertex[3].z);
		glVertex3f(vertex[7].x, vertex[7].y, vertex[7].z);
	glEnd();
}

void display(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(1.0,1.0,1.0);
	glMatrixMode(GL_MODELVIEW);

	drawCube(cube_vertex);
	glBegin(GL_LINES);
		glColor3f(1,0,0);
		glVertex3f(0,0,0);
		glVertex3f(5, 0, 0);

		glColor3f(0,1,0);
		glVertex3f(0,0,0);
		glVertex3f(0,3,0);

		glColor3f(0,0,1);
		glVertex3f(0,0,0);
		glVertex3f(0,0,3);
	glEnd();

	glFinish();
	Option();
}
```

```
void transform(Point in_v, float mat[3][3], Point &out_v)
{
	out_v.x = in_v.x * mat[0][0] + in_v.y * mat[1][0] + in_v.z * mat[2][0];
	out_v.y = in_v.x * mat[0][1] + in_v.y * mat[1][1] + in_v.z * mat[2][1];
	out_v.z = in_v.x * mat[0][2] + in_v.y * mat[1][2] + in_v.z * mat[2][2];
}

void transformCube(Point source_v[8], float mat[3][3], Point dest_v[8])
{
	int i=0;

	for(i=0;i<8;i++)
		transform(source_v[i], mat, dest_v[i]);
}
void Option()
{
	int opt;
	Point cube_dest[8];

	printf("\n1. X- axis \n 2. Y - axis \n");
	printf("Enter your option:");
	scanf("%d",&opt);

	switch(opt)
	{
		case 1: {
				float mat[3][3]={  {1,0,0},
						   {0,1,0},
						   {0,0,-1}
						};

				transformCube(cube_vertex,mat,  cube_dest);
				glColor3f(0.5, 0.8, 0.5);
				drawCube(cube_dest);
				glFinish();
			}
			break;

		case 2: {
				float mat[3][3]={  {-1,0,0},
						   {0,1,0},
						   {0,0,1}
						};

				transformCube(cube_vertex,mat,  cube_dest);
				glColor3f(0.5, 0.8, 0.5);
```

```
                        drawCube(cube_dest);
                        glFinish();
                    }
                    break;

            default:
                    return;
        }
}

void main(int argc, char**argv)
{
        glutInit(&argc,argv);       //initialize GLUT
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);   //initialize display mode
        glutInitWindowSize(600,600);    //set display-window width & height
        glutInitWindowPosition(450,50);         //set display-window upper-left position
        glutCreateWindow("Cube Reflection");        //create display-window with a title
        init();                         //initialze OpenGL
        glutDisplayFunc(display);           //call graphics to be displayed on the window
        glutMainLoop();     //display everything and wait
}
```

Output:

When you execute this program, you will be shown options on console window as below:

1. X axis
2. Y axis

Enter your option:

At the same time, a frame buffer will be displayed as in Figure 1. Once you choose the option as 1, the cube will be reflected on X axis and output will be as shown in Figure 2. If you choose option 2, the reflection on Y axis will be as shown in Figure 3.

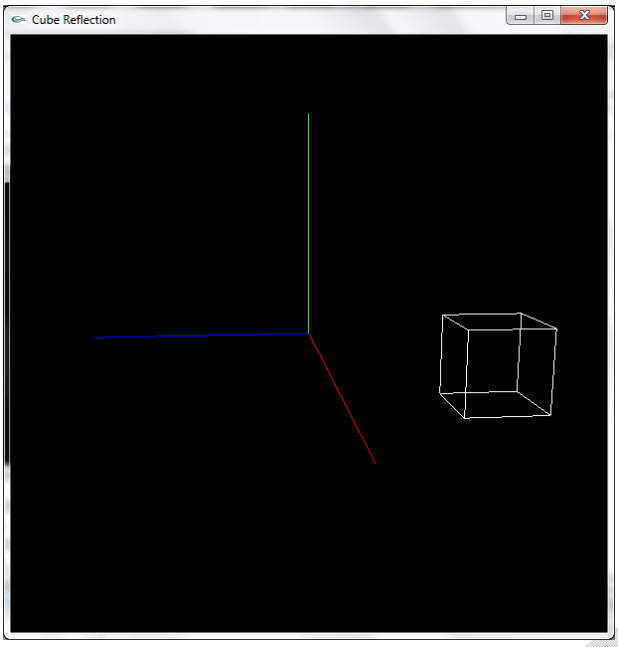


Figure 1. Original Cube

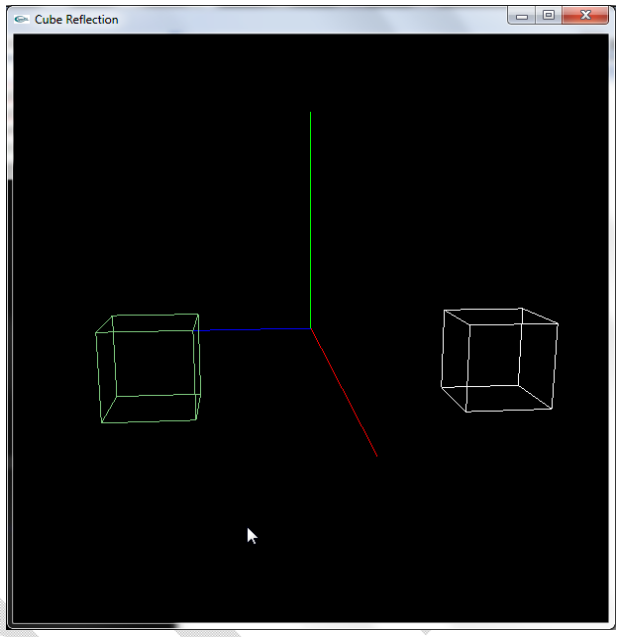


Figure 2. Reflection along X axis

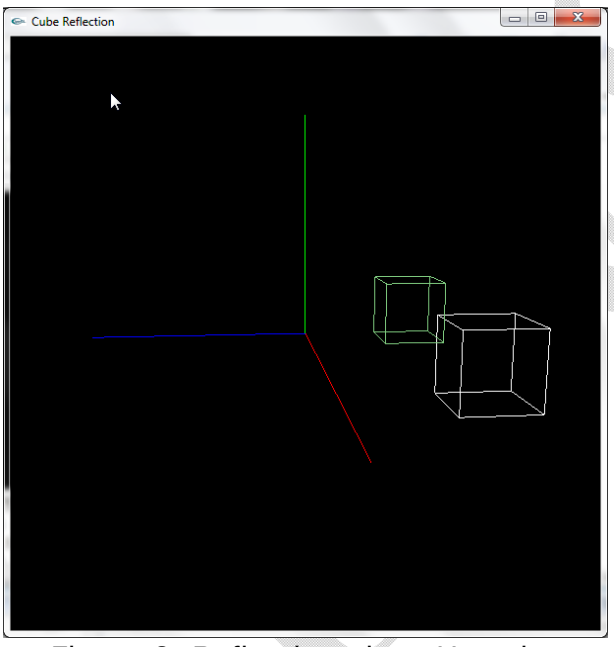


Figure 3. Reflection along Y –axis

10. Write a program to create (without using built-in function) a square and implement shear algorithm along (i) X axis (ii) Y axis

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

```
void init();
void display();
void drawPolygon(int, int, int, int, int, int, int, int);
void setPixel(GLint, GLint);

void main(int argc, char**argv)
{
   glutInit(&argc,argv);                        //initialize GLUT
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);   //initialize display mode
   glutInitWindowSize(300,300);   //set display-window width & height
   glutInitWindowPosition(600,50);         //set display-window upper-left position
   glutCreateWindow("Implementation of Shear Algorithm");
   init();         //initialze OpenGL
   glutDisplayFunc(display);         //call graphics to be displayed on the window
   glutMainLoop();     //display everything and wait
}

void init(void)
{
   glClearColor(1.0,1.0,1.0,0.0);   //set display-window background color to white
   glMatrixMode(GL_PROJECTION);         //set projection paramaters
   gluOrtho2D(0.0,300.0,0.0,300.0);
}

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

void drawPolygon(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)
{
   glColor3f(0.0, 0.0, 0.0);

   glBegin(GL_LINES);
         glVertex2i(x1, y1);
         glVertex2i(x2, y2);
   glEnd();

   glBegin(GL_LINES);
         glVertex2i(x2, y2);
         glVertex2i(x3, y3);
   glEnd();
```

```
    glBegin(GL_LINES);
        glVertex2i(x3, y3);
        glVertex2i(x4, y4);
    glEnd();

    glBegin(GL_LINES);
        glVertex2i(x4, y4);
        glVertex2i(x1, y1);
    glEnd();

    glFlush();
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,0.0,0.0);
    glPointSize(2.0);

    GLint x0=100;       // setting the starting point of square
    GLint y0=100;

    GLint x1 = 150;    //setting the third point of square
    GLint y1 = 150;

    drawPolygon(x0, y0, x1, y0, x1, y1, x0, y1);  //draw square

    int opt;
    float sh;
    GLint x0_New, x1_New, y0_New, y1_New ;


    for(;;)
    {
        printf(" \n 1. Shear along X-axis \n 2. Shear along Y-axis \n 3. Exit\n");
        printf("Enter your option:");
        scanf("%d", & opt);

        switch(opt)
        {
        case 1: printf("\nEnter shear parameter: ");
                scanf("%f", &sh);
                x0_New = x0+ sh * y1;
                x1_New = x1+ sh * y1;
                glClear(GL_COLOR_BUFFER_BIT);
                glColor3f(1.0,0.0,0.0);
                glPointSize(2.0);
```

```
                drawPolygon(x0, y0, x1, y0, x1_New, y1, x0_New, y1);
                break;
        case 2:
                printf("\nEnter shear parameter: ");
                scanf("%f", &sh);
                y0_New = y0+ sh * y1;
                y1_New = y1+ sh * y1;
                glClear(GL_COLOR_BUFFER_BIT);
                glColor3f(1.0,0.0,0.0);
                glPointSize(2.0);

                drawPolygon(x0, y0, x1, y0_New, x1, y1_New, x0, y1);
                break;
        default: exit(0);
        }
    }
}
```

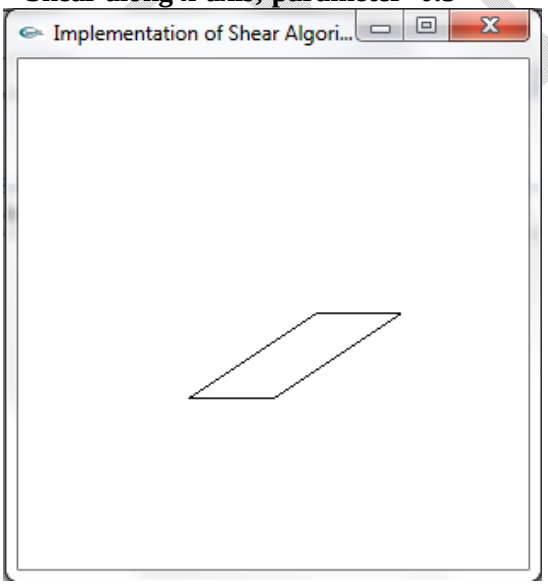
OUTPUT:

Once you execute this program, a console window will be displaying following text and a frame-buffer window will be showing a square in it. Choose the required option and also the shearing parameter.

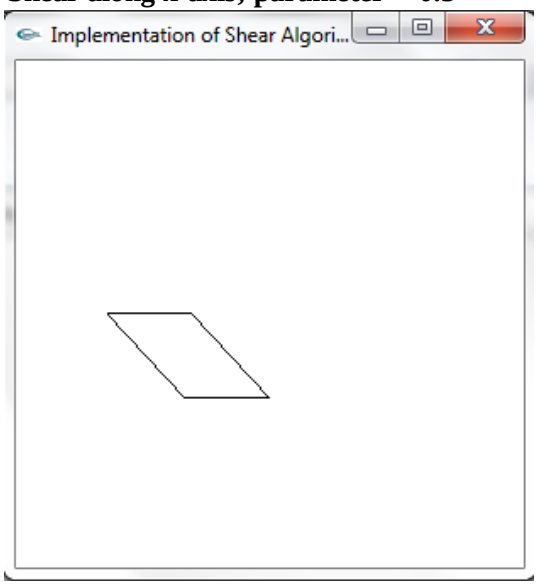
```
1. Shear along X-axis
2. Shear along Y-axis
3. Exit
Enter your option: 1
Enter shear parameter: 0.5
```

Sample output screens have been shown below.

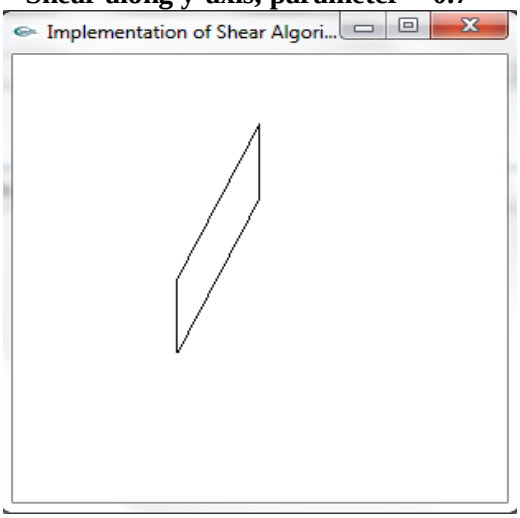
Shear along x-axis; parameter=0.5



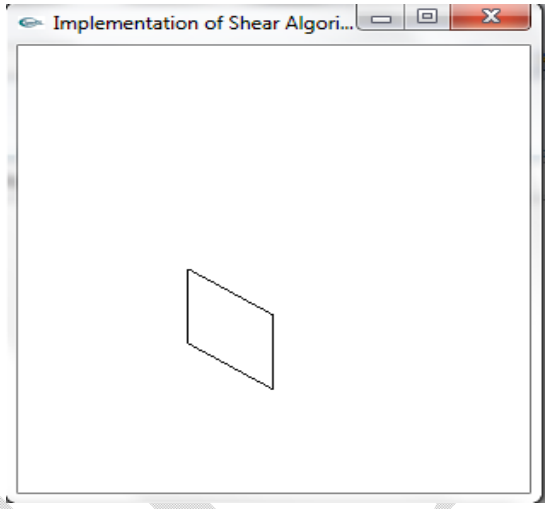
Shear along x-axis; parameter= -0.3



Shear along y-axis, parameter = 0.7



Shear along y-axis, parameter = -0.2



11. Write a program to animate a flag using Bezier curve algorithm.

```
#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;
```

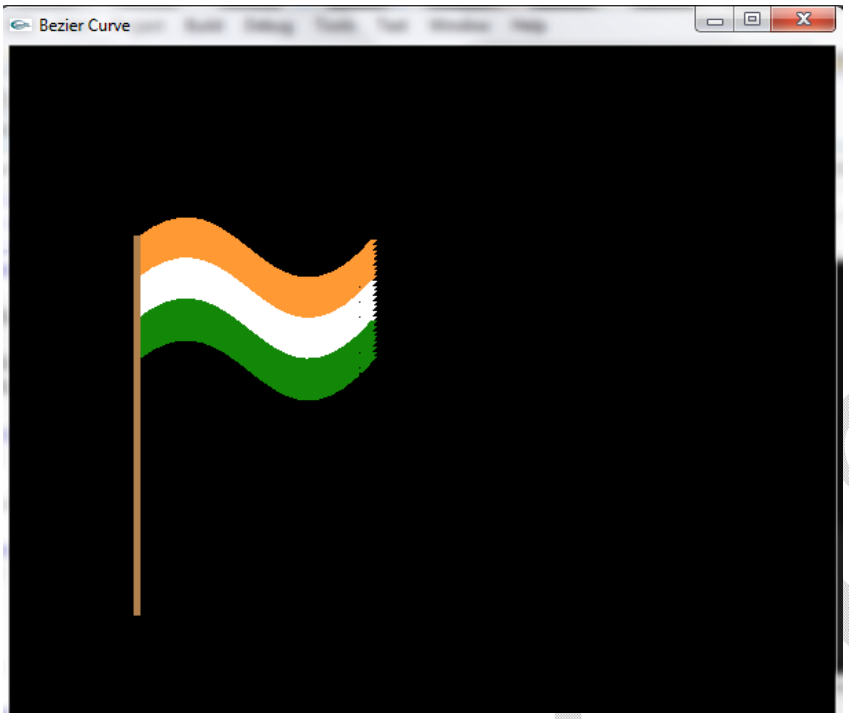
```
        }
    }

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()
    {
        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 * 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 color code
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();
}

void winReshapeFun(GLint newWidth, GLint newHeight)
{
    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:

Once you execute the above program, you will find a waving flag. The below given figure is a still-image of the same.



12. Write a program to clip lines using Liang-Barsky algorithm.

```
#include <GL/glut.h>
#define true 1
#define false 0

double xmin=50,ymin=50, xmax=100,ymax=100; // Window boundaries
double xvmin=200,yvmin=200,xvmax=300,yvmax=300; // Viewport boundaries

int cliptest(double p, double q, double *t1, double *t2)
{
        double t;
        if(p) t=q/p;  // to check whether p
        if(p < 0.0)    // potentially entry point, update te
        {
                if( t > *t1) *t1=t;
                if( t > *t2) return(false); // line portion is outside
        }
        else
                if(p > 0.0)    //  Potentially leaving point, update tl
                {
                        if( t < *t2) *t2=t;
                        if( t < *t1) 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, te=0.0, tl=1.0;
        if(cliptest(-dx,x0-xmin,&te,&tl))               // inside test wrt left edge
                if(cliptest(dx,xmax-x0,&te,&tl))        // inside test wrt right edge
                        if(cliptest(-dy,y0-ymin,&te,&tl))  // inside test wrt bottom edge
                                if(cliptest(dy,ymax-y0,&te,&tl)) // inside test wrt top edge
                                {
                                        if( tl < 1.0 )
                                        {
                                                x1 = x0 + tl*dx;
                                                y1 = y0 + tl*dy;
                                        }
                                        if( te > 0.0 )
                                        {
                                                x0 = x0 + te*dx;
                                                y0 = y0 + te*dy;
                                        }
                                        // Window to viewport mappings
                                        // Scale parameters
                                        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;
                                        //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();
                                }
```

```
}// end of line clipping

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);
	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();
	LiangBarskyLineClipAndDraw(x0,y0,x1,y1);
	glFlush();
}

void init()
{
	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);
}

int main(int argc, char** argv)
{
	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutInitWindowSize(500,500);
	glutInitWindowPosition(250,100);
	glutCreateWindow("Liang Barsky Line Clipping Algorithm");
	glutDisplayFunc(display);
	init();
	glutMainLoop();
}
```

Output: