

Table of Contents

Experiment No	Experiment Name	Page No
1	Draw the shape of a Star	2-3
2	Translation of Star Shape by taking user input of Tx & Ty	3-6
3	Rotations of Star Shape by taking user input of angle	6-9
4	Reflection of Star Shape	9-12
5	DDA Line Drawing Algorithm by taking user input	12-15
6	Bresenham Line Drawing Algorithm	15-17
7	Cohen Sutherland Line Clipping Algorithm	17-22
8	Sutherland Hodgman Polygon Clipping Algorithm	22-31
9	Weiler-Atherton Clipping Algorithm	31-38
10	Bresenham Circle Drawing Algorithm	38-40
11	Midpoint Circle Drawing Algorithm	40-42

Experiment No: 01

Experiment Name: Draw a shape (Star) in OpenGL.

Objectives: To draw a star shape.

Code:

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
#include <stdlib.h>
void consturct()
{
    glBegin(GL_TRIANGLES);

    glVertex2f(-4, 4);
    glVertex2f(-4, -4);
    glVertex2f(-10, 0);

    glVertex2f(10, 0);
    glVertex2f(4, -4);
    glVertex2f(4, 4);

    glVertex2f(4, -4);
    glVertex2f(0, -10);
    glVertex2f(-4, -4);

    glVertex2f(0, 10);
    glVertex2f(4, 4);
    glVertex2f(-4, 4);

    glEnd();
    glBegin(GL_QUADS);

    glVertex2f(4, 4);
    glVertex2f(-4, 4);
    glVertex2f(-4, -4);
    glVertex2f(4, -4);

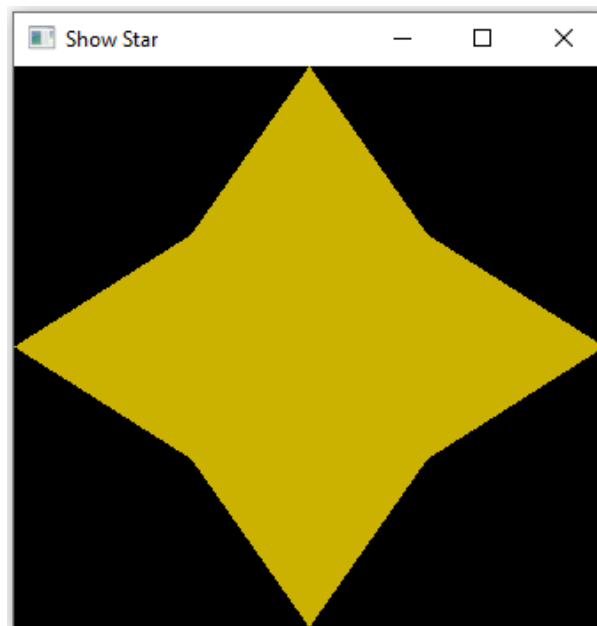
    glEnd();
    glFlush();
}
void reset(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-10, 10, -10, 10);
    glMatrixMode(GL_MODELVIEW);
}
void print()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.8, 0.7, 0.0);
    glLoadIdentity();
}
```

```

    consturct();
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutCreateWindow("Show Star");
    glutDisplayFunc(print);
    glutReshapeFunc(reset);
    glutMainLoop();
    return 0;
}

```

Output:



Result and Discussion: The OpenGL program successfully draws a star within a 400x400 window. The use of GL_TRIANGLES and GL_QUADS primitives, along with appropriate vertex coordinates, creates the desired star pattern.

Experiment No: 02

Experiment Name: Translation of a shape (Star) with user input (Tx, Ty) in OpenGL.

Objective: To translate a star shape.

Code:

```

#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

```

```

#include <stdlib.h>
#include<bits/stdc++.h>
using namespace std;
float tx,ty;
void consturct()
{
    glBegin(GL_TRIANGLES);
        glVertex2f(5, 0);
        glVertex2f(2, -2);
        glVertex2f(2, 2);
        glVertex2f(0, 5);
        glVertex2f(2, 2);
        glVertex2f(-2, 2);
        glVertex2f(-2, 2);
        glVertex2f(-2, -2);
        glVertex2f(-5, 0);
        glVertex2f(2, -2);
        glVertex2f(0, -5);
        glVertex2f(-2, -2);
    glEnd();
    glBegin(GL_QUADS);
        glVertex2f(2, 2);
        glVertex2f(-2, 2);
        glVertex2f(-2, -2);
        glVertex2f(2, -2);
    glEnd();
    glFlush();
}
void reset(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-10, 10, -10, 10);
    glMatrixMode(GL_MODELVIEW);
}
void print()
{

```

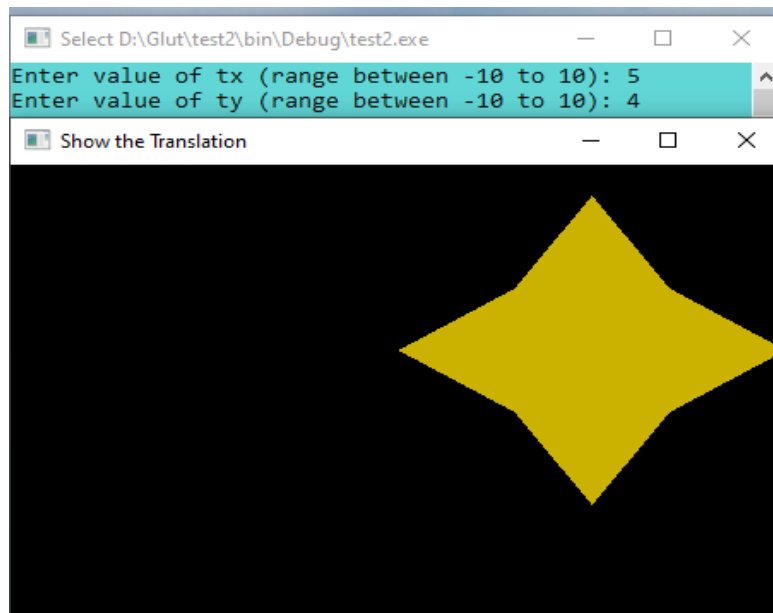
```

    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.8, 0.7, 0.0);
    glLoadIdentity();
    glTranslatef(tx, ty, 0.0);
    consturct();
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(400, 400);
    glutCreateWindow("Show the Translation");
    printf("Enter value of tx (range between -10 to 10): ");
    cin>>tx;
    printf("Enter value of ty (range between -10 to 10): ");
    cin >> ty;
    glutDisplayFunc(print);
    glutReshapeFunc(reset);
    glutMainLoop();
    return 0;
}

```

Output:



Result and Discussion: The OpenGL program successfully translates the star shape based on user input values for Tx and Ty. The translation operation is applied to the star, allowing dynamic positioning within the window, and demonstrating the interactive capabilities of the program.

Experiment No: 03

Experiment Name: Rotate a shape (Star) in OpenGL.

Objective: To rotate a star shape.

Code:

```
#include <windows.h>
#include <iostream>
#include <stdlib.h>
#ifdef __APPLE__
#include <OpenGL/OpenGL.h>
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
#include <bits/stdc++.h>
using namespace std;
float ang;
float cAng = 0.0f;
void build()
{
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);
    glClearColor(0.7f, 0.5f, 0.0f, 0.3f);
}
void Resize(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (double)w / (double)h, 1.0, 200.0);
}
void keychk(unsigned char key, int x, int y)
{

```

```

switch (key)
{
    case 27:
        exit(0);
    }
}

void construct()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotatef(-cAng, 0.0f, 1.0f, 0.0f);
    glTranslatef(0.0f, 0.0f, -5.0f);
    glPushMatrix();
    glTranslatef(0.0f, -1.0f, 0.0f);
    glRotatef(ang, 0.0f, 0.0f, -1.0f);
    glBegin(GL_POLYGON);
    glVertex2f(0.2,0.2);
    glVertex2f(0.2,-0.2);
    glVertex2f(-0.2,-0.2);
    glVertex2f(-0.2,0.2);
    glEnd();
    glBegin(GL_POLYGON);
    glVertex2f(0.5,0.0);
    glVertex2f(0.2,0.2);
    glVertex2f(0.2,-0.2);
    glEnd();
    glBegin(GL_POLYGON);
    glVertex2f(0.0,0.5);
    glVertex2f(-0.2,0.2);
    glVertex2f(0.2,0.2);
    glEnd();
    glBegin(GL_POLYGON);
    glVertex2f(-0.5,0.0);
    glVertex2f(-0.2,-0.2);
    glVertex2f(-0.2,0.2);
    glEnd();
}

```

```

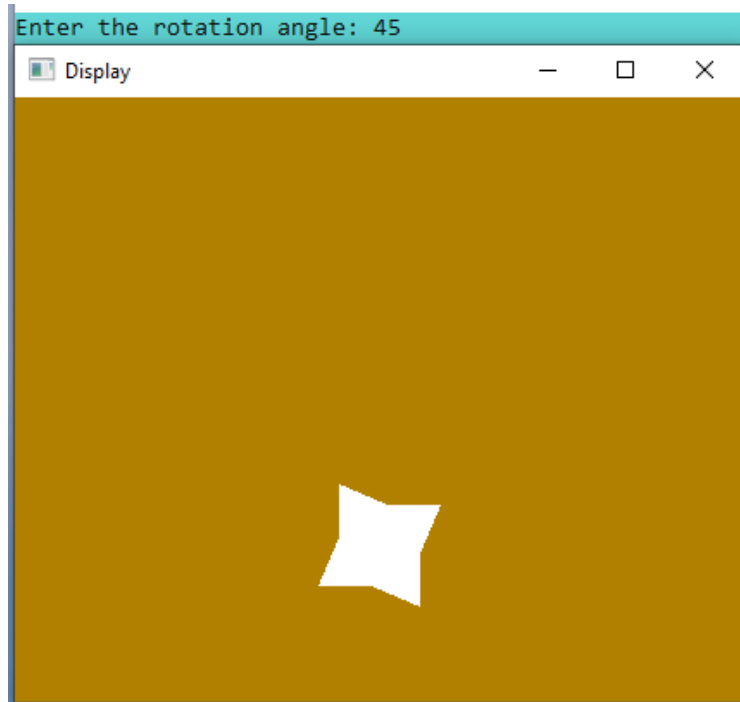
    glBegin(GL_POLYGON);
    glVertex2f(0.0,-0.5);
    glVertex2f(0.2,-0.2);
    glVertex2f(-0.2,-0.2);
    glEnd();
    glPopMatrix();
    glutSwapBuffers();
}

void modify(int value)
{
    ang += 2.0f;
    if (ang > 360)
    {
        ang -= 360;
    }
    glutPostRedisplay();
    glutTimerFunc(25, modify, 0);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(700, 500);
    glutInitWindowPosition(200, 200);
    cout<<"Enter the rotation angle: ";
    cin>>ang;
    glutCreateWindow("Display");
    build();
    glutDisplayFunc(construct);
    glutKeyboardFunc(keychk);
    glutReshapeFunc(Resize);
    glutTimerFunc(25, modify, 0);
    glutMainLoop();
    return 0;
}

```


Output:



Result and Discussion: The OpenGL program successfully rotates the star shape based on user input for the rotation angle. Pressing 'r' in the display window triggers rotation mode, allowing users to dynamically specify the angle and observe the rotation effect on the star, demonstrating the interactive rotational capabilities of the program.

Experiment No: 04

Experiment Name: Reflection of a shape (Star) in all quadrants in OpenGL.

Objective: To draw reflections of a star shape in all quadrants.

Code:

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
#include <stdlib.h>
#include <bits/stdc++.h>
using namespace std;
void reset(int w, int h)
{
    glViewport(0, 0, w, h);
```

```

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-10, 10, -10, 10, -10, 10);
    glMatrixMode(GL_MODELVIEW);
}

void construct()
{
    glBegin(GL_POLYGON);
    glVertex2f(3, 6);
    glVertex2f(5, 6);
    glVertex2f(5, 4);
    glVertex2f(3, 4);
    glEnd();
    glBegin(GL_POLYGON);
    glVertex2f(4, 7.5);
    glVertex2f(5, 6);
    glVertex2f(3, 6);
    glEnd();
    glBegin(GL_POLYGON);
    glVertex2f(5, 6);
    glVertex2f(6.5, 5);
    glVertex2f(5, 4);
    glEnd();
    glBegin(GL_POLYGON);
    glVertex2f(5, 4);
    glVertex2f(4, 2.5);
    glVertex2f(3, 4);
    glEnd();
    glBegin(GL_POLYGON);
    glVertex2f(3, 4);
    glVertex2f(1.5, 5);
    glVertex2f(3, 6);
    glEnd();
}

void print()
{
    glClear(GL_COLOR_BUFFER_BIT);

```

```

    glColor3f(1.0, 1.0, 1.0);
    glLoadIdentity();
    construct();
    glPushMatrix();
    glScalef(-1.0, 1.0, 1.0);
    glColor3f(1.0, 0.0, 0.0);
    construct();
    glPopMatrix();
    glPushMatrix();
    glScalef(1.0, -1.0, 1.0);
    glColor3f(0.0, 1.0, 0.0);
    construct();
    glPopMatrix();
    glPushMatrix();
    glScalef(-1.0, -1.0, 1.0);
    glColor3f(0.0, 0.0, 1.0);
    construct();
    glPopMatrix();
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(700, 700);
    glutInitWindowPosition (100, 100);
    glutCreateWindow("Reflection Window");
    glutDisplayFunc(print);
    glutReshapeFunc(reset);
    glutMainLoop();

    return 0;
}

```

Output:



Result and Discussion: The OpenGL program successfully draws reflections of the star shape in all quadrants by applying horizontal, vertical, and combined horizontal-vertical reflections. Different colors (red, green, and blue) are used for each reflection, illustrating the versatility of transformations in creating mirrored images of the original star shape.

Experiment No: 05

Experiment Name: Draw a line between two points using DDA line drawing algorithm in OpenGL.

Objective: To draw a line between two points using DDA line drawing algorithm.

Code:

```
#include<GL/glut.h>
#include<math.h>
#include<bits/stdc++.h>
using namespace std;
float x_1, x_2, y_1,y_2;
int sgn(float a)
```

```

{
    if(a==0)
        return 0;
    if(a<0)
        return -1;
    else
        return 1;
}

void Line()
{
    float dy,dx, length;
    dy = y_2 - y_1;
    dx = x_2 - x_1;
    if(abs(dx)>=abs(dy))
        length = abs(dx);
    else
        length = abs(dy);
    float xin,yin;
    xin = (x_2-x_1)/length;
    yin = (y_2-y_1)/length;
    float x,y;
    x = x_1 + 0.5 * sgn(xin);
    y = y_1 + 0.5 * sgn(yin);
    int i = 0;
    while(i<=length)
    {
        cout<<"\nx = "<< x <<" y = "<<y;
        glBegin(GL_POINTS);
        glVertex2i(x,y);
        glEnd();
        x = x + xin;
        y = y + yin;
        i++;
    }
    glFlush();
}

void init(void)
{
    glClearColor(0,0,0,0);

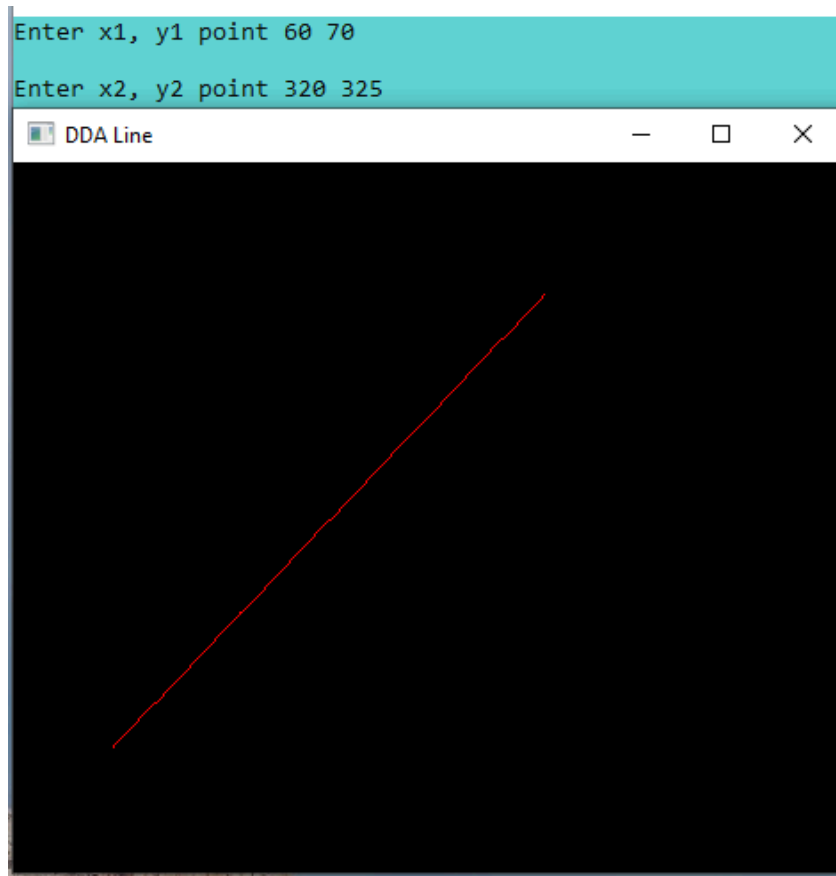
```

```

    glColor3f(1.0,0.0,0.0);
    gluOrtho2D(0,500,0,400);
    glClear(GL_COLOR_BUFFER_BIT);
}
int main(int argc,char** argv )
{ cout<<"Enter x1, y1 point\n";
  cin>>x_1>>y_1;
  cout<<"Enter x2, y2 point\n";
  cin>>x_2>>y_2;
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(500,400);
  glutCreateWindow("DDA Line");
  init();
  glutDisplayFunc(Line); glutMainLoop();
  return 0;
}

```

Output:



Result and Discussion: The OpenGL program successfully implements the DDA line drawing algorithm to draw a line between two user-specified points. The algorithm calculates incremental steps in both x and y directions, ensuring a smooth and accurate representation of the line, and demonstrating the effectiveness of the DDA algorithm for basic line drawing.

Experiment No: 06

Experiment Name: Draw a line between two points using Bresenham line drawing algorithm in OpenGL.

Objective: To draw a line between two points using Bresenham line drawing algorithm.

Code:

```
#include <GL/glut.h>
#include <windows.h>
#include <stdio.h>
GLint x0,y0,xEnd,yEnd;
void init()
{
    glClearColor(1.0,1.0,1.0,0.0);
    glColor3f(1.0f,0.0f,0.0f);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0,600.0,0.0,600.0);
}
void setPixel(GLint xcoordinate, GLint ycoordinate)
{
    glBegin(GL_POINTS);
    glVertex2i(xcoordinate,ycoordinate);
    glEnd();
    glFlush();
}
void lineBA(GLint x0,GLint y0,GLint xEnd,GLint yEnd)
{
    GLint dx = xEnd-x0;
    GLint dy = yEnd-y0;
    GLint steps,k;
    steps=dx;
    GLint x,y,p0=(2*dy)-dx;
    setPixel(x0,y0);
```

```

x=x0;
y=y0;
for(k=0; k<steps; k++)
{
    if(p0<0)
    {
        p0=p0+(2*dy);
        x+=1;
    }
    else
    {
        p0=p0+(2*dy)-(2*dx);
        x+=1;
        y+=1;
    }
    setPixel(x,y);
}
}

void readInput()
{
    printf("Enter x0, y0, xEnd, yEnd : \n");
    scanf("%i %i %i %i",&x0,&y0,&xEnd,&yEnd);

}

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    lineBA(x0,y0,xEnd,yEnd);
}

int main(int argc,char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(600,600);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Breshnam's Line Drawing Algorithm");
    readInput();
}

```

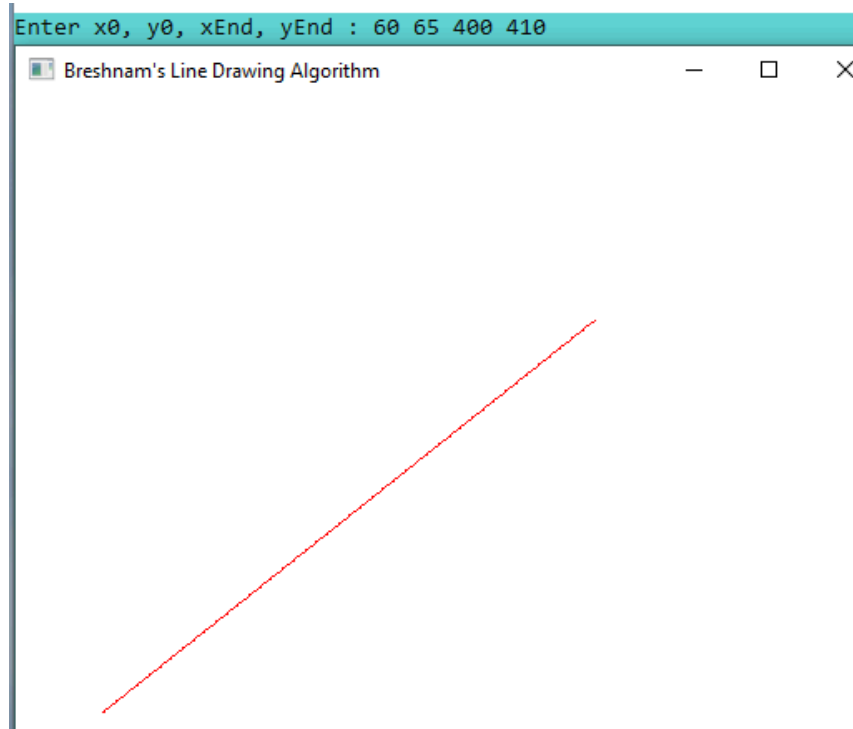


```

    glutDisplayFunc(Display);
    init();
    glutMainLoop();
    return 0;
}

```

Output:



Result and Discussion: The OpenGL program successfully utilizes the Bresenham line drawing algorithm to draw a line between two user-specified points. The Bresenham algorithm, with its arithmetic, efficiently handles various line slopes and produces accurate results, demonstrating its effectiveness for line rendering.

Experiment No: 07

Experiment Name: Clip a line inside a window using Cohen Sutherland line clipping algorithm in OpenGL.

Objective: To clip a line inside a window using Cohen Cohen-Sutherland line clipping algorithm.

Code:

```

#include <GL/glut.h>
#define SCREEN_WIDTH 640
#define SCREEN_HEIGHT 480
typedef struct
{
    GLfloat x, y;

```

```

} Point;
const GLint WIN_LEFT_BIT = 0x01;
const GLint WIN_RIGHT_BIT = 0x02;
const GLint WIN_BOTTOM_BIT = 0x04;
const GLint WIN_TOP_BIT = 0x08;
void init_graph(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(SCREEN_WIDTH, SCREEN_HEIGHT);
    glutCreateWindow(argv[0]);
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glPointSize(1.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, SCREEN_WIDTH, 0, SCREEN_HEIGHT);
}
void close_graph()
{
    glutMainLoop();
}
void swap_points(Point *p1, Point *p2)
{
    Point t = *p1;
    *p1 = *p2;
    *p2 = t;
}
void swap_codes(GLint *x, GLint *y)
{
    GLint t = *x;
    *x = *y;
    *y = t;
}
GLint inside(GLint code)
{
    return !code;
}

```

```

GLint accept(GLint code1, GLint code2)
{
    return !(code1 | code2);
}

GLint reject(GLint code1, GLint code2)
{
    return code1 & code2;
}

GLint encode(Point p1, Point win_min, Point win_max)
{
    GLint code = 0x00;
    if (p1.x < win_min.x) code |= WIN_LEFT_BIT;
    if (p1.x > win_max.x) code |= WIN_RIGHT_BIT;
    if (p1.y < win_min.y) code |= WIN_BOTTOM_BIT;
    if (p1.y > win_max.y) code |= WIN_TOP_BIT;
    return code;
}

GLint round(GLfloat a)
{
    return (GLint) (a + 0.5f);
}

void line_clip(Point p1, Point p2, Point win_min, Point win_max)
{
    GLint code1, code2;
    GLint done = 0, plot_line = 0;
    GLfloat m = 0;
    if (p1.x != p2.x)
    {
        m = (p2.y - p1.y) / (p2.x - p1.x);
    }
    while (!done)
    {
        code1 = encode(p1, win_min, win_max);
        code2 = encode(p2, win_min, win_max);
        if (accept(code1, code2))
        {
            done = 1;

```

```

    plot_line = 1;
}
else if (reject(code1, code2))
{
    done = 1;
}
else
{
    if (inside(code1))
    {
        swap_points(&p1, &p2);
        swap_codes(&code1, &code2);
    }

    if (code1 & WIN_LEFT_BIT)
    {
        p1.y += (win_min.x - p1.x) * m;
        p1.x = win_min.x;
    }
    else if (code1 & WIN_RIGHT_BIT)
    {
        p1.y += (win_max.x - p1.x) * m;
        p1.x = win_max.x;
    }
    else if (code1 & WIN_BOTTOM_BIT)
    {
        if (p1.x != p2.x)
            p1.x += (win_min.y - p1.y) / m;
        p1.y = win_min.y;
    }
    else if (code1 & WIN_TOP_BIT)
    {
        if (p1.x != p2.x)
            p1.x += (win_max.y - p1.y) / m;
        p1.y = win_max.y;
    }
}
}

```

```

    }
    if (plot_line)
    {
        glColor3f(1, 0, 0);
        glLineWidth(2);
        glBegin(GL_LINES);
        glVertex2i(round(p1.x), round(p1.y));
        glVertex2i(round(p2.x), round(p2.y));
        glEnd();
        glFlush();
    }
}

void draw_window(Point win_min, Point win_max)
{
    glColor3f(0, 0, 0);
    glBegin(GL_LINE_LOOP);
    glVertex2i(round(win_min.x), round(win_min.y));
    glVertex2i(round(win_min.x), round(win_max.y));
    glVertex2i(round(win_max.x), round(win_max.y));
    glVertex2i(round(win_max.x), round(win_min.y));
    glEnd();
    glFlush();
}

void init_clip()
{
    glClear(GL_COLOR_BUFFER_BIT);
    Point win_min = {60, 60};
    Point win_max = {470, 290};
    draw_window(win_min, win_max);
    Point p1 = {50, 50};
    Point p2 = {490, 310};
    glColor3f(0, 0, 1);
    glBegin(GL_LINES);
    glVertex2i(round(p1.x), round(p1.y));
    glVertex2i(round(p2.x), round(p2.y));
    glEnd();
    line_clip(p1, p2, win_min, win_max);
}

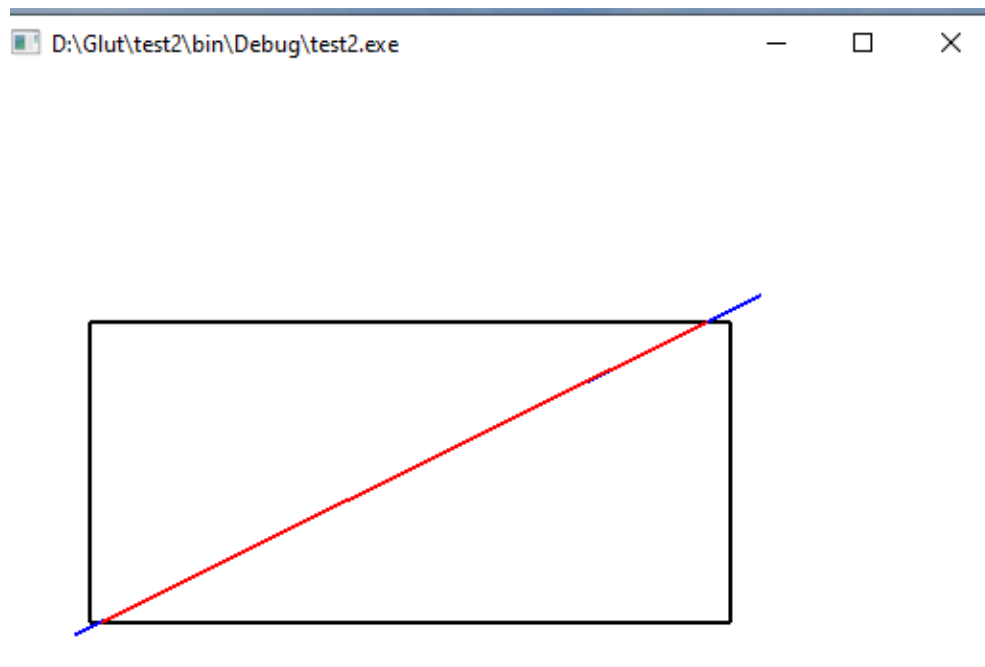
```

```

}
int main(int argc, char **argv)
{
    init_graph(argc, argv);
    glutDisplayFunc(init_clip);
    close_graph();
    return EXIT_SUCCESS;
}

```

Output:



Result and Discussion: The OpenGL program successfully implements the Cohen-Sutherland line clipping algorithm to clip a line segment within a specified window. The red line represents the clipped portion, ensuring that only the segment inside the defined clipping window is displayed, demonstrating the effectiveness of the Cohen-Sutherland algorithm for line clipping.

Experiment No: 8

Experiment Name: Clip a polygon inside a window using Sutherland-Hodgman polygon clipping algorithm in OpenGL.

Objective: To clip a polygon inside a window using Sutherland-Hodgman polygon clipping algorithm.

Code:

```

#include<stdio.h>
#include<GL/gl.h>
#include<GL/glu.h>
#include<GL/glut.h>

```

```

#include<math.h>
typedef struct
{
    float x;
    float y;
} PT;

int n;
int i,j;
PT p1,p2,p[20],pp[20];
void left()
{
    i=0,j=0;
    for(i=0; i<n; i++)
    {
        if(p[i].x<p1.x && p[i+1].x>=p1.x)
        {
            if(p[i+1].x-p[i].x!=0)
            {
                pp[j].y=(p[i+1].y-p[i].y)/(p[i+1].x-p[i].x)*(p1.x-p[i].x)+p[i].y;
            }
            else
            {
                pp[j].y=p[i].y;
            }
            pp[j].x=p1.x;
            j++;
            pp[j].x=p[i+1].x;
            pp[j].y=p[i+1].y;
            j++;
        }

        if(p[i].x>=p1.x && p[i+1].x>=p1.x)
        {
            pp[j].y=p[i+1].y;
            pp[j].x=p[i+1].x;
        }
    }
}

```

```

        j++;
    }
    if(p[i].x>=p1.x && p[i+1].x<p1.x)
    {
        if(p[i+1].x-p[i].x!=0)
        {
            pp[j].y=(p[i+1].y-p[i].y)/(p[i+1].x-p[i].x)*(p1.x-p[i].x)+p[i].y;
        }
        else
        {
            pp[j].y=p[i].y;
        }
        pp[j].x=p1.x;
        j++;
    }
}

for(i=0; i<j; i++)
{
    p[i].x=pp[i].x;
    p[i].y=pp[i].y;
}

p[i].x=pp[0].x;
p[i].y=pp[0].y;
n=j;
}

void right()
{
    i=0,j=0;
    for(i=0; i<n; i++)
    {
        if(p[i].x>p2.x && p[i+1].x<=p2.x)
        {
            if(p[i+1].x-p[i].x!=0)
            {
                pp[j].y=(p[i+1].y-p[i].y)/(p[i+1].x-p[i].x)*(p2.x-p[i].x)+p[i].y;

```



```

    }
    else
    {
        pp[j].y=p[i].y;
    }
    pp[j].x=p2.x;
    j++;
    pp[j].x=p[i+1].x;
    pp[j].y=p[i+1].y;
    j++;
}

if(p[i].x<=p2.x && p[i+1].x<=p2.x)
{
    pp[j].y=p[i+1].y;
    pp[j].x=p[i+1].x;
    j++;
}

if(p[i].x<=p2.x && p[i+1].x>p2.x)
{
    if(p[i+1].x-p[i].x!=0)
    {
        pp[j].y=(p[i+1].y-p[i].y)/(p[i+1].x-p[i].x)*(p2.x-p[i].x)+p[i].y;
    }
    else
    {
        pp[j].y=p[i].y;
    }
    pp[j].x=p2.x;
    j++;
}

}
for(i=0; i<j; i++)
{
    p[i].x=pp[i].x;

```

```

        p[i].y=pp[i].y;
    }
    p[i].x=pp[0].x;
    p[i].y=pp[0].y;

}

void top()
{
    i=0,j=0;
    for(i=0; i<n; i++)
    {
        if(p[i].y>p2.y && p[i+1].y<=p2.y)
        {
            if(p[i+1].y-p[i].y!=0)
            {
                pp[j].x=(p[i+1].x-p[i].x)/(p[i+1].y-p[i].y)*(p2.y-p[i].y)+p[i].x;
            }
            else
            {
                pp[j].x=p[i].x;
            }
            pp[j].y=p2.y;
            j++;
            pp[j].x=p[i+1].x;
            pp[j].y=p[i+1].y;
            j++;
        }
        if(p[i].y<=p2.y && p[i+1].y<=p2.y)
        {
            pp[j].y=p[i+1].y;
            pp[j].x=p[i+1].x;
            j++;
        }
        if(p[i].y<=p2.y && p[i+1].y>p2.y)
        {
            if(p[i+1].y-p[i].y!=0)
            {

```

```

        pp[j].x=(p[i+1].x-p[i].x)/(p[i+1].y-p[i].y)*(p2.y-p[i].y)+p[i].x;
    }
    else
    {
        pp[j].x=p[i].x;
    }
    pp[j].y=p2.y;
    j++;
}
}
for(i=0; i<j; i++)
{
    p[i].x=pp[i].x;
    p[i].y=pp[i].y;
}
p[i].x=pp[0].x;
p[i].y=pp[0].y;
n=j;
}
void bottom()
{
    i=0,j=0;
    for(i=0; i<n; i++)
    {
        if(p[i].y<p1.y && p[i+1].y>=p1.y)
        {
            if(p[i+1].y-p[i].y!=0)
            {
                pp[j].x=(p[i+1].x-p[i].x)/(p[i+1].y-p[i].y)*(p1.y-p[i].y)+p[i].x;
            }
            else
            {
                pp[j].x=p[i].x;
            }
            pp[j].y=p1.y;
            j++;
            pp[j].x=p[i+1].x;

```

```

        pp[j].y=p[i+1].y;
        j++;
    }
    if(p[i].y>=p1.y && p[i+1].y>=p1.y)
    {
        pp[j].x=p[i+1].x;
        pp[j].y=p[i+1].y;
        j++;
    }
    if(p[i].y>=p1.y && p[i+1].y<p1.y)
    {
        if(p[i+1].y-p[i].y!=0)
        {
            pp[j].x=(p[i+1].x-p[i].x)/(p[i+1].y-p[i].y)*(p1.y-p[i].y)+p[i].x;
        }
        else
        {
            pp[j].x=p[i].x;
        }
        pp[j].y=p1.y;
        j++;
    }
}
for(i=0; i<j; i++)
{
    p[i].x=pp[i].x;
    p[i].y=pp[i].y;
}
p[i].x=pp[0].x;
p[i].y=pp[0].y;
n=j;
}
void drawpolygon()
{
    glColor3f(1.0,0.0,0.0);
    for(i=0; i<n-1; i++)
    {

```

```

        glBegin(GL_LINES);
        glVertex2d(p[i].x,p[i].y);
        glVertex2d(p[i+1].x,p[i+1].y);
        glEnd();
    }
    glBegin(GL_LINES);
    glVertex2d(p[i].x,p[i].y);
    glVertex2d(p[0].x,p[0].y);
    glEnd();
}

void myMouse(int button, int state, int x, int y)
{
    if(button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
    {
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_LINE_LOOP);
        glVertex2f(p1.x,p1.y);
        glVertex2f(p2.x,p1.y);
        glVertex2f(p2.x,p2.y);
        glVertex2f(p1.x,p2.y);
        glEnd();
        left(); right();
        top(); bottom();
        drawpolygon();
    }
    glFlush();
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.4,1.0,0.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(p1.x, p1.y);
    glVertex2f(p2.x,p1.y);
    glVertex2f(p2.x,p2.y);
    glVertex2f(p1.x,p2.y);
    glEnd();
}

```

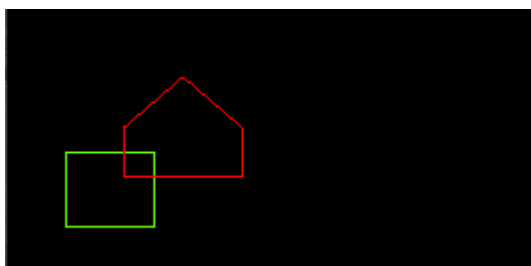
```

        drawpolygon();
        glFlush();
    }
void init(void)
{
    glClearColor(0.0,0.0,0.0,0.0);
    gluOrtho2D(0,500,0,500);
}
int main(int argc, char**argv)
{
    printf("Enter Window Coordinates: ");
    printf("\nPlease Enter two Points: ");
    printf("Enter P1(x,y) & P2(x,y: ");
    scanf("%f %f %f %f", &p1.x,&p1.y,&p2.x,&p2.y);
    printf("Enter the no. of vertices: ");
    scanf("%d", &n);
    for(i=0; i<n; i++)
    {
        printf("Enter V%d(x%d,y%d): ", i+1, i+1, i+1);
        scanf("%f", &p[i].x);
        scanf("%f", &p[i].y);
    }
    p[i].x=p[0].x;
    p[i].y=p[0].y;
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(640,480);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Sutherland Hodgman Polygon Clipping Algorithm");
    init();
    glutDisplayFunc(display);
    glutMouseFunc(myMouse);
    glFlush();
    glutMainLoop();
    return 0;
}

```

Output:

```
D:\Glut\test2\bin\Debug\test2.exe
Enter Window Coordinates:
Please Enter two Points: Enter P1(x,y) & P2(x,y: 20 20 50 50
Enter the no. of vertices: 5
Enter V1(x1,y1): 40 40
Enter V2(x2,y2): 40 60
Enter V3(x3,y3): 60 80
Enter V4(x4,y4): 80 60
Enter V5(x5,y5): 80 40
```



Result and Discussion: The OpenGL program successfully implements the Sutherland-Hodgman polygon clipping algorithm to clip a user-defined polygon within a specified window. The grey portion represents the original polygon, and the green portion shows the clipped result inside the defined clipping window, demonstrating the effectiveness of the Sutherland-Hodgman algorithm for polygon clipping.

Experiment No: 09

Experiment Name: Weiler-Atherton Clipping Algorithm using OpenGL.

Objectives: To clip a polygon against a complex clipping window by performing set operations on the polygon and window.

Code:

```
#include <iostream>
#include <cstring>
#include <stdio>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <vector>
#include <list>
#include <algorithm>
#include <functional>
#include <GL/glut.h>
#include <GL/glu.h>
#include <GL/gl.h>
#define Size 600
using namespace std;
typedef float Color[3];
struct Point
{
    int x, y;
};
typedef struct IntersectionPoint
```

```

{
    int pointFlag;
    int index0, index1;
    Point p;
    bool inFlag;
    int dis;
} IP;
class Pg
{
public:
    vector<Point> pts;
    Pg(void);
    ~Pg(void);
    void drawPgLine(Color c);
};
Pg::Pg(void)
{
}
Pg::~~Pg(void)
{
}
void Pg::drawPgLine(Color c)
{
    glColor3fv(c);
    glLineWidth(2.0);
    glBegin(GL_LINE_LOOP);
    int size = pts.size();
    for (int i = 0; i < size; i++)
        glVertex2i(pts[i].x, pts[i].y);
    glEnd();
}
bool isPointInsidePg(Point p, Pg& py)
{
    int cnt = 0, size = py.pts.size();
    for (int i = 0; i < size; i++)
    {
        Point p1 = py.pts[i];
        Point p2 = py.pts[(i + 1) % size];
        if (p1.y == p2.y) continue;
        if (p.y < min(p1.y, p2.y)) continue;
        if (p.y >= max(p1.y, p2.y)) continue;
        double x = (double)(p.y - p1.y) * (double)(p2.x - p1.x) / (double)(p2.y - p1.y) + p1.x;
        if (x > p.x) cnt++;
    }
    return (cnt % 2 == 1);
}
int cross(Point& p0, Point& p1, Point& p2)
{
    return ((p2.x - p0.x) * (p1.y - p0.y) - (p1.x - p0.x) * (p2.y - p0.y));
}
bool onSegment(Point& p0, Point& p1, Point& p2)
{
    int minx = min(p0.x, p1.x), maxx = max(p0.x, p1.x);
    int miny = min(p0.y, p1.y), maxy = max(p0.y, p1.y);
    if (p2.x >= minx && p2.x <= maxx && p2.y >= miny && p2.y <= maxy) return true;
    return false;
}

```



```

}
bool segmentsIntersect(Point& p1, Point& p2, Point& p3, Point& p4)
{
    int d1 = cross(p3, p4, p1);
    int d2 = cross(p3, p4, p2);
    int d3 = cross(p1, p2, p3);
    int d4 = cross(p1, p2, p4);
    if (((d1 > 0 && d2 < 0) || (d1 < 0 && d2 > 0)) &&
        ((d3 > 0 && d4 < 0) || (d3 < 0 && d4 > 0)))
        return true;
    if (d1 == 0 && onSegment(p3, p4, p1)) return true;
    if (d2 == 0 && onSegment(p3, p4, p2)) return true;
    if (d3 == 0 && onSegment(p1, p2, p3)) return true;
    if (d4 == 0 && onSegment(p1, p2, p4)) return true;
    return false;
}
Point getIntersectPoint(Point p1, Point p2, Point p3, Point p4)
{
    Point p;
    int b1 = (p2.y - p1.y) * p1.x + (p1.x - p2.x) * p1.y;
    int b2 = (p4.y - p3.y) * p3.x + (p3.x - p4.x) * p3.y;
    int D = (p2.x - p1.x) * (p4.y - p3.y) - (p4.x - p3.x) * (p2.y - p1.y);
    int D1 = b2 * (p2.x - p1.x) - b1 * (p4.x - p3.x);
    int D2 = b2 * (p2.y - p1.y) - b1 * (p4.y - p3.y);
    p.x = D1 / D;
    p.y = D2 / D;
    return p;
}
void generateIntersectPoints(Pg& pyclick, Pg& py, list<IP>& ipList)
{
    int clipSize = pyclick.pts.size(), pySize = py.pts.size();

    for (int i = 0; i < clipSize; i++)
    {
        Point p1 = pyclick.pts[i];
        Point p2 = pyclick.pts[(i + 1) % clipSize];
        for (int j = 0; j < pySize; j++)
        {
            Point p3 = py.pts[j];
            Point p4 = py.pts[(j + 1) % pySize];
            if (segmentsIntersect(p1, p2, p3, p4))
            {
                IP ip;
                ip.index0 = j;
                ip.index1 = i;
                ip.p = getIntersectPoint(p1, p2, p3, p4);
                ipList.push_back(ip);
            }
        }
    }
}
int getDistance(Point& p1, Point& p2)
{
    return (p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y);
}
bool distanceComparator(IP& ip1, IP& ip2)

```

```

{
    return ip1.dis < ip2.dis;
}
void generateList(Pg& py, list<IP>& iplist, list<IP>& comlist, int index)
{
    int size = py.pts.size();
    list<IP>::iterator it;

    for (int i = 0; i < size; i++)
    {
        Point p1 = py.pts[i];
        IP ip;
        ip.pointFlag = 0;
        ip.p = p1;
        comlist.push_back(ip);
        list<IP> oneSeg;
        for (it = iplist.begin(); it != iplist.end(); it++)
        {
            if ((index == 0 && i == it->index0) ||
                (index == 1 && i == it->index1))
            {
                it->dis = getDistance(it->p, p1);
                it->pointFlag = 1;
                oneSeg.push_back(*it);
            }
        }
        oneSeg.sort(distanceComparator);
        for (it = oneSeg.begin(); it != oneSeg.end(); it++)
            comlist.push_back(*it);
    }
}
void getPgPointInOut(list<IP>& Pglist, Pg& pyclip)
{
    bool inFlag;
    list<IP>::iterator it;
    for (it = Pglist.begin(); it != Pglist.end(); it++)
    {
        if (it->pointFlag == 0)
        {
            if (isPointInsidePg(it->p, pyclip))
                inFlag = true;
            else inFlag = false;
        }
        else
        {
            inFlag = !inFlag;
            it->inFlag = inFlag;
        }
    }
}
bool operator==(Point& p1, Point& p2)
{
    return p1.x == p2.x && p1.y == p2.y;
}
void getClipPointInOut(list<IP>& cliplist, list<IP>& Pglist)
{

```

```

list<IP>::iterator it, it1;
for (it = cliplist.begin(); it != cliplist.end(); it++)
{
    if (it->pointFlag == 0) continue;
    for (it1 = Pglist.begin(); it1 != Pglist.end(); it1++)
    {
        if (it1->pointFlag == 0) continue;
        if (it->p == it1->p) it->inFlag = it1->inFlag;
    }
}
}
void generateClipArea(list<IP>& Pglist, list<IP>& cliplist)
{
    list<IP>::iterator it, it1;
    Pg py;
    Color c = { 0.0, 0.0, 1.0 };

    for (it = Pglist.begin(); it != Pglist.end(); it++)
        if (it->pointFlag == 1 && it->inFlag) break;
    py.pts.clear();

    while (true)
    {
        if (it == Pglist.end()) break;
        py.pts.push_back(it->p);
        for (; it != Pglist.end(); it++)
        {
            if (it->pointFlag == 1 && !it->inFlag) break;
            py.pts.push_back(it->p);
        }
        for (it1 = cliplist.begin(); it1 != cliplist.end(); it1++)
            if (it1->p == it->p) break;

        for (; it1 != cliplist.end(); it1++)
        {
            if (it1->pointFlag == 1 && it1->inFlag) break;
            py.pts.push_back(it1->p);
        }
        if (py.pts[0] == it1->p)
        {
            py.drawPgLine(c);
            py.pts.clear();
            for (; it1 != Pglist.end(); it1++)
                if (it1->pointFlag == 1 && it1->inFlag) break;
            continue;
        }
        for (; it != Pglist.end(); it++)
            if (it->p == it1->p) break;
    }
}
void weilerAtherton(Pg& pyclip, Pg& py)
{
    list<IP> iplist, Pglist, cliplist;
    generateIntersectPoints(pyclip, py, iplist);
    generateList(py, iplist, Pglist, 0);
    generateList(pyclip, iplist, cliplist, 1);
}

```

```

    getPgPointInOut(Pglist, pycclip);
    getClipPointInOut(cliplist, Pglist);
    generateClipArea(Pglist, cliplist);
}
void init()
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glColor3f(1.0, 0.0, 0.0);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, Size - 1, 0.0, Size - 1);
}
void GenerateRandomSimplePg(Pg &G, int M)
{
    Point P;
    G.pts.clear();
    for (int i = 0; i < M; ++i)
    {
        bool flag;
        do
        {
            P.x = rand() % Size;
            P.y = rand() % Size;
            flag = true;
            for (int j = 1; j < i - 1; ++j)
                if (segmentsIntersect(G.pts[j - 1], G.pts[j], G.pts[i - 1], P))
                {
                    flag = false;
                    break;
                }
            if (flag && i == M - 1)
            {
                for (int j = 2; j < i; ++j)
                    if (segmentsIntersect(G.pts[j - 1], G.pts[j], P, G.pts[0]))
                    {
                        flag = false;
                        break;
                    }
            }
        } while (!flag);
        G.pts.push_back(P);
    }
}
void KeyboardAction(unsigned char key, int x, int y)
{
    exit(0);
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glEnable(GL_POINT_SMOOTH);
    Pg pycclip, py;
    //GenerateRandomSimplePg(pycclip, 4);
    //GenerateRandomSimplePg(py, 4);

```

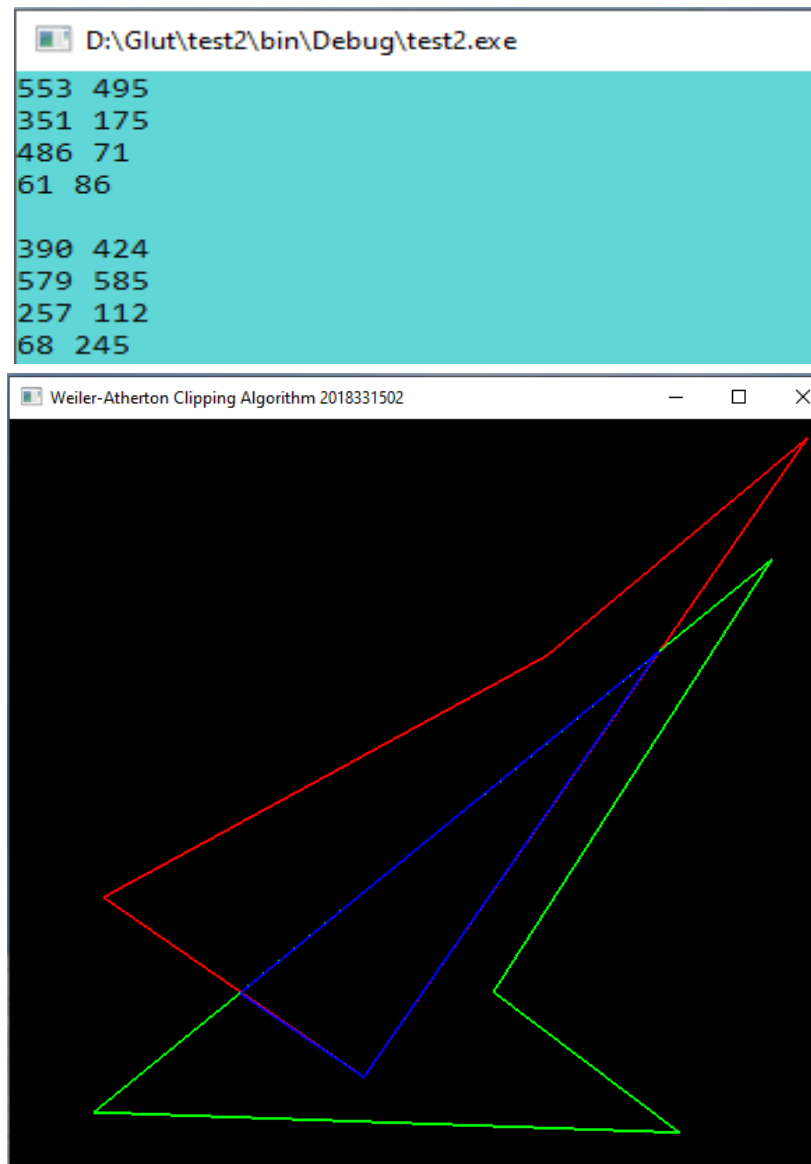
```

Point p1, p2, p3, p4;
p1.x = 553, p1.y = 495;
p2.x = 351, p2.y = 175;
p3.x = 486, p3.y = 71;
p4.x = 61, p4.y = 86;
pyclip.pts.push_back(p1);
pyclip.pts.push_back(p2);
pyclip.pts.push_back(p3);
pyclip.pts.push_back(p4);
Point p5, p6, p7, p8;
p5.x = 390, p5.y = 424;
p6.x = 579, p6.y = 585;
p7.x = 257, p7.y = 112;
p8.x = 68, p8.y = 245;
py.pts.push_back(p5);
py.pts.push_back(p6);
py.pts.push_back(p7);
py.pts.push_back(p8);
int size = pyclick.pts.size();
for (int i = 0; i < size; ++i)
    cout << pyclick.pts[i].x << " " << pyclick.pts[i].y << endl;
cout << endl;
size = py.pts.size();
for (int i = 0; i < size; ++i)
    cout << py.pts[i].x << " " << py.pts[i].y << endl;
Color a = { 1.0, 0.0, 0.0 };
Color b = { 0.0, 1.0, 0.0 };
py.drawPgLine(a);
pyclip.drawPgLine(b);
weilerAtherton(pyclick, py);

glFlush();
}
int main(int argc, char **argv)
{
    srand(time(NULL));
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(Size, Size);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Weiler-Atherton Clipping Algorithm 2018331502");
    glutKeyboardFunc(KeyboardAction);
    glutDisplayFunc(display);
    init();
    glutMainLoop();
    return 0;
}

```

Output:



Result and Discussion: The implemented Weiler Atherton polygon clipping algorithm successfully clips the example polygon inside the specified window. The original green window polygon is displayed along with the red clipped subject polygon, displaying the blue portion is the original visible part of the following clipping algorithm, demonstrating the effective application of the algorithm in isolating the visible region.

Experiment No: 10

Experiment Name: Draw a circle using Bresenham circle drawing algorithm in OpenGL.

Objective: To draw a circle using Bresenham circle drawing algorithm.

Code:

```
#include <GL/glut.h>
#include<bits/stdc++.h>
using namespace std;
int radius,centerX,centerY;
void drawCircle(int x, int y) {
```

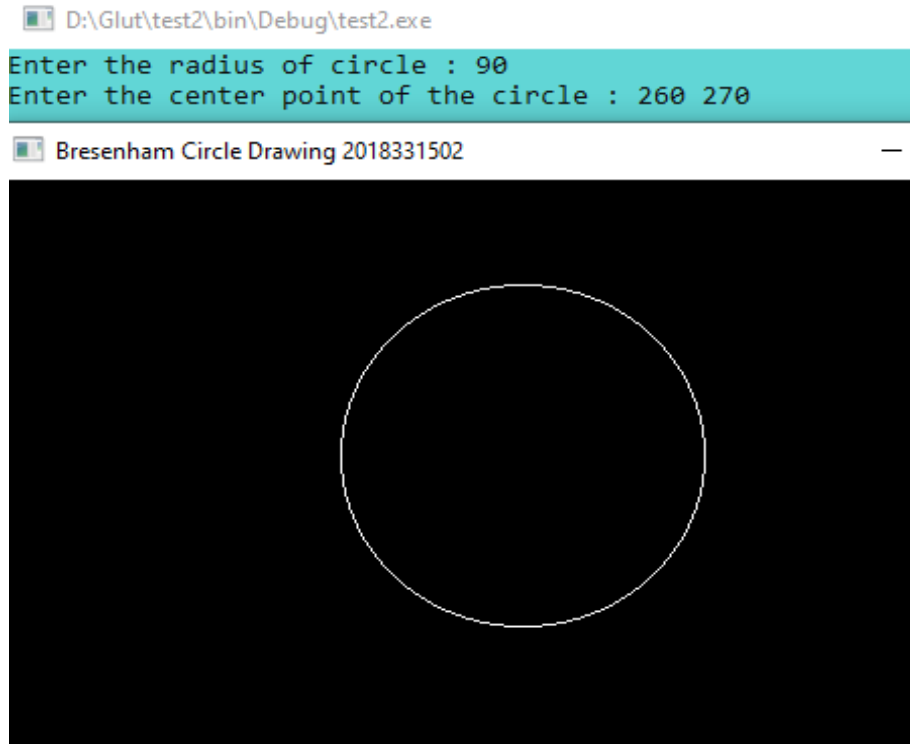
```

    glBegin(GL_POINTS);
    glVertex2i(centerX + x, centerY + y);
    glVertex2i(centerX + x, centerY - y);
    glVertex2i(centerX - x, centerY + y);
    glVertex2i(centerX - x, centerY - y);
    glVertex2i(centerX + y, centerY + x);
    glVertex2i(centerX + y, centerY - x);
    glVertex2i(centerX - y, centerY + x);
    glVertex2i(centerX - y, centerY - x);
    glEnd();
}
void bresenhamCircle() {
    int x = 0, y = radius;
    int decision = 3 - 2 * radius;

    while (x <= y) {
        drawCircle(x, y);
        if (decision < 0) {
            decision += 4 * x + 6;
            x++;
        } else {
            decision += 4 * (x - y) + 10;
            x++;
            y--;
        }
    }
}
void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    bresenhamCircle();
    glFlush();
}
void reshape(int width, int height) {
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, width, 0, height);
    glMatrixMode(GL_MODELVIEW);
}
int main(int argc, char** argv) {
    cout<<"Enter the radius of circle : ";
    cin>>radius;
    cout<<"Enter the center point of the circle : ";
    cin>>centerX>>centerY;
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 480);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Bresenham Circle Drawing 2018331502");
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glutMainLoop();
    return 0;
}

```

Output:



Result and Discussion:

The OpenGL program successfully implements the Bresenham circle drawing algorithm to draw a circle with a specified center and radius. The algorithm efficiently calculates and plots points in the eight octants of the circle, producing an accurate and symmetrical representation, demonstrating the effectiveness of Bresenham's algorithm for circle drawing.

Experiment No: 11

Experiment Name: Draw a circle using Mid-Point circle drawing algorithm in OpenGL.

Objective: To draw a circle using Mid-Point circle drawing algorithm.

Code:

```
#include <GL/glut.h>
#include<bits/stdc++.h>
using namespace std;
int radius ,centerX , centerY;
void drawCircle(int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2i(centerX + x, centerY + y);
    glVertex2i(centerX + x, centerY - y);
    glVertex2i(centerX - x, centerY + y);
    glVertex2i(centerX - x, centerY - y);
    glVertex2i(centerX + y, centerY + x);
    glVertex2i(centerX + y, centerY - x);
    glVertex2i(centerX - y, centerY + x);
    glVertex2i(centerX - y, centerY - x);
}
```

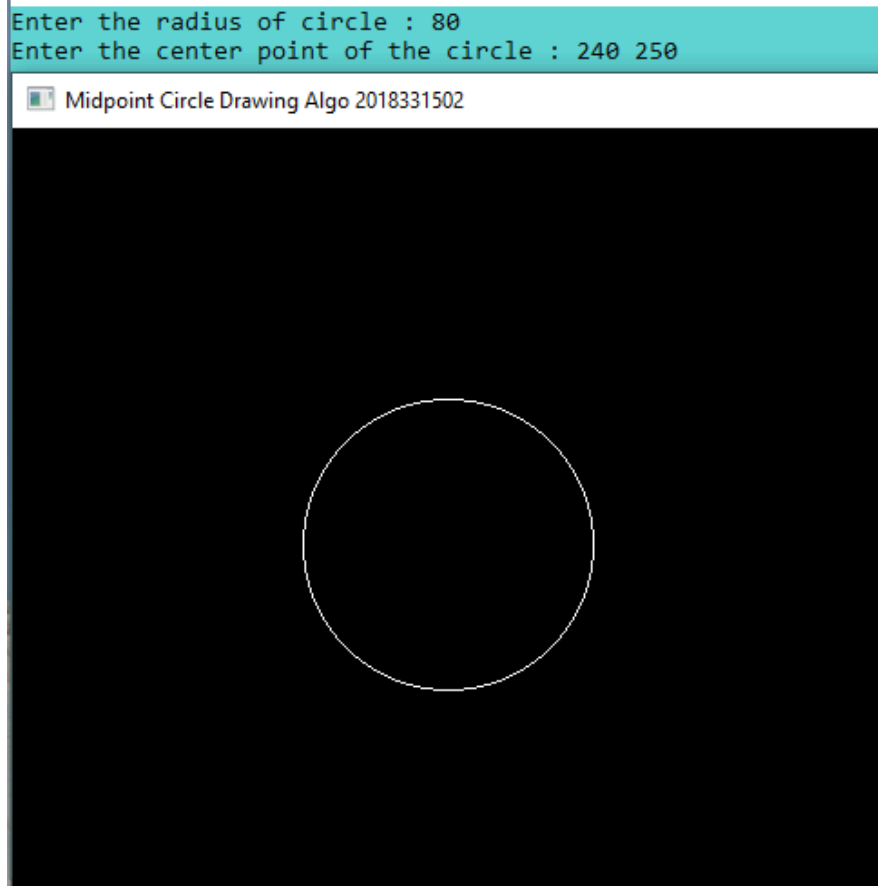


```

    glEnd();
}
void midpointCircle()
{
    int x = 0;
    int y = radius;
    int decision = 1 - radius;
    while (x <= y)
    {
        drawCircle(x, y);
        if (decision < 0)
        {
            decision += 2 * x + 3;
            x++;
        }
        else
        {
            decision += 2 * (x - y) + 5;
            x++;
            y--;
        }
    }
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    midpointCircle();
    glFlush();
}
void reshape(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, width, 0, height);
    glMatrixMode(GL_MODELVIEW);
}
int main(int argc, char** argv)
{
    cout<<"Enter the radius of circle : ";
    cin>>radius;
    cout<<"Enter the center point of the circle : ";
    cin>>centerX>>centerY;
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Midpoint Circle Drawing Algo 2018331502");
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glutMainLoop();
    return 0;
}

```

Output:



Result & Discussion:

The OpenGL program successfully implements the Mid-Point circle drawing algorithm to draw a circle with a specified center and radius. This algorithm efficiently calculates and plots points in the circle's octants, producing an accurate and symmetrical representation. The Mid-Point circle drawing algorithm is known for its efficiency in minimizing calculations and is widely used for drawing circles.