**NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY**

**Computer Graphics (COCSE64)**

**Practical File**



**Submitted by :**

Ashish Kumar

(2019UCO1518)

|  |  |
| --- | --- |
| Exp no. | Experiment Name |
| 1. | Generating line primitives using DDA |
| 2. | Generating line primitives using Bresenham’s Approach |
| 3. | Generating circle using bresenham’s approach |
| 4. | Generating circle using mid point algorithm |
| 5. | Generating ellipse using mid-point approach |
| 6. | Generating hyperbola using mid point algorithm |
| 7. | Implement line clipping approach using cohen sutherland |
| 8. | Implement line clipping approach using liang barsky / cyrus beck |
| 9. | Implement line clipping approach using mid-point subdivision |

**1 .Generating line primitives using DDA**

#include <iostream>

#include <graphics.h>

#include <cmath>

#include <time.h>

using namespace std;

//function to generate the line

void DDALine(int x0, int y0, int x1, int y1){

    int dx = x1 - x0;

    int dy = y1 - y0;

    int step = (abs(dx) > abs(dy))? abs(dx) : abs(dy);

    float x\_step = (float)dx/step;

    float y\_step = (float)dy/step;

    float x = x0;

    float y = y0;

    for(int i = 0; i < step; i++){

        putpixel(round(x), round(y), WHITE);

        // cout << round(x) << " " << round(y) << endl;

        x += x\_step;

        y += y\_step;

        delay(10);

    }

}

//driver function

int main(){

    initwindow(500,500);

    int x0, y0, x1, y1;

    cout << "Enter the coordinates of the points: ";

    cin >> x0 >> y0 >> x1 >> y1;

    DDALine(x0, y0, x1, y1);

    delay(100);

    getch();

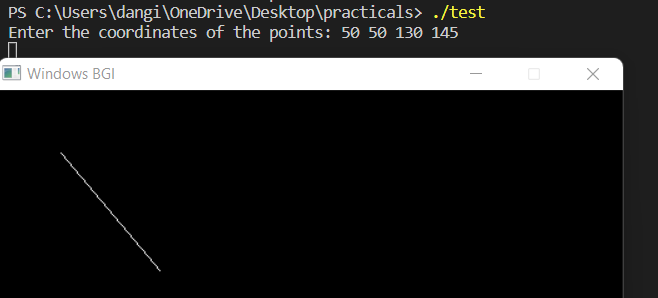
    delay(10000);

    closegraph();

    return 0;

}

**Output :**

****

**2 . Generating Line using Bresenham’s Approach**

// LINE USING BRESENHAM

/\* Assumptions :

1) Line is drawn from left to right.

2) x1 < x2 and y1 < y2

3) Slope of the line is between 0 and 1.

We draw a line from lower left to upper

right.

\*/

#include <bits/stdc++.h>

#include <graphics.h>

using namespace std;

// function for line generation

void bresenham(int x1, int y1, int x2, int y2)

{

    int m\_new = 2 \* (y2 - y1);

    int slope\_error\_new = m\_new - (x2 - x1);

    for (int x = x1, y = y1; x <= x2; x++)

    {

        putpixel(x, y, WHITE);

        cout << x << " " << y << endl;

        slope\_error\_new += m\_new;

        if (slope\_error\_new >= 0)

        {

            y++;

            slope\_error\_new -= 2 \* (x2 - x1);

        }

        delay(10);

    }

}

int main()

{

    initwindow(500, 500);

    int x1, y1, x2, y2;

    cout << "Enter the coordinates of the points: ";

    cin >> x1 >> y1 >> x2 >> y2;

    bresenham(x1, y1, x2, y2);

    delay(100);

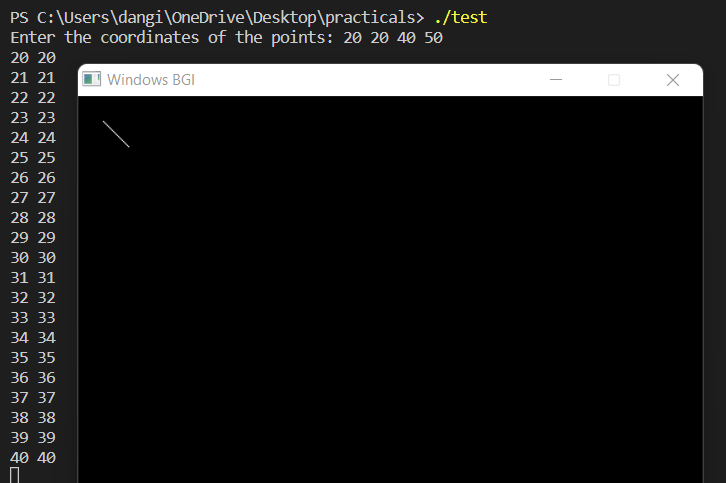
    getch();

    closegraph();

    return 0;

}

**Output :**



**3. Generating circle using bresenham’s approach**

#include <bits/stdc++.h>

using namespace std;

#include <graphics.h>

void draw(int x, int y)

{

    putpixel(x + 200, y + 200, WHITE);

    delay(1);

    putpixel(x + 200, -y + 200, WHITE);

    delay(1);

    putpixel(-x + 200, -y + 200, WHITE);

    delay(1);

    putpixel(-x + 200, y + 200, WHITE);

    delay(1);

    putpixel(y + 200, x + 200, WHITE);

    delay(1);

    putpixel(y + 200, -x + 200, WHITE);

    delay(1);

    putpixel(-y + 200, x + 200, WHITE);

    delay(1);

    putpixel(-y + 200, -x + 200, WHITE);

}

void circle1(int a, int b, int c)

{

    float p = 3 - 2 \* c;

    int x = 0, y = c;

    cout << x << ", " << y << "\n";

    putpixel(x + a, y + b, WHITE);

    while (x <= y)

    {

        if (p < 0)

        {

            p = p + 4 \* x + 6;

            x = x + 1;

            draw(x, y);

        }

        else

        {

            p = p + 4 \* (x - y) + 10;

            x = x + 1;

            y = y - 1;

            draw(x, y);

        }

    }

}

int main()

{

    initwindow(720, 720);

    int a = 150, b = 150, c = 100;

    circle1(a, b, c);

    getch();

    closegraph();

    return 0;

}

**Output :**



**4. Generating circle using Mid point approach**

using namespace std;

#include <graphics.h>

void draw(int x, int y)

{

    putpixel(x + 200, y + 200, WHITE);

    delay(1);

    putpixel(x + 200, -y + 200, WHITE);

    delay(1);

    putpixel(-x + 200, -y + 200, WHITE);

    delay(1);

    putpixel(-x + 200, y + 200, WHITE);

    delay(1);

    putpixel(y + 200, x + 200, WHITE);

    delay(1);

    putpixel(y + 200, -x + 200, WHITE);

    delay(1);

    putpixel(-y + 200, x + 200, WHITE);

    delay(1);

    putpixel(-y + 200, -x + 200, WHITE);

}

void circle1(int a, int b, int c)

{

    float p = 3 - 2 \* c;

    int x = 0, y = c;

    cout << x << ", " << y << "\n";

    putpixel(x + a, y + b, WHITE);

    while (x <= y)

    {

        if (p < 0)

        {

            p = p + 4 \* x + 6;

            x = x + 1;

            draw(x, y);

        }

        else

        {

            p = p + 4 \* (x - y) + 10;

            x = x + 1;

            y = y - 1;

            draw(x, y);

        }

    }

}

int main()

{

    initwindow(720, 720);

    int a = 150, b = 150, c = 100;

    circle1(a, b, c);

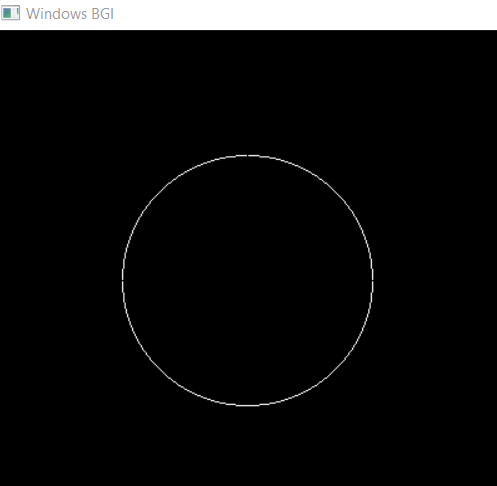
    getch();

    closegraph();

    return 0;

}

**Output :**



**5.**  **Generating ellipse using mid point algorithm**

#include <stdio.h>

#include <conio.h>

#include <graphics.h>

#include <math.h>

void disp();

float x, y;

int xc, yc;

int main()

{

    float p1, p2;

    initwindow(720, 720);

    int a, b;

    printf("\*\*\* Ellipse Generating Algorithm \*\*\*\n");

    printf("Enter the value of Xc\t");

    scanf("%d", &xc);

    printf("Enter the value of yc\t");

    scanf("%d", &yc);

    printf("Enter X axis length\t");

    scanf("%d", &a);

    printf("Enter Y axis length\t");

    scanf("%d", &b);

    x = 0;

    y = b;

    disp();

    p1 = (b \* b) - (a \* a \* b) + (a \* a) / 4;

    while ((2.0 \* b \* b \* x) <= (2.0 \* a \* a \* y))

    {

        x++;

        if (p1 <= 0)

            p1 = p1 + (2.0 \* b \* b \* x) + (b \* b);

        else

        {

            y--;

            p1 = p1 + (2.0 \* b \* b \* x) + (b \* b) - (2.0 \* a \* a \* y);

        }

        disp();

        x = -x;

        disp();

        x = -x;

        delay(50);

    }

    x = a;

    y = 0;

    disp();

    p2 = (a \* a) + 2.0 \* (b \* b \* a) + (b \* b) / 4;

    while ((2.0 \* b \* b \* x) > (2.0 \* a \* a \* y))

    {

        y++;

        if (p2 > 0)

            p2 = p2 + (a \* a) - (2.0 \* a \* a \* y);

        else

        {

            x--;

            p2 = p2 + (2.0 \* b \* b \* x) - (2.0 \* a \* a \* y) + (a \* a);

        }

        disp();

        y = -y;

        disp();

        y = -y;

        delay(50);

    }

    getch();

    closegraph();

    return 0;

}

void disp()

{

    putpixel(xc + x, yc + y, 7);

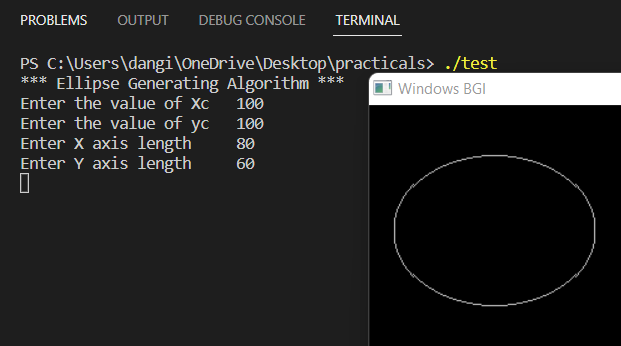
    putpixel(xc - x, yc + y, 7);

    putpixel(xc + x, yc - y, 7);

    putpixel(xc + x, yc - y, 7);

}

**Output :**

****

**6. Generating hyperbola using Mid point algorithm**

#include <bits/stdc++.h>

using namespace std;

#include <graphics.h>

// // Driver program

void draw(int x, int y)

{

    putpixel(x + 200, y + 200, GREEN);

    delay(1);

    putpixel(x + 200, -y + 200, GREEN);

    delay(1);

    putpixel(-x + 200, -y + 200, GREEN);

    delay(1);

    putpixel(-x + 200, y + 200, GREEN);

    delay(1);

}

void hyp(int a, int b)

{

    double p = (float)(1 / 4 + a) \* (b \* b) - (float)1 \* (a \* a);

    cout << p << "\n";

    int x = a, y = 0;

    putpixel(x + 200, y + 200, GREEN);

    while (y < (b \* b) / (sqrt(a \* a - b \* b)))

    {

        if (p > 0)

        {

            p = p - (2 \* y + 3) \* (a \* a);

            y = y + 1;

            draw(x, y);

        }

        else

        {

            p = p + (2 \* (x + 1) \* (b \* b)) - (2 \* y + 3) \* (a \* a);

            y = y + 1;

            x = x + 1;

            draw(x, y);

        }

        cout << p << "\n";

    }

    p = (x + 1) \* (x + 1) \* b \* b - (y + 1 / 2) \* (y + 1 / 2) \* a \* a - a \* a \* b \* b;

    while (y < 300)

    {

        cout << "aa";

        if (p > 0)

        {

            p = p + (2 \* x + 3) \* b \* b - a \* a \* (2 \* (y + 1));

            y = y + 1;

            x = x + 1;

            draw(x, y);

        }

        else

        {

            p = p + (2 \* x + 3) \* b \* b;

            x = x + 1;

            draw(x, y);

        }

    }

}

int main()

{

    initwindow(1000, 1000);

    int a = 40, b = 20;

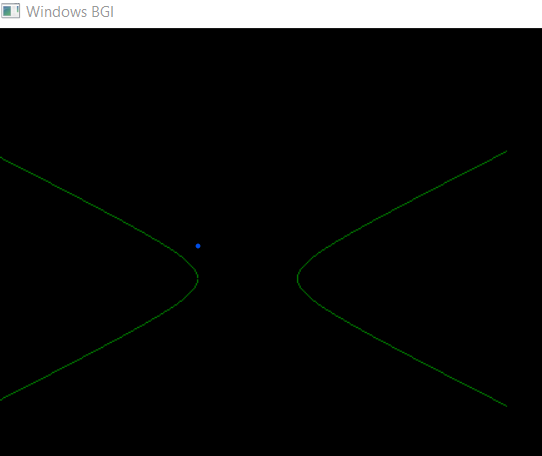
    hyp(a, b);

    getch();

    return 0;

}

**Output :**

****

**7. Implement Line Clipping approach using Cohen Sutherland**

#include <iostream>

#include <GL/glut.h>

using namespace std;

// Defining region codes

const int INSIDE = 0; // 0000

const int LEFT = 1;   // 0001

const int RIGHT = 2;  // 0010

const int BOTTOM = 4; // 0100

const int TOP = 8;    // 1000

const int x\_max = 700;

const int y\_max = 500;

const int x\_min = 100;

const int y\_min = 100;

void drawLine(int x1, int y1, int x2, int y2)

{

    glBegin(GL\_LINES);

    glVertex2i(x1, y1);

    glVertex2i(x2, y2);

    glEnd();

    glFlush();

}

int computeCode(double x, double y)

{ // initialized as being inside

    int code = INSIDE;

    if (x < x\_min) // to the left of rectangle

        code |= LEFT;

    else if (x > x\_max) // to the right of rectangle

        code |= RIGHT;

    if (y < y\_min) // below the rectangle

        code |= BOTTOM;

    else if (y > y\_max) // above the rectangle

        code |= TOP;

    return code;

}

void lc\_cs()

{

    int x1, y1, x2, y2;

    cout << "Enter the first point: ";

    cin >> x1 >> y1;

    cout << "Enter the second point: ";

    cin >> x2 >> y2;

    // Compute region codes for P1, P2

    int code1 = computeCode(x1, y1);

    int code2 = computeCode(x2, y2);

    // Initialize line as outside the rectangular window

    bool accept = false;

    while (true)

    {

        if ((code1 == 0) && (code2 == 0))

        {

            accept = true;

            break;

        }

        else if (code1 & code2)

        {

            break;

        }

        else

        {

            int code\_out;

            double x, y;

            if (code1 != 0)

                code\_out = code1;

            else

                code\_out = code2;

            if (code\_out & TOP)

            {

                x = x1 + (x2 - x1) \* (y\_max - y1) / (y2 - y1);

                y = y\_max;

            }

            else if (code\_out & BOTTOM)

            {

                x = x1 + (x2 - x1) \* (y\_min - y1) / (y2 - y1);

                y = y\_min;

            }

            else if (code\_out & RIGHT)

            {

                y = y1 + (y2 - y1) \* (x\_max - x1) / (x2 - x1);

                x = x\_max;

            }

            else if (code\_out & LEFT)

            {

                y = y1 + (y2 - y1) \* (x\_min - x1) / (x2 - x1);

                x = x\_min;

            }

            if (code\_out == code1)

            {

                x1 = x;

                y1 = y;

                code1 = computeCode(x1, y1);

            }

            else

            {

                x2 = x;

                y2 = y;

                code2 = computeCode(x2, y2);

            }

        }

    }

    if (accept)

    {

        cout << "Line accepted from " << x1 << ", "

             << y1 << " to " << x2 << ", " << y2 << endl;

        drawLine(x1, y1, x2, y2);

        // Here the user can add code to display the rectangle

        // along with the accepted (portion of) lines

    }

    else

        cout << "Line rejected" << endl;

}

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

{

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowPosition(100, 200);

    glutInitWindowSize(800, 600);

    glutCreateWindow("Cohen Sutherland Line Clipping Algorithm");

    glClearColor(1, 1, 1, 1);

    glColor3f(0, 0, 0);

    glClear(GL\_COLOR\_BUFFER\_BIT);

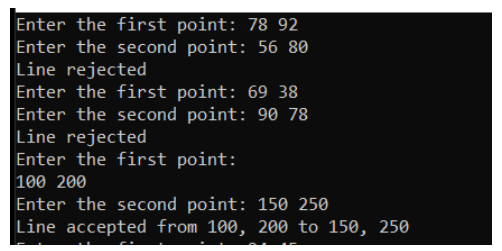
    gluOrtho2D(0, 800, 0, 600);

    glutDisplayFunc(lc\_cs);

    glutMainLoop();

}

**Output :**

****

**8. Implement Line Clipping approach using Liang Barsky / Cyrus Beck**

#include <iostream>

#include <GL/glut.h>

using namespace std;

// Defining region codes

const int INSIDE = 0; // 0000

const int LEFT = 1;   // 0001

const int RIGHT = 2;  // 0010

const int BOTTOM = 4; // 0100

const int TOP = 8;    // 1000

const int x\_max = 700;

const int y\_max = 500;

const int x\_min = 100;

const int y\_min = 100;

void drawLine(int x1, int y1, int x2, int y2)

{

    glBegin(GL\_LINES);

    glVertex2i(x1, y1);

    glVertex2i(x2, y2);

    glEnd();

    glFlush();

}

int computeCode(double x, double y)

{

    // initialized as being inside

    int code = INSIDE;

    if (x < x\_min) // to the left of rectangle

        code |= LEFT;

    else if (x > x\_max) // to the right of rectangle

        code |= RIGHT;

    if (y < y\_min) // below the rectangle

        code |= BOTTOM;

    else if (y > y\_max) // above the rectangle

        code |= TOP;

    return code;

}

void lc\_cs()

{

    int x1, y1, x2, y2;

    cout << "Enter the first point: ";

    cin >> x1 >> y1;

    cout << "Enter the second point: ";

    cin >> x2 >> y2;

    // Compute region codes for P1, P2

    int code1 = computeCode(x1, y1);

    int code2 = computeCode(x2, y2);

    // Initialize line as outside the rectangular window

    bool accept = false;

    while (true)

    {

        if ((code1 == 0) && (code2 == 0))

        {

            accept = true;

            break;

        }

        else if (code1 & code2)

        {

            break;

        }

        else

        {

            int code\_out;

            double x, y;

            if (code1 != 0)

                code\_out = code1;

            else

                code\_out = code2;

            if (code\_out & TOP)

            {

                x = x1 + (x2 - x1) \* (y\_max - y1) / (y2 - y1);

                y = y\_max;

            }

            else if (code\_out & BOTTOM)

            {

                x = x1 + (x2 - x1) \* (y\_min - y1) / (y2 - y1);

                y = y\_min;

            }

            else if (code\_out & RIGHT)

            {

                y = y1 + (y2 - y1) \* (x\_max - x1) / (x2 - x1);

                x = x\_max;

            }

            else if (code\_out & LEFT)

            {

                y = y1 + (y2 - y1) \* (x\_min - x1) / (x2 - x1);

                x = x\_min;

            }

            if (code\_out == code1)

            {

                x1 = x;

                y1 = y;

                code1 = computeCode(x1, y1);

            }

            else

            {

                x2 = x;

                y2 = y;

                code2 = computeCode(x2, y2);

            }

        }

    }

    if (accept)

    {

        cout << "Line accepted from " << x1 << ", "

             << y1 << " to " << x2 << ", " << y2 << endl;

        drawLine(x1, y1, x2, y2);

        // Here the user can add code to display the rectangle

        // along with the accepted (portion of) lines

    }

    else

        cout << "Line rejected" << endl;

}

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

{

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowPosition(100, 200);

    glutInitWindowSize(800, 600);

    glutCreateWindow("Cohen Sutherland Line Clipping Algorithm");

    glClearColor(1, 1, 1, 1);

    glColor3f(0, 0, 0);

    glClear(GL\_COLOR\_BUFFER\_BIT);

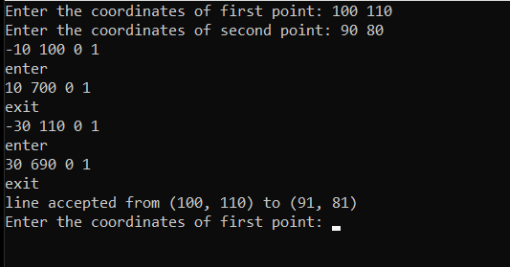
    gluOrtho2D(0, 800, 0, 600);

    glutDisplayFunc(lc\_cs);

    glutMainLoop();

}

**Output :**



**9. Implement Line Clipping approach using Mid-Point Subdivision**

#include <iostream>

#include <GL/glut.h>

using namespace std;

#define XWMIN 100

#define XWMAX 400

#define YWMIN 100

#define YWMAX 400

void drawLine(int x1, int y1, int x2, int y2)

{

    glBegin(GL\_LINES);

    glVertex2i(x1, y1);

    glVertex2i(x2, y2);

    glEnd();

    glFlush();

}

void drawWindow(int xmin, int ymin, int xmax, int ymax)

{

    glBegin(GL\_LINE\_LOOP);

    glVertex2i(xmin, ymin);

    glVertex2i(xmin, ymax);

    glVertex2i(xmax, ymax);

    glVertex2i(xmax, ymin);

    glEnd();

    glFlush();

}

int calcCode(int x, int y)

{

    int code = 0;

    if (x < XWMIN)

    {

        // left

        code |= 1;

    }

    else if (x > XWMAX)

    { // right

        code |= 2;

    }

    if (y < YWMIN)

    { // top

        code |= 4;

    }

    else if (y > YWMAX)

    { // bottom

        code |= 8;

    }

    return code;

}

void clipLine(int &xc1, int &yc1, int &xc2, int &yc2, int x1, int y1, int x2, int y2)

{

    int xc11, yc11, xc12, yc12, xc21, yc21, xc22, yc22;

    int code1 = calcCode(x1, y1), code2 = calcCode(x2, y2);

    if (x1 == (x1 + x2) / 2 && y1 == (y1 + y2) / 2)

    {

        xc1 = x1;

        xc2 = x2;

        yc1 = y1;

        yc2 = y2;

        return;

    }

    if ((code1 | code2) == 0)

    {

        // completely inside

        xc1 = x1;

        yc1 = y1;

        xc2 = x2;

        yc2 = y2;

        return;

    }

    else if ((code1 & code2) != 0)

    {

        // completely outside

        xc1 = -1;

        yc1 = -1;

        xc2 = -1;

        yc2 = -1;

        return;

    }

    // clipping candidate

    clipLine(xc11, yc11, xc21, yc21, x1, y1, (x1 + x2) / 2, (y1 + y2) / 2);

    clipLine(xc12, yc12, xc22, yc22, (x1 + x2) / 2, (y1 + y2) / 2, x2, y2);

    if (xc21 == xc12 && yc21 == yc12)

    {

        xc1 = xc11;

        yc1 = yc11;

        xc2 = xc22;

        yc2 = yc22;

    }

    else if (xc11 == -1 && xc21 == -1 && yc11 == -1 && yc21 == -1)

    { // first point

        xc1 = xc12;

        xc2 = xc22;

        yc1 = yc12;

        yc2 = yc22;

    }

    else

    {

        // second point invalid

        xc1 = xc11;

        xc2 = xc21;

        yc1 = yc11;

        yc2 = yc21;

    }

}

void mpsd()

{

    int x0, y0, x1, y1;

    cout << "Enter first point: ";

    cin >> x0 >> y0;

    cout << "Enter second point: ";

    cin >> x1 >> y1;

    glClear(GL\_COLOR\_BUFFER\_BIT);

    drawWindow(XWMIN, YWMIN, XWMAX, YWMAX);

    glColor3f(1, 0, 0);

    drawLine(x0, y0, x1, y1);

    int xc1, yc1, xc2, yc2;

    clipLine(xc1, yc1, xc2, yc2, x0, y0, x1, y1);

    glColor3f(0, 0, 1);

    if (xc1 != -1 && yc1 != -1 && xc2 != -1 && yc2 != -1)

        drawLine(xc1, yc1, xc2, yc2);

    cout << xc1 << " " << yc1 << " " << xc2 << " " << yc2 << endl;

}

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

{

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowPosition(100, 200);

    glutInitWindowSize(800, 600);

    glutCreateWindow("Mid point subdivision");

    glClearColor(1, 1, 1, 0);

    glColor3f(0, 0, 0);

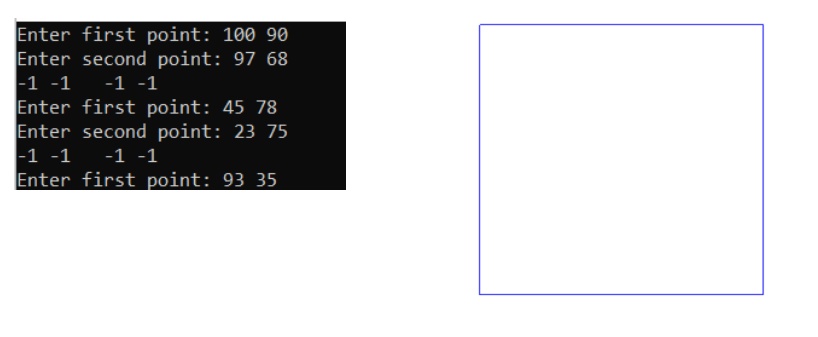
    gluOrtho2D(0, 800, 0, 600);

    glutDisplayFunc(mpsd);

    glutMainLoop();

}

**Output :**

****