NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COMPUTER GRAPHICS
(COCSE64)

PRACTICAL FILE

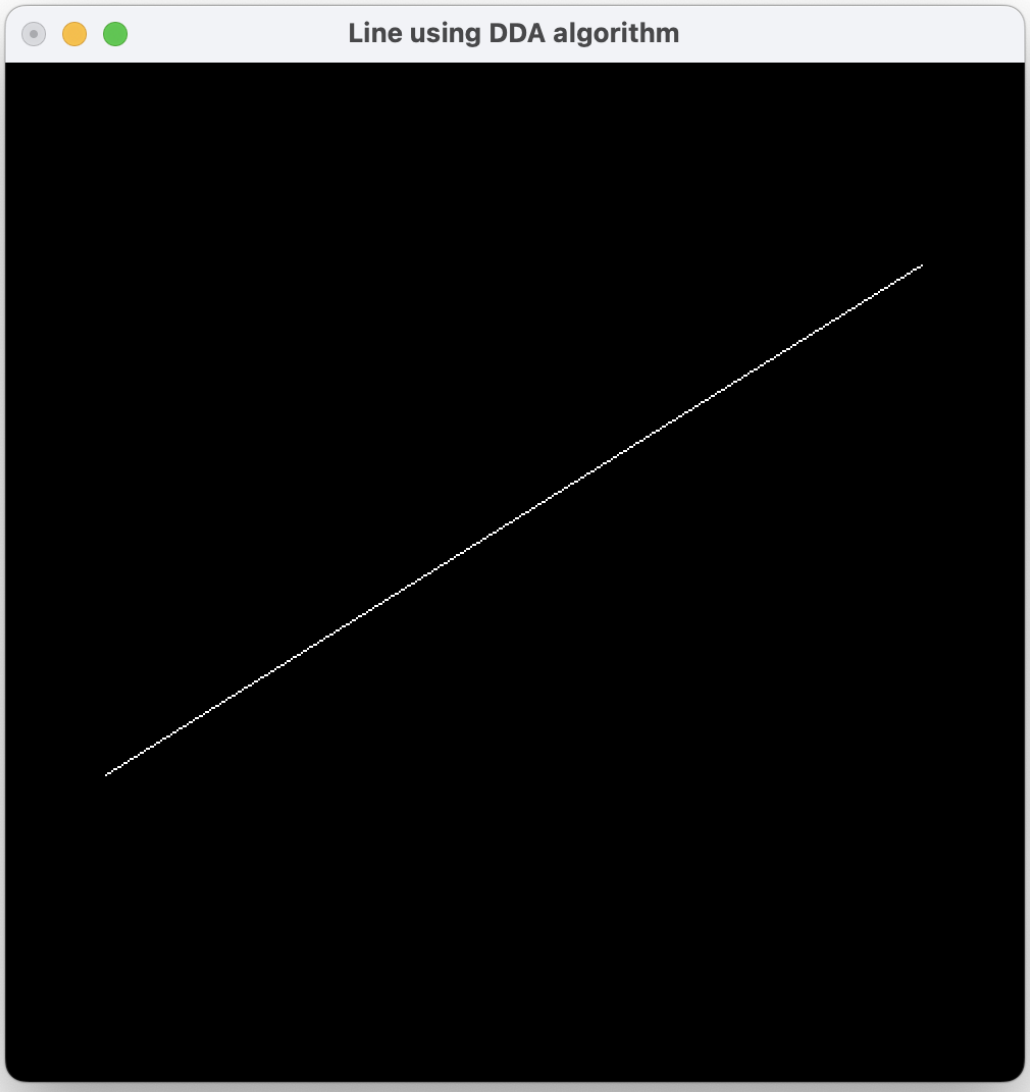
Tanvee Balhara
2019UCO1705
Computer Engineering
Batch - 2

INDEX

1) Scan conversion of line using DDA algorithm

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

void dda(int x0, int y0, int x1, int y1) {
    glBegin(GL_POINTS);
    glColor3f(1.0, 1.0, 1.0);

    int dx = x1 - x0,
        dy = y1 - y0;

    float steps = dx > dy ? dx : dy;
    float xinc = dx / steps,
          yinc = dy / steps;

    float x = x0, y = y0;
    glVertex2i(x, y);

    for (int i = 0; i < steps; i++) {
        x += xinc;
        y += yinc;
        glVertex2i(round(x), round(y));
    }

    glEnd();
}

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    dda(50, 150, 450, 400);
    glFlush();
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);

    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Line using DDA algorithm");
```

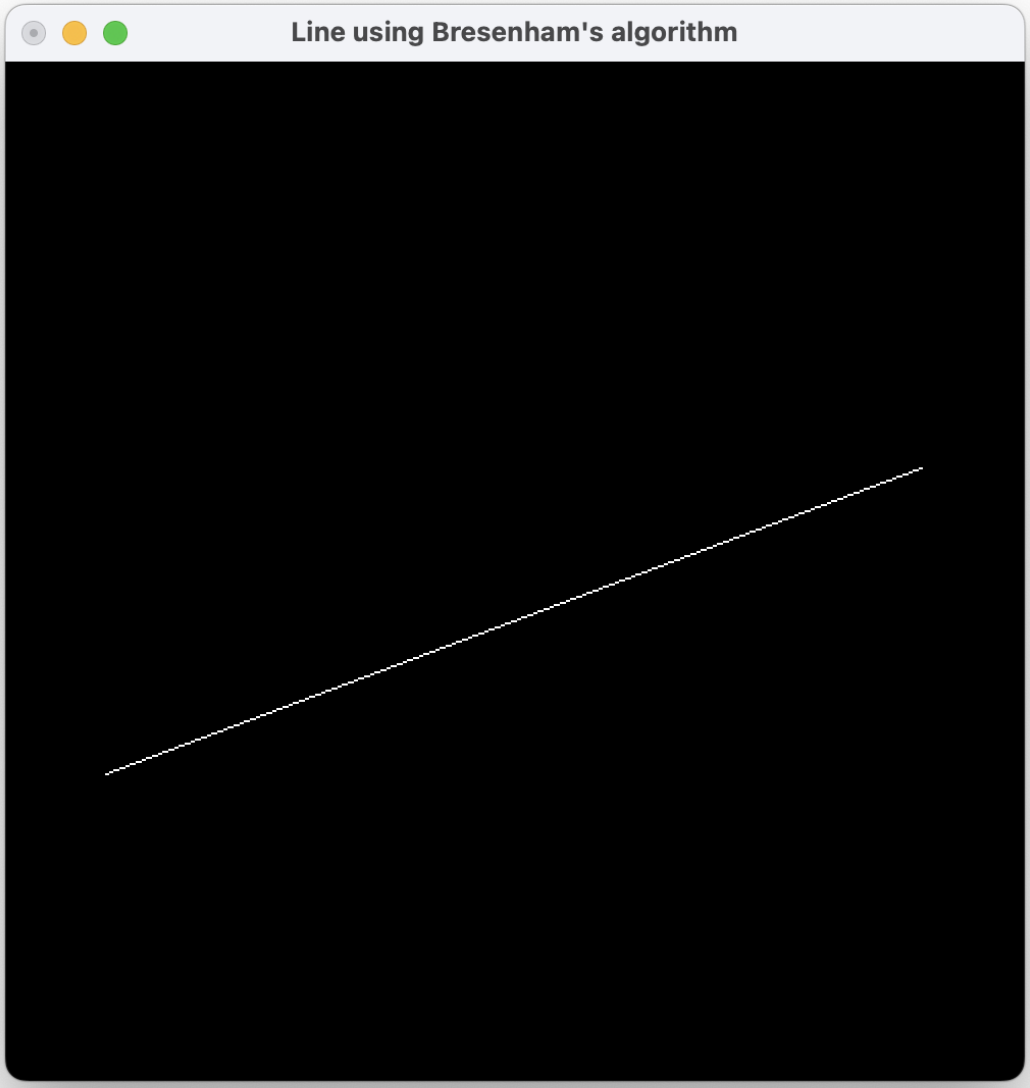
```
    gluOrtho2D(0,500,0,500);
    glutDisplayFunc(display);
    glutMainLoop();
}
```



2) Scan conversion of line using Bresenham's algorithm

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

void bresenham(int x0, int y0, int x1, int y1) {
    glBegin(GL_POINTS);
    glColor3f(1.0, 1.0, 1.0);

    int dx = x1 - x0, dy = y1 - y0;
    int cdx = x1 * y0 - x0 * y1;

    int x = x0, y = y0;
    int dk = 2 * dx * y - 2 * dy * (x + 1) - 2 * cdx + dx;
    glVertex2i(x, y);

    while (x != x1) {
        if (dk > 0) {
            dk = dk - 2 * dy;
        } else {
            dk = dk + 2 * (dx - dy);
            y = y + 1;
        }
        x = x + 1;
        glVertex2i(x, y);
    }

    glEnd();
}

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    bresenham(50, 150, 450, 300);
    glFlush();
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
```

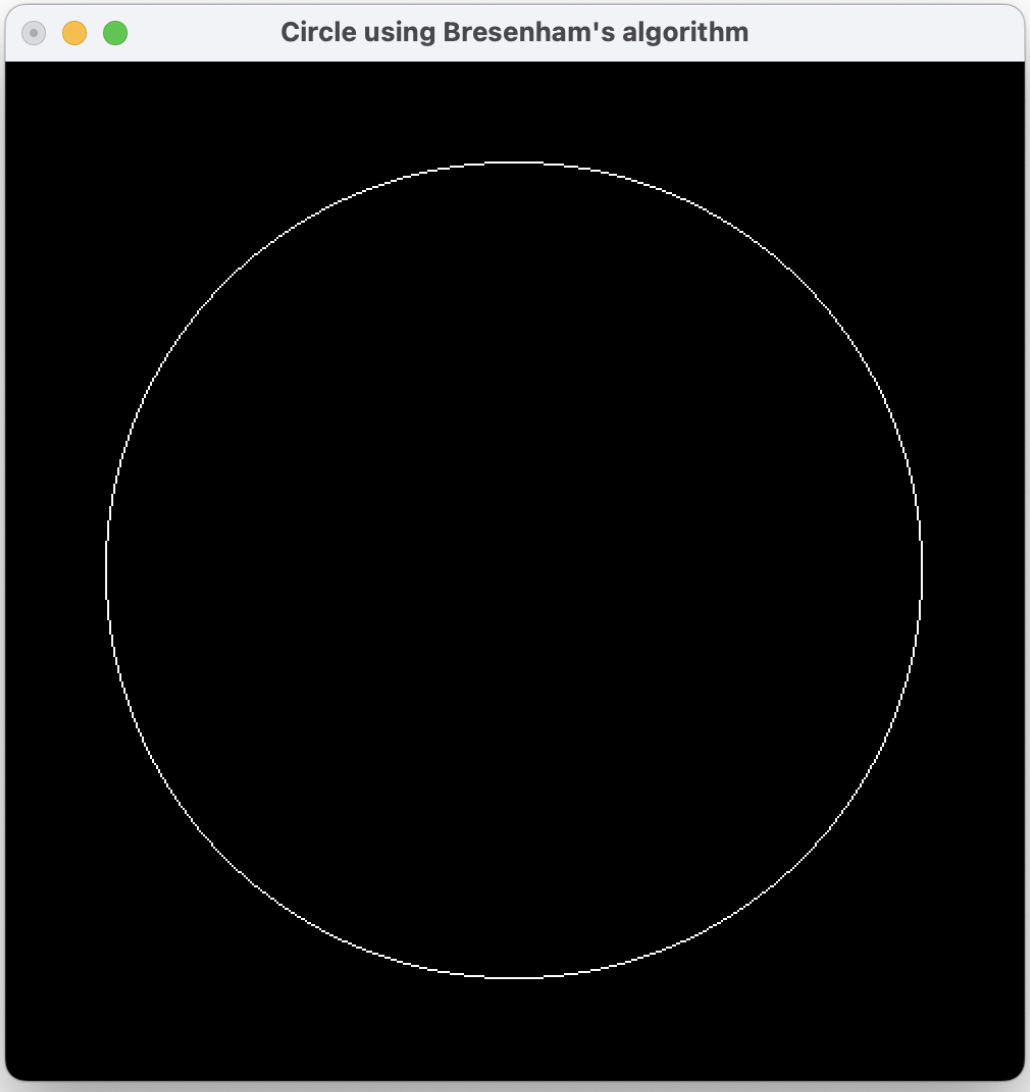
```
    glutCreateWindow("Line using Bresenham's algorithm");
    gluOrtho2D(0,500,0,500);
    glutDisplayFunc(display);
    glutMainLoop();
}
```



3) Scan conversion of circle using Bresenham's algorithm

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

void drawSymmetricalPoints(int x, int y) {
    glVertex2i( x,  y); glVertex2i( y,  x);
    glVertex2i(-x,  y); glVertex2i( y, -x);
    glVertex2i( x, -y); glVertex2i(-y,  x);
    glVertex2i(-x, -y); glVertex2i(-y, -x);
}

void bresenham(int r) {
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_POINTS);

    int x = 0, y = r;
    int dk = 3 - 2 * r;
    drawSymmetricalPoints(x, y);

    while (x < y) {
        if (dk > 0) {
            dk = dk + 4 * (x - y) + 10;
            y = y - 1;
        } else {
            dk = dk + 4 * x + 6;
        }
        x = x + 1;
        drawSymmetricalPoints(x, y);
    }

    glEnd();
}

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    bresenham(200);
    glFlush();
}
int main(int argc, char **argv) {
    glutInit(&argc, argv);
```

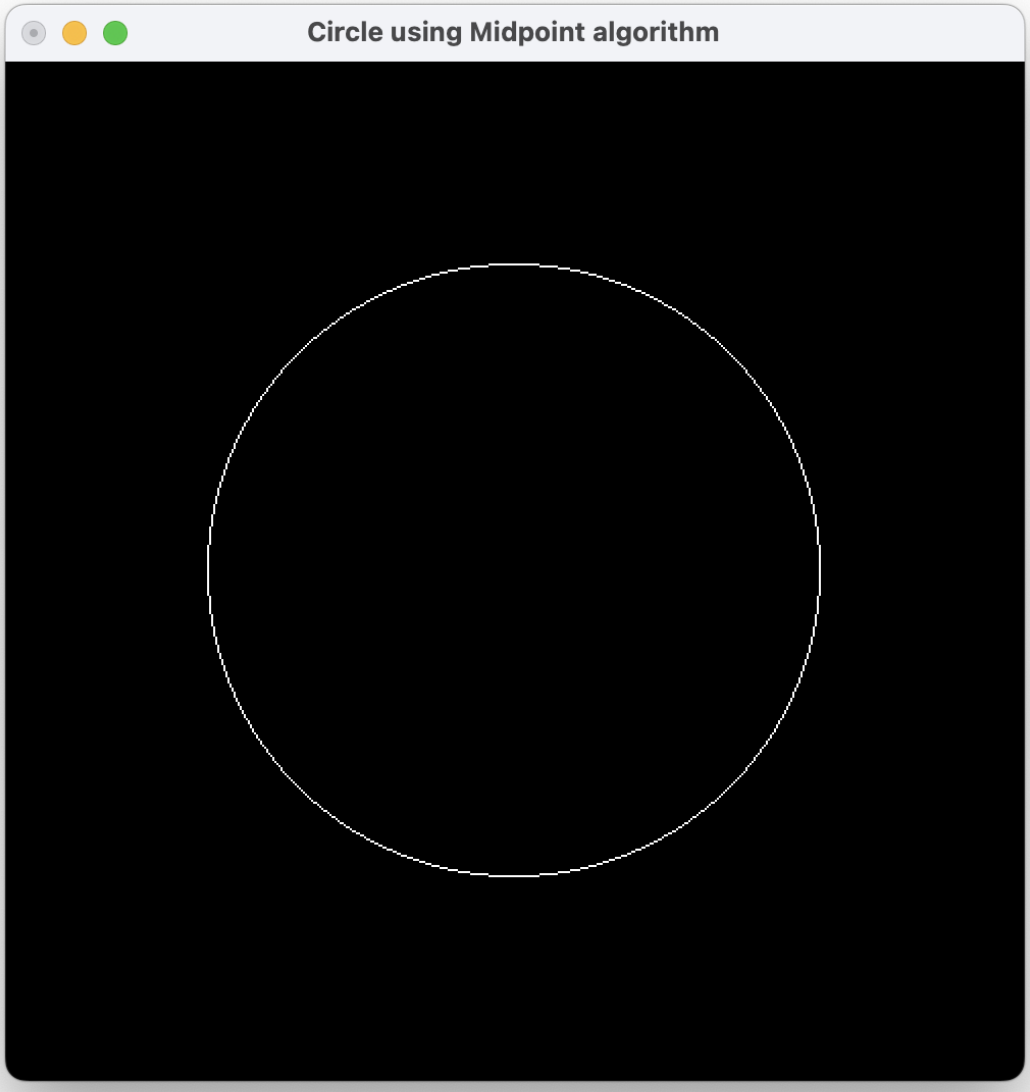
```
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Circle using Bresenham's algorithm");
    glLoadIdentity();
    gluOrtho2D(-250,250,-250,250);
    glutDisplayFunc(display);
    glutMainLoop();
}
```



4) Scan conversion of circle using Midpoint algorithm

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

void drawSymmetricalPoints(int x, int y) {
    glVertex2i( x,  y); glVertex2i( y,  x);
    glVertex2i(-x,  y); glVertex2i( y, -x);
    glVertex2i( x, -y); glVertex2i(-y,  x);
    glVertex2i(-x, -y); glVertex2i(-y, -x);
}

void midpoint(int r) {
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_POINTS);

    int x = 0, y = r;
    float dk = 5.0/4 - r;
    drawSymmetricalPoints(x, y);

    while (x < y) {
        if (dk > 0) {
            dk = dk + 5 + 2 * (x - y);
            y = y - 1;
        } else {
            dk = dk + 3 + 2 * x;
        }
        x = x + 1;
        drawSymmetricalPoints(x, y);
    }
    glEnd();
}

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    midpoint(150);
    glFlush();
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
```

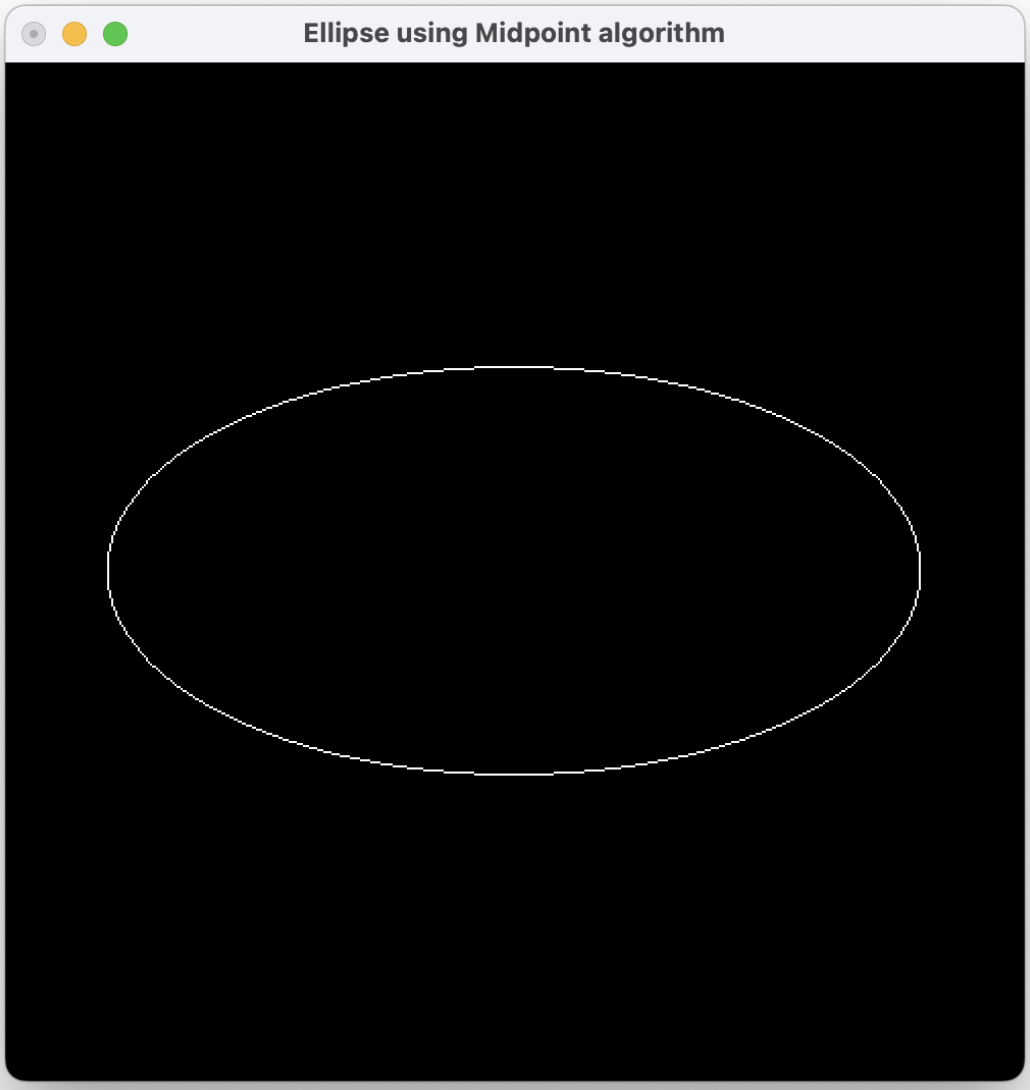
```
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Circle using Midpoint algorithm");
    glLoadIdentity();
    gluOrtho2D(-250,250,-250,250);
    glutDisplayFunc(display);
    glutMainLoop();
}
```



5) Scan conversion of ellipse using Midpoint algorithm

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

void drawSymmetricalPoints(int x, int y) {
    glVertex2i(x, y);
    glVertex2i(-x, y);
    glVertex2i(x, -y);
    glVertex2i(-x, -y);
}

void midpoint(int a, int b) {
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_POINTS);

    int x = 0, y = b;
    float dk = b * b - a * a * b + a * a / 4.0;
    drawSymmetricalPoints(x, y);

    while (x * b * b <= y * a * a) {
        if (dk > 0) {
            dk =
            dk + b * b * (2 * x + 3) + 2 * a * a * (1 - y);
            y = y - 1;
        } else {
            dk = dk + b * b * (2 * x + 3);
        }
        x = x + 1;
        drawSymmetricalPoints(x, y);
    }

    dk = a * b * b + b * b / 4.0 + a * a;
    while (y > 0) {
        if (dk > 0) {
            dk = dk + a * a * (3 - 2 * y);
        } else {
            dk =
            dk + a * a * (3 - 2 * y) + 2 * b * b * (x + 1);
            x = x + 1;
        }
```

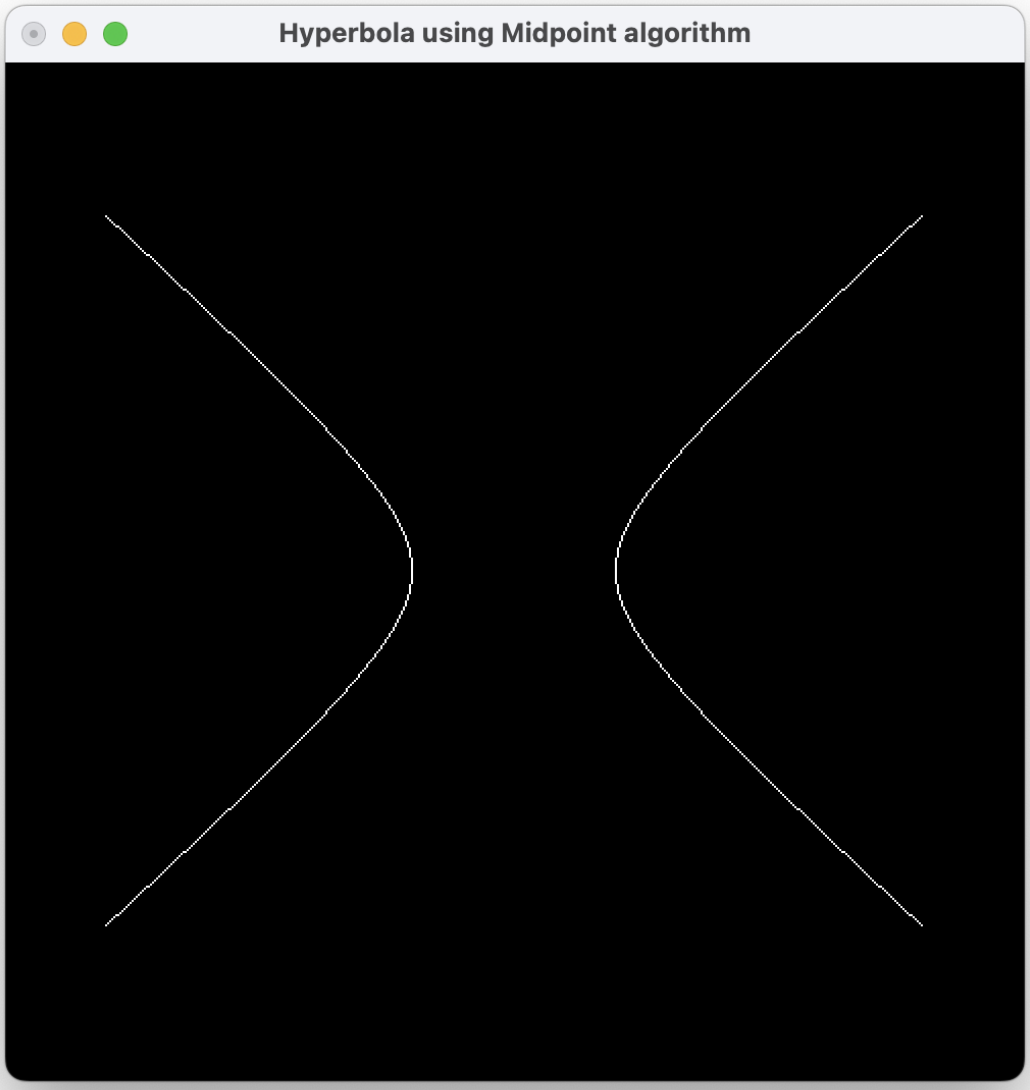
```
        y = y - 1;
        drawSymmetricalPoints(x, y);
    }

    glEnd();
}

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    midpoint(200, 100);
    glFlush();
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Ellipse using Midpoint algorithm");
    glLoadIdentity();
    gluOrtho2D(-250,250,-250,250);
    glutDisplayFunc(display);
    glutMainLoop();
}
```


Ellipse using Midpoint algorithm

6) Scan conversion of hyperbola using Midpoint algorithm

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

void drawSymmetricalPoints(int x, int y) {
    glVertex2i(x, y);
    glVertex2i(-x, y);
    glVertex2i(x, -y);
    glVertex2i(-x, -y);
}

void midpoint(int a, int b) {
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_POINTS);

    float bsq = b * b, asq = a * a;

    int x = a, y = 0;
    float dk = bsq / 4 + a * bsq - asq;
    drawSymmetricalPoints(x, y);

    while (asq * y < bsq * x && x < 200) {
        if (dk > 0) {
            dk += - 3 * asq - 2 * asq * y;
        } else {
            dk += 2 * bsq - 3 * asq + 2 * (bsq*x - asq*y);
            x = x + 1;
        }
        y = y + 1;
        drawSymmetricalPoints(x, y);
    }

    if (a > b) {
        dk = 2 * a * bsq + bsq - asq / 4;
        while (x < 200) {
            if (dk > 0) {
                dk += 3*bsq - 2*asq + 2 * (bsq*x - asq*y);
                y = y + 1;
            } else {
                dk += 3 * bsq + 2 * bsq * x;
```

```
            }
            x = x + 1;
            drawSymmetricalPoints(x, y);
        }
    }

    glEnd();
}

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    midpoint(50, 45);
    glFlush();
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Hyperbola using Midpoint algorithm");
    glLoadIdentity();
    gluOrtho2D(-250,250,-250,250);
    glutDisplayFunc(display);
    glutMainLoop();
}
```

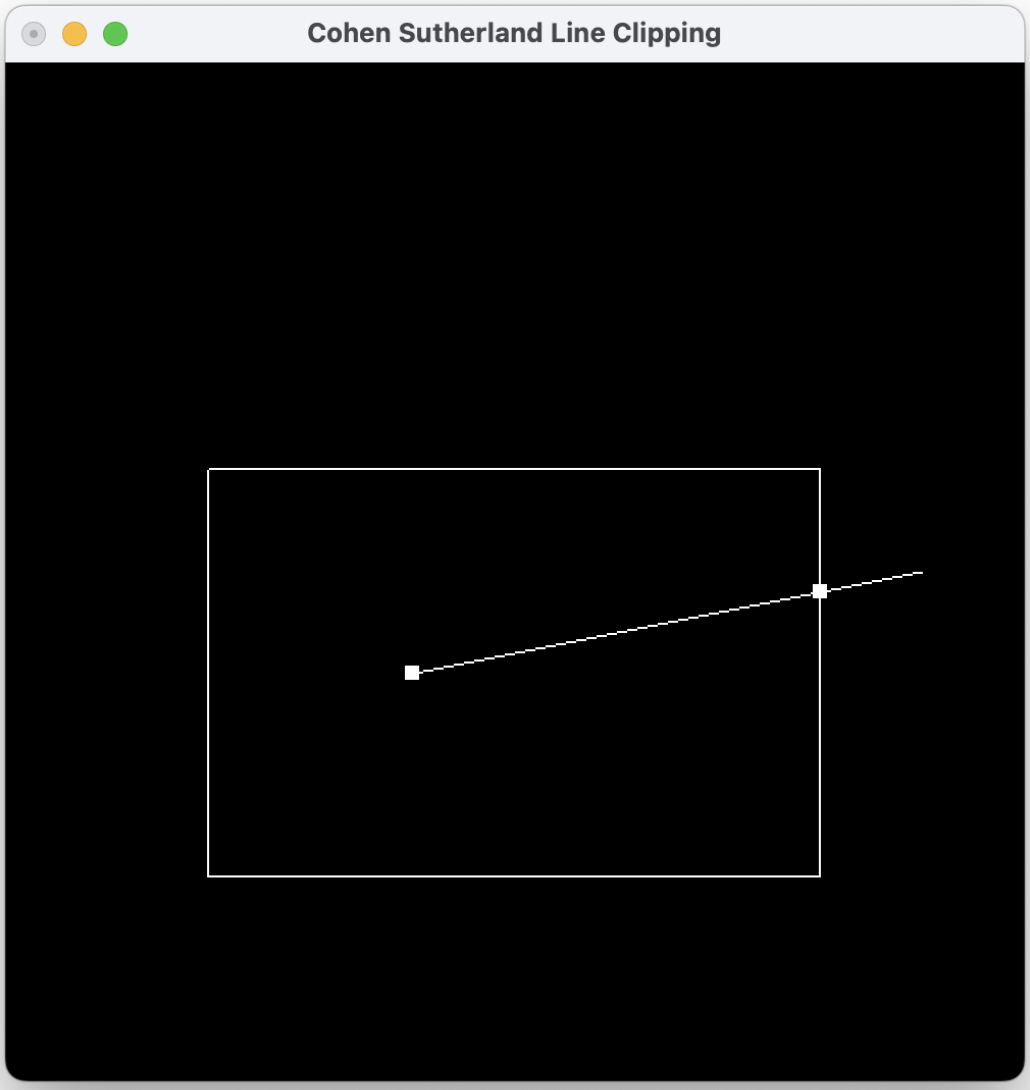

Hyperbola using Midpoint algorithm

7) Line clipping using Cohen-Sutherland algorithm

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

#define LEFT_EDGE   0x1
#define RIGHT_EDGE  0x2
#define BOTTOM_EDGE 0x4
#define TOP_EDGE    0x8

#define INSIDE(a)   (!a)
#define REJECT(a,b) (a&b)
#define ACCEPT(a,b) (!(a|b))

int xwmin = 100, xwmax = 400, ywmin = 100, ywmax = 300;

void line(float x1, float y1, float x2, float y2) {
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINES);
    glVertex2d(x1, y1);
    glVertex2d(x2, y2);
    glEnd();
}

void point(float x, float y) {
    glColor3f(1.0, 1.0, 1.0);
    glPointSize(7.0);
    glBegin(GL_POINTS);
    glVertex2d(x, y);
    glEnd();
}

void swap(float *a, float *b) {
    float temp = *a;
    *a = *b;
    *b = temp;
}

void clippingWindow() {
    glColor3f(1.0, 1.0, 1.0);
    glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
```

```
    glBegin(GL_POLYGON);
    glVertex2i(xwmin, ywmin);
    glVertex2i(xwmin, ywmax);
    glVertex2i(xwmax, ywmax);
    glVertex2i(xwmax, ywmin);
    glEnd();
}

unsigned char encode (int x, int y) {
    unsigned char code = 0x00;
    if (x < xwmin) code |= LEFT_EDGE;
    if (x > xwmax) code |= RIGHT_EDGE;
    if (y < ywmin) code |= BOTTOM_EDGE;
    if (y > ywmax) code |= TOP_EDGE;
    return code;
}

void clipLine(float x1, float y1, float x2, float y2) {
    unsigned char code1, code2;
    int done = 0, draw = 0;

    while (!done) {
        code1 = encode(x1, y1);
        code2 = encode(x2, y2);
        if (ACCEPT(code1, code2)) {
            done = draw = 1;
        } else if (REJECT(code1, code2)) {
            done = 1;
        } else {
            if (INSIDE(code1)) {
                swap(&x1, &x2);
                swap(&y1, &y2);
                char t = code1;
                code1 = code2;
                code2 = t;
            }
            float m = 0;
            if (x1 != x2) {
                m = (y2 - y1) * 1.0 / (x2 - x1);
            }
```

```
            if (code1 & LEFT_EDGE) {
                y1 += (xwmin - x1) * m;
                x1 = xwmin;
            } else if (code1 & RIGHT_EDGE) {
                y1 += (xwmax - x1) * m;
                x1 = xwmax;
            } else if (code1 & BOTTOM_EDGE) {
                if (x1 != x2)
                    x1 += (ywmin - y1) / m;
                y1 = ywmin;
            } else if (code1 & TOP_EDGE) {
                if (x1 != x2)
                    x1 += (ywmax - y1) / m;
                y1 = ywmax;
            }
        }
    }
    if (draw) {
        point(x1, y1);
        point(x2, y2);
    }
}

void display(void) {
    glClearColor(1.0, 1.0, 1.0, 0.0);
    clippingWindow();
    line(200, 200, 450, 250);
    clipLine(200, 200, 450, 250);
    glFlush();
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Cohen Sutherland Line Clipping");
    gluOrtho2D(0,500,0,500);
    glutDisplayFunc(display);
    glutMainLoop();
}
```

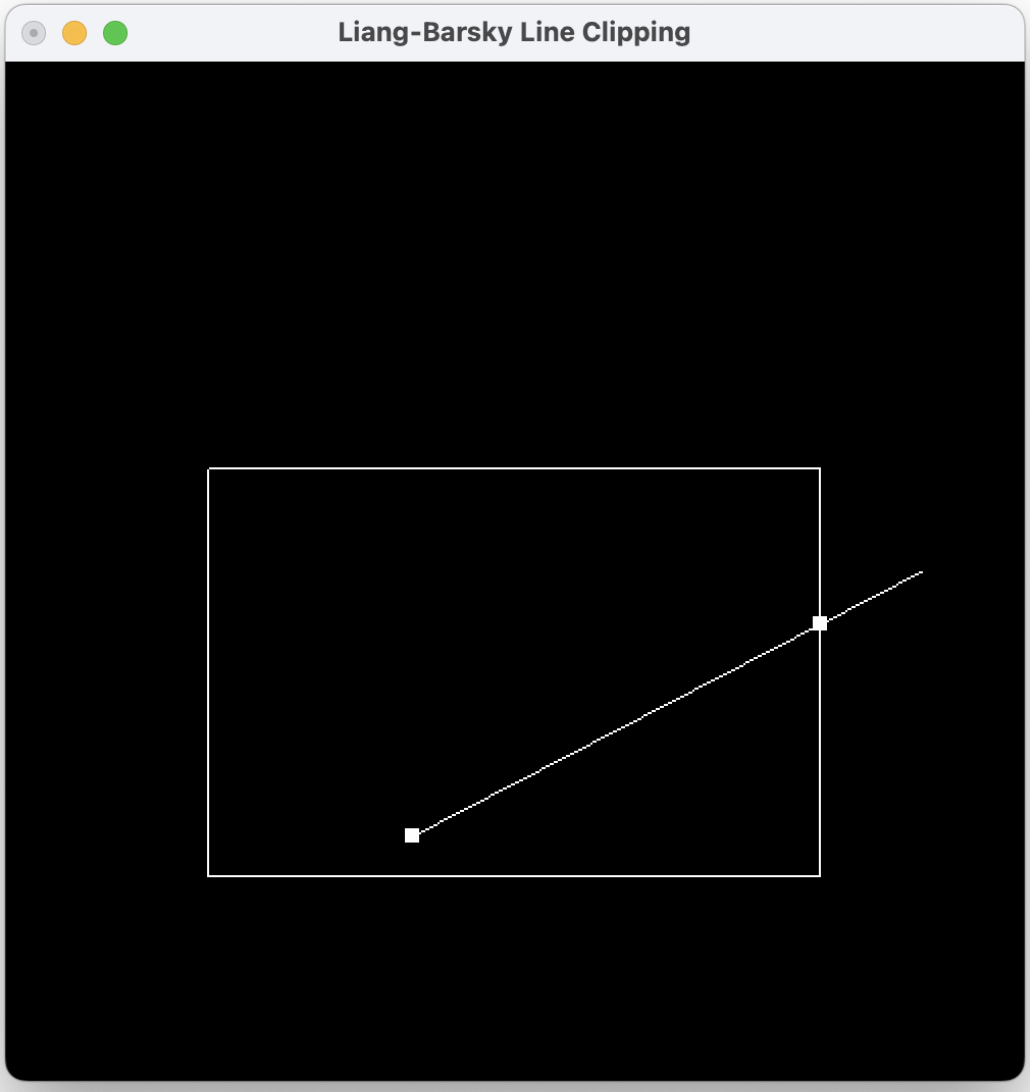

Cohen Sutherland Line Clipping

8) Line clipping using Liang-Barsky algorithm

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

int xwmin = 100, xwmax = 400, ywmin = 100, ywmax = 300;

int clipTest(float p, float q, float *u1, float*u2) {
    if (p == 0) {
        if (q < 0) return 0;
    } else if (p < 0) {
        float r = q / p;
        if (r > *u2) return 0;
        else if (r > *u1) *u1 = r;
    } else {
        float r = q / p;
        if (r < *u1) return 0;
        else if (r < *u2) *u2 = r;
    }
    return 1;
}

void line(float x1, float y1, float x2, float y2) {
    glColor3f(1.0, 1.0, 1.0);
    glLineWidth(1.0);
    glBegin(GL_LINES);
    glVertex2d(x1, y1);
    glVertex2d(x2, y2);
    glEnd();
}

void point(float x, float y) {
    glColor3f(1.0, 1.0, 1.0);
    glPointSize(7.0);
    glBegin(GL_POINTS);
    glVertex2d(x, y);
    glEnd();
}

void clippingWindow() {
    glColor3f(1.0, 1.0, 1.0);
```

```
    glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
    glBegin(GL_POLYGON);
    glVertex2i(xwmin, ywmin);
    glVertex2i(xwmin, ywmax);
    glVertex2i(xwmax, ywmax);
    glVertex2i(xwmax, ywmin);
    glEnd();
}

void clipLine(float x1, float y1, float x2, float y2) {
    float u1 = 0.0, u2 = 1.0, dx = x2 - x1, dy = y2 - y1;
    if (
        clipTest(-dx, x1 - xwmin, &u1, &u2) &&
        clipTest( dx, xwmax - x1, &u1, &u2) &&
        clipTest(-dy, y1 - ywmin, &u1, &u2) &&
        clipTest( dy, ywmax - y1, &u1, &u2)
    )  {
        if (u2 < 1) {
            x2 = x1 + u2 * dx;
            y2 = y1 + u2 * dy;
        }
        if (u1 > 0) {
            x1 += u1 * dx;
            y1 += u1 * dy;
        }
        point(x1, y1);
        point(x2, y2);
    }
}

void display(void) {
    glClearColor(1.0, 1.0, 1.0, 0.0);
    clippingWindow();
    line(200, 120, 450, 250);
    clipLine(200, 120, 450, 250);
    glFlush();
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitWindowSize(500, 500);
```

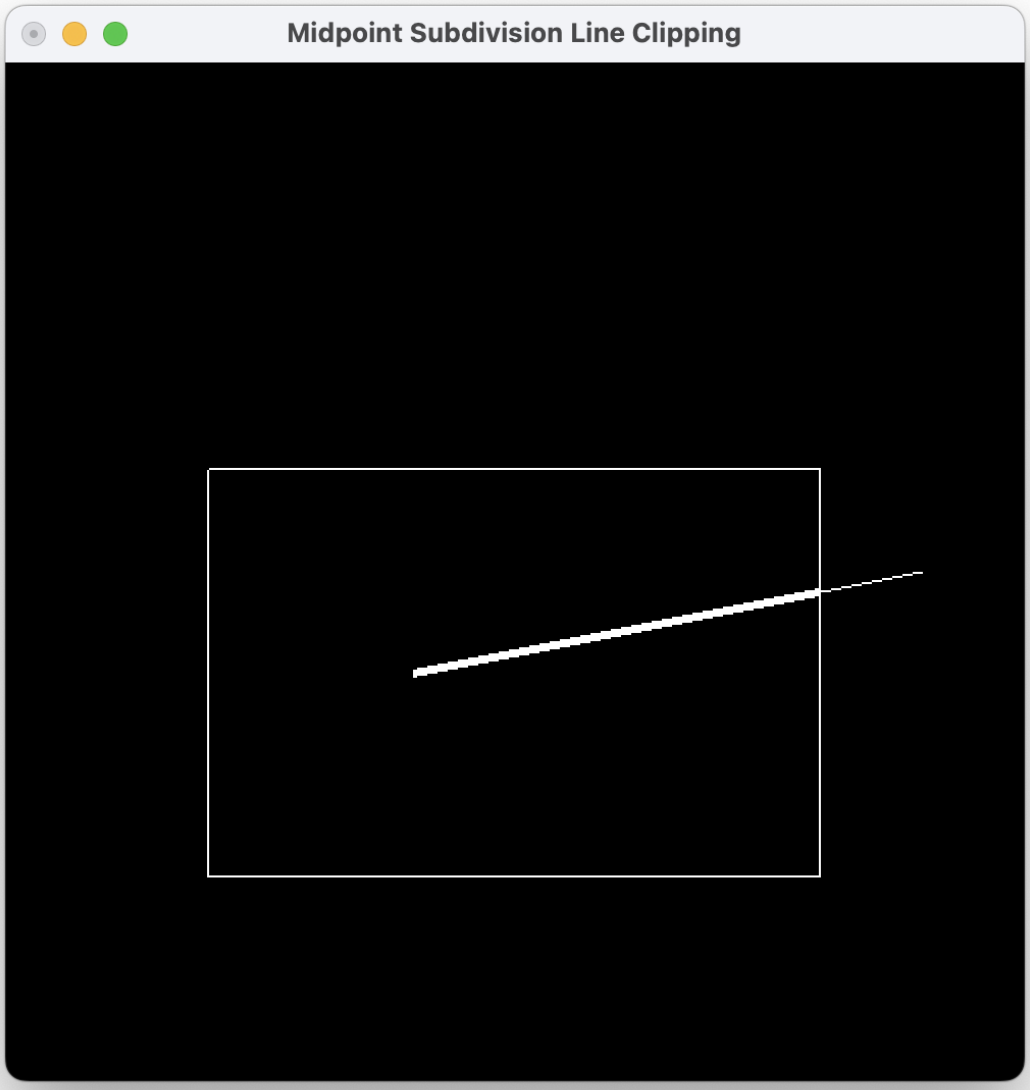
```
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Liang-Barsky Line Clipping");
    gluOrtho2D(0,500,0,500);
    glutDisplayFunc(display);
    glutMainLoop();
}
```



9) Line clipping using Midpoint Subdivision algorithm

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

#define LEFT_EDGE   0x1
#define RIGHT_EDGE  0x2
#define BOTTOM_EDGE 0x4
#define TOP_EDGE    0x8

#define REJECT(a,b) (a&b)
#define ACCEPT(a,b) (!(a|b))

int xwmin = 100, xwmax = 400, ywmin = 100, ywmax = 300;

void line(float x1, float y1, float x2, float y2, int
width) {
    glColor3f(1.0, 1.0, 1.0);
    glLineWidth(width);
    glBegin(GL_LINES);
    glVertex2d(x1, y1);
    glVertex2d(x2, y2);
    glEnd();
}

void clippingWindow() {
    glColor3f(1.0, 1.0, 1.0);
    glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
    glBegin(GL_POLYGON);
    glVertex2i(xwmin, ywmin);
    glVertex2i(xwmin, ywmax);
    glVertex2i(xwmax, ywmax);
    glVertex2i(xwmax, ywmin);
    glEnd();
}

unsigned char encode (float x, float y) {
    unsigned char code = 0x00;
    if (x < xwmin) code |= LEFT_EDGE;
    if (x > xwmax) code |= RIGHT_EDGE;
    if (y < ywmin) code |= BOTTOM_EDGE;
```

```
    if (y > ywmax) code |= TOP_EDGE;
    return code;
}

void clipLine(float x1, float y1, float x2, float y2, int
c) {
    if (c > 100) return;
    unsigned char code1 = encode(x1, y1),
                  code2 = encode(x2, y2);
    if (REJECT(code1, code2)) {
        return;
    } else if (ACCEPT(code1, code2)) {
        line(x1, y1, x2, y2, 4);
        return;
    } else {
        float xmid = (x1 + x2) / 2.0;
        float ymid = (y1 + y2) / 2.0;
        clipLine(x1, y1, xmid, ymid, c+1);
        clipLine(xmid, ymid, x2, y2, c+1);
    }
}

void display(void) {
    glClearColor(1.0, 1.0, 1.0, 0.0);
    clippingWindow();
    line(200, 200, 450, 250, 1);
    clipLine(200, 200, 450, 250,0);
    glFlush();
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Midpoint Subdivision Line Clipping");
    gluOrtho2D(0,500,0,500);
    glutDisplayFunc(display);
    glutMainLoop();
}
```


Midpoint Subdivision Line Clipping