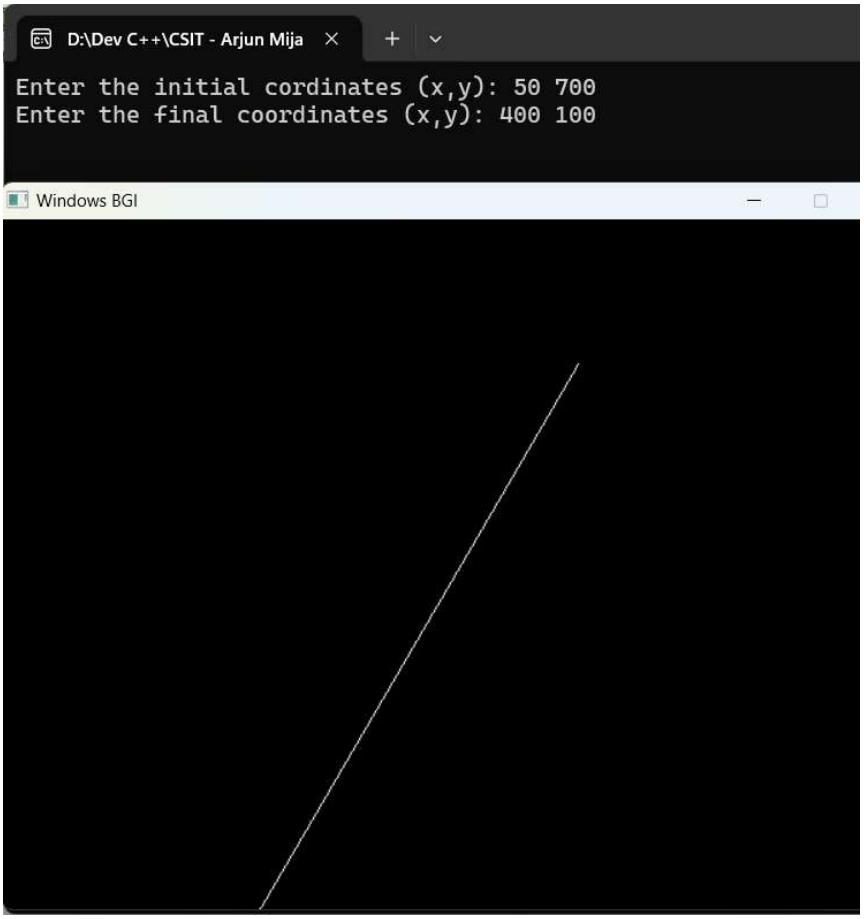
1. DDA line drawing algorithm to generate a line.

Source code:

```
// C program for DDA line generation
#include <iostream>
#include <graphics.h>
#include <math.h>
#include <conio.h>

// DDA Function for line generation
void DDA(int X0, int Y0, int X1, int Y1)
{
    int gd = DETECT, gm;
    // Initialize graphics function
    initgraph(&gd, &gm, "");
    int dx = X1 - X0;
    int dy = Y1 - Y0;
    int steps = fabs(dx) > fabs(dy) ? fabs(dx) : fabs(dy);
    float Xinc = dx / (float)steps;
    float Yinc = dy / (float)steps;
    float X = X0;
    float Y = Y0;
    for (int i = 0; i <= steps; i++) {
        putpixel(round(X), round(Y), WHITE);
        X += Xinc;
        Y += Yinc;
    }
```

```
    getch();
    closegraph();


    printf("\n\nDDA Completed.");
}
int main()
{
    int X0, Y0, X1, Y1;
    // scanning the points for drawing
    printf("Enter the initial cordinates (x,y): ");
    scanf("%d %d", &X0, &Y0);
    printf("Enter the final coordinates (x,y): ");
    scanf("%d %d", &X1, &Y1);
    // implement the DDA algorithm code
    DDA(X0, Y0, X1, Y1);
    getch();
    return 0;
}
```

Output:



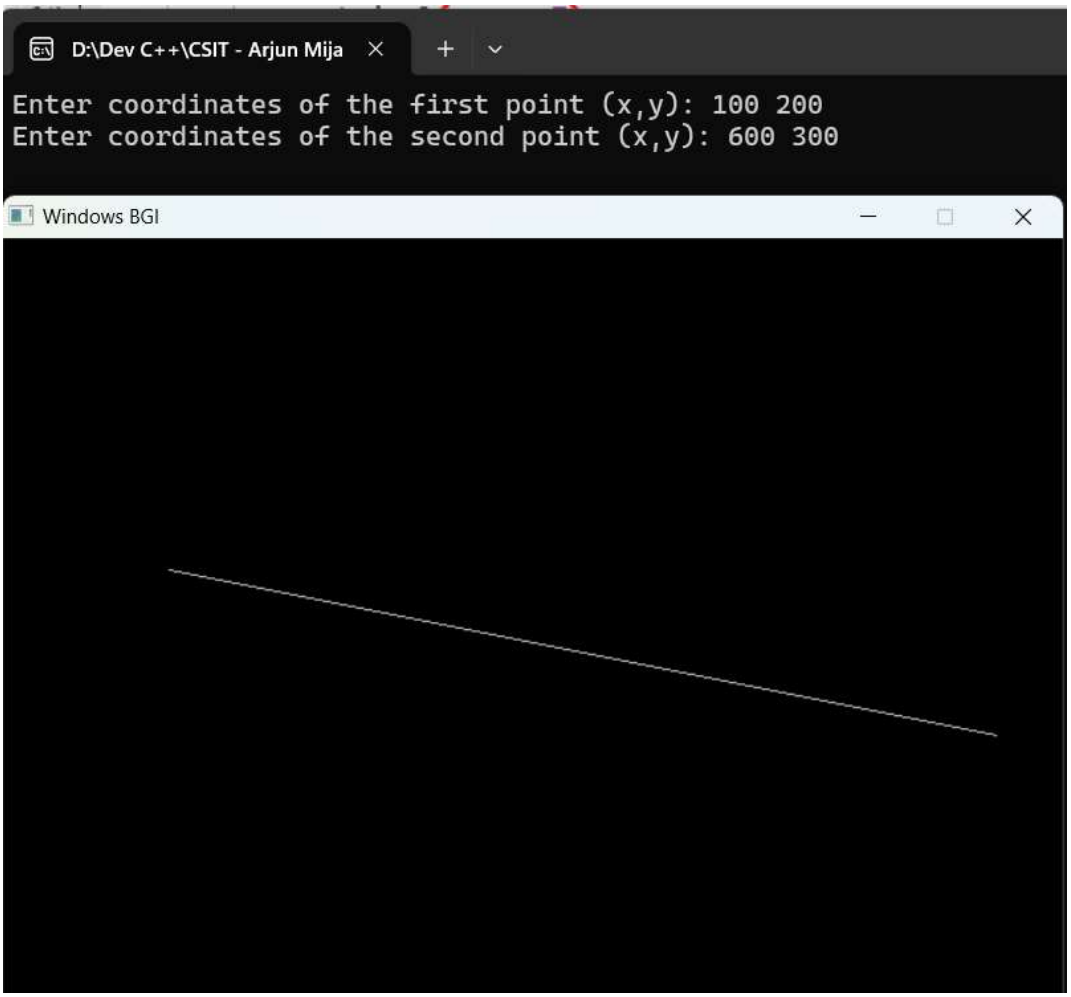
2. Bresenham's line drawing algorithm to generate a line

Source code:

```
#include <stdio.h>
#include <graphics.h>

void drawline(int x0, int y0, int x1, int y1) {
   int dx, dy, p, x, y;
   dx = x1 - x0;
   dy = y1 - y0;
   x = x0;
   y = y0;
   p = 2 * dy - dx;

   while (x < x1) {
      if (p >= 0) {
         putpixel(x, y, 7);
         y = y + 1;
         p = p + 2 * dy - 2 * dx;
      } else {
         putpixel(x, y, 7);
         p = p + 2 * dy;
      }
      x = x + 1;
   }
}
```

```
int main() {
    int gdriver = DETECT, gmode, error, x0, y0, x1, y1;
    initgraph(&gdriver, &gmode, "");

    printf("Enter coordinates of the first point (x,y): ");
    scanf("%d%d", &x0, &y0);
    printf("Enter coordinates of the second point (x,y): ");
    scanf("%d%d", &x1, &y1);

    drawline(x0, y0, x1, y1);
    getch();
    closegraph();
    return 0;
}
```

Output:



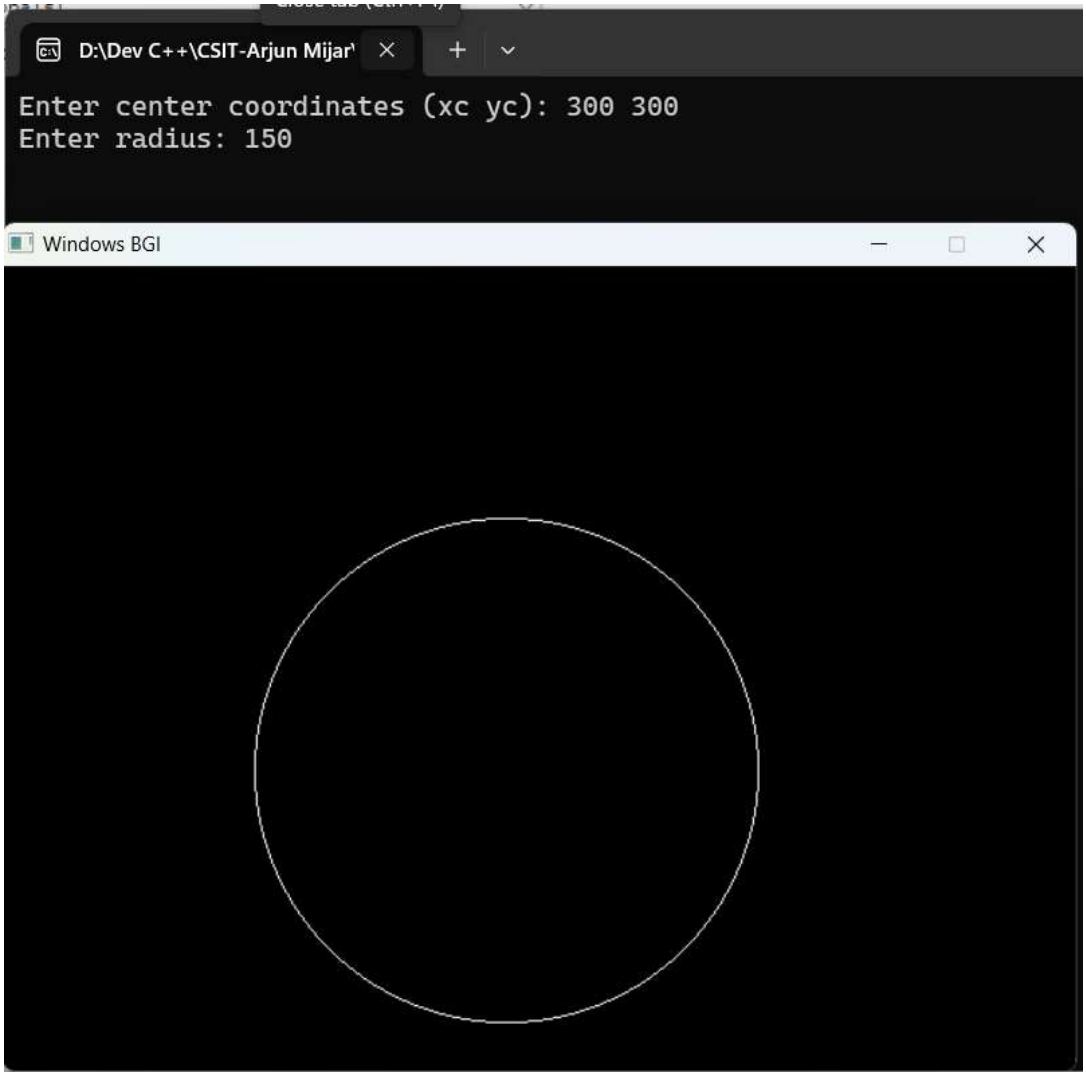
3. Mid-point circle drawing algorithm to draw an ellipse

Source Code:

```
#include <stdio.h>
#include <graphics.h>
void draw_circle(int xc, int yc, int radius) {
    int x = 0;
    int y = radius;
    int p = 1 - radius;
    while (x <= y) {
        putpixel(xc + x, yc + y, WHITE);
        putpixel(xc - x, yc + y, WHITE);
        putpixel(xc + x, yc - y, WHITE);
        putpixel(xc - x, yc - y, WHITE);
        putpixel(xc + y, yc + x, WHITE);
        putpixel(xc - y, yc + x, WHITE);
        putpixel(xc + y, yc - x, WHITE);
        putpixel(xc - y, yc - x, WHITE);
        if (p < 0) {
            p += 2 * x + 3;
        } else {
            p += 2 * (x - y) + 5;
            y--;
        }
        x++;
    }
}
```

```
int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");
    int xc, yc, radius;
    printf("Enter center coordinates (xc yc): ");
    scanf("%d %d", &xc, &yc);
    printf("Enter radius: ");
    scanf("%d", &radius);
    draw_circle(xc, yc, radius);
    getch();
    closegraph();
    return 0;
}
```

Output:



4. Mid-point ellipse drawing algorithm to draw a circle.

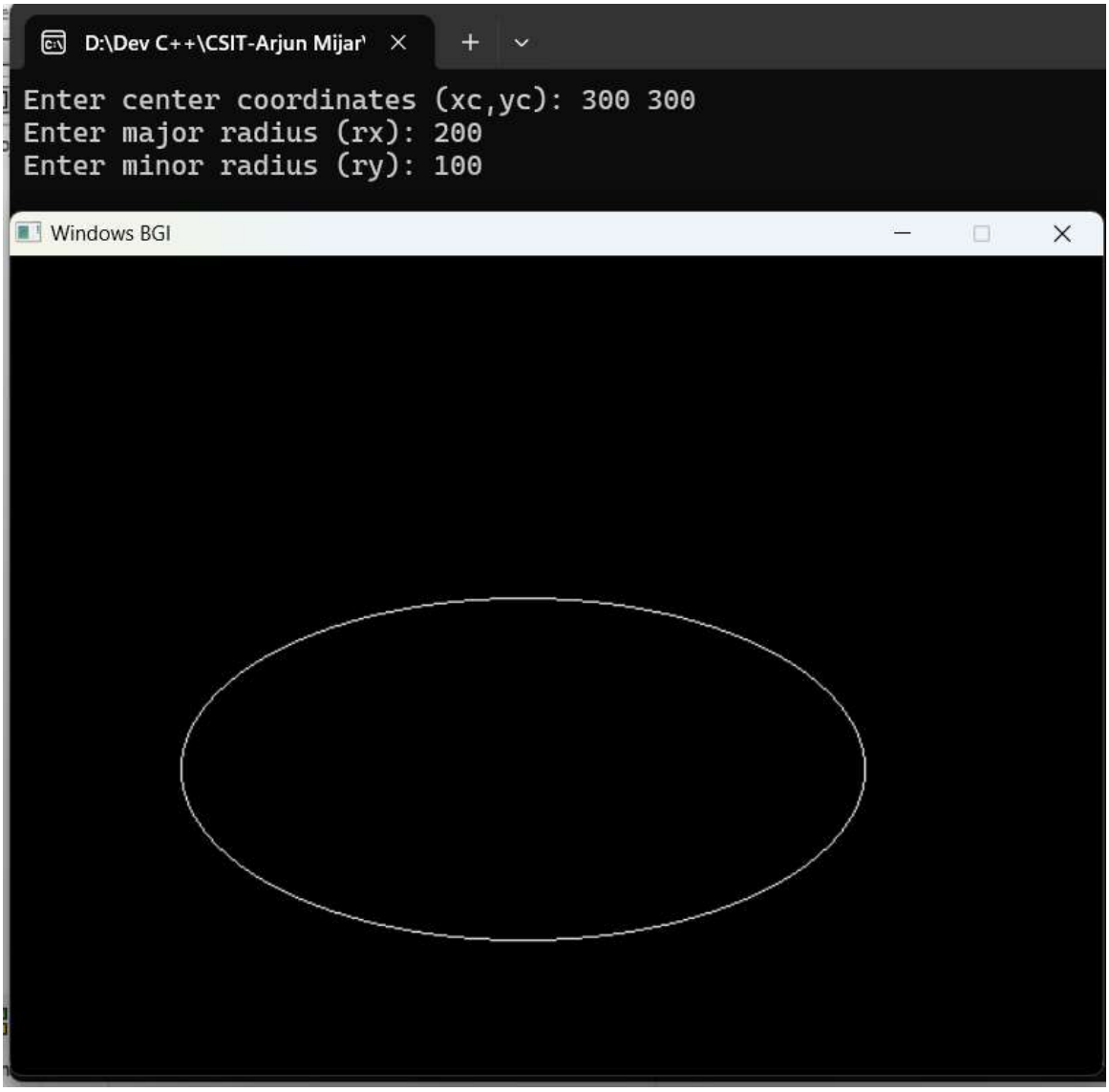
Source Code:

```
#include <stdio.h>
#include <graphics.h>

void draw_ellipse(int xc, int yc, int rx, int ry) {
   int x, y, p1, p2;
   x = 0;
   y = ry;
   p1 = ry * ry - rx * rx * ry + 0.25 * rx * rx;

   while (2 * ry * ry * x < 2 * rx * rx * y) {
      putpixel(xc + x, yc + y, WHITE);
      putpixel(xc - x, yc + y, WHITE);
      putpixel(xc + x, yc - y, WHITE);
      putpixel(xc - x, yc - y, WHITE);
      if (p1 < 0) {
         x++;
         p1 += 2 * ry * ry * x + ry * ry;
      } else {
         x++;
         y--;
         p1 += 2 * ry * ry * x - 2 * rx * rx * y + ry * ry;
      }
   }
```

```
    p2 = ry * ry * (x + 0.5) * (x + 0.5) + rx * rx * (y - 1) * (y - 1) - rx * rx * ry * ry;

    while (y >= 0) {
        putpixel(xc + x, yc + y, WHITE);
        putpixel(xc - x, yc + y, WHITE);
        putpixel(xc + x, yc - y, WHITE);
        putpixel(xc - x, yc - y, WHITE);

        if (p2 > 0) {
            y--;
            p2 += -2 * rx * rx * y + rx * rx;
        } else {
            x++;
            y--;
            p2 += 2 * ry * ry * x - 2 * rx * rx * y + rx * rx;
        }
    }
}

int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\Turboc3\\BGI");

    int xc, yc, rx, ry;
    printf("Enter center coordinates (xc,yc): ");
    scanf("%d %d", &xc, &yc);
```

```
    printf("Enter major radius (rx): ");
    scanf("%d", &rx);
    printf("Enter minor radius (ry): ");
    scanf("%d", &ry);

    draw_ellipse(xc, yc, rx, ry);

    getch();
    closegraph();
    return 0;
}
```

Output:



5. Implementation of 2D transformations

Source Code:

```
#include <iostream>
#include <complex>
#include <conio.h>
#include <graphics.h>
#include <math.h>
using namespace std;

typedef complex<double> point;
#define x real()
#define y imag()

void displaymenu()
{
   cout << "Press 1 for translation." << endl;
   cout << "Press 2 for rotation." << endl;
   cout << "Press 3 for scaling." << endl;
   cout << "Press 4 for reflection." << endl;
}

void reflectionmenu()
{
   cout << "Press 1 for reflection through x-axis." << endl;
   cout << "Press 2 for reflection through y-axis." << endl;
}
```

```
point translation(point A, int a, int b)
{
   point B(A.real() + a, A.imag() + b);
   return B;
}

point rotation(point A, int a, int b, float angl)
{
   point C = translation(A, -a, -b);
   point B(((C.real() * cos(angl)) - (C.imag() * sin(angl))), ((C.real() * sin(angl)) + (C.imag()) * cos(angl)));
   point D = translation(B, +a, +b);
   return D;
}

point scaling(point A, int a, int b)
{
   point B(A.real() * a, A.imag() * b);
   return B;
}

point reflectionthx(point A)
{
   point C(A.real(), -A.imag());
   point B = translation(C, 0, 600);
   return B;
}
```

```
point reflectionthy(point A)
{
   point C(-A.real(), A.imag());
   point B = translation(C, 600, 0);
   return B;
}

int drawpolygon(point W, point X, point Y, point Z)
{
   line(W.real(), W.imag(), Z.real(), Z.imag());
   delay(200);
   line(Z.real(), Z.imag(), Y.real(), Y.imag());
   delay(200);
   line(Y.real(), Y.imag(), X.real(), X.imag());
   delay(200);
   line(X.real(), X.imag(), W.real(), W.imag());
   delay(200);
   return 0;
}

int main()
{
   int gd = DETECT, gm;
   point E, F, G, H;
   int x1, y1, x2, y2, x3, y3, x4, y4, choice, subchoice;
```

```
cout << "Enter four coordinates of polygon (one in a single line): ";
cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3 >> x4 >> y4;

initgraph(&gd, &gm, "");
point A(x1, y1);
point B(x2, y2);
point C(x3, y3);
point D(x4, y4);
drawpolygon(A, B, C, D);
displaymenu();

cout << "Enter your choice: ";
cin >> choice;

switch (choice)
{
case 1:
   int a, b;
   cout << "Enter translation distances: ";
   cin >> a >> b;
   E = translation(A, a, b);
   F = translation(B, a, b);
   G = translation(C, a, b);
   H = translation(D, a, b);
   setcolor(BLUE);
```

```
    drawpolygon(E, F, G, H);
    getch();
    break;

case 2:
    float angle, ang;
    int c, d; // pivot
    cout << "Enter pivot point for rotation: ";
    cin >> c >> d;
    cout << "Enter angle through which you want to rotate: ";
    cin >> ang;
    angle = (ang * 3.14) / 180;
    E = rotation(A, c, d, angle);
    F = rotation(B, c, d, angle);
    G = rotation(C, c, d, angle);
    H = rotation(D, c, d, angle);
    setcolor(BLUE);
    drawpolygon(E, F, G, H);
    break;

case 3:
    int sx, sy;
    cout << "Enter scaling factors (Sx, Sy): ";
    cin >> sx >> sy;
    E = scaling(A, sx, sy);
    F = scaling(B, sx, sy);
```

```
    G = scaling(C, sx, sy);
    H = scaling(D, sx, sy);
    setcolor(RED);
    drawpolygon(E, F, G, H);
    break;

case 4:
    reflectionmenu();
    cout << "Choose type of reflection: ";
    cin >> subchoice;
    switch (subchoice)
    {
    case 1:
        E = reflectionthx(A);
        F = reflectionthx(B);
        G = reflectionthx(C);
        H = reflectionthx(D);
        setcolor(BLUE);
        drawpolygon(E, F, G, H);
        break;

    case 2:
        E = reflectionthy(A);
        F = reflectionthy(B);
        G = reflectionthy(C);
        H = reflectionthy(D);
```

```
            setcolor(BLUE);
            drawpolygon(E, F, G, H);
            break;
        }
        break;
    default:
        cout << "Invalid Choice.." << endl;
        break;
    }
    getch();
    closegraph();
    return 0;
}
```

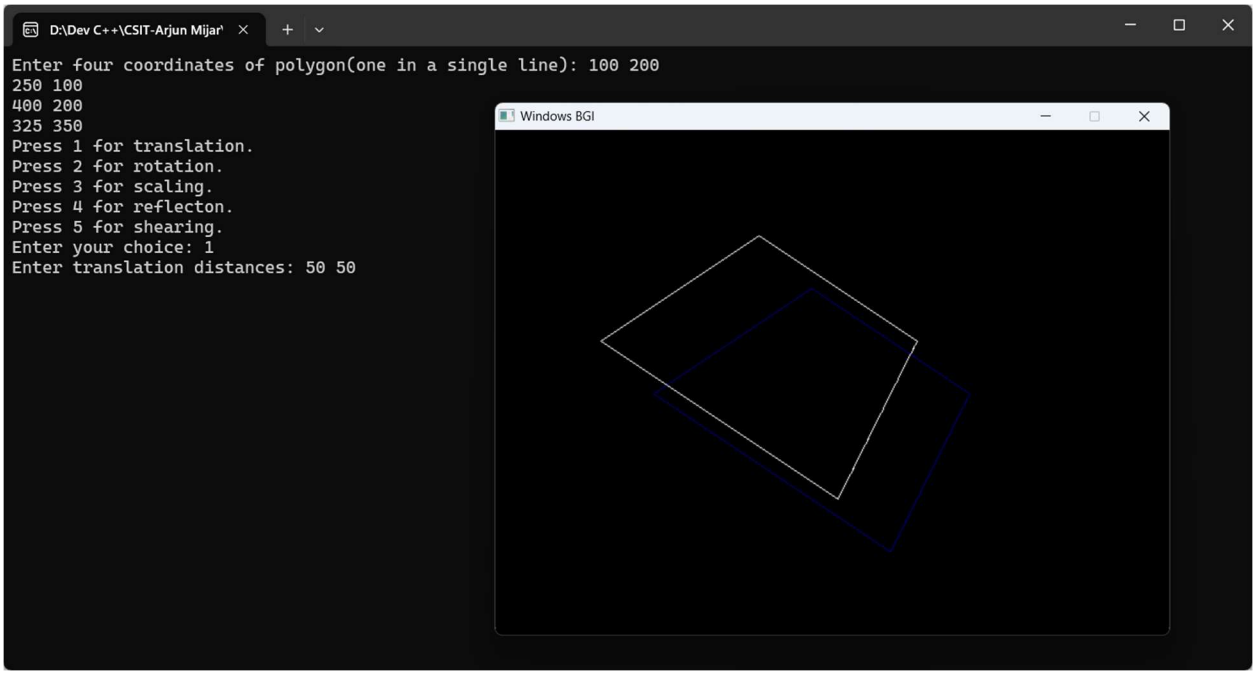


Figure: Output for translation

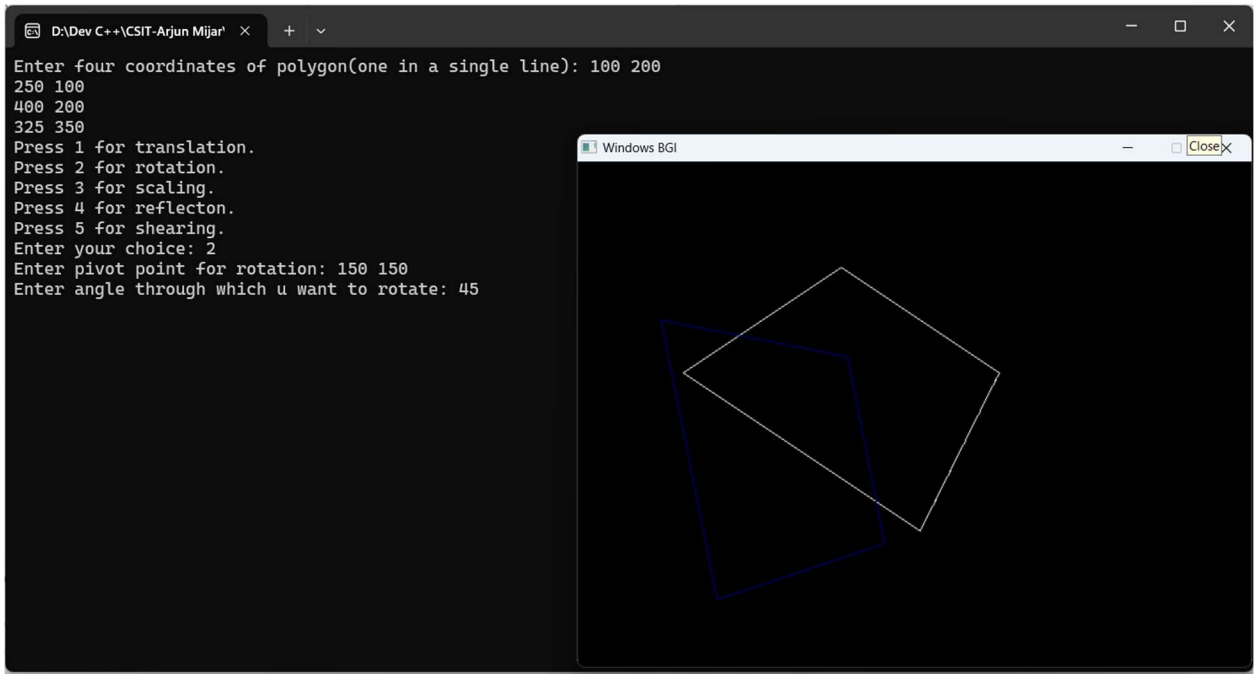


Figure: Output for rotation

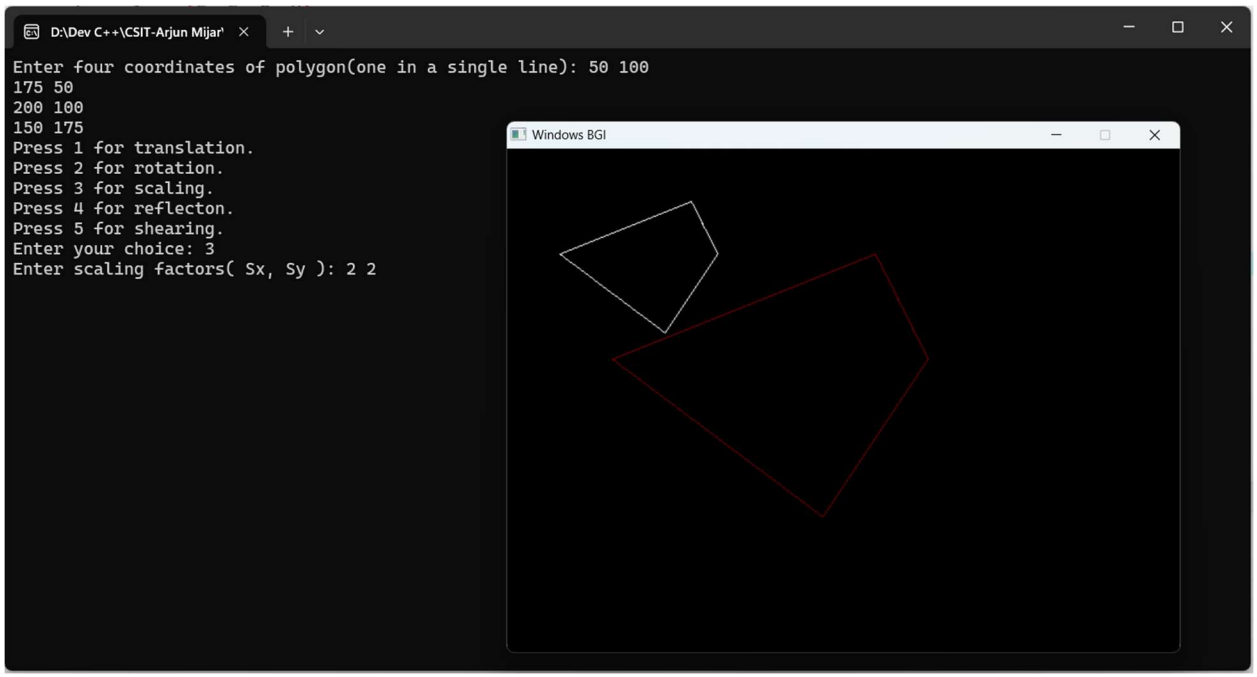


Figure: Output for scaling

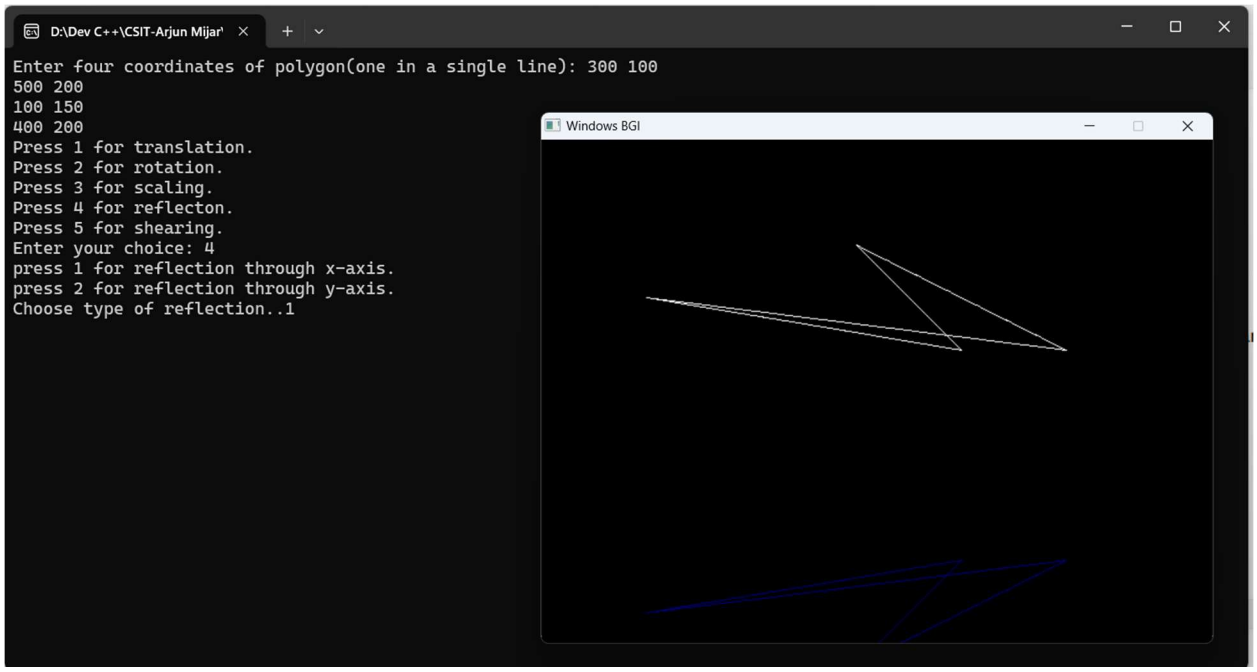


Figure: Output for reflection on x-axis

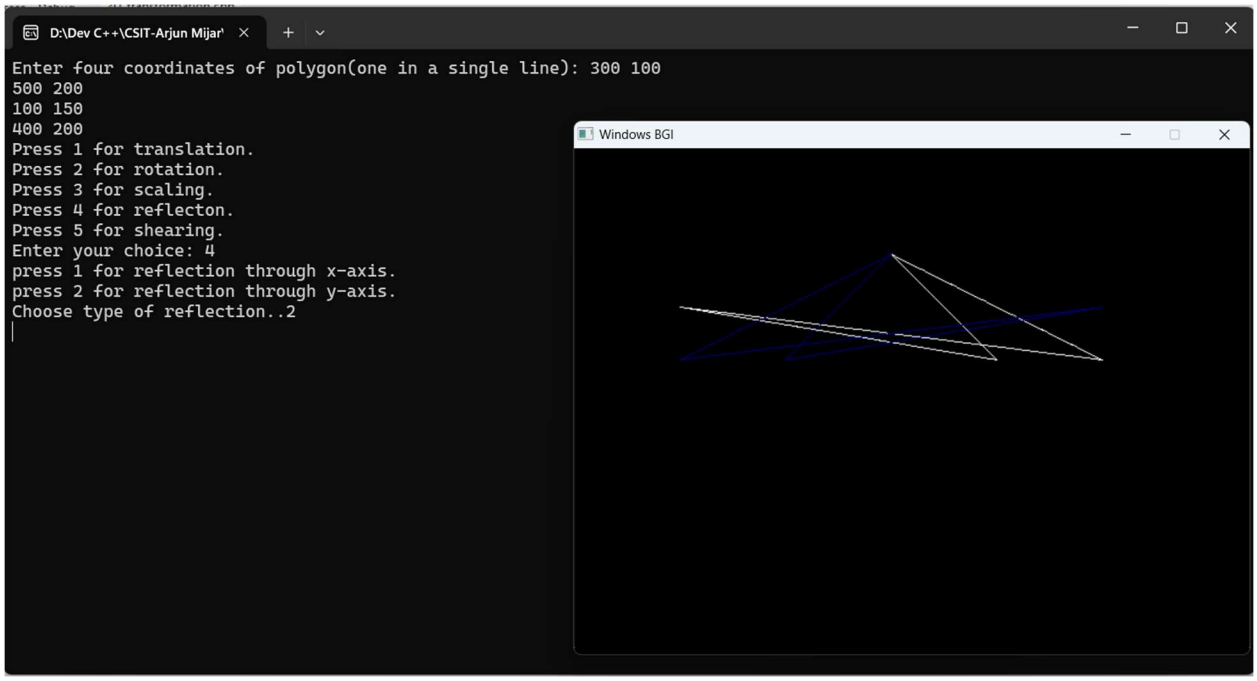


Figure: Output for reflection on y-axis

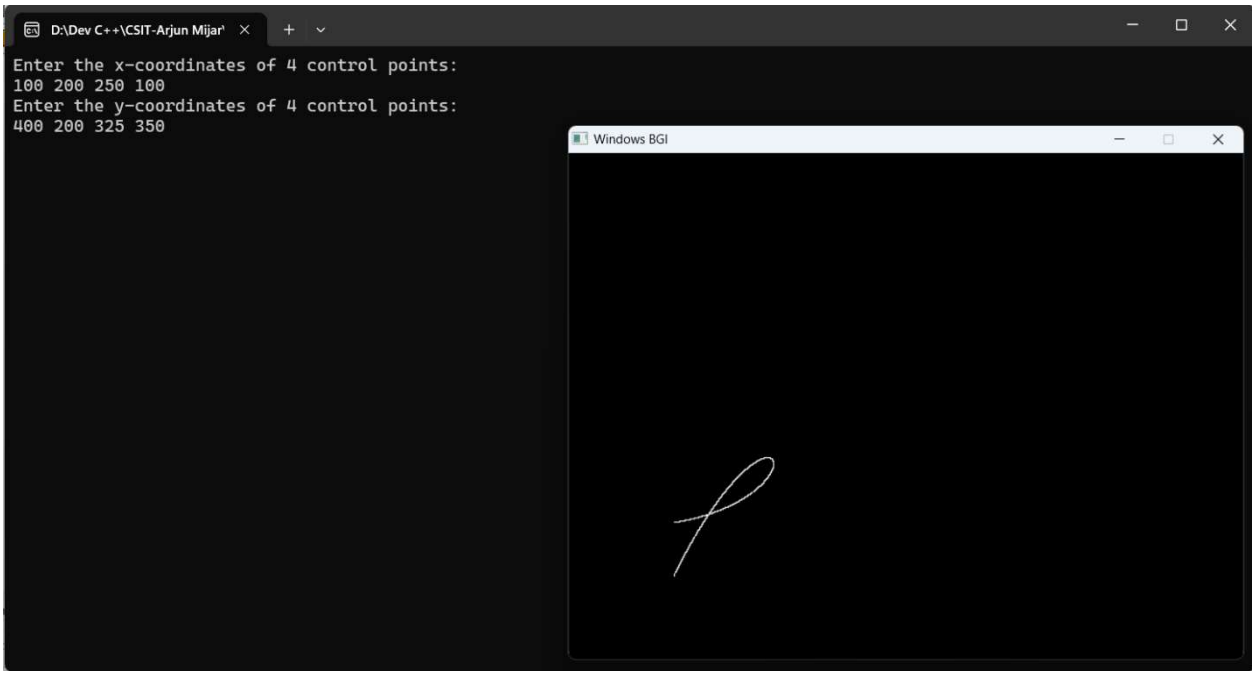
6. Bezier curve implementation using c program

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <math.h>

void bezier(int x[], int y[]) {
   int gd = DETECT, gm;
   int i;
   double t;

   initgraph(&gd, &gm, NULL);

   for (t = 0; t <= 1; t += 0.0001) {
      double u = 1 - t;
      double x_pixel = pow(u, 3) * x[0] + 3 * t * pow(u, 2) * x[1] + 3 * pow(t, 2) * u * x[2] + pow(t, 3) *
x[3];
      double y_pixel = pow(u, 3) * y[0] + 3 * t * pow(u, 2) * y[1] + 3 * pow(t, 2) * u * y[2] + pow(t, 3) *
y[3];
      putpixel((int)x_pixel, (int)y_pixel, WHITE);
   }

  getch();
   closegraph();
}
```

```
int main() {
    int x[4], y[4]; // Control points

    printf("Enter the x-coordinates of 4 control points:\n");
    for (int i = 0; i < 4; i++) {
        scanf("%d", &x[i]);
    }

    printf("Enter the y-coordinates of 4 control points:\n");
    for (int i = 0; i < 4; i++) {
        scanf("%d", &y[i]);
    }
    bezier(x, y);
    return 0;
}
```

Output:



7. Implementation of 3D transformations

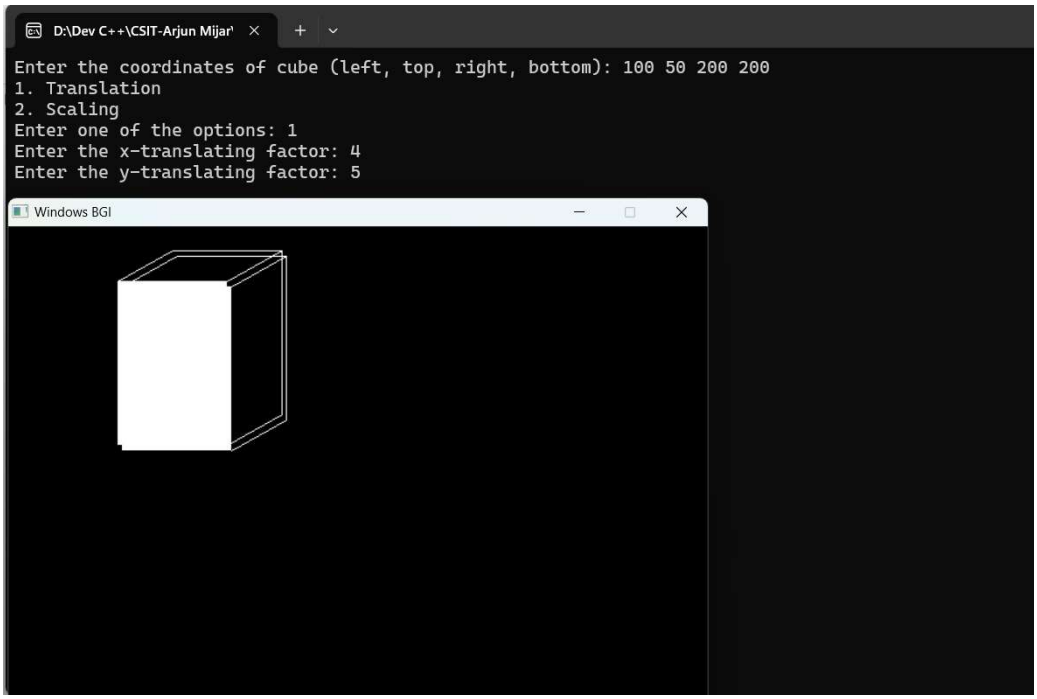
Source Code:

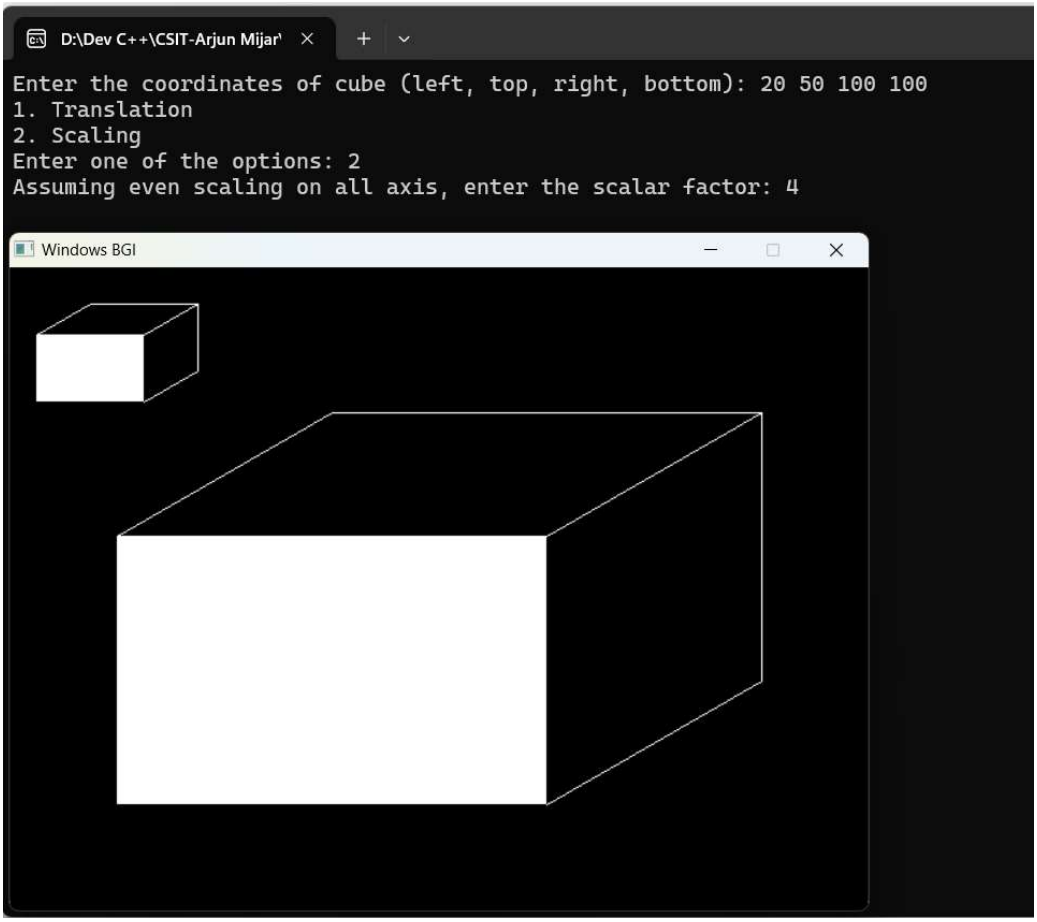
```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <stdlib.h>
#include <math.h>
int translation(int n, int tn) {
   return (n + tn);}
int scalar(int k, int n) {
   return (k * n);}
int main() {
   int gd = DETECT, gm, left, right, bottom, top, depth, k;
   initgraph(&gd, &gm, "");
   printf("BIJAY DHAKAL\n");
   printf("Enter the coordinates of cube (left, top, right, bottom): ");
   scanf("%d %d %d %d", &left, &top, &right, &bottom);
   depth = fabs(right - left) / 2;
   bar3d(left, top, right, bottom, depth, 1);
   printf("1. Translation\n2. Scaling\n");
   printf("Enter one of the options: ");
   scanf("%d", &k);
   switch (k) {
      case 1:
         printf("Enter the x-translating factor: ");
```

```
    scanf("%d", &k);
    left = translation(left, k);
    right = translation(right, k);
    printf("Enter the y-translating factor: ");
    scanf("%d", &k);
    top = translation(top, k);
    bottom = translation(bottom, k);
    depth = fabs(right - left) / 2;
    bar3d(left, top, right, bottom, depth, 1);
    getch();
    break;
  case 2:
    printf("Assuming even scaling on all axis, enter the scalar factor: ");
    scanf("%d", &k);
    left = scalar(left, k);
    right = scalar(right, k);
    top = scalar(top, k);
    bottom = scalar(bottom, k);
    depth = fabs(right - left) / 2;
    bar3d(left, top, right, bottom, depth, 1);
    getch();
    break;
  default:
    printf("Invalid selection!");
    break;
```

```
    }    closegraph();
    return 0;
}
```

Output





8. Implementation of Cohen Sutherland's line clipping algorithm

Source Code:

```
#include <stdio.h>
#include <graphics.h>

// Define region codes for Cohen-Sutherland algorithm
#define INSIDE 0 // 0000
#define LEFT 1   // 0001
#define RIGHT 2  // 0010
#define BOTTOM 4 // 0100
#define TOP 8    // 1000

// Function to compute region code for a point (x, y)
int computeCode(int x, int y, int xmin, int ymin, int xmax, int ymax) {
    int code = INSIDE;

    if (x < xmin)
        code |= LEFT;
    else if (x > xmax)
        code |= RIGHT;
    if (y < ymin)
        code |= BOTTOM;
    else if (y > ymax)
        code |= TOP;

    return code;
```

```
}
// Cohen-Sutherland line clipping algorithm
void cohenSutherland(int x1, int y1, int x2, int y2, int xmin, int ymin, int xmax, int ymax) {
    int code1, code2;
    int accept = 0, done = 0;
    code1 = computeCode(x1, y1, xmin, ymin, xmax, ymax);
    code2 = computeCode(x2, y2, xmin, ymin, xmax, ymax);
    do {
        // If both endpoints lie inside the window
        if (code1 == 0 && code2 == 0) {
            accept = 1;
            done = 1;
        }
        // If both endpoints lie outside the window
        else if (code1 & code2) {
            done = 1;
        }
        // If one of the endpoints is inside the window
        else {
            int x, y;
            int code = code1 ? code1 : code2;
            // Calculate intersection point
            if (code & TOP) {
                x = x1 + (x2 - x1) * (ymax - y1) / (y2 - y1);
                y = ymax;
            } else if (code & BOTTOM) {
```

```
            x = x1 + (x2 - x1) * (ymin - y1) / (y2 - y1);
            y = ymin;
        } else if (code & RIGHT) {
            y = y1 + (y2 - y1) * (xmax - x1) / (x2 - x1);
            x = xmax;
        } else if (code & LEFT) {
            y = y1 + (y2 - y1) * (xmin - x1) / (x2 - x1);
            x = xmin;
        }
        // Update the endpoint
        if (code == code1) {
            x1 = x;
            y1 = y;
            code1 = computeCode(x1, y1, xmin, ymin, xmax, ymax);
        } else {
            x2 = x;
            y2 = y;
            code2 = computeCode(x2, y2, xmin, ymin, xmax, ymax);
        }
    }
} while (!done);

// If line is accepted, draw it
if (accept) {
    setcolor(RED);
    line(x1, y1, x2, y2);
```

```
    }
}
int main() {
    int gd = DETECT, gm;
    int xmin, ymin, xmax, ymax;
    int x1, y1, x2, y2;
    printf("Enter the window boundaries (xmin ymin xmax ymax): ");
    scanf("%d %d %d %d", &xmin, &ymin, &xmax, &ymax);
    printf("Enter the endpoints of the line (x1 y1 x2 y2): ");
    scanf("%d %d %d %d", &x1, &y1, &x2, &y2);

    initgraph(&gd, &gm, "");
    // Draw the window
    rectangle(xmin, ymin, xmax, ymax);

    // Draw the original line
    setcolor(WHITE);
    line(x1, y1, x2, y2);
    // Apply Cohen-Sutherland algorithm
    cohenSutherland(x1, y1, x2, y2, xmin, ymin, xmax, ymax);
    getch();
    closegraph();
    return 0;
}
```

9. Implementation of Liang – Barsky line clipping algorithm

Source Code:

```
#include <stdio.h>
#include <graphics.h>
// Liang-Barsky line clipping algorithm
void liangBarsky(int x1, int y1, int x2, int y2, int xmin, int ymin, int xmax, int ymax) {
    int p[4], q[4];
    float u1 = 0, u2 = 1;
    int dx = x2 - x1;
    int dy = y2 - y1;
    int xdelta = xmin - x1;
    int ydelta = ymin - y1;
    p[0] = -dx;  p[1] = dx;  p[2] = -dy;  p[3] = dy;
    q[0] = xdelta;  q[1] = xmax - x1;  q[2] = ydelta;  q[3] = ymax - y1;
    for (int i = 0; i < 4; i++) {
        if (p[i] == 0) {
            if (q[i] < 0)
                return; // Line is parallel to clipping boundary and outside
        } else {
            float u = (float)q[i] / p[i];
            if (p[i] < 0 && u > u1)
                u1 = u;
            else if (p[i] > 0 && u < u2)
                u2 = u;}}
    if (u1 < u2) {
        int x1_clip = x1 + u1 * dx;
```

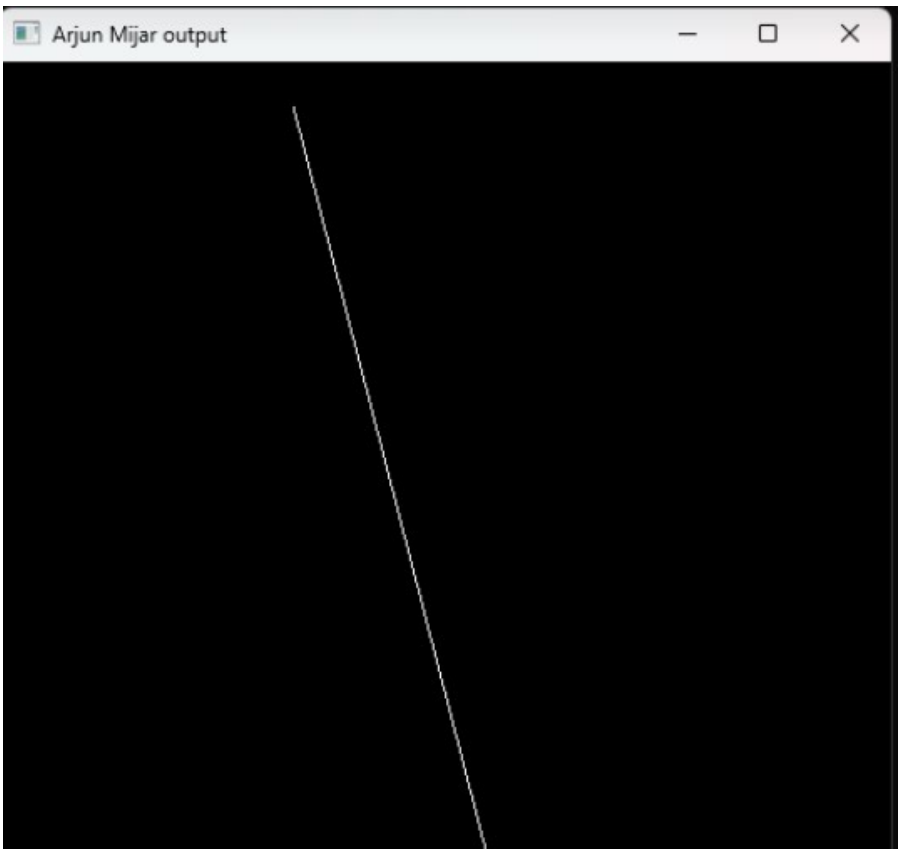
```
        int y1_clip = y1 + u1 * dy;
        int x2_clip = x1 + u2 * dx;
        int y2_clip = y1 + u2 * dy;
        setcolor(RED);
        line(x1_clip, y1_clip, x2_clip, y2_clip);
    }}
int main() {
    int gd = DETECT, gm;
    int xmin, ymin, xmax, ymax;
    int x1, y1, x2, y2;
    printf("Enter the window boundaries (xmin ymin xmax ymax): ");
    scanf("%d %d %d %d", &xmin, &ymin, &xmax, &ymax);
    printf("Enter the endpoints of the line (x1 y1 x2 y2): ");
    scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
    initgraph(&gd, &gm, "");
    // Draw the window
    rectangle(xmin, ymin, xmax, ymax);
    // Draw the original line
    setcolor(WHITE);
    line(x1, y1, x2, y2);
    // Apply Liang-Barsky algorithm
    liangBarsky(x1, y1, x2, y2, xmin, ymin, xmax, ymax);
    getch();
    closegraph();
    return 0;
}
```

11.To draw a line using OpenGL

Source Code:

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <windows.h>
#include <GL/glut.h>
#endif
#include <stdlib.h>
void init(void) {
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glViewport(10, 0, 500, 500);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0, 500, 500, 0, 1, -1);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}void drawLines(void) {
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glPointSize(3.0);
   glBegin(GL_LINES);
      glVertex2i(275, 450);
      glVertex2i(165, 25);
   glEnd();
   glFlush();}
```

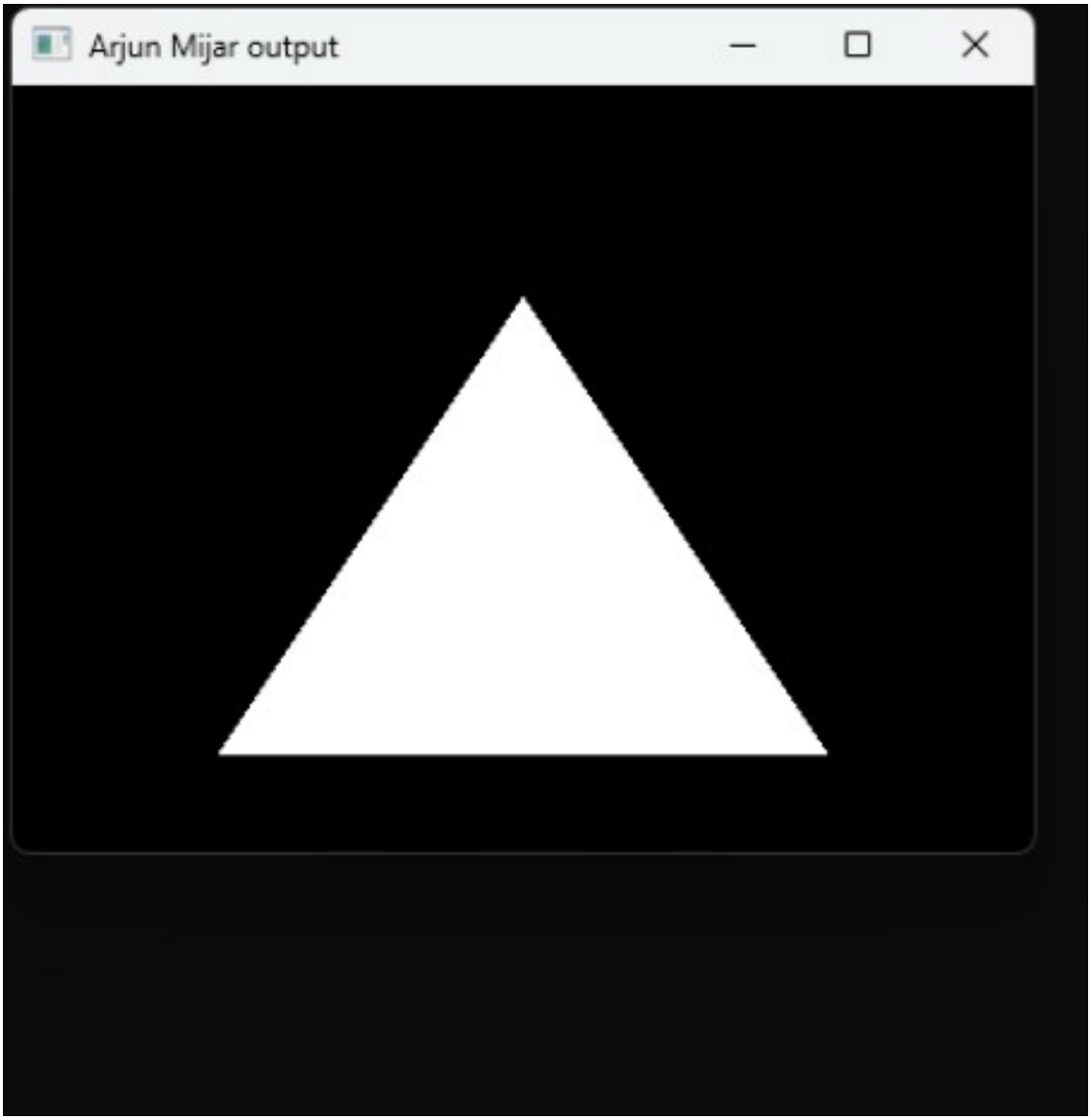
```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitWindowPosition(10, 10);
    glutInitWindowSize(500, 500);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("bijay line output");
    init();
    glutDisplayFunc(drawLines);
    glutMainLoop();
    return 0;
}
```



12.To draw triangle using OpenGL

Source Code:

```
#include <windows.h>
#ifndef __APPLE__
#include <GL/glut.h>
#endif
#include <stdlib.h>
void display() {
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POLYGON);
      glColor3f(1.0, 1.0, 1.0); glVertex3f(-0.6, -0.75, 0.5);
      glColor3f(1.0, 1.0, 1.0); glVertex3f(0.6, -0.75, 0.0);
      glColor3f(1.0, 1.0, 1.0); glVertex3f(0.0, 0.45, 0.0);
   glEnd();
   glFlush();
}
int main(int argc, char **argv) {
   glutInit(&argc, argv);
   glutInitWindowPosition(80, 80);
   glutInitWindowSize(400, 300);
   glutCreateWindow("bijay Triangle output");
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```


Arjun Mijar output