SRI SIVASUBRAMANIYA NADAR COLLEGE OF ENGINEERING

(AN AUTONOMOUS INSTITUTION,
AFFILIATED TO ANNA UNIVERSITY)

Rajiv Gandhi Salai (OMR), Kalavakkam - 603 110.

LABORATORY RECORD

NAME : KRITHIKA SWAMINATHAN

Reg. No. : 205001057

Dept. : CSE Sem. : VII Sec. : A

SSN

SRI SIVASUBRAMANIYA NADAR COLLEGE OF ENGINEERING, CHENNAI

(AN AUTONOMOUS INSTITUTION, AFFILIATED TO ANNA UNIVERSITY)

BONAFIDE CERTIFICATE

Certified that this is the bonafide record of the practical work done in the UCS1712 - GRAPHICS AND MULTIMEDIA Laboratory by

Name KRITHIKA SWAMINATHAN

Register Number 205001057

Semester VII

Branch COMPUTER SCIENCE AND ENGINEERING

Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam.

During the Academic year 2023-24

27/11/23.

Faculty

Head of the Department

Submitted for the END SEMESTER Practical Examination held at SSNCE on 28/11/2023

Internal Examiner

External Examiner

INDEX

Name : KRITHIKA SWAMINATHAN Reg. No. 205001057

Sem : VII Sec : A

Ex. No.	Date of Expt.	Title of the Experiment	Page No.	Signature of the Faculty	Remarks
1.	07/08/2023	Study of Basic Output Primitives in C++ using OpenGL	1		
2.	21/08/2023	DDA Line Drawing Algorithm	11	11/09/23.	
3.	28/08/2023	Bresenham Line Drawing Algorithm	25		
4.	04/09/2023	Midpoint Circle Drawing Algorithm	38		
5.	11/09/2023	2D Transformations	43		
6.	25/09/2023	2D Composite Transformations and Windowing	59	28/10/23.	
7.	09/10/2023	Cohen Sutherland Line Clipping	79		
8.	16/10/2023	3D Transformations in C++ using OpenGL	88		
9.	26/10/2023	3D Projections in C++ using OpenGL	97		
10.	30/10/2023	Creating a 3D Scene in C++ using OpenGL	102		
11.	30/10/2023	Image Editing and Manipulation	109	16/11/23.	
12.	06/11/2023	Creating 2D Animation	115		
13.	06/11/2023	Mini Project	117		

Exercise 1 – Study of Basic Output Primitives in C++ using OpenGL

Date: 14/08/2023

Aim:

To complete the following tasks:

a. To create an output window using OPENGL and to draw the following basic output primitives
– POINTS, LINES, LINE_STRIP, LINE_LOOP, TRIANGLES, QUADS, QUAD_STRIP, POLYGON.
b. To create an output window and draw a checkerboard using OpenGL.
c. To create an output window and draw a house using POINTS,LINES,TRIANGLES and QUADS/POLYGON.

Code:

```
#define GL_SILENCE_DEPRECATION

#include<GLUT/glut.h>

void myInit() {
	glClearColor(0.6,0.2,1.0,0.0);
	//glClearColor(0.4,0.2,1.0,0.0);
	glColor3f(0.0f,0.0f,0.5f);
	glPointSize(10);
	glMatrixMode(GL_PROJECTION);
	//glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
	glLineWidth(4);
	glLoadIdentity();
	gluOrtho2D(0.0,640.0,0.0,480.0);
}

//question a

void displayPoints() {
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_POINTS);
	glVertex2d(150,100);
	glVertex2d(100,230);
	glVertex2d(170,130);
	glVertex2d(300,350);
	glVertex2d(400,200);
```

```
	glVertex2d(400,350);
	glEnd();
	glFlush();
}

void displayLines() {
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_LINES);
	glColor4f(1,1,0.4,1);
	glVertex2d(300,200);
	glVertex2d(350,300);
	glEnd();
	glFlush();
}

void displayLineStrip() {
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_LINE_STRIP);
	glColor4f(1,1,0.3,1);
	glVertex2d(200,200);
	glVertex2d(200,350);
	glVertex2d(300,200);
	glVertex2d(300,350);
	glVertex2d(400,350);
	glVertex2d(400,200);
	glVertex2d(500,200);
	glVertex2d(500,350);
	glEnd();
	glFlush();
}

void displayLineLoop() {
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_LINE_LOOP);
	glColor4f(1,1,0.4,1);
	glVertex2d(300,200);
	glVertex2d(300,350);
	glVertex2d(400,350);
	glVertex2d(400,200);
	glEnd();
	glFlush();
}

void displayTriangles() {
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_TRIANGLES);
```

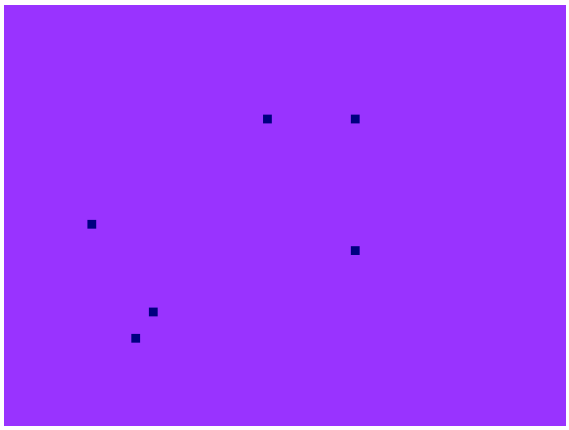
```
	glColor4f(1,1,0.4,1);
	glVertex2d(300,200);
	glVertex2d(350,300);
	glVertex2d(400,200);
	glEnd();
	glFlush();
}

void displayQuads() {
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_QUADS);
	glColor4f(1,1,0.4,1);
	glVertex2d(300,200);
	glVertex2d(300,350);
	glVertex2d(400,350);
	glVertex2d(400,200);
	glEnd();
	glFlush();
}

void displayQuadStrip() {
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_QUAD_STRIP);
	glColor4f(1,1,0.3,1);
	glVertex2d(200,200);
	glVertex2d(200,350);
	glVertex2d(300,200);
	glVertex2d(300,350);
	glVertex2d(400,350);
	glVertex2d(400,200);
	glVertex2d(500,200);
	glVertex2d(500,350);
	glEnd();
	glFlush();
}

void displayPolygon() {
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_POLYGON);
	glColor4f(1,1,0.4,1);
	glVertex2d(300,200);
	glVertex2d(300,300);
	glVertex2d(350,350);
	glVertex2d(400,300);
	glVertex2d(400,200);
	glEnd();
```

```
	glFlush();
}

//question b

/*
void drawCheckerBoard() {
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_QUAD_STRIP);
	glColor4f(1,1,1,1);

	for (int i=0; i<=600; i+=20) {
	for (int j=0; j<=600; j+=20) {
		glVertex2d(i, j);
	}
	}

	glEnd();
	glFlush();
}
*/

void drawChecker(int size)
{
	int i = 0;
	int j = 0;
	for (i = 0; i < 20 ; ++i) {
	for (j = 0; j < 16; ++j) {
		if((i + j)%2 == 0) // if i + j is even
		glColor3f( 1.0, 1.0, 1.0);
		else
		glColor3f( 0.0, 0.0, 0.0);
		glRecti(i*size, j*size, (i+1)*size, (j+1)*size);
	}
	}
	glFlush();
}

void checkerboard(void) {
	glClear(GL_COLOR_BUFFER_BIT);	// clear the screen

	drawChecker(32);
}

//question c
```

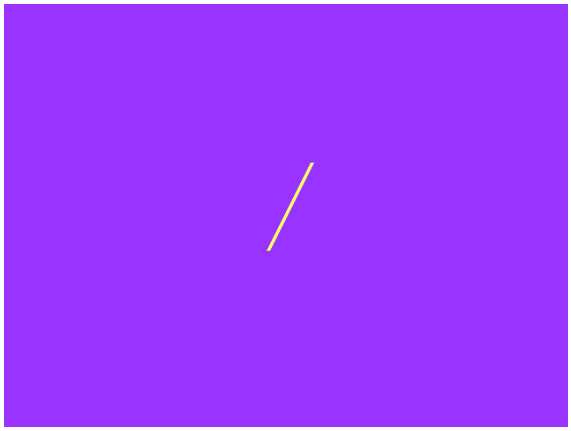
```
void displayRectangle(int x, int y, int w, int h) {
	glBegin(GL_POLYGON);
	glVertex2d(x,y);
	glVertex2d(x+w,y);
	glVertex2d(x+w,y+h);
	glVertex2d(x,y+h);
	glEnd();
}

void displayTriangle(int x, int y, int w, int h) {
	glBegin(GL_TRIANGLES);
	glVertex2d(x,y);
	glVertex2d(x+w,y);
	glVertex2d(x+(w/2),y+h);
	glEnd();
}

void displayHouse() {
	glClear(GL_COLOR_BUFFER_BIT);
	glColor4f(0.768,0.643,0.517,1);
	displayRectangle(200,0,150,150);
	glColor4f(0.5,0,0,1);
	displayTriangle(200, 150, 150, 100);
	glColor4f(0.36,0.15,0,1);
	displayRectangle(250,0,50,80);
	glColor4f(0,0,0,1);
	displayRectangle(280,30,10,10);
	glColor4f(0,0.2,0,1);
	displayRectangle(0,0,200,30);
	glColor4f(0,0.2,0,1);
	displayRectangle(350,0,400,30);
	glFlush();
}

int main(int argc,char* argv[]) {
	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
	glutInitWindowSize(640,480);
	glutCreateWindow("Shapes");
	glutDisplayFunc(displayPolygon);
	myInit();
	glutMainLoop();
	return 1;
}
```

Output:

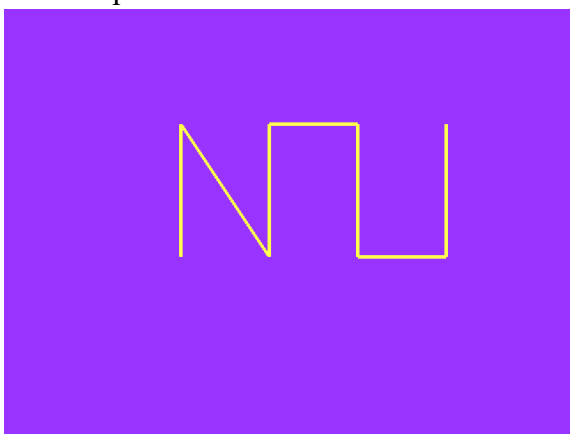
Points:



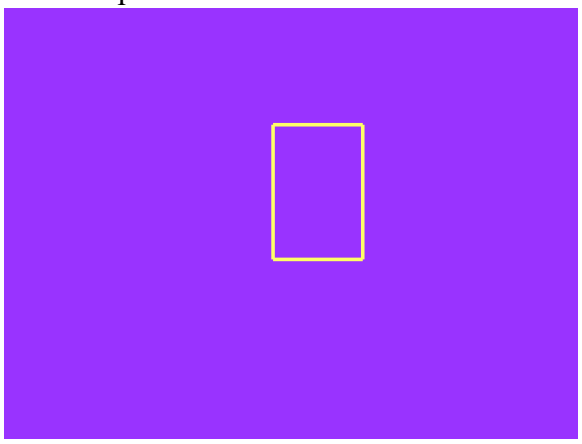
Line:



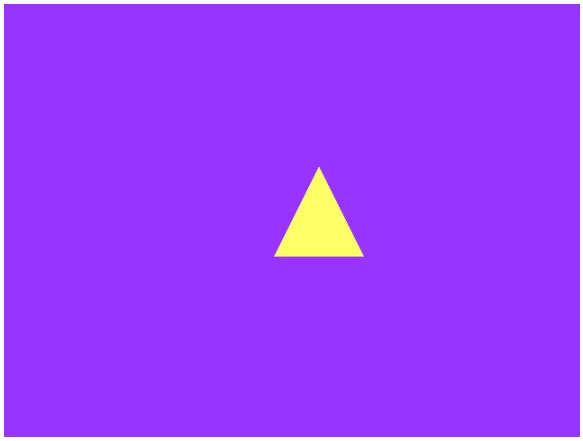
Line Strip:



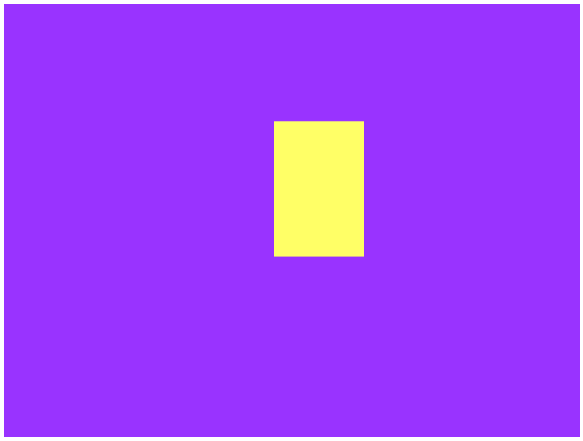
Line Loop:



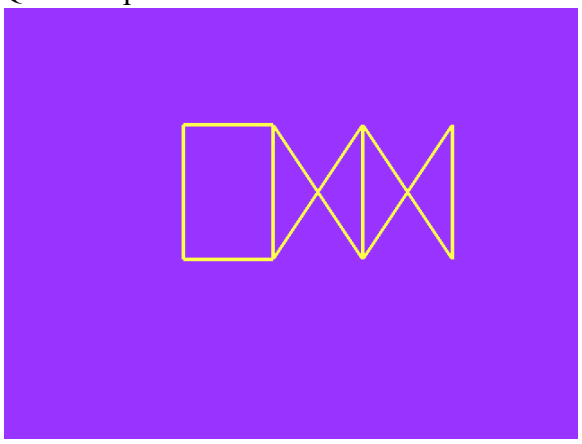
Triangle:



Quadrilateral:

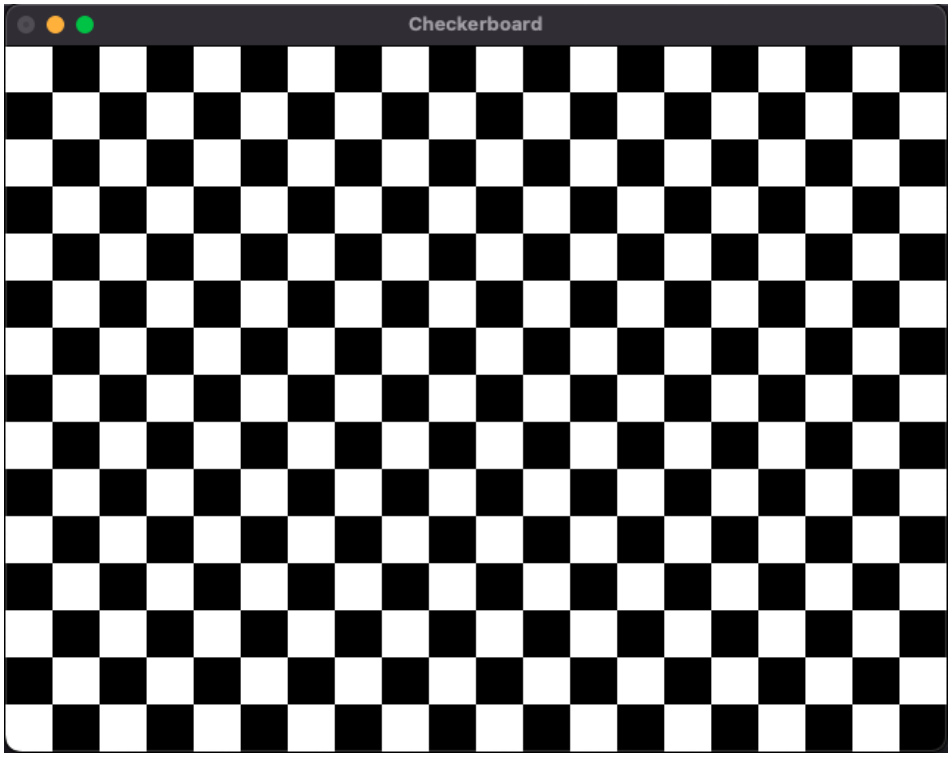


Quad Strip:

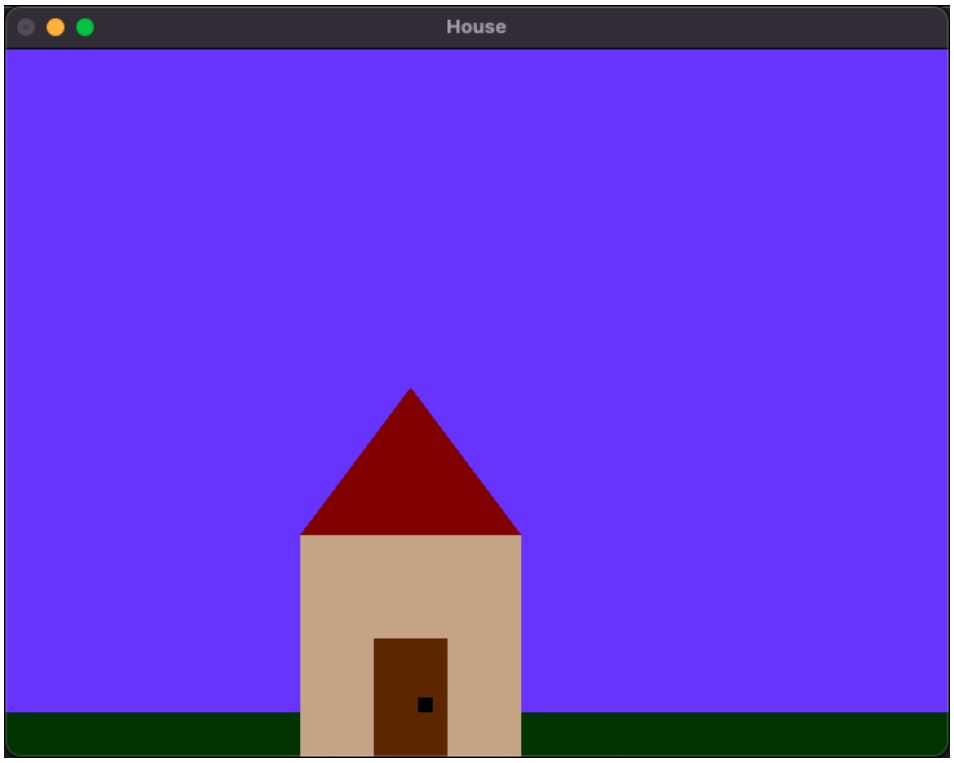


Polygon:

Checkerboard:



House:



Learning outcomes:

- Various shapes were drawn using the OpenGL library.
- Graphics were incorporated into C++ on xcode to display a checkerboard and draw a house.

Exercise 2 – DDA Line Drawing Algorithm

Date: 28/08/2023

Aim:

To plot points that make up the line with endpoints (x0,y0) and (xn,yn) using the DDA line drawing algorithm.

a. Case 1: +ve slope Left to Right line
b. Case 2: +ve slope Right to Left line
c. Case 3: -ve slope Left to Right line
d. Case 4: -ve slope Right to Left line

Each case has two subdivisions:
(i) |m|<= 1 (ii) |m|>1

Algorithm:

1. Input line endpoints, (x1,y1) and (x2, y2)
2. Set pixel at position (x1,y1)
3. Calculate slope m=(y2-y1)/(x2-x1)
4. **For +ve slope (left to right)**
   a. **Case |m|≤1**: Sample at unit x intervals and compute each successive y.
      Repeat the following steps until (x2, y2) is reached:
      yk+1 = yk + m where(m=y/ x)
      xk+1 = xk + 1
      Set pixel at position (xk+1,Round(yk+1))
   b. **Case |m|>1**: Sample at unit y intervals and compute each successive x.
      Repeat the following steps until (x2, y2) is reached:
      xk+1 = xk + 1/m
      yk+1 = yk + 1
      Set pixel at position (Round(xk+1), yk+1)
5. **For +ve slope (right endpoint to left endpoint)**
   a. **Case |m|≤1**: Sample at unit x intervals and compute each successive y.
      Repeat the following steps until (x2, y2) is reached:
      yk+1 = yk - m where(m=y/ x)
      xk+1 = xk - 1
      Set pixel at position (xk+1,Round(yk+1))
   b. **Case |m|>1**: Sample at unit y intervals and compute each successive x.
      Repeat the following steps until (x2, y2) is reached:
      xk+1 = xk - 1/m
      yk+1 = yk - 1

set pixel at position (Round(xk+1), yk+1)

6. **For -ve slope (left to right)**
   a. **Case |m|≤1**: Sample at unit x intervals and compute each successive y.
   Repeat the following steps until (x2, y2) is reached:
   yk+1 = yk - m where(m=y/ x)
   xk+1 = xk + 1
   Set pixel at position (xk+1,Round(yk+1))
   b. **Case |m|>1**: Sample at unit y intervals and compute each successive x.
   Repeat the following steps until (x2, y2) is reached:
   xk+1 = xk + 1/m
   yk+1 = yk - 1
   Set pixel at position (Round(xk+1), yk+1)
7. **For -ve slope (right endpoint to left endpoint)**
   a. **Case |m|≤1**: Sample at unit x intervals and compute each successive y.
   Repeat the following steps until (x2, y2) is reached:
   yk+1 = yk + m where(m=y/ x)
   xk+1 = xk - 1
   Set pixel at position (xk+1,Round(yk+1))
   b. **Case |m|>1**: Sample at unit y intervals and compute each successive x.
   Repeat the following steps until (x2, y2) is reached:
   xk+1 = xk - 1/m
   yk+1 = yk + 1
   set pixel at position (Round(xk+1), yk+1)

Code:

```
#define GL_SILENCE_DEPRECATION

#include<GLUT/glut.h>
#include<iostream>
#include<math.h>
using namespace std;

void myInit() {
	glClearColor(0.6,0.2,1.0,0.0);
	//glClearColor(0.4,0.2,1.0,0.0);
	glColor3f(0.0f,0.0f,0.5f);
	//glPointSize(10);
	glMatrixMode(GL_PROJECTION);
```

```
	//glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
	glLineWidth(2);
	glLoadIdentity();
	gluOrtho2D(0.0,640.0,0.0,480.0);
}

void displayPoint(int x, int y) {
	glBegin(GL_POINTS);
	glVertex2d(x+320,y+240);
	glEnd();
}

void displayLine(int x1, int y1, int x2, int y2) {
	glBegin(GL_LINES);
	glColor4f(1,1,0.4,1);
	glVertex2d(x1+320,y1+240);
	glVertex2d(x2+320,y2+240);
	glEnd();
}

void drawPlane() {
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_LINES);
	glColor4f(1,1,1,1);
	//y-axis
	displayLine(0,-240,0,240);
	//x-axis
	displayLine(-320,0,320,0);
	glEnd();
	glFlush();
}

void plotInput() {
	glClear(GL_COLOR_BUFFER_BIT);

	int x1, y1, x2, y2;
	float m;
	int dir;

	cout << "Enter start coords: ";
	cin >> x1 >> y1;
	cout << "Enter end coords: ";
	cin >> x2 >> y2;
	 cout << endl;

	drawPlane();
```

```
m = float(y2 - y1)/float(x2 - x1);
cout << "Slope: " << m << endl;
dir = x1 <= x2 ? 1 : 2;
if (dir == 1) {
cout << "Direction: L to R" << endl;
}
else {
cout << "Direction: R to L" << endl;
}

if (m >= 0) {
if (dir == 1) {
	if (abs(m) <= 1) {
	float y = y1;
	cout << "x " << "y " << "round(y)" << endl;
	for (int x = x1; x <= x2; x++) {
		y += m;
		cout << x << " " << y << " " << round(y) << endl;
		displayPoint(x, round(y));
	}
	}
	else {
	float x = x1;
	cout << "x " << "round(x)" << "y " << endl;
	for (int y = y1; y <= y2; y++) {
		x += (1/m);
		cout << x << " " << round(x) << " " << y << endl;
		displayPoint(round(x), y);
	}
	}
}
else {
	if (abs(m) <= 1) {
	float y = y1;
	cout << "x " << "y " << "round(y)" << endl;
	for (int x = x1; x >= x2; x--) {
		y -= m;
		cout << x << " " << y << " " << round(y) << endl;
		displayPoint(x, round(y));
	}
	}
	else {
	float x = x1;
	cout << "x " << "round(x)" << "y " << endl;
	for (int y = y1; y >= y2; y--) {
```

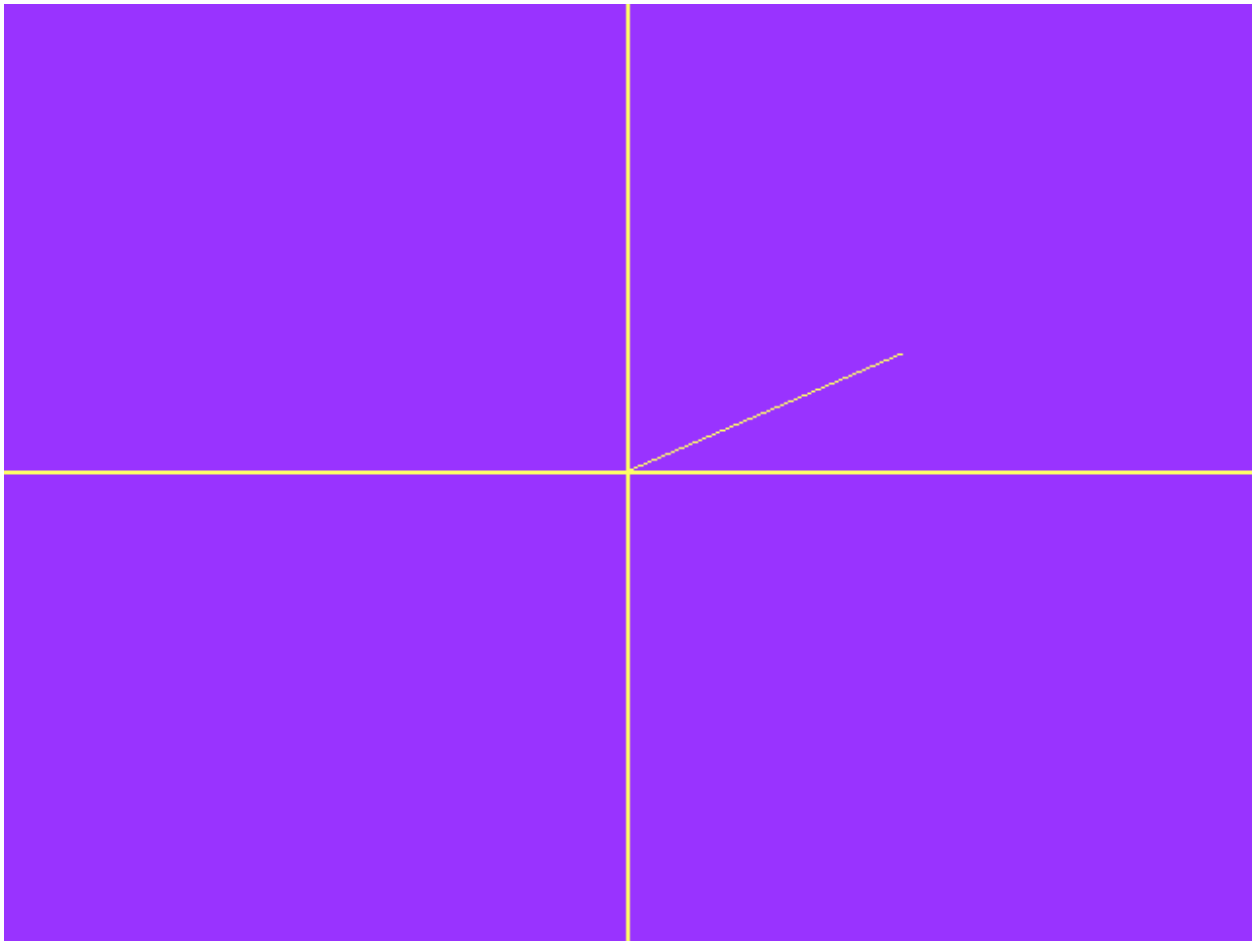
```
                x -= (1/m);
                cout << x << " " << round(x) << " " << y << endl;
                displayPoint(round(x), y);
        }
        }
}
}
else {
m = abs(m);
if (dir == 1) {
        if (abs(m) <= 1) {
        float y = y1;
        cout << "x " << "y " << "round(y)" << endl;
        for (int x = x1; x <= x2; x++) {
                y -= m;
                cout << x << " " << y << " " << round(y) << endl;
                displayPoint(x, round(y));
        }
        }
        else {
        float x = x1;
        cout << "x " << "round(x)" << "y " << endl;
        for (int y = y1; y >= y2; y--) {
                x += (1/m);
                cout << x << " " << round(x) << " " << y << endl;
                displayPoint(round(x), y);
        }
        }
}
else {
        if (abs(m) <= 1) {
        float y = y1;
        cout << "x " << "y " << "round(y)" << endl;
        for (int x = x1; x >= x2; x--) {
                y += m;
                cout << x << " " << y << " " << round(y) << endl;
                displayPoint(x, round(y));
        }
        }
        else {
        float x = x1;
        cout << "x " << "round(x)" << "y " << endl;
        for (int y = y1; y <= y2; y++) {
                x -= (1/m);
                cout << x << " " << round(x) << " " << y << endl;
                displayPoint(round(x), y);
```

```
                }
                }
        }
        }

        glFlush();
}

int main(int argc,char* argv[]) {
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
        glutInitWindowSize(640,480);
        glutCreateWindow("DDA Line Drawing Algorithm");
        glutDisplayFunc(plotInput);
        myInit();
        glutMainLoop();
        return 1;
}
```

Output:

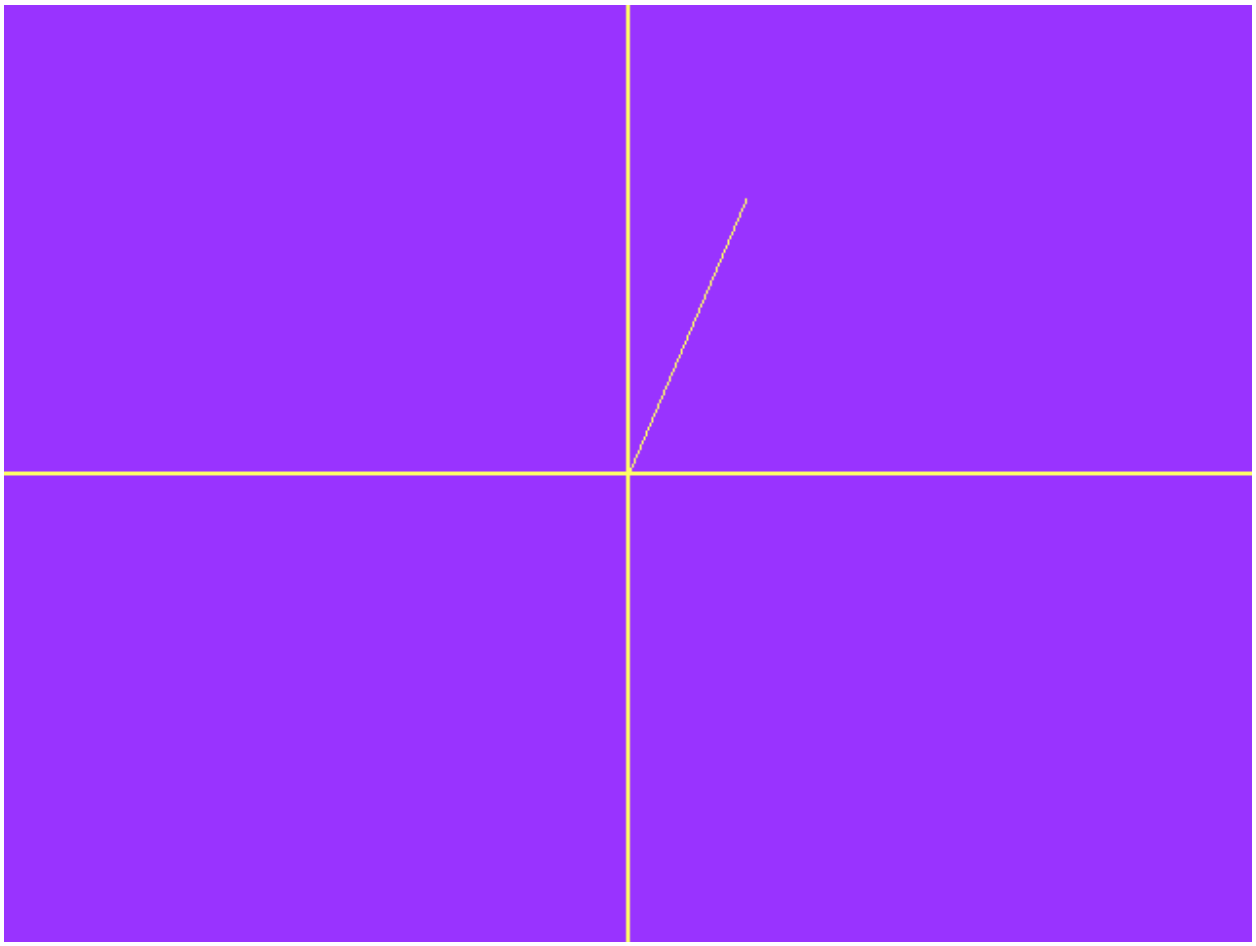
Case 1: +ve slope, L to R, |m|<=1

```
Enter start coords: 0 0
Enter end coords: 140 60

Slope: 0.428571
Direction: L to R
x y round(y)
```



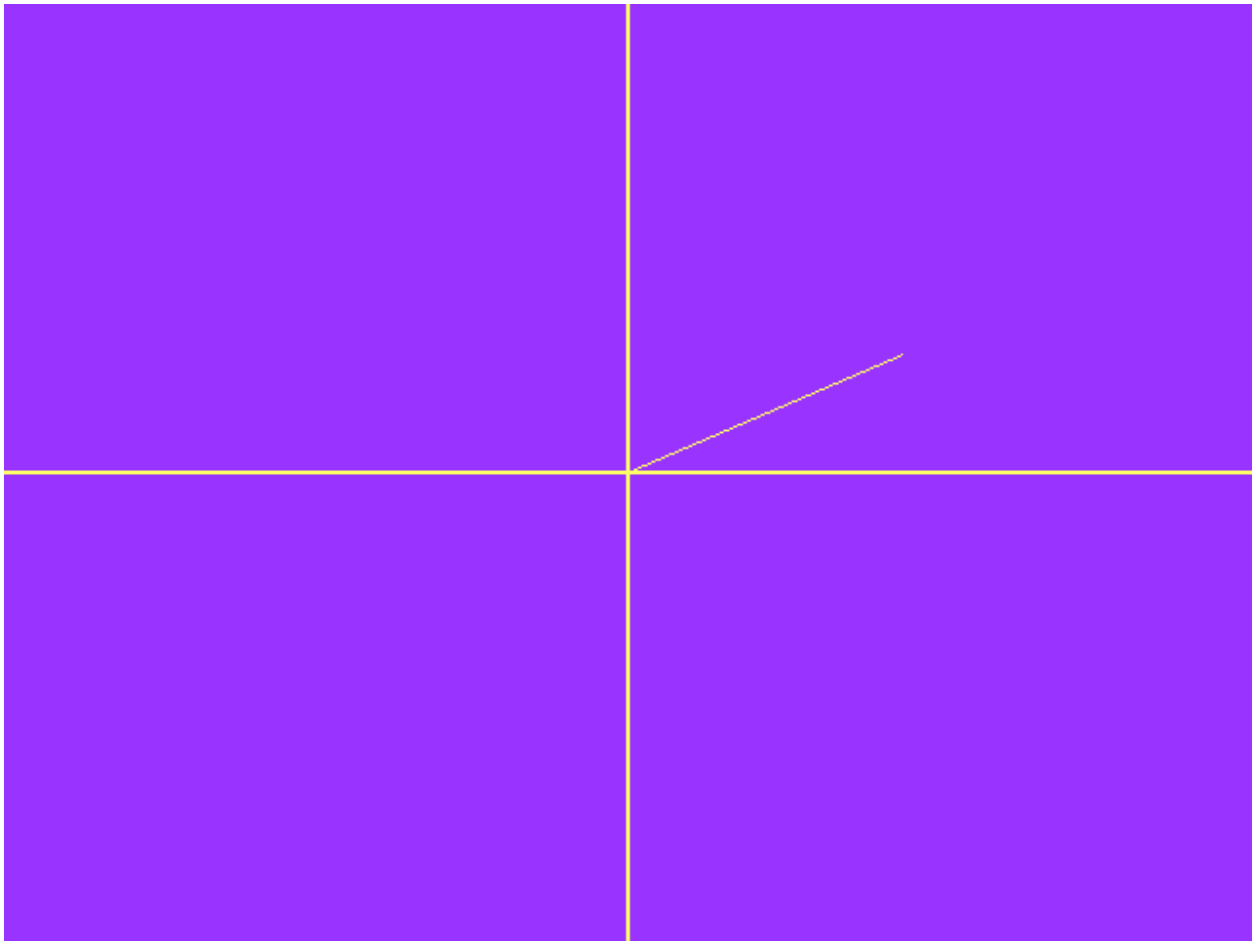
Case 2: +ve slope, L to R, |m|>1

```
Enter start coords: 0 0
Enter end coords: 60 140

Slope: 2.33333
Direction: L to R
x round(x)y
```



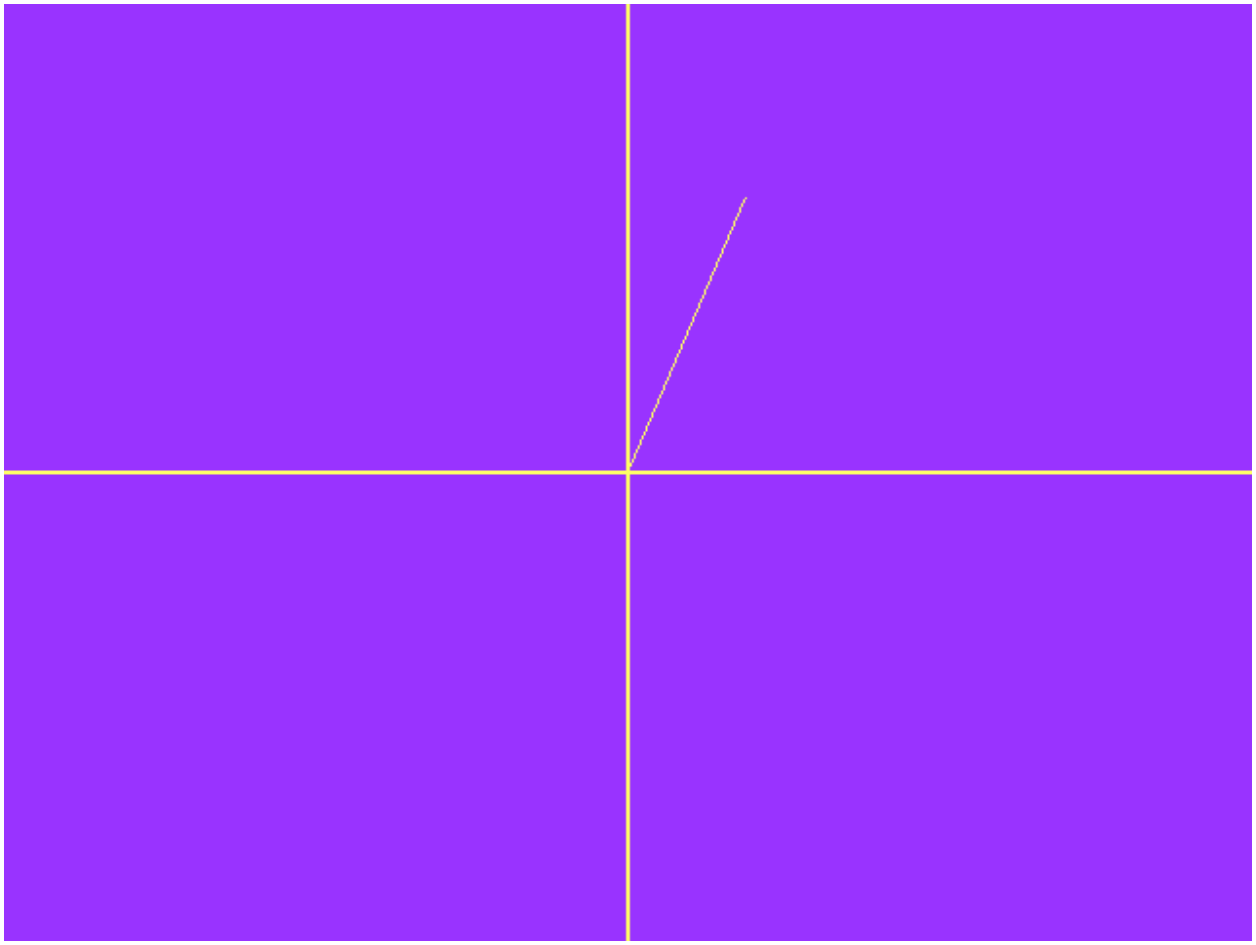
Case 3: +ve slope, R to L, |m|<=1

```
Enter start coords: 140 60
Enter end coords: 0 0

Slope: 0.428571
Direction: R to L
x y round(y)
```



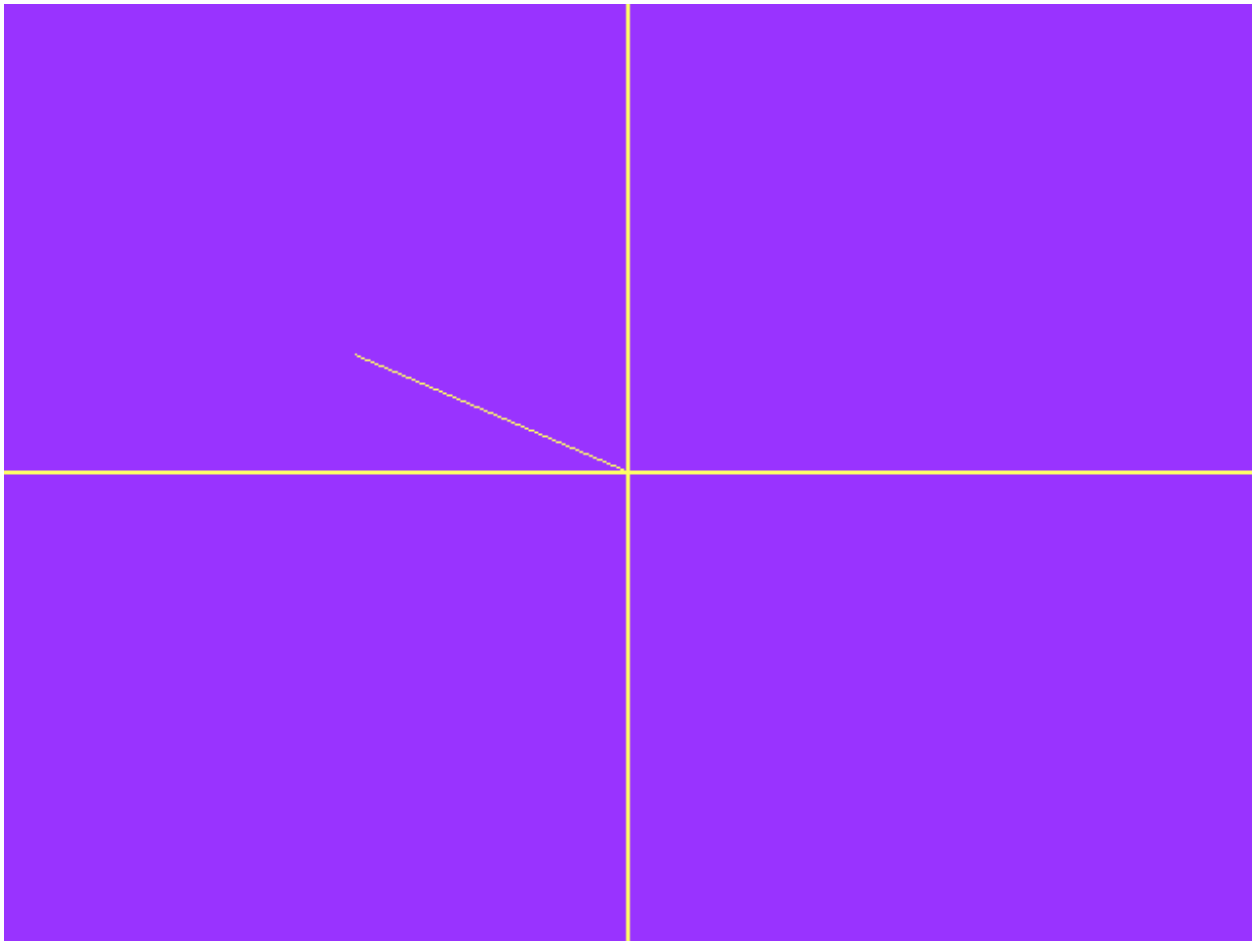
Case 4: +ve slope, R to L, |m|>1

```
Enter start coords: 60 140
Enter end coords: 0 0

Slope: 2.33333
Direction: R to L
x round(x)y
```



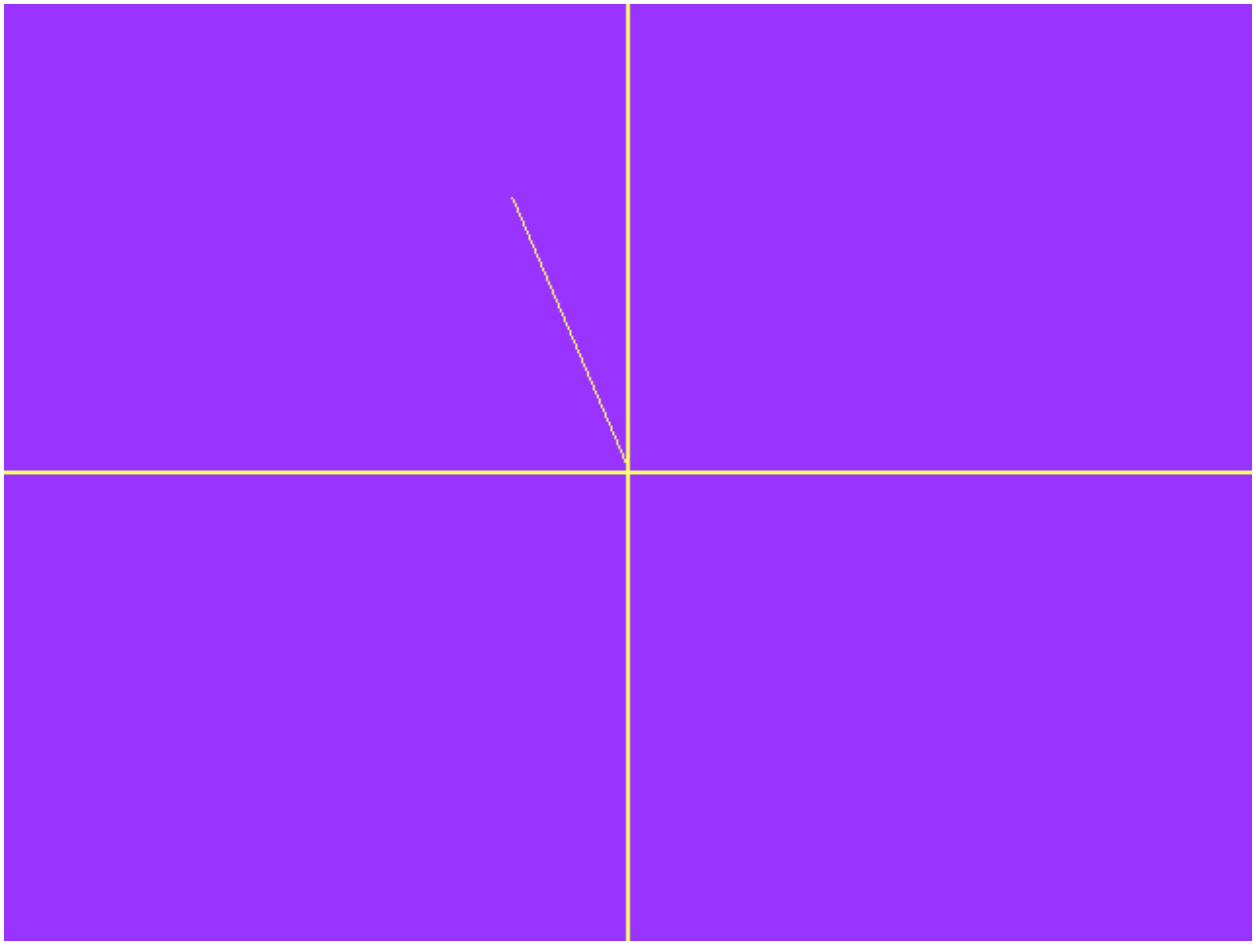
Case 5: -ve slope, L to R, |m|<=1

```
Enter start coords: -140 60
Enter end coords: 0 0

Slope: -0.428571
Direction: L to R
x y round(y)
```



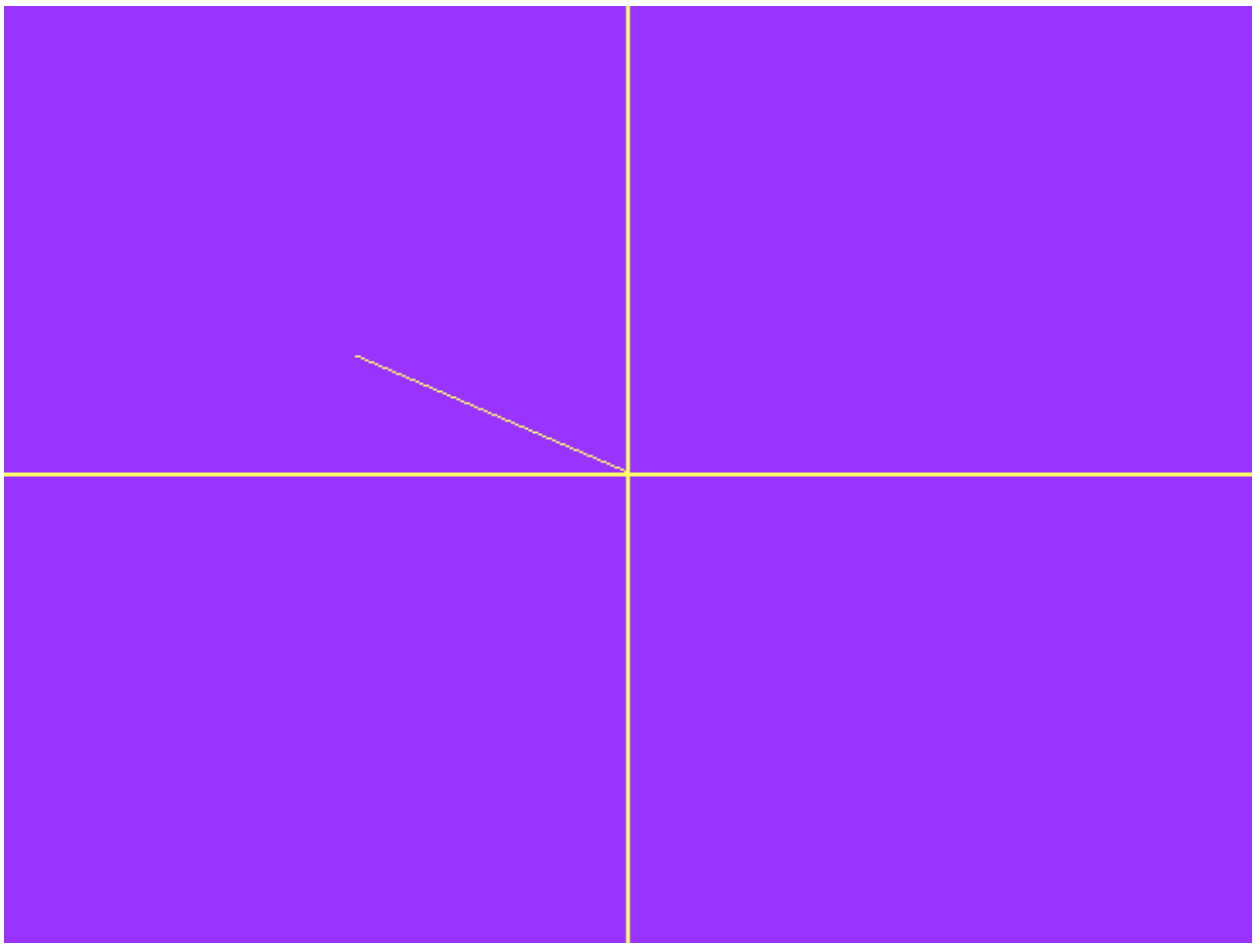
Case 6: -ve slope, L to R, |m|>1

```
Enter start coords: -60 140
Enter end coords: 0 0

Slope: -2.33333
Direction: L to R
x round(x)y
```



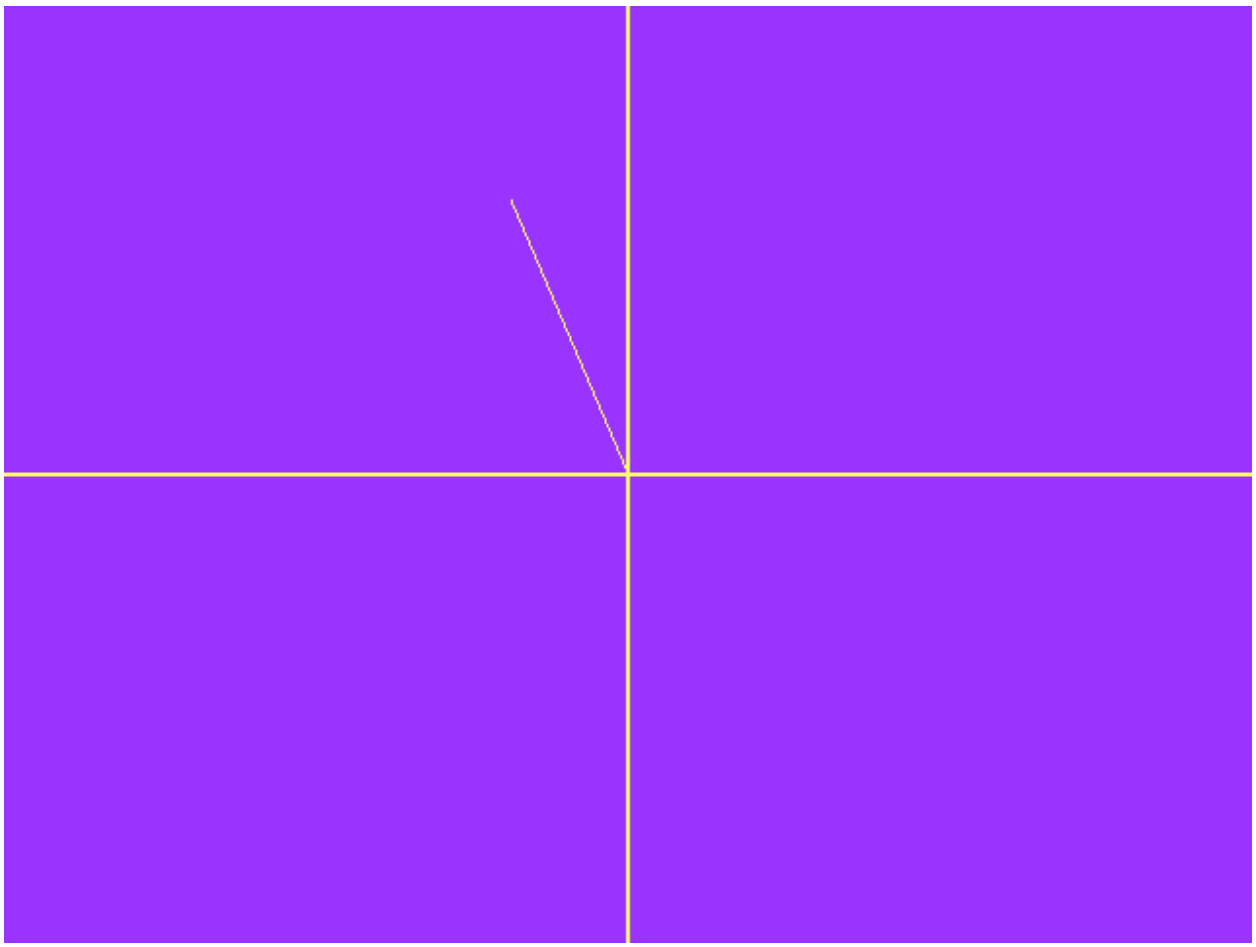
Case 7: -ve slope, R to L, |m|<=1

```
Enter start coords: 0 0
Enter end coords: -140 60

Slope: -0.428571
Direction: R to L
x y round(y)
```



Case 8: -ve slope, R to L, |m|>1

```
Enter start coords: 0 0
Enter end coords: -60 140

Slope: -2.33333
Direction: R to L
x round(x)y
```



Learning outcomes:

- The DDA Line Drawing Algorithm was implemented.
- The implementation for all 8 cases of the algorithm was tested.

Exercise 3 – Bresenham Line Drawing Algorithm

Date: 28/08/2023

Aim:

To plot points that make up the line with endpoints (x0,y0) and (xn,yn) using the Bresenham line drawing algorithm.

e. Case 1: +ve slope Left to Right line
f. Case 2: +ve slope Right to Left line
g. Case 3: -ve slope Left to Right line
h. Case 4: -ve slope Right to Left line

Each case has two subdivisions:
(i) |m|<= 1 (ii) |m|>1

Algorithm:

1. Input line endpoints, (x1,y1) and (x2, y2)
2. Set pixel at position (x1,y1)
3. Calculate dy=(y2-y1) and dx=(x2-x1)
4. Calculate slope m=dy/dx
5. Calculate parameter P = (2 * dy) - dx;
6. Set x = x1, y = y1
7. If P < 0
    a. Plot (x + 1, y)
    b. P = P + 2 * dy
8. Else if P >= 0
    a. Plot (x + 1, y + 1)
    b. P = P + 2 * dy - 2 * dx
9. Repeat until x = x2
10. Modify for all cases accordingly

Code:

```
#define GL_SILENCE_DEPRECATION

#include<GL/glut.h>
#include<iostream>
#include<math.h>
using namespace std;
```

```
void myInit() {
    glClearColor(0.6, 0.2, 1.0, 0.0);
    //glClearColor(0.4,0.2,1.0,0.0);
    glColor3f(0.0f, 0.0f, 0.5f);
    //glPointSize(10);
    glMatrixMode(GL_PROJECTION);
    //glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glLineWidth(2);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}

void displayPoint(int x, int y) {
    glBegin(GL_POINTS);
    glVertex2d(x + 320, y + 240);
    glEnd();
}

void displayLine(int x1, int y1, int x2, int y2) {
    glBegin(GL_LINES);
    glColor4f(1, 1, 0.4, 1);
    glVertex2d(x1 + 320, y1 + 240);
    glVertex2d(x2 + 320, y2 + 240);
    glEnd();
}

void drawPlane() {
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
    glColor4f(1, 1, 1, 1);
    //y-axis
    displayLine(0, -240, 0, 240);
    //x-axis
    displayLine(-320, 0, 320, 0);
    glEnd();
    glFlush();
}

void plotBresenhamLine() {
    int x1, y1, x2, y2;
    float dx, dy, m;
    int dir;
    float P;
```

```
cout << "Enter start coords: ";
 cin >> x1 >> y1;
 cout << "Enter end coords: ";
 cin >> x2 >> y2;
 cout << endl;

 dx = float(x2 - x1);
 dy = float(y2 - y1);
 m = dy / dx;
 cout << "Slope: " << m << endl;
 if (abs(m) > 1) {
     int temp1 = x1, temp2 = x2;
     x1 = y1; x2 = y2;
     y1 = temp1; y2 = temp2;

     int temp = dx;
     dx = dy; dy = temp;
 }

 dir = x1 <= x2 ? 1 : 2;
 if (dir == 1) {
     cout << "Direction: L to R" << endl;
 }
 else {
     cout << "Direction: R to L" << endl;
     int tempX = x1, tempY = y1;
     x1 = x2; y1 = y2;
     x2 = tempX; y2 = tempY;

     dx = -dx; dy = -dy;
 }

 cout << "x y\n";
 cout << x1 << " " << y1 << endl;

 if (m >= 0) {
     P = (2 * dy) - dx;
     displayPoint(x1, y1);

     int y = y1;
     for (int x = x1 + 1; x <= x2; x++) {
         if (P < 0) {
             P += 2 * dy;
         }
         else {
             y++;
```

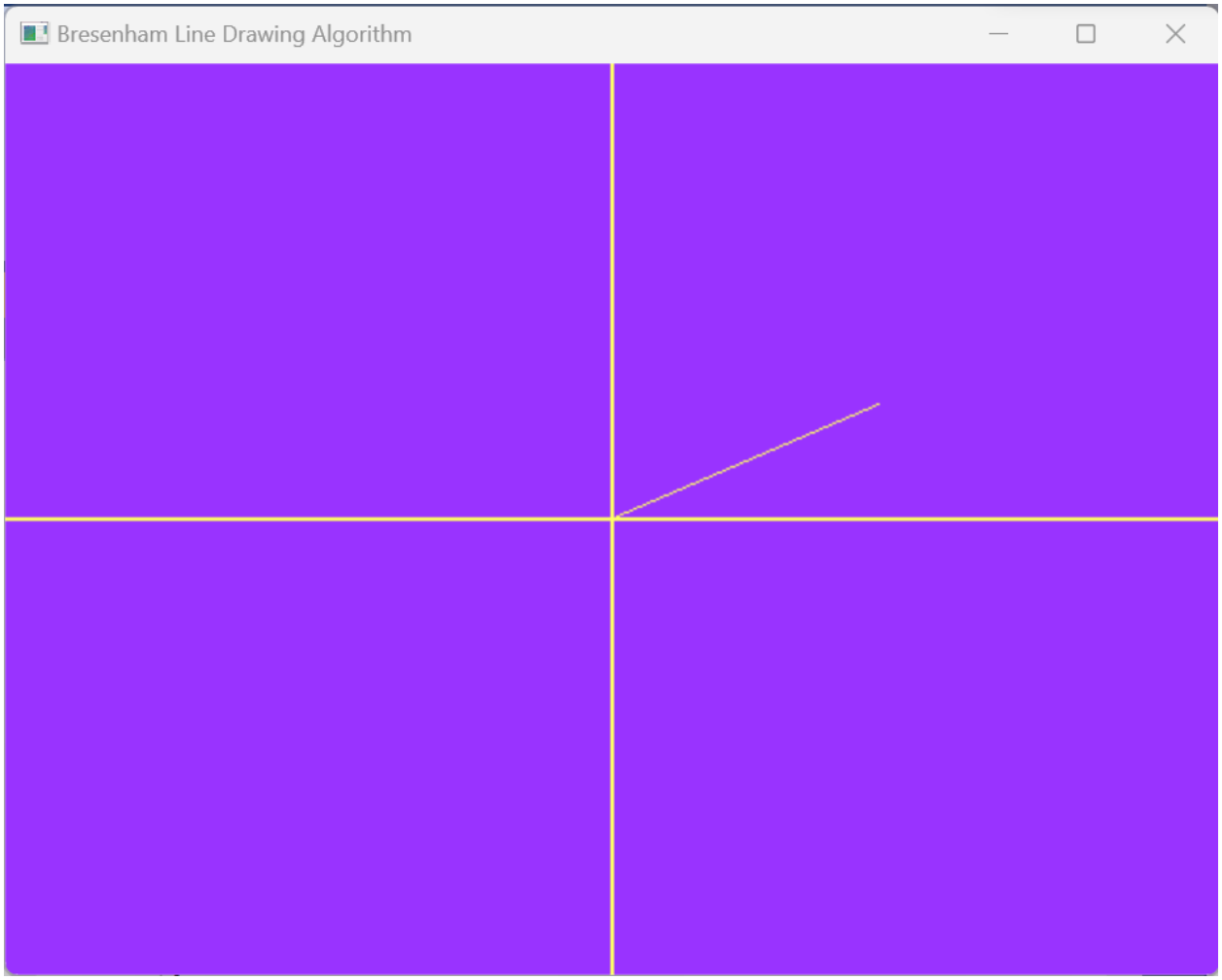
```
                P += 2 * (dy - dx);
            }
            if (abs(m) <= 1) {
                displayPoint(x, y);
                cout << x << " " << y << endl;
            }
            else {
                displayPoint(y, x);
                cout << y << " " << x << endl;
            }
        }
    }
    else {
        dx = abs(dx); dy = abs(dy);

        P = (2 * dy) - dx;
        displayPoint(x1, y1);

        int y = y1;
        for (int x = x1 + 1; x <= x2; x++) {
            if (P < 0) {
                P += 2 * dy;
            }
            else {
                y--;
                P += 2 * (dy - dx);
            }
            if (abs(m) <= 1) {
                displayPoint(x, y);
                cout << x << " " << y << endl;
            }
            else {
                displayPoint(y, x);
                cout << y << " " << x << endl;
            }
        }
    }
}

void plotChart() {
    glClear(GL_COLOR_BUFFER_BIT);

    drawPlane();
    plotBresenhamLine();

    glFlush();
```

```
}

int main(int argc, char* argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowSize(640, 480);
    glutCreateWindow("Bresenham Line Drawing Algorithm");
    glutDisplayFunc(plotChart);
    myInit();
    glutMainLoop();
    return 1;
}
```

Output:

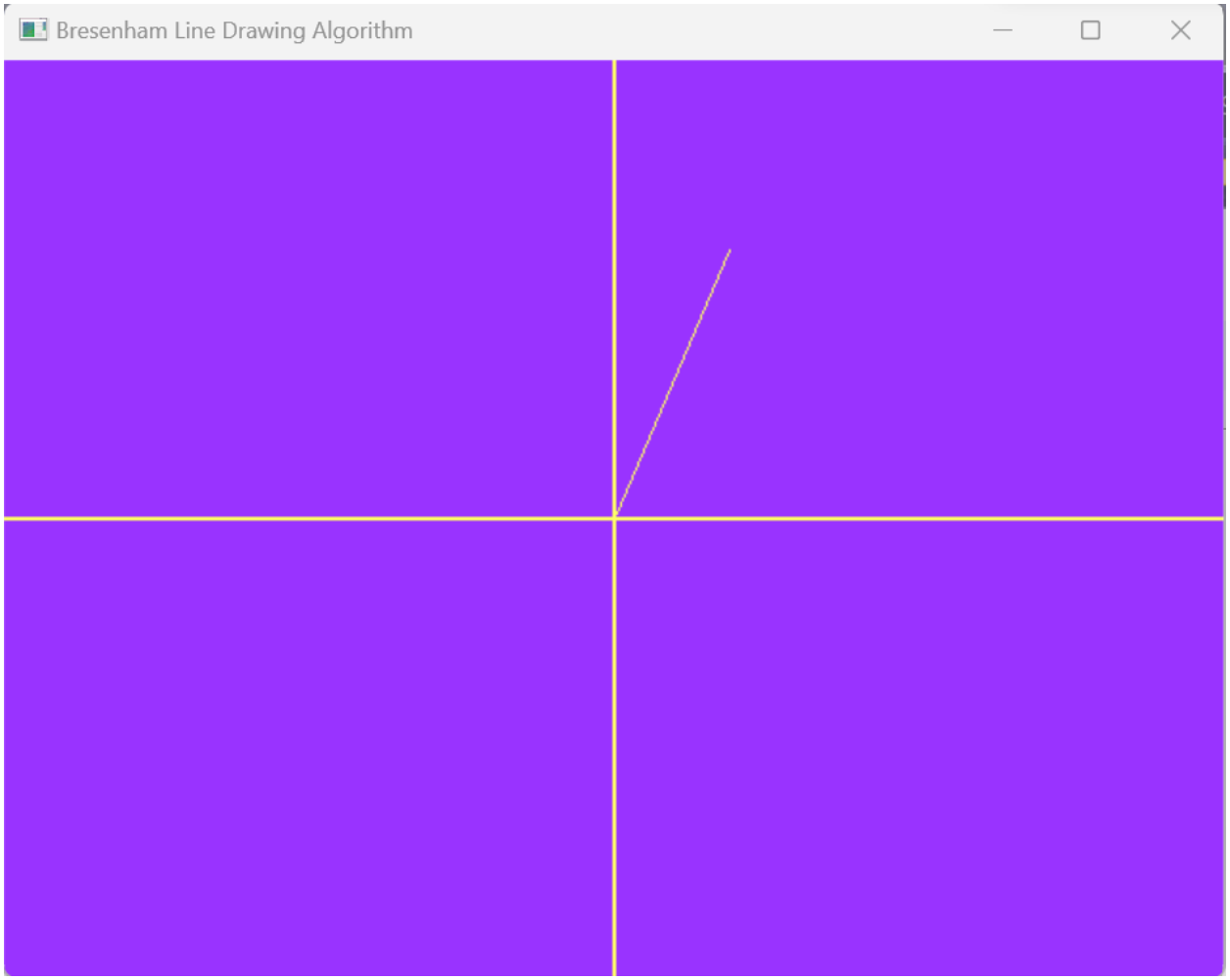
Case 1: +ve slope, L to R, |m|<=1

```
E:\SSN\Sem 7\Graphics and M
Enter start coords: 0 0
Enter end coords: 140 60

Slope: 0.428571
Direction: L to R
x y
0 0
1 0
2 1
3 1
4 2
5 2
6 3
7 3
8 3
9 4
```

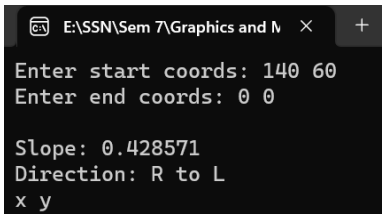


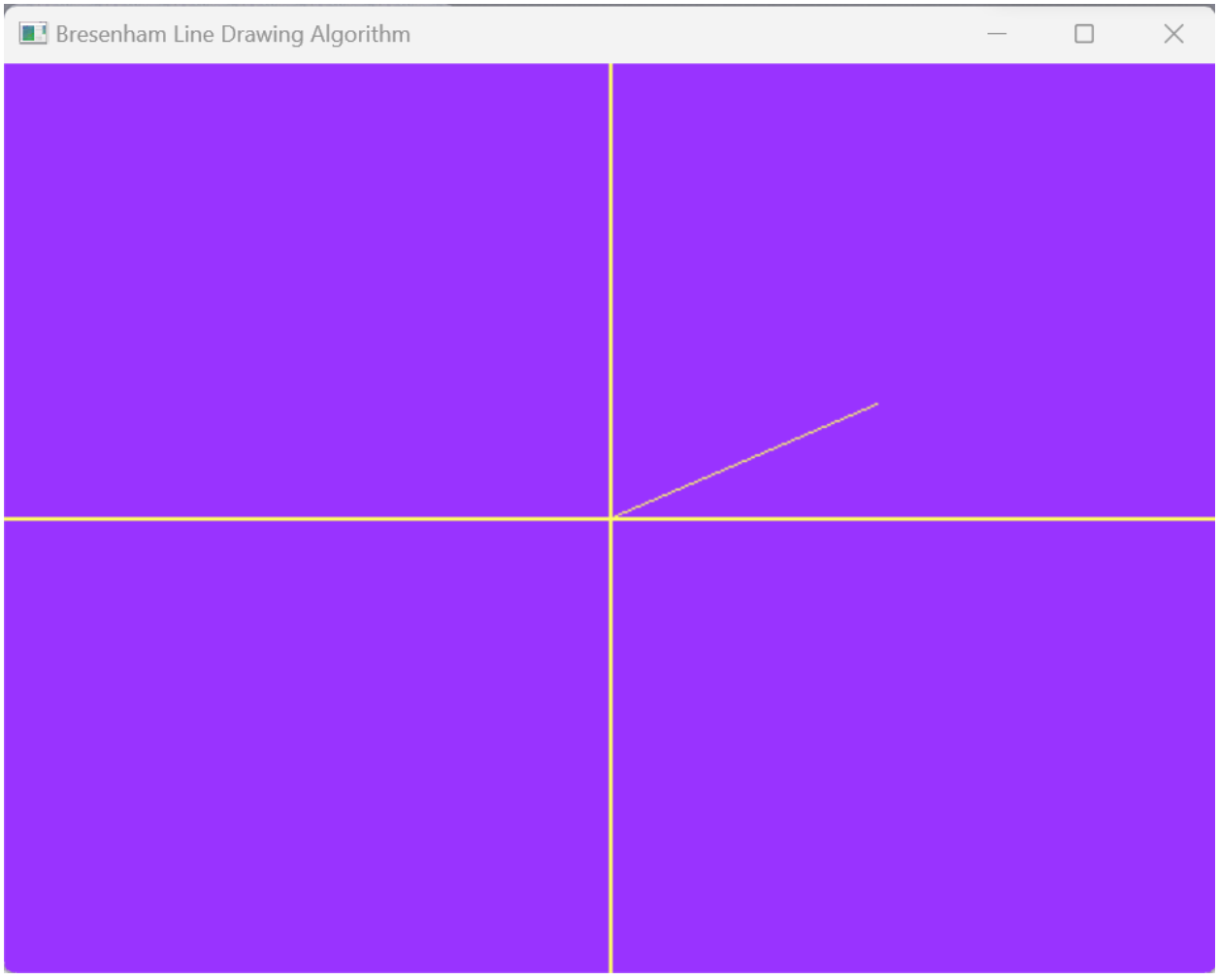
Case 2: +ve slope, L to R, |m|>1

```
E:\SSN\Sem 7\Graphics and M
Enter start coords: 0 0
Enter end coords: 60 140

Slope: 2.33333
Direction: L to R
x y
0 0
0 1
1 2
1 3
2 4
2 5
3 6
3 7
3 8
4 9
```

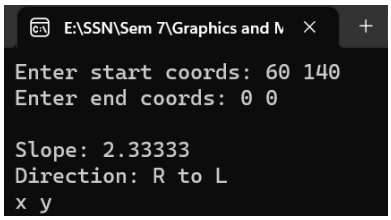


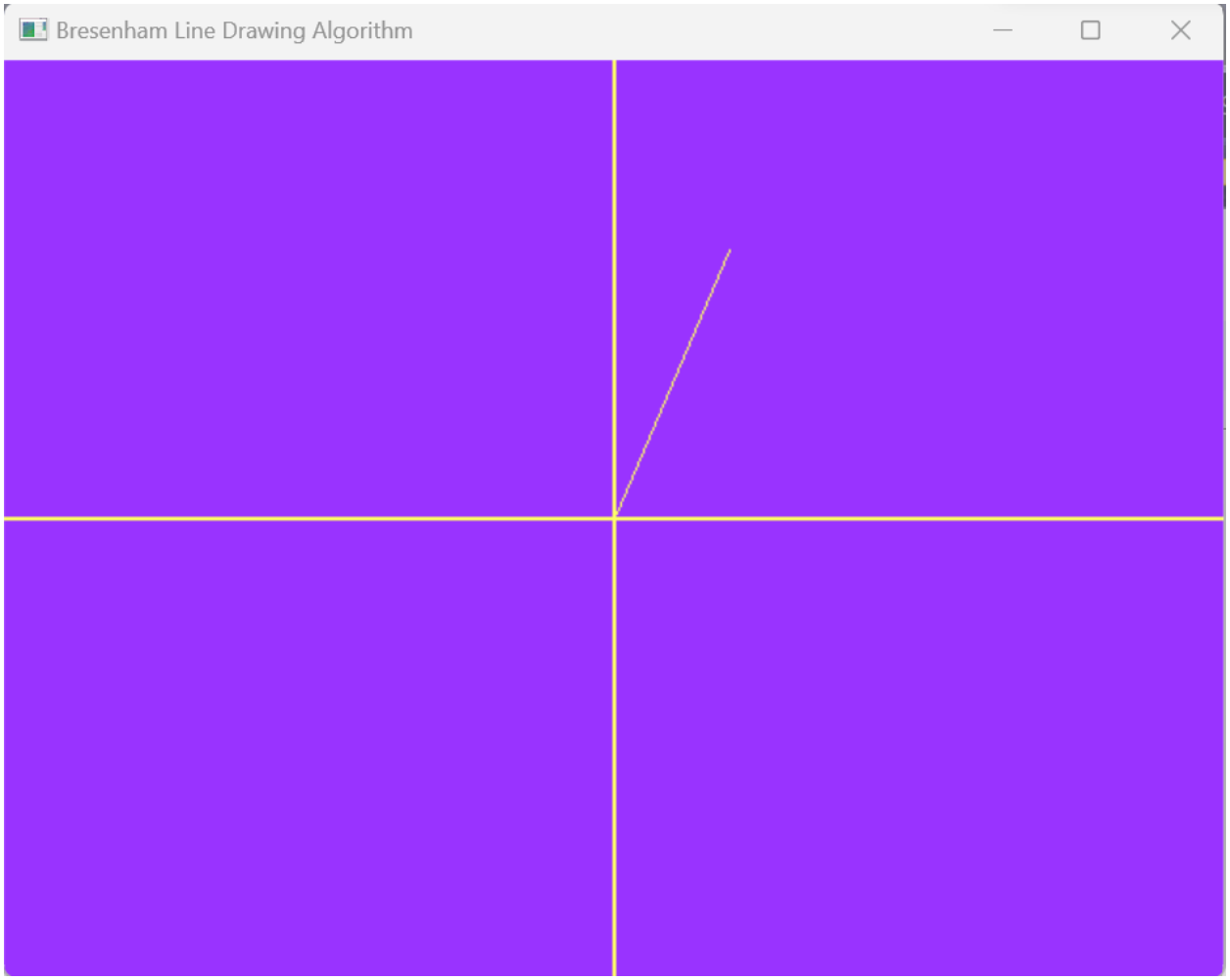
Case 3: +ve slope, R to L, |m|<=1





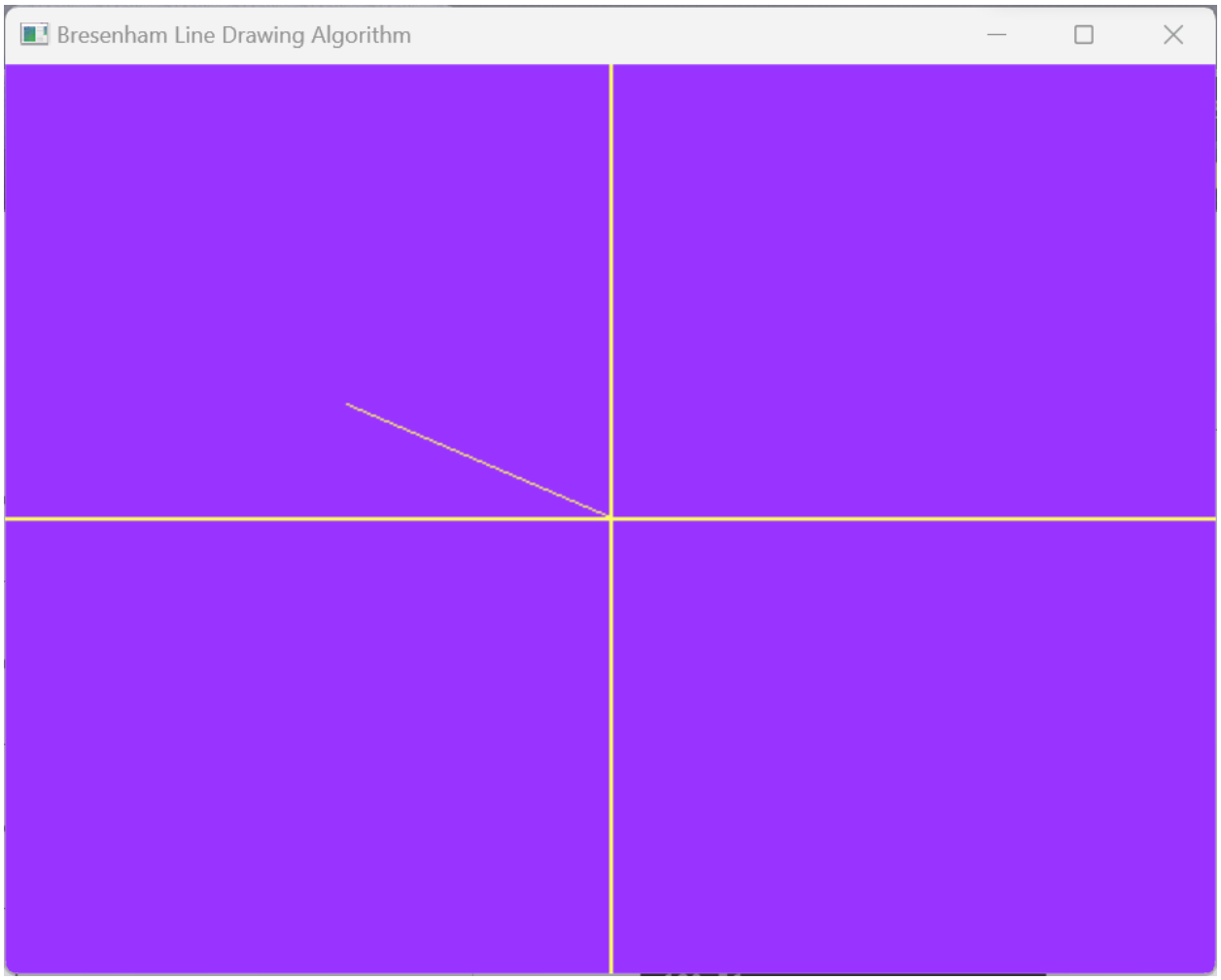
Case 4: +ve slope, R to L, |m|>1



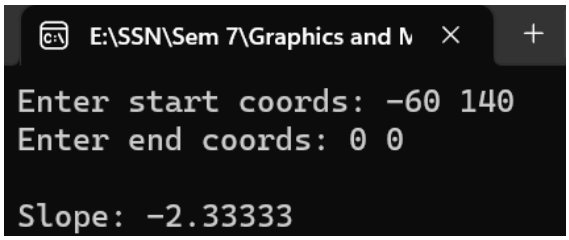


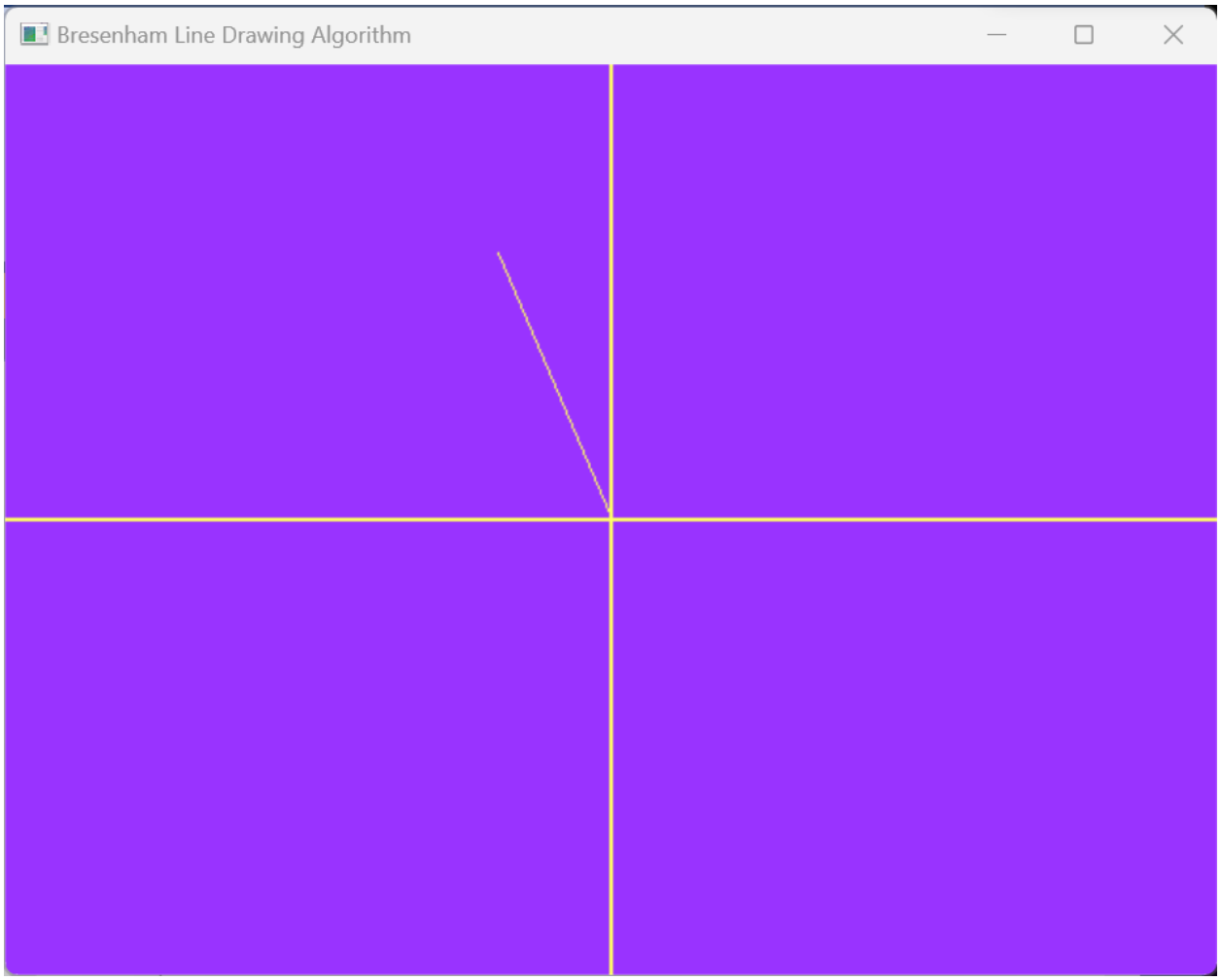
Case 5: -ve slope, L to R, |m|<=1

```
E:\SSN\Sem 7\Graphics and M
Enter start coords: -140 60
Enter end coords: 0 0

Slope: -0.428571
Direction: L to R
x y
-140 60
-139 60
-138 59
-137 59
-136 58
-135 58
-134 57
-133 57
-132 57
-131 56
-130 56
```

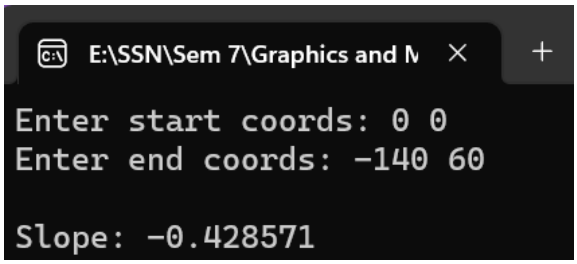


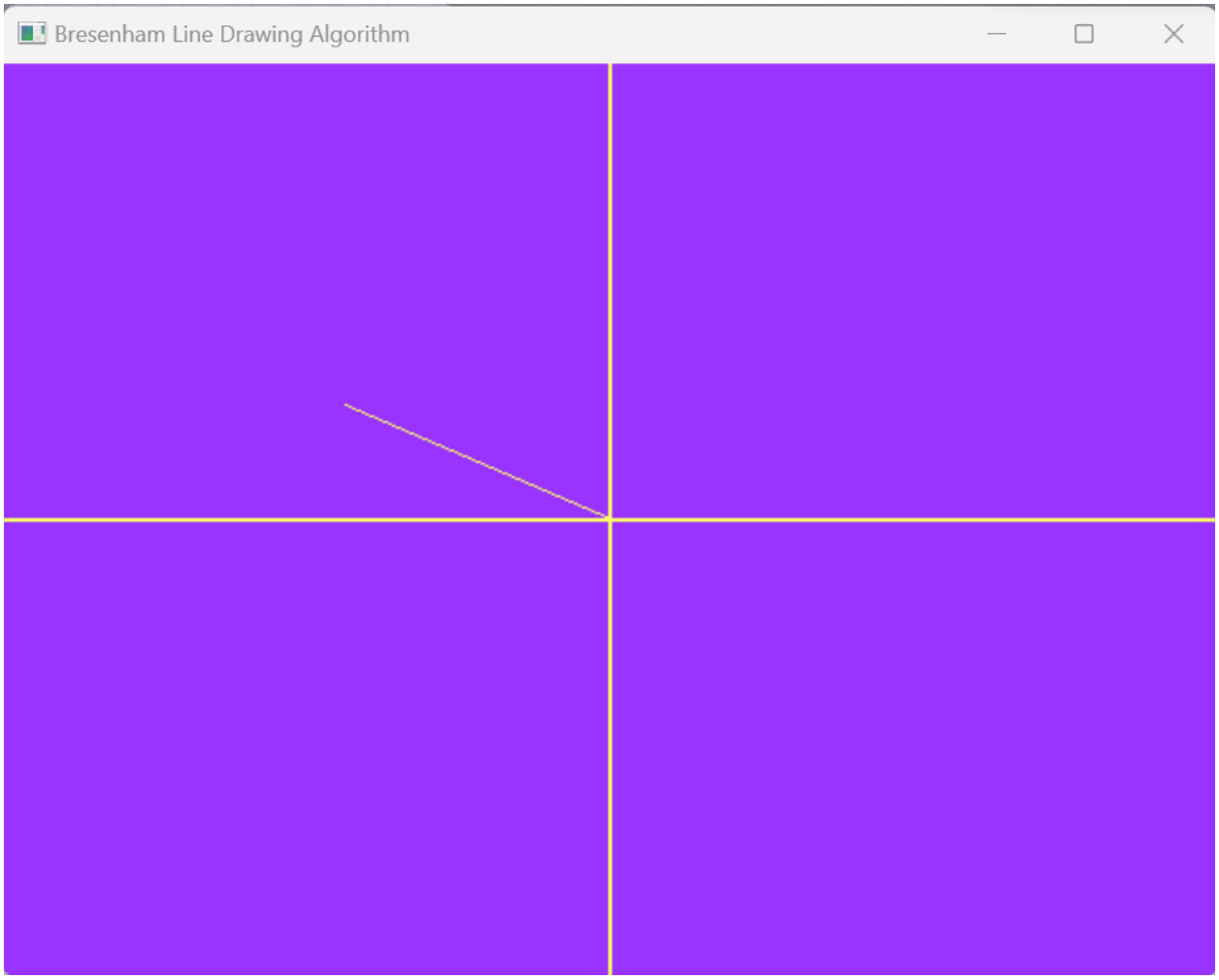
Case 6: -ve slope, L to R, |m|>1



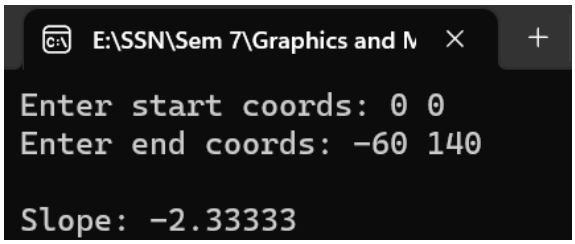


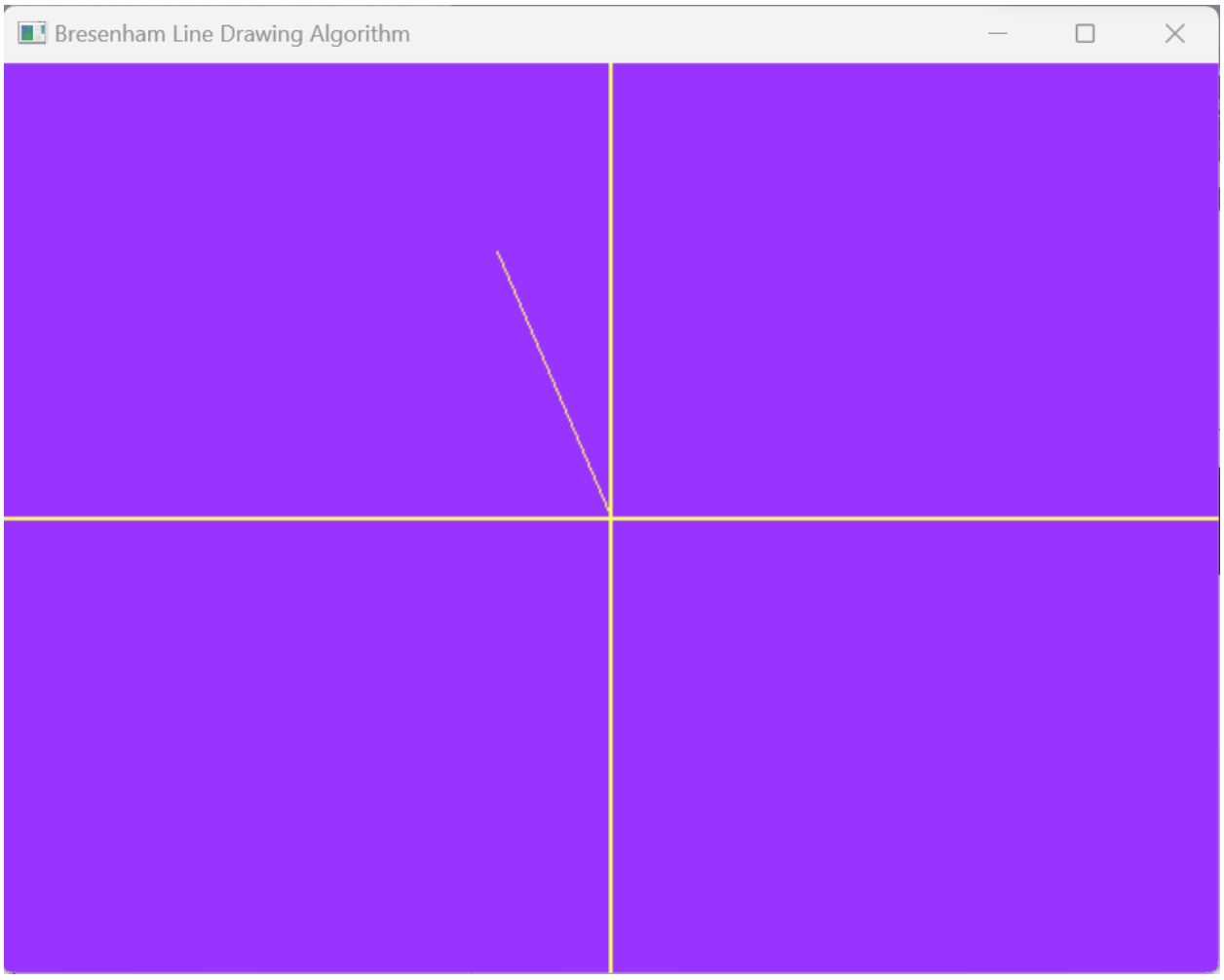
Case 7: -ve slope, R to L, |m|<=1





Case 8: -ve slope, R to L, |m|>1





Learning outcomes:

- The Bresenham Line Drawing Algorithm was implemented.
- The implementation for all 8 cases of the algorithm was tested.

Exercise 4 – Midpoint Circle Drawing Algorithm

Date: 11/09/2023

Aim:

To plot points that make up the circle with center (xc,yc) and radius r using the Midpoint circle drawing algorithm. Give atleast 2 test cases.

Case 1: With center (0,0)
Case 2: With center (xc,yc)

Algorithm:

1. Input circle center (xc, yc) and radius r.
2. Plot the first point (x, y) = (0, r) and its corresponding point in each quadrant.
3. Compute P = 1 - r.
4. Repeat until x >= y:
    a. Increment x = x + 1
    b. If P < 0,
        i. Assign P = P + 2x + 1
    Else if P >= 0,
        i. Decrement y = y - 1
        ii. Assign P = P + 2x - 2y + 1
    c. Plot (x, y) and its corresponding point in each octet.

Code:

```
#define GL_SILENCE_DEPRECATION

#include<GL/glut.h>
#include<iostream>
#include<math.h>
using namespace std;

void myInit() {
    glClearColor(0.6, 0.2, 1.0, 0.0);
    glColor3f(0.0f, 0.0f, 0.5f);
    glMatrixMode(GL_PROJECTION);
    glLineWidth(2);
```

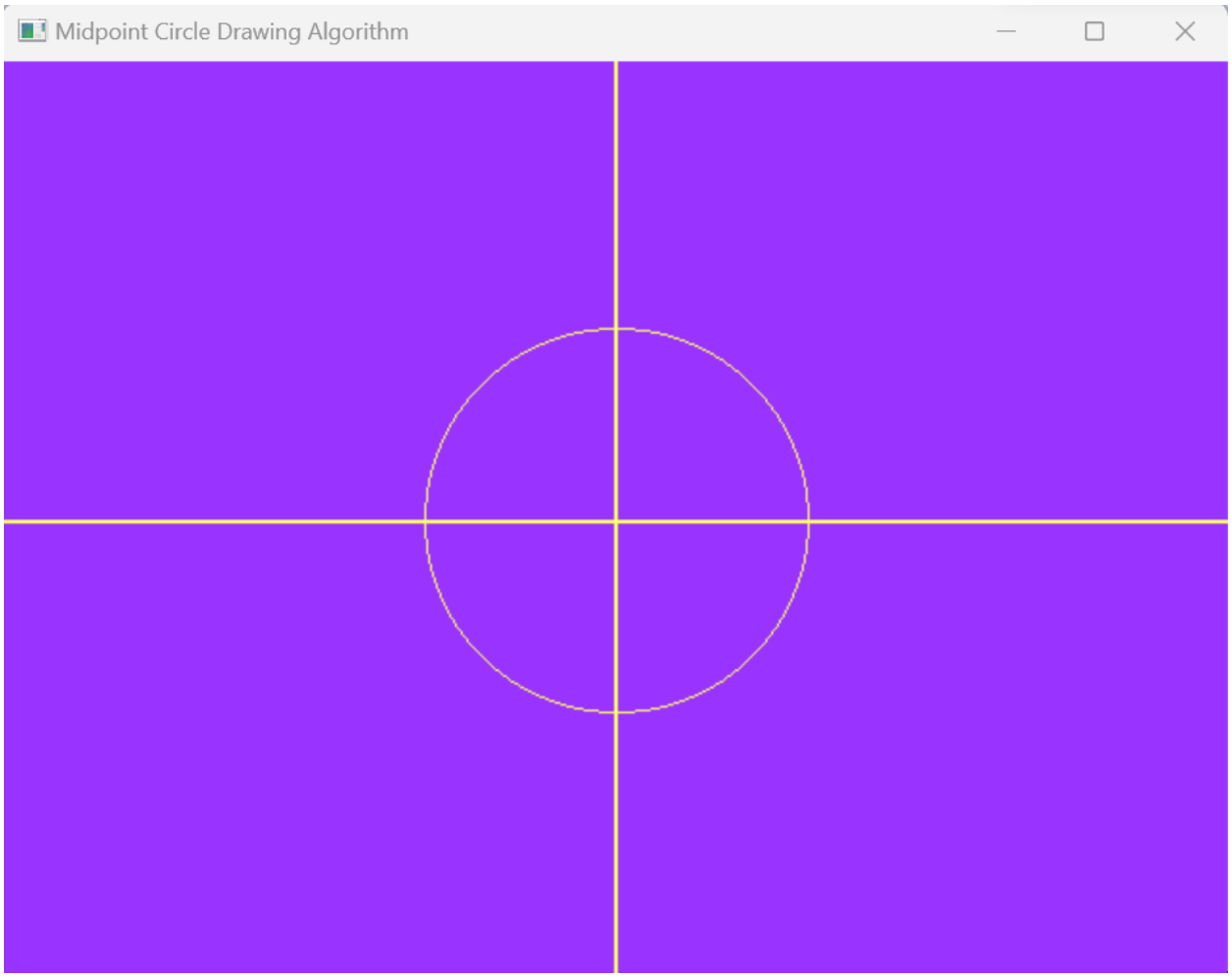
```
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}

void displayPoint(int x, int y) {
    glBegin(GL_POINTS);
    glVertex2d(x + 320, y + 240);
    glEnd();
}

void displayLine(int x1, int y1, int x2, int y2) {
    glBegin(GL_LINES);
    glColor4f(1, 1, 0.4, 1);
    glVertex2d(x1 + 320, y1 + 240);
    glVertex2d(x2 + 320, y2 + 240);
    glEnd();
}

void drawPlane() {
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
    glColor4f(1, 1, 1, 1);

    //y-axis
    displayLine(0, -240, 0, 240);
    //x-axis
    displayLine(-320, 0, 320, 0);
    glEnd();
    glFlush();
}

void plotAllOctets(int xc, int yc, int x, int y) {
    displayPoint(xc + x, yc + y);
    displayPoint(xc - x, yc + y);
    displayPoint(xc + x, yc - y);
    displayPoint(xc - x, yc - y);
    displayPoint(xc + y, yc + x);
    displayPoint(xc - y, yc + x);
    displayPoint(xc + y, yc - x);
    displayPoint(xc - y, yc - x);
}

void plotMidpointCircle() {
    int xc, yc, r;
    int P, x, y;
```

```
    cout << "Enter center coords: ";
    cin >> xc >> yc;
    cout << "Enter radius: ";
    cin >> r;
    cout << endl;

    plotAllOctets(xc, yc, 0, r);
    P = 1 - r;
    x = 0; y = r;

    while (x < y) {
        x++;
        if (P < 0) {
            P += (2 * x) + 1;
        }
        else {
            y--;
            P += 2 * (x - y) + 1;
        }
        plotAllOctets(xc, yc, x, y);
    }
}

void plotChart() {
    glClear(GL_COLOR_BUFFER_BIT);

    drawPlane();
    plotMidpointCircle();

    glFlush();
}

int main(int argc, char* argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowSize(640, 480);
    glutCreateWindow("Midpoint Circle Drawing Algorithm");
    glutDisplayFunc(plotChart);
    myInit();
    glutMainLoop();
    return 1;
}
```

Output:

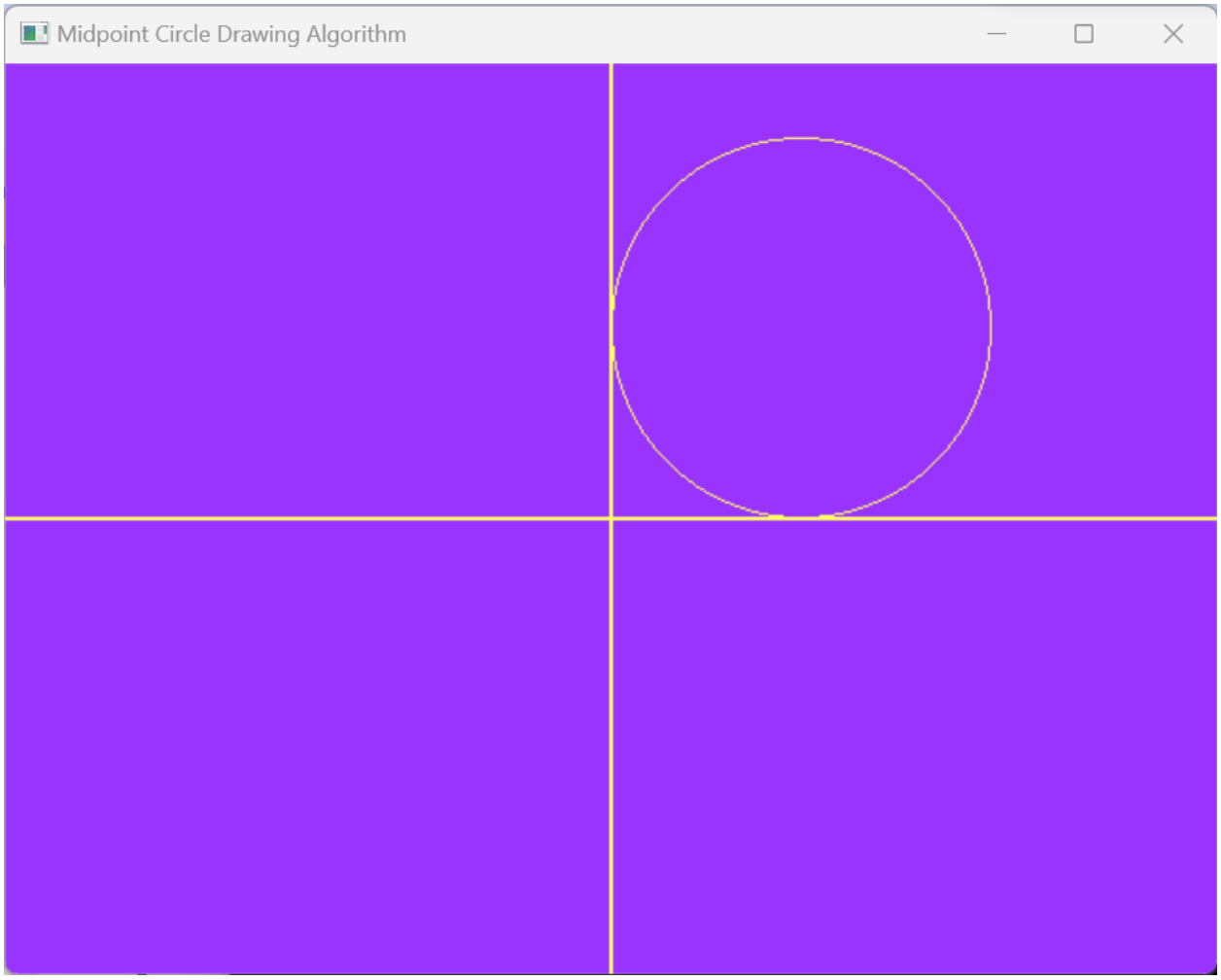
Case 1: circle with center at origin

```
Enter center coords: 0 0
Enter radius: 100
```



Case 2: circle with center not at origin

```
Enter center coords: 100 100
Enter radius: 100
```



Learning outcomes:

- The Midpoint Circle Drawing Algorithm was implemented.
- The implementation for the cases with the center of the circle at the origin and the center not at the origin was tested.

Exercise 5 – 2D Transformations

Date: 18/09/2023

Aim:

To apply the following 2D transformations on objects and to render the final output along with the original object.

1) Translation
2) Rotation
    a) about origin
    b) with respect to a fixed point (xr,yr)
3) Scaling with respect to
    a) origin - Uniform Vs Differential Scaling
    b) fixed point (xf,yf)
4) Reflection with respect to
    a) x-axis
    b) y-axis
    c) origin
    d) the line x=y
5) Shearing
    a) x-direction shear
    b) y-direction shear

Algorithm:

1. Get points of the object as input.
2. Draw the object.
3. Transform each vertex of the object.
4. Draw the object with the transformed vertices.

Code:

```
#define GL_SILENCE_DEPRECATION

#include<GL/glut.h>
#include<iostream>
#include<math.h>
using namespace std;
```

```
float toRad(float xDeg) {
	return xDeg * 3.14159 / 180;
}

void myInit() {
	glClearColor(0.6, 0.2, 1.0, 0.0); // violet
	glColor3f(0.0f, 0.0f, 0.5f); //dark blue
	//glPointSize(10);
	glMatrixMode(GL_PROJECTION);
	glLineWidth(2);
	glLoadIdentity();
	gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}

void displayPoint(float x, float y) {
	glBegin(GL_POINTS);
	glVertex2d(x + 320, y + 240);
	glEnd();
}

void displayHomogeneousPoint(float* h) {
	float x = *(h + 0);
	float y = *(h + 1);
	glColor4f(0, 1, 0.4, 1); //green
	displayPoint(x, y);
}

void displayLine(int x1, int y1, int x2, int y2) {
	glBegin(GL_LINES);
	glVertex2d(x1 + 320, y1 + 240);
	glVertex2d(x2 + 320, y2 + 240);
	glEnd();
}

void displayTriangle(int x1, int y1, int x2, int y2, int x3, int y3) {
	glBegin(GL_TRIANGLES);
	glVertex2d(x1 + 320, y1 + 240);
	glVertex2d(x2 + 320, y2 + 240);
	glVertex2d(x3 + 320, y3 + 240);
	glEnd();
}

void displayTransformedTriangle(float* p1, float* p2, float* p3) {
	float x1 = *(p1 + 0);
	float y1 = *(p1 + 1);
	float x2 = *(p2 + 0);
```

```
	float y2 = *(p2 + 1);
	float x3 = *(p3 + 0);
	float y3 = *(p3 + 1);

	glColor4f(0, 1, 0.4, 1); //green
	displayTriangle(x1, y1, x2, y2, x3, y3);
}

void drawPlane() {
	glClear(GL_COLOR_BUFFER_BIT);
	glColor4f(1, 1, 0.4, 1); //yellow
	displayLine(-320, 0, 320, 0); //x-axis
	displayLine(0, -240, 0, 240); //y-axis
	glFlush();
}

void printMenu() {
	cout << "1 - Translation" << endl;
	cout << "2 - Rotation about origin" << endl;
	cout << "3 - Rotation wrt fixed point" << endl;
	cout << "4 - Scaling wrt origin" << endl;
	cout << "5 - Scaling wrt fixed point" << endl;
	cout << "6 - Reflection wrt x-axis" << endl;
	cout << "7 - Reflection wrt y-axis" << endl;
	cout << "8 - Reflection wrt origin" << endl;
	cout << "9 - Reflection wrt line x=y" << endl;
	cout << "10 - Shearing along x-dir" << endl;
	cout << "11 - Shearing along y-dir" << endl;
	cout << "0 - All done" << endl;
}

void printMatrix(float* arr, int m, int n)
{
	int i, j;
	for (i = 0; i < m; i++) {
		for (j = 0; j < n; j++)
			cout << *((arr + i*n) + j) << " ";
		cout << endl;
	}
}

float* mulMatrix(float* a, int m1, int n1, float* b, int m2, int n2) {
	if (n1 != m2) {
		cout << "Multiplication Input Error" << endl;
		return NULL;
	}
```

```
	float* res = new float[m1 * n2];
	for (int i = 0; i < m1; i++) {
		for (int j = 0; j < n2; j++) {
			*((res + i*n2) + j) = 0;
			for (int k = 0; k < n1; k++) {
				*((res + i*n2) + j) += *((a + i*n1) + k) * *((b + k*n2) +
j);
			}
		}
	}
	return res;
}

void printPoint(float* P) {
	printMatrix(P, 3, 1);
}

void printMatrix3(float* M) {
	printMatrix(M, 3, 3);
}

float* transformPoint(float* m, float* p) {
	return mulMatrix(m, 3, 3, p, 3, 1);
}

float* mulTransforms(float* m1, float* m2) {
	return mulMatrix(m1, 3, 3, m2, 3, 3);
}

float* getTransformationMatrix() {
	cout << "COMPOSITE TRANSFORMATION" << endl;
	float* compositeMatrix = new float[3 * 3];
	for (int i = 0; i < 3; i++) {
		for (int j = 0; j < 3; j++) {
			compositeMatrix[i*3 + j] = (i == j) ? 1 : 0;
		}
	}

	printMenu();
	int ch;
	do {
		cout << "\nChoose required transformation: ";
		cin >> ch;

		switch (ch) {
```

```
                case 1: {
                        cout << "TRANSLATION" << endl;

                        float tx, ty;
                        cout << "Enter translation values: ";
                        cin >> tx >> ty;

                        float T[3][3] = {
                                {1, 0, tx},
                                {0, 1, ty},
                                {0, 0, 1}
                        };

                        float* temp = mulTransforms((float*)T, compositeMatrix);
                        delete[] compositeMatrix;
                        compositeMatrix = temp;

                        break;
                }

                case 2: {
                        cout << "ROTATION ABOUT ORIGIN" << endl;

                        float angle;
                        cout << "Enter rotation angle: ";
                        cin >> angle;

                        float theta = toRad(angle);
                        float c = cos(theta);
                        float s = sin(theta);

                        float R[3][3] = {
                                {c, -s, 0},
                                {s, c, 0},
                                {0, 0, 1}
                        };

                        float* temp = mulTransforms((float*)R, compositeMatrix);
                        delete[] compositeMatrix;
                        compositeMatrix = temp;

                        break;
                }

                case 3: {
                        cout << "ROTATION WRT FIXED POINT" << endl;
```

```
            float angle;
            cout << "Enter rotation angle: ";
            cin >> angle;

            float theta = toRad(angle);
            float c = cos(theta);
            float s = sin(theta);

            float xr, yr;
            cout << "Enter fixed point coords: ";
            cin >> xr >> yr;

            float R[3][3] = {
                  {c, -s, (xr * (1-c)) + (yr * s)},
                  {s, c, (yr * (1-c)) - (xr * s)},
                  {0, 0, 1}
            };

            float* temp = mulTransforms((float*)R, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;

            break;
      }

      case 4: {
            cout << "SCALING WRT ORIGIN" << endl;

            float sx, sy;
            cout << "Enter scaling factor values: ";
            cin >> sx >> sy;

            float S[3][3] = {
                  {sx, 0, 0},
                  {0, sy, 0},
                  {0, 0, 1}
            };

            float* temp = mulTransforms((float*)S, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;

            break;
      }
```

```
		case 5: {
			cout << "SCALING WRT FIXED POINT" << endl;

			float sx, sy;
			cout << "Enter scaling factor values: ";
			cin >> sx >> sy;

			float xf, yf;
			cout << "Enter fixed point coords: ";
			cin >> xf >> yf;

			float S[3][3] = {
				{sx, 0, xf * (1-sx)},
				{0, sy, yf * (1-sy)},
				{0, 0, 1}
			};

			float* temp = mulTransforms((float*)S, compositeMatrix);
			delete[] compositeMatrix;
			compositeMatrix = temp;

			break;
		}

		case 6: {
			cout << "REFLECTION WRT X-AXIS" << endl;

			float RF[3][3] = {
				{1, 0, 0},
				{0, -1, 0},
				{0, 0, 1}
			};

			float* temp = mulTransforms((float*)RF, compositeMatrix);
			delete[] compositeMatrix;
			compositeMatrix = temp;

			break;
		}

		case 7: {
			cout << "REFLECTION WRT Y-AXIS" << endl;

			float RF[3][3] = {
				{-1, 0, 0},
				{0, 1, 0},
```

```
                        {0, 0, 1}
                };

                float* temp = mulTransforms((float*)RF, compositeMatrix);
                delete[] compositeMatrix;
                compositeMatrix = temp;

                break;
        }

        case 8: {
                cout << "REFLECTION WRT ORIGIN" << endl;

                float RF[3][3] = {
                        {-1, 0, 0},
                        {0, -1, 0},
                        {0, 0, 1}
                };

                float* temp = mulTransforms((float*)RF, compositeMatrix);
                delete[] compositeMatrix;
                compositeMatrix = temp;

                break;
        }

        case 9: {
                cout << "REFLECTION WRT LINE X=Y" << endl;

                float RF[3][3] = {
                        {0, 1, 0},
                        {1, 0, 0},
                        {0, 0, 1}
                };

                float* temp = mulTransforms((float*)RF, compositeMatrix);
                delete[] compositeMatrix;
                compositeMatrix = temp;

                break;
        }

        case 10: {
                cout << "SHEARING ALONG X-DIR" << endl;

                float shx, yref = 0;
```

```
                        cout << "Enter shear value: ";
                        cin >> shx;
                        cout << "Enter yref value: ";
                        cin >> yref;

                        float SH[3][3] = {
                              {1, shx,-shx * yref},
                              {0, 1, 0},
                              {0, 0, 1}
                        };

                        float* temp = mulTransforms((float*)SH, compositeMatrix);
                        delete[] compositeMatrix;
                        compositeMatrix = temp;

                        break;
                  }

                  case 11: {
                        cout << "SHEARING ALONG Y-DIR" << endl;

                        float shy, xref = 0;
                        cout << "Enter shear value: ";
                        cin >> shy;
                        cout << "Enter yref value: ";
                        cin >> xref;

                        float SH[3][3] = {
                              {1, 0, 0},
                              {shy, 1, -shy * xref},
                              {0, 0, 1}
                        };

                        float* temp = mulTransforms((float*)SH, compositeMatrix);
                        delete[] compositeMatrix;
                        compositeMatrix = temp;

                        break;
                  }

                  case 0: {
                        cout << "ALL DONE" << endl;
                  }

                  default: break;
            }
```

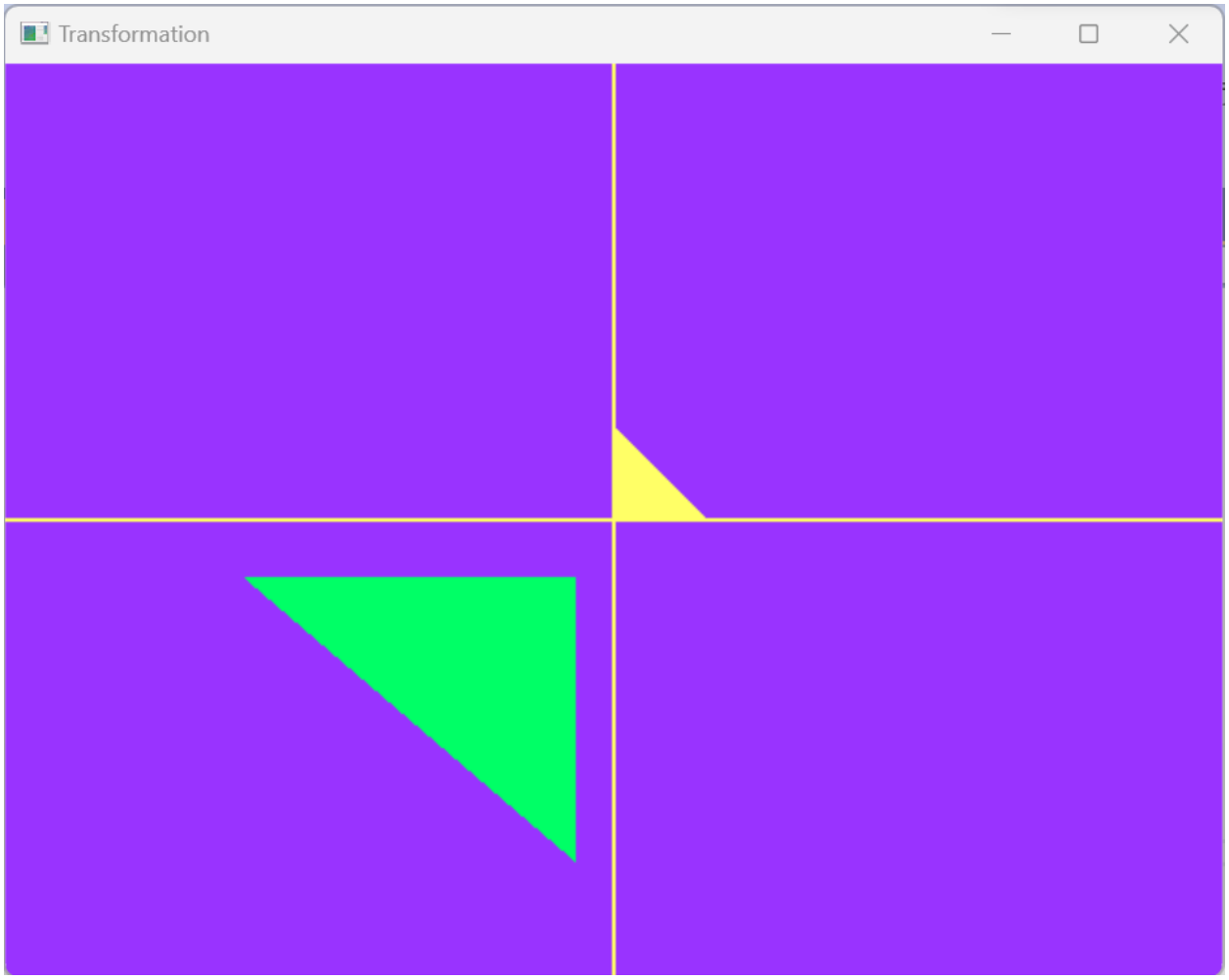
```
	} while (ch != 0);

	return compositeMatrix;
}

void plotTransform()
{
	cout << "TRANSFORMATION OF A TRIANGLE" << endl;

	//Point P1
	float x1, y1;
	cout << "Enter point P1 coords: ";
	cin >> x1 >> y1;

	float* P1 = new float[3]{ {x1}, {y1}, {1} };
	cout << "Homogeneous representation of P1: " << endl;
	printPoint(P1);
	cout << endl;

	//Point P2
	float x2, y2;
	cout << "Enter point P2 coords: ";
	cin >> x2 >> y2;

	float* P2 = new float[3] { {x2}, { y2 }, { 1 } };
	cout << "Homogeneous representation of P2: " << endl;
	printPoint(P2);
	cout << endl;

	//Point P3
	float x3, y3;
	cout << "Enter point P3 coords: ";
	cin >> x3 >> y3;

	float* P3 = new float[3] { {x3}, { y3 }, { 1 } };
	cout << "Homogeneous representation of P3: " << endl;
	printPoint(P3);
	cout << endl;

	//plot triangle
	displayTriangle(x1, y1, x2, y2, x3, y3);

	float* M = getTransformationMatrix();
	if (M != NULL) {
		cout << "\nTransformation Matrix: " << endl;
		printMatrix3(M);
```

```
            cout << "\nP1': " << endl;
            float* Q1 = transformPoint(M, P1);
            printPoint(Q1);

            cout << "\nP2': " << endl;
            float* Q2 = transformPoint(M, P2);
            printPoint(Q2);

            cout << "\nP3': " << endl;
            float* Q3 = transformPoint(M, P3);
            printPoint(Q3);

            displayTransformedTriangle(Q1, Q2, Q3);

            delete[] Q1;
            delete[] Q2;
            delete[] Q3;
      }
      delete[] M;
      delete[] P1;
      delete[] P2;
      delete[] P3;
}

void plotChart() {
      glClear(GL_COLOR_BUFFER_BIT);

      drawPlane();
      plotTransform();

      glFlush();
}

int main(int argc, char* argv[]) {
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
      glutInitWindowSize(640, 480);
      glutCreateWindow("Transformation");
      glutDisplayFunc(plotChart);
      myInit();
      glutMainLoop();
      return 1;
}
```

Output:

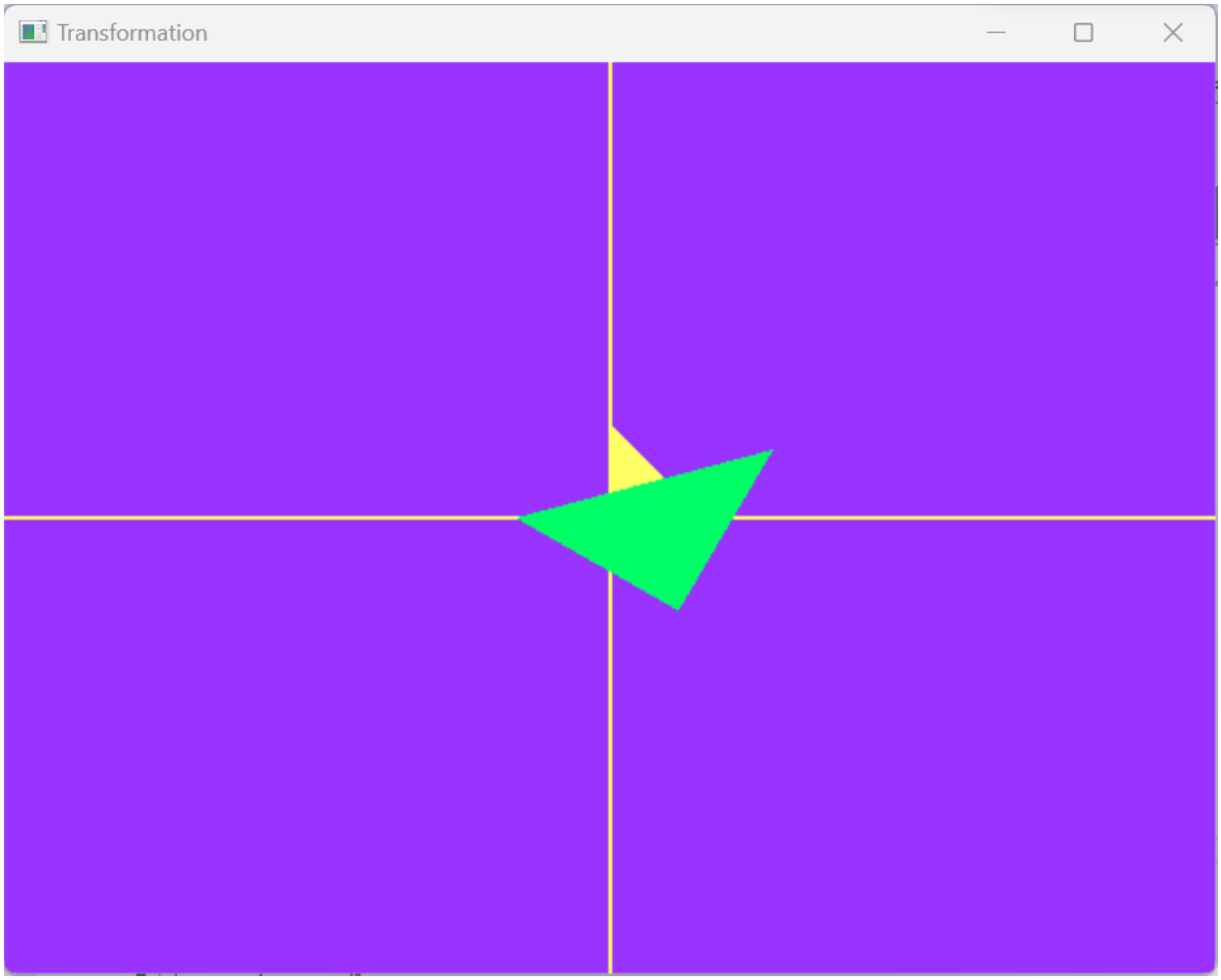
```
TRANSFORMATION OF A TRIANGLE
Enter point P1 coords: 0 0
Homogeneous representation of P1:
0
0
1

Enter point P2 coords: 0 50
Homogeneous representation of P2:
0
50
1

Enter point P3 coords: 50 0
Homogeneous representation of P3:
50
0
1

COMPOSITE TRANSFORMATION
1 - Translation
2 - Rotation about origin
3 - Rotation wrt fixed point
4 - Scaling wrt origin
5 - Scaling wrt fixed point
6 - Reflection wrt x-axis
7 - Reflection wrt y-axis
8 - Reflection wrt origin
9 - Reflection wrt line x=y
10 - Shearing along x-dir
11 - Shearing along y-dir
0 - All done
```

```
SCALING WRT ORIGIN
Enter scaling factor values: 3.5 3

Choose required transformation: 8
REFLECTION WRT ORIGIN

Choose required transformation: 1
TRANSLATION
Enter translation values: -20 -30

Choose required transformation: 0
ALL DONE

Transformation Matrix:
-3.5 0 -20
0 -3 -30
0 0 1

P1':
-20
-30
1

P2':
-20
-180
1

P3':
-195
-30
1
```



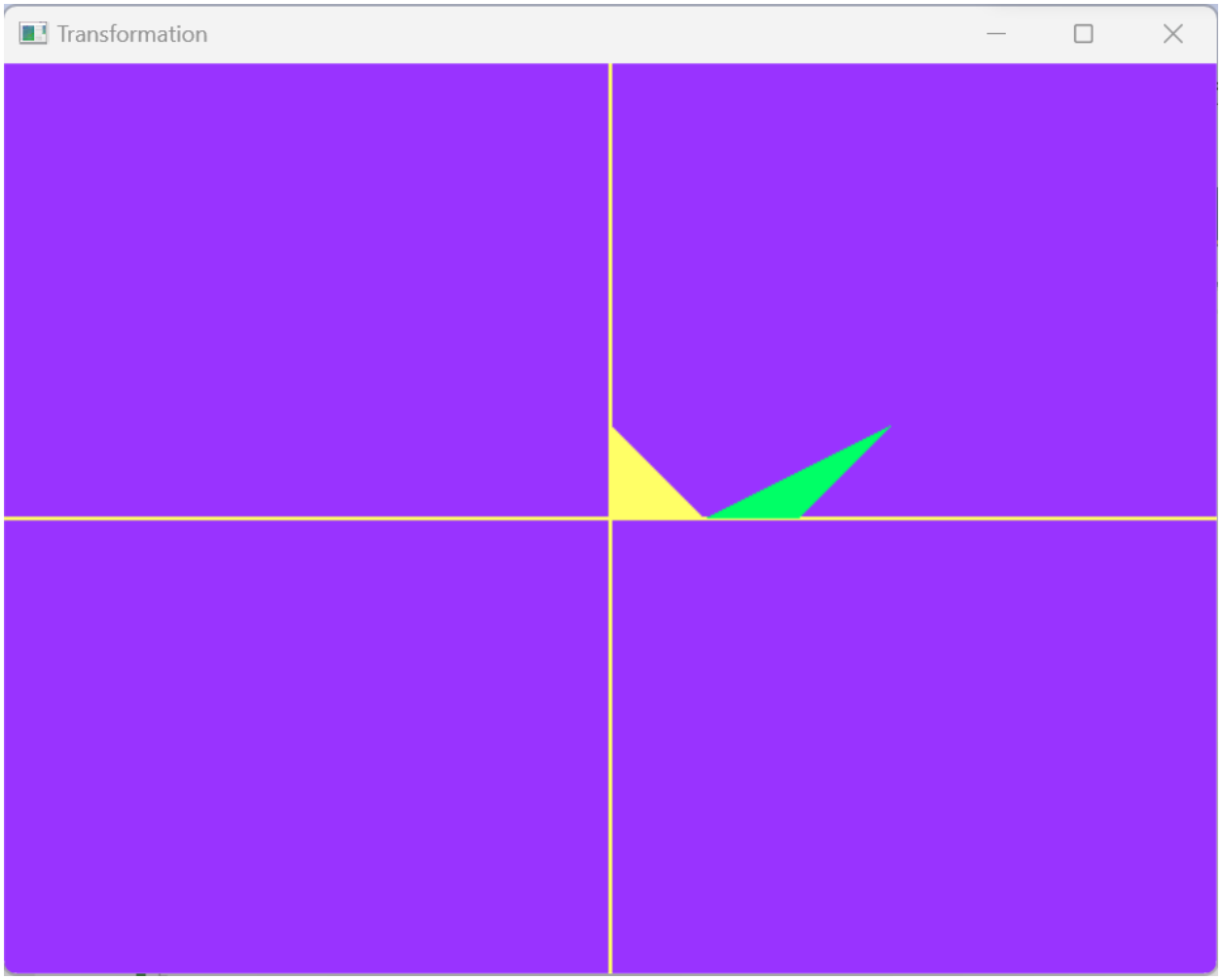

```
Choose required transformation: 3
ROTATION WRT FIXED POINT
Enter rotation angle: 30
Enter fixed point coords: 50 0

Choose required transformation: 5
SCALING WRT FIXED POINT
Enter scaling factor values: 2 2
Enter fixed point coords: 50 0

Choose required transformation: 7
REFLECTION WRT Y-AXIS

Choose required transformation: 0
ALL DONE
```

```
Transformation Matrix:
-1.73205 0.999999 36.6026
0.999999 1.73205 -50
0 0 1

P1':
36.6026
-50
1

P2':
-50
0
1

P3':
86.6025
36.6026
1
```



```
Choose required transformation: 9
REFLECTION WRT LINE X=Y

Choose required transformation: 10
SHEARING ALONG X-DIR
Enter shear value: 2
Enter yref value: -25

Choose required transformation: 0
ALL DONE

Transformation Matrix:
2 1 50
1 0 0
0 0 1
```

```
P1':
50
0
1

P2':
150
50
1

P3':
100
0
1
```



Learning outcomes:

- 2D transformations were applied on objects using OpenGL in C++.
- The implementation for each transformation and the composite transformation were tested for points and objects.

Exercise 6 – 2D Composite Transformations and Windowing

Date: 25/09/2023

Aim:

a. To compute the composite transformation matrix for any 2 transformations input by the user and apply it on the object.
   1) Translation
   2) Rotation
      a) about origin
      b) with respect to a fixed point (xr,yr)
   3) Scaling with respect to
      a) origin - Uniform Vs Differential Scaling
      b) fixed point (xf,yf)
   4) Reflection with respect to
      a) x-axis
      b) y-axis
      c) origin
      d) the line x=y
   5) Shearing
      a) x-direction shear
      b) y-direction shear

b. Create a window with any 2D object and a different sized viewport. Apply window to viewport transformation on the object. Display both window and viewport.

Algorithm:

6a:

1. Get points of the object as input.
2. Draw the object.
3. Transform each vertex of the object.
4. Draw the object with the transformed vertices.

6b:

1. Store the window dimensions and the viewport dimensions.
2. Get points of the object as input and draw it on the window.
3. Apply window to viewport transformation on the object as:
   a. Sx = (xvmax - xvmin) / (xwmax - xwmin)
   b. xv = xvmin + (xw - xwmin) * Sx
   c. Similarly, for the y-coordinates.
4. Draw the object on the viewport.

Code:

6a. Composite Transformations:

```
#define GL_SILENCE_DEPRECATION

#include<GL/glut.h>
#include<iostream>
#include<math.h>
using namespace std;

float toRad(float xDeg) {
	return xDeg * 3.14159 / 180;
}

void myInit() {
	glClearColor(0.6, 0.2, 1.0, 0.0); // violet
	glColor3f(0.0f, 0.0f, 0.5f); //dark blue
	//glPointSize(10);
	glMatrixMode(GL_PROJECTION);
	glLineWidth(2);
	glLoadIdentity();
	gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}

void displayPoint(float x, float y) {
	glBegin(GL_POINTS);
	glVertex2d(x + 320, y + 240);
	glEnd();
}

void displayHomogeneousPoint(float* h) {
	float x = *(h + 0);
	float y = *(h + 1);
	glColor4f(0, 1, 0.4, 1); //green
	displayPoint(x, y);
}

void displayLine(int x1, int y1, int x2, int y2) {
	glBegin(GL_LINES);
	glVertex2d(x1 + 320, y1 + 240);
	glVertex2d(x2 + 320, y2 + 240);
	glEnd();
}
```

```
void displayTriangle(int x1, int y1, int x2, int y2, int x3, int y3) {
	glBegin(GL_TRIANGLES);
	glVertex2d(x1 + 320, y1 + 240);
	glVertex2d(x2 + 320, y2 + 240);
	glVertex2d(x3 + 320, y3 + 240);
	glEnd();
}

void displayTransformedTriangle(float* p1, float* p2, float* p3) {
	float x1 = *(p1 + 0);
	float y1 = *(p1 + 1);
	float x2 = *(p2 + 0);
	float y2 = *(p2 + 1);
	float x3 = *(p3 + 0);
	float y3 = *(p3 + 1);

	glColor4f(0, 1, 0.4, 1); //green
	displayTriangle(x1, y1, x2, y2, x3, y3);
}

void drawPlane() {
	glClear(GL_COLOR_BUFFER_BIT);
	glColor4f(1, 1, 0.4, 1); //yellow
	displayLine(-320, 0, 320, 0); //x-axis
	displayLine(0, -240, 0, 240); //y-axis
	glFlush();
}

void printMenu() {
	cout << "1 - Translation" << endl;
	cout << "2 - Rotation about origin" << endl;
	cout << "3 - Rotation wrt fixed point" << endl;
	cout << "4 - Scaling wrt origin" << endl;
	cout << "5 - Scaling wrt fixed point" << endl;
	cout << "6 - Reflection wrt x-axis" << endl;
	cout << "7 - Reflection wrt y-axis" << endl;
	cout << "8 - Reflection wrt origin" << endl;
	cout << "9 - Reflection wrt line x=y" << endl;
	cout << "10 - Shearing along x-dir" << endl;
	cout << "11 - Shearing along y-dir" << endl;
	cout << "0 - All done" << endl;
}

void printMatrix(float* arr, int m, int n)
{
	int i, j;
```

```
	for (i = 0; i < m; i++) {
		for (j = 0; j < n; j++)
			cout << *((arr + i*n) + j) << " ";
		cout << endl;
	}
}

float* mulMatrix(float* a, int m1, int n1, float* b, int m2, int n2) {
	if (n1 != m2) {
		cout << "Multiplication Input Error" << endl;
		return NULL;
	}

	float* res = new float[m1 * n2];
	for (int i = 0; i < m1; i++) {
		for (int j = 0; j < n2; j++) {
			*((res + i*n2) + j) = 0;
			for (int k = 0; k < n1; k++) {
				*((res + i*n2) + j) += *((a + i*n1) + k) * *((b + k*n2) +
j);
			}
		}
	}
	return res;
}

void printPoint(float* P) {
	printMatrix(P, 3, 1);
}

void printMatrix3(float* M) {
	printMatrix(M, 3, 3);
}

float* transformPoint(float* m, float* p) {
	return mulMatrix(m, 3, 3, p, 3, 1);
}

float* mulTransforms(float* m1, float* m2) {
	return mulMatrix(m1, 3, 3, m2, 3, 3);
}

float* getTransformationMatrix() {
	cout << "COMPOSITE TRANSFORMATION" << endl;
	float* compositeMatrix = new float[3 * 3];
	for (int i = 0; i < 3; i++) {
```

```
		for (int j = 0; j < 3; j++) {
			compositeMatrix[i*3 + j] = (i == j) ? 1 : 0;
		}
	}

	printMenu();
	int ch;
	do {
		cout << "\nChoose required transformation: ";
		cin >> ch;

		switch (ch) {
			case 1: {
				cout << "TRANSLATION" << endl;

				float tx, ty;
				cout << "Enter translation values: ";
				cin >> tx >> ty;

				float T[3][3] = {
					{1, 0, tx},
					{0, 1, ty},
					{0, 0, 1}
				};

				float* temp = mulTransforms((float*)T, compositeMatrix);
				delete[] compositeMatrix;
				compositeMatrix = temp;

				break;
			}

			case 2: {
				cout << "ROTATION ABOUT ORIGIN" << endl;

				float angle;
				cout << "Enter rotation angle: ";
				cin >> angle;

				float theta = toRad(angle);
				float c = cos(theta);
				float s = sin(theta);

				float R[3][3] = {
					{c, -s, 0},
					{s, c, 0},
```

```
                        {0, 0, 1}
                };

                float* temp = mulTransforms((float*)R, compositeMatrix);
                delete[] compositeMatrix;
                compositeMatrix = temp;

                break;
        }

        case 3: {
                cout << "ROTATION WRT FIXED POINT" << endl;

                float angle;
                cout << "Enter rotation angle: ";
                cin >> angle;

                float theta = toRad(angle);
                float c = cos(theta);
                float s = sin(theta);

                float xr, yr;
                cout << "Enter fixed point coords: ";
                cin >> xr >> yr;

                float R[3][3] = {
                        {c, -s, (xr * (1-c)) + (yr * s)},
                        {s, c, (yr * (1-c)) - (xr * s)},
                        {0, 0, 1}
                };

                float* temp = mulTransforms((float*)R, compositeMatrix);
                delete[] compositeMatrix;
                compositeMatrix = temp;

                break;
        }

        case 4: {
                cout << "SCALING WRT ORIGIN" << endl;

                float sx, sy;
                cout << "Enter scaling factor values: ";
                cin >> sx >> sy;

                float S[3][3] = {
```

```
                        {sx, 0, 0},
                        {0, sy, 0},
                        {0, 0, 1}
                    };

                    float* temp = mulTransforms((float*)S, compositeMatrix);
                    delete[] compositeMatrix;
                    compositeMatrix = temp;

                    break;
                }

                case 5: {
                    cout << "SCALING WRT FIXED POINT" << endl;

                    float sx, sy;
                    cout << "Enter scaling factor values: ";
                    cin >> sx >> sy;

                    float xf, yf;
                    cout << "Enter fixed point coords: ";
                    cin >> xf >> yf;

                    float S[3][3] = {
                        {sx, 0, xf * (1-sx)},
                        {0, sy, yf * (1-sy)},
                        {0, 0, 1}
                    };

                    float* temp = mulTransforms((float*)S, compositeMatrix);
                    delete[] compositeMatrix;
                    compositeMatrix = temp;

                    break;
                }

                case 6: {
                    cout << "REFLECTION WRT X-AXIS" << endl;

                    float RF[3][3] = {
                        {1, 0, 0},
                        {0, -1, 0},
                        {0, 0, 1}
                    };

                    float* temp = mulTransforms((float*)RF, compositeMatrix);
```

```
			delete[] compositeMatrix;
			compositeMatrix = temp;

			break;
		}

		case 7: {
			cout << "REFLECTION WRT Y-AXIS" << endl;

			float RF[3][3] = {
				{-1, 0, 0},
				{0, 1, 0},
				{0, 0, 1}
			};

			float* temp = mulTransforms((float*)RF, compositeMatrix);
			delete[] compositeMatrix;
			compositeMatrix = temp;

			break;
		}

		case 8: {
			cout << "REFLECTION WRT ORIGIN" << endl;

			float RF[3][3] = {
				{-1, 0, 0},
				{0, -1, 0},
				{0, 0, 1}
			};

			float* temp = mulTransforms((float*)RF, compositeMatrix);
			delete[] compositeMatrix;
			compositeMatrix = temp;

			break;
		}

		case 9: {
			cout << "REFLECTION WRT LINE X=Y" << endl;

			float RF[3][3] = {
				{0, 1, 0},
				{1, 0, 0},
				{0, 0, 1}
			};
```

```
            float* temp = mulTransforms((float*)RF, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;

            break;
        }

        case 10: {
            cout << "SHEARING ALONG X-DIR" << endl;

            float shx, yref = 0;
            cout << "Enter shear value: ";
            cin >> shx;
            cout << "Enter yref value: ";
            cin >> yref;

            float SH[3][3] = {
                {1, shx,-shx * yref},
                {0, 1, 0},
                {0, 0, 1}
            };

            float* temp = mulTransforms((float*)SH, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;

            break;
        }

        case 11: {
            cout << "SHEARING ALONG Y-DIR" << endl;

            float shy, xref = 0;
            cout << "Enter shear value: ";
            cin >> shy;
            cout << "Enter yref value: ";
            cin >> xref;

            float SH[3][3] = {
                {1, 0, 0},
                {shy, 1, -shy * xref},
                {0, 0, 1}
            };

            float* temp = mulTransforms((float*)SH, compositeMatrix);
```

```
                        delete[] compositeMatrix;
                        compositeMatrix = temp;

                        break;
                }

                case 0: {
                        cout << "ALL DONE" << endl;
                }

                default: break;
            }
        } while (ch != 0);

        return compositeMatrix;
}

void plotTransform()
{
        cout << "TRANSFORMATION OF A TRIANGLE" << endl;

        //Point P1
        float x1, y1;
        cout << "Enter point P1 coords: ";
        cin >> x1 >> y1;

        float* P1 = new float[3]{ {x1}, {y1}, {1} };
        cout << "Homogeneous representation of P1: " << endl;
        printPoint(P1);
        cout << endl;

        //Point P2
        float x2, y2;
        cout << "Enter point P2 coords: ";
        cin >> x2 >> y2;

        float* P2 = new float[3] { {x2}, { y2 }, { 1 } };
        cout << "Homogeneous representation of P2: " << endl;
        printPoint(P2);
        cout << endl;

        //Point P3
        float x3, y3;
        cout << "Enter point P3 coords: ";
        cin >> x3 >> y3;
```

```
	float* P3 = new float[3] { {x3}, { y3 }, { 1 } };
	cout << "Homogeneous representation of P3: " << endl;
	printPoint(P3);
	cout << endl;

	//plot triangle
	displayTriangle(x1, y1, x2, y2, x3, y3);

	float* M = getTransformationMatrix();
	if (M != NULL) {
		cout << "\nTransformation Matrix: " << endl;
		printMatrix3(M);

		cout << "\nP1': " << endl;
		float* Q1 = transformPoint(M, P1);
		printPoint(Q1);

		cout << "\nP2': " << endl;
		float* Q2 = transformPoint(M, P2);
		printPoint(Q2);

		cout << "\nP3': " << endl;
		float* Q3 = transformPoint(M, P3);
		printPoint(Q3);

		displayTransformedTriangle(Q1, Q2, Q3);

		delete[] Q1;
		delete[] Q2;
		delete[] Q3;
	}
	delete[] M;
	delete[] P1;
	delete[] P2;
	delete[] P3;
}

void plotChart() {
	glClear(GL_COLOR_BUFFER_BIT);

	drawPlane();
	plotTransform();

	glFlush();
}
```

```
int main(int argc, char* argv[]) {
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
	glutInitWindowSize(640, 480);
	glutCreateWindow("Transformation");
	glutDisplayFunc(plotChart);
	myInit();
	glutMainLoop();
	return 1;
}
```

6b. Window to Viewport Transformation:

```
#define GL_SILENCE_DEPRECATION

#include<GL/glut.h>
#include<iostream>
#include<math.h>
using namespace std;

float* G1 = new float[3];
float* G2 = new float[3];
float* G3 = new float[3];

float xwmax = 1280, ywmax = 960;
float xvmax = 640, yvmax = 480;

void myInit_win1(float x, float y) {
	glClearColor(0.6, 0.2, 1.0, 0.0); // violet
	glColor3f(0.0f, 0.0f, 0.5f); //dark blue
	glMatrixMode(GL_PROJECTION);
	glLineWidth(2);
	glLoadIdentity();
	gluOrtho2D(0.0, x, 0.0, y);
}

void myInit_win2(float x, float y) {
	glClearColor(0.0, 0.0, 0.5, 0.0); // dark blue
	glColor3f(0.5f, 0.3f, 0.5f); //
	glMatrixMode(GL_PROJECTION);
	glLineWidth(2);
	glLoadIdentity();
	gluOrtho2D(0.0, x, 0.0, y);
}

void displayLine(int x1, int y1, int x2, int y2, int xc, int yc) {
```

```
	glBegin(GL_LINES);
	glVertex2d(x1 + xc, y1 + yc);
	glVertex2d(x2 + xc, y2 + yc);
	glEnd();
}

void displayTriangle(int x1, int y1, int x2, int y2, int x3, int y3, int xc, int
yc) {
	glBegin(GL_TRIANGLES);
	glVertex2d(x1 + xc, y1 + yc);
	glVertex2d(x2 + xc, y2 + yc);
	glVertex2d(x3 + xc, y3 + yc);
	glEnd();
}

void displayLineWindow(int x1, int y1, int x2, int y2) {
	displayLine(x1, y1, x2, y2, xwmax/2, ywmax/2);
}

void displayLineViewport(int x1, int y1, int x2, int y2) {
	displayLine(x1, y1, x2, y2, xvmax/2, yvmax/2);
}

void displayTriangleWindow(int x1, int y1, int x2, int y2, int x3, int y3) {
	displayTriangle(x1, y1, x2, y2, x3, y3, xwmax/2, ywmax/2);
}

void displayTriangleViewport(int x1, int y1, int x2, int y2, int x3, int y3) {
	displayTriangle(x1, y1, x2, y2, x3, y3, xvmax/2, yvmax/2);
}

void displayHomogeneousTriangle(float* p1, float* p2, float* p3) {
	float x1 = *(p1 + 0);
	float y1 = *(p1 + 1);
	float x2 = *(p2 + 0);
	float y2 = *(p2 + 1);
	float x3 = *(p3 + 0);
	float y3 = *(p3 + 1);

	displayTriangleViewport(x1, y1, x2, y2, x3, y3);
}

void printMatrix(float* arr, int m, int n)
{
	int i, j;
	for (i = 0; i < m; i++) {
```

```
			for (j = 0; j < n; j++)
				cout << *((arr + i * n) + j) << " ";
			cout << endl;
		}
}

float* mulMatrix(float* a, int m1, int n1, float* b, int m2, int n2) {
		if (n1 != m2) {
			cout << "Multiplication Input Error" << endl;
			return NULL;
		}

		float* res = new float[m1 * n2];
		for (int i = 0; i < m1; i++) {
			for (int j = 0; j < n2; j++) {
				*((res + i * n2) + j) = 0;
				for (int k = 0; k < n1; k++) {
					*((res + i * n2) + j) += *((a + i * n1) + k) * *((b + k * 
n2) + j);
				}
			}
		}
		return res;
}

void printPoint(float* P) {
		printMatrix(P, 3, 1);
}

float* transformPoint(float* m, float* p) {
		return mulMatrix(m, 3, 3, p, 3, 1);
}

void drawPlane(int xmax, int ymax) {
		int hx = xmax/2, hy = ymax/2;
		glClear(GL_COLOR_BUFFER_BIT);
		glColor4f(1, 1, 0.4, 1); //yellow
		displayLine(-hx, 0, hx, 0, hx, hy); //x-axis
		displayLine(0, -hy, 0, hy, hx, hy); //y-axis
		glFlush();
}

void plotWindow()
{
		cout << "GET INPUTS FOR WINDOW IMAGE" << endl;
```

```
	//Point P1
	float x1, y1;
	cout << "Enter point P1 coords: ";
	cin >> x1 >> y1;

	float* P1 = new float[3] { {x1}, { y1 }, { 1 } };
	cout << "Homogeneous representation of P1: " << endl;
	printPoint(P1);
	cout << endl;
	G1 = P1;

	//Point P2
	float x2, y2;
	cout << "Enter point P2 coords: ";
	cin >> x2 >> y2;

	float* P2 = new float[3] { {x2}, { y2 }, { 1 } };
	cout << "Homogeneous representation of P2: " << endl;
	printPoint(P2);
	cout << endl;
	G2 = P2;

	//Point P3
	float x3, y3;
	cout << "Enter point P3 coords: ";
	cin >> x3 >> y3;

	float* P3 = new float[3] { {x3}, { y3 }, { 1 } };
	cout << "Homogeneous representation of P3: " << endl;
	printPoint(P3);
	cout << endl;
	G3 = P3;

	//plot triangle
	displayTriangleWindow(x1, y1, x2, y2, x3, y3);
}

void plotViewport() {
	cout << "GETTING INPUTS FROM WINDOW IMAGE" << endl;

	float Sx = xvmax/xwmax, Sy = yvmax/ywmax;
	//Scaling matrix: Window to Viewport
	float S[3][3] = {
		{Sx, 0, 0},
		{0, Sy, 0},
		{0, 0, 1}
```

```
	};

	//Point P1
	float* P1 = transformPoint((float*)S, G1);
	cout << "Homogeneous representation of P1: " << endl;
	printPoint(P1);
	cout << endl;

	//Point P2
	float* P2 = transformPoint((float*)S, G2);
	cout << "Homogeneous representation of P2: " << endl;
	printPoint(P2);
	cout << endl;

	//Point P3
	float* P3 = transformPoint((float*)S, G3);
	cout << "Homogeneous representation of P3: " << endl;
	printPoint(P3);
	cout << endl;

	//plot triangle
	displayHomogeneousTriangle(P1, P2, P3);

	delete[] P1;
	delete[] P2;
	delete[] P3;
}

void plotChart_win1() {
	myInit_win1(xwmax, ywmax);
	glClear(GL_COLOR_BUFFER_BIT);
	cout << "Window" << endl;

	drawPlane(xwmax, ywmax);
	plotWindow();

	glFlush();
	glutSwapBuffers();
}

void plotChart_win2() {
	myInit_win2(xvmax, yvmax);
	glClear(GL_COLOR_BUFFER_BIT);
	cout << "Viewport" << endl;

	drawPlane(xvmax, yvmax);
```

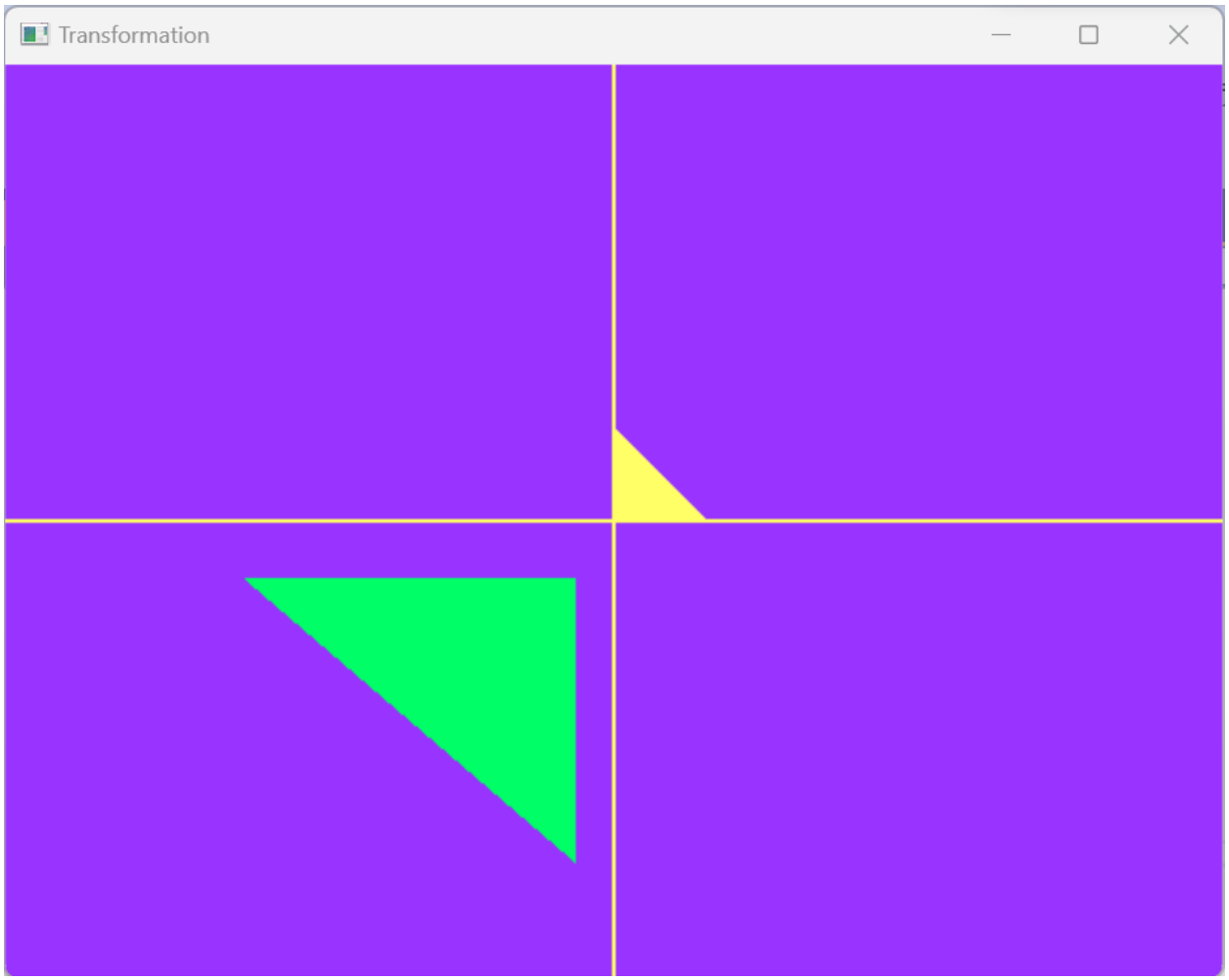
```
	plotViewport();

	glFlush();
	glutSwapBuffers();
}

int main(int argc, char* argv[]) {
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);

	glutInitWindowSize(xwmax, ywmax);
	glutInitWindowPosition(0, 0);
	int win1 = glutCreateWindow("Window");

	glutInitWindowSize(xvmax, yvmax);
	glutInitWindowPosition(0, 0);
	int win2 = glutCreateWindow("Viewport");

	glutSetWindow(win1);
	glutDisplayFunc(plotChart_win1);

	glutSetWindow(win2);
	glutDisplayFunc(plotChart_win2);

	glutMainLoop();
	return 1;
}
```

Output:

6a. Composite Transformations

```
TRANSFORMATION OF A TRIANGLE
Enter point P1 coords: 0 0
Homogeneous representation of P1:
0
0
1

Enter point P2 coords: 0 50
Homogeneous representation of P2:
0
50
1

Enter point P3 coords: 50 0
Homogeneous representation of P3:
50
0
1

COMPOSITE TRANSFORMATION
1 - Translation
2 - Rotation about origin
3 - Rotation wrt fixed point
4 - Scaling wrt origin
5 - Scaling wrt fixed point
6 - Reflection wrt x-axis
7 - Reflection wrt y-axis
8 - Reflection wrt origin
9 - Reflection wrt line x=y
10 - Shearing along x-dir
11 - Shearing along y-dir
0 - All done
```

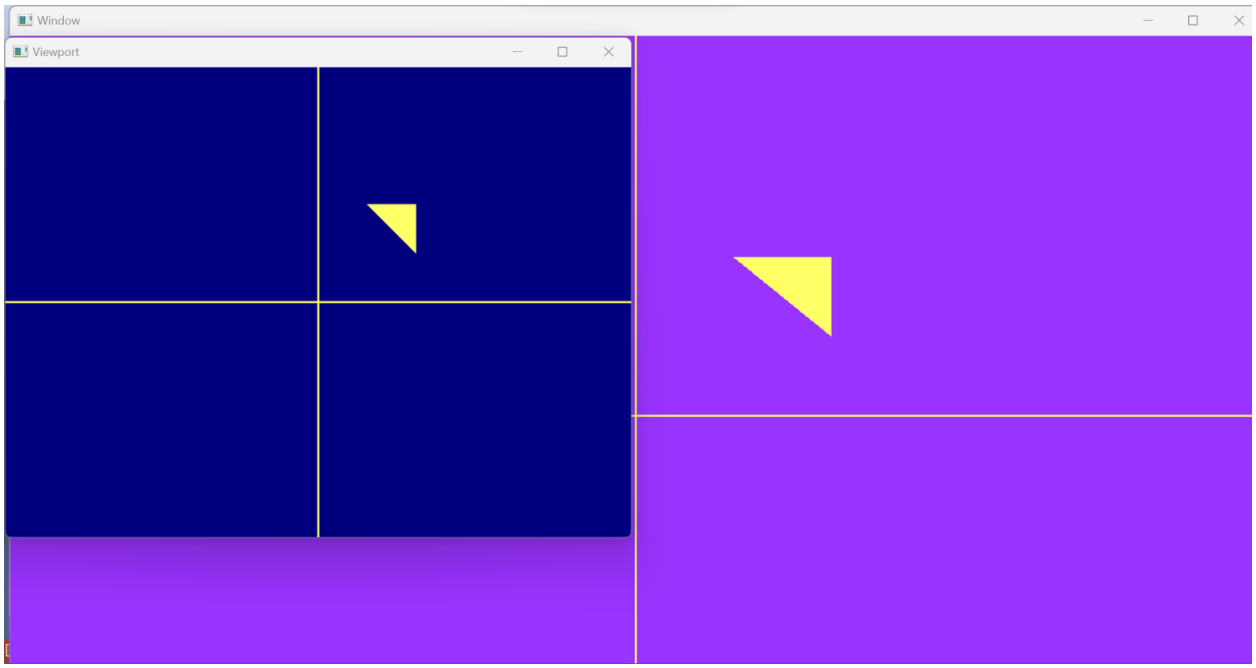
```
SCALING WRT ORIGIN
Enter scaling factor values: 3.5 3

Choose required transformation: 8
REFLECTION WRT ORIGIN

Choose required transformation: 1
TRANSLATION
Enter translation values: -20 -30

Choose required transformation: 0
ALL DONE

Transformation Matrix:
-3.5 0 -20
0 -3 -30
0 0 1

P1':
-20
-30
1

P2':
-20
-180
1

P3':
-195
-30
1
```



6b. Window to Viewport Transformation

```
Window
GET INPUTS FOR WINDOW IMAGE
Enter point P1 coords: 100 200
Homogeneous representation of P1:
100
200
1

Enter point P2 coords: 200 100
Homogeneous representation of P2:
200
100
1

Enter point P3 coords: 200 200
Homogeneous representation of P3:
200
200
1
```

```
Viewport
GETTING INPUTS FROM WINDOW IMAGE
Homogeneous representation of P1:
50
100
1

Homogeneous representation of P2:
100
50
1

Homogeneous representation of P3:
100
100
1
```



Learning outcomes:

- 2D composite transformations were applied on objects using OpenGL in C++.
- The implementation for each transformation and the composite transformation were tested for points and objects.
- Window to Viewport transformation was implemented using OpenGL in C++.

Exercise 7 – Cohen Sutherland Line Clipping

Date: 14/10/2023

Aim:

To apply Cohen Sutherland line clipping on a line (x1,y1) (x2,y2) with respect to a clipping window (XWmin,YWmin) (XWmax,YWmax).
After clipping with an edge, display the line segment with the calculated intermediate intersection points.

Algorithm:

1. Get the coordinates of the clipping window as input from the user and draw it.
2. Get the endpoints of the line from the user and draw the line.
3. Repeat until the line is trivially accepted or rejected:
    a. Compute the region codes for the endpoints.
    b. Perform bitwise OR on the region codes. If the result is 0000, the line is trivially accepted.
    c. Perform bitwise AND on the region codes. If the result is not 0000, the line is trivially rejected.
    d. Compute an intersection point the line has with the clipping window.
    e. Clip the line by setting the intersection point as one of the endpoints of the line.

Code:

```
#define GL_SILENCE_DEPRECATION

#include<GL/glut.h>
#include<iostream>
#include<math.h>
using namespace std;

float xmin, xmax, ymin, ymax;
char letter = 'A';

void myInit() {
	glClearColor(0.6, 0.2, 1.0, 0.0); // violet
	glColor3f(0.0f, 0.0f, 0.5f); //dark blue
	glPointSize(5);
	glMatrixMode(GL_PROJECTION);
	glLineWidth(1.5);
```

```
	glLoadIdentity();
	gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}

void labelPoint(float x, float y, char label) {
	//glColor4f(0, 0, 0, 1);
	glRasterPos2f(x + 320, y + 240);
	glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, label);
}

void displayPoint(float x, float y) {
	glBegin(GL_POINTS);
	glVertex2d(x + 320, y + 240);
	glEnd();
}

void displayLine(int x1, int y1, int x2, int y2) {
	glBegin(GL_LINES);
	glVertex2d(x1 + 320, y1 + 240);
	glVertex2d(x2 + 320, y2 + 240);
	glEnd();
}

void displayQuads(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) {
	glBegin(GL_QUADS);
	glVertex2d(x1 + 320, y1 + 240);
	glVertex2d(x2 + 320, y2 + 240);
	glVertex2d(x3 + 320, y3 + 240);
	glVertex2d(x4 + 320, y4 + 240);
	glEnd();
}

void drawPlane() {
	glClear(GL_COLOR_BUFFER_BIT);
	glColor4f(1, 1, 0.4, 1); //yellow
	displayLine(-320, 0, 320, 0); //x-axis
	displayLine(0, -240, 0, 240); //y-axis
	glFlush();
}

void printMatrix(float* arr, int m, int n)
{
	int i, j;
	for (i = 0; i < m; i++) {
		for (j = 0; j < n; j++)
			cout << *((arr + i * n) + j) << " ";
```

```
			cout << endl;
		}
}

void printRegionCode(float* RC) {
		printMatrix(RC, 4, 1);
}

float* getRegionCode(float x, float y) {
		float* TBRL = new float[4];

		TBRL[0] = (y > ymax) ? 1 : 0;
		TBRL[1] = (y < ymin) ? 1 : 0;
		TBRL[2] = (x > xmax) ? 1 : 0;
		TBRL[3] = (x < xmin) ? 1 : 0;

		return TBRL;
}

bool isTrivialAccept(float* OR) {
		for (int i = 0; i < 4; i++) {
			if (OR[i]==1) {
				cout << "not accept" << endl;
				return false;
			}
		}
		cout << "accept" << endl;
		return true;
}

bool isTrivialReject(float* AND) {
		for (int i = 0; i < 4; i++) {
			if (AND[i]==1) {
				cout << "reject" << endl;
				return true;
			}
		}
		cout << "not reject" << endl;
		return false;
}

int firstRegion(float* OR) {
		int i;
		for (i = 0; OR[i] == 0; i++);
		return i;
}
```

```
void plotClippingWindow() {
	glColor4f(0.8, 0.8, 0.8, 1); //grey-white
	displayQuads(xmin, ymin, xmin, ymax, xmax, ymax, xmax, ymin);
}

void plotInputLine(float x1, float y1, float x2, float y2) {
	glColor4f(0, 0, 0.5, 1); //dark blue
	displayLine(x1, y1, x2, y2);
}

void applyCSLineClipping(float x1, float y1, float x2, float y2) {
	//compute region codes of the endpoints of the line
	float* RC1 = getRegionCode(x1, y1);
	cout << "\nRegion code of P1: " << endl;
	printRegionCode(RC1);
	cout << endl;

	float* RC2 = getRegionCode(x2, y2);
	cout << "\nRegion code of P2: " << endl;
	printRegionCode(RC2);
	cout << endl;

	//clipping the line
	bool algoEnd = false;
	int green = 1;
	do {
		//compute bitwise OR and AND
		float* OR = new float[4];
		float* AND = new float[4];
		for (int i = 0; i < 4; i++) {
			OR[i] = max(RC1[i], RC2[i]);
			AND[i] = min(RC1[i], RC2[i]);
		}
		cout << "\nOR: " << endl;
		printRegionCode(OR);
		cout << "AND: " << endl;
		printRegionCode(AND);

		//check if line has been clipped
		if (isTrivialAccept(OR) || isTrivialReject(AND)) {
			cout << "\nend" << endl;
			algoEnd = true;
			//display clipped line
			glColor4f(1, 0, 0, 1); //red
			displayLine(x1, y1, x2, y2);
```

```
		}
		else {
			//compute intersection point
			int r = firstRegion(OR);
			cout << "r: " << r << endl;
			float x = x1, y = y1;

			//slope of the line
			float m = (y2 - y1) / (x2 - x1);

			//change line colour
			glColor4f(0, green, 0, 1); green -= 0.2; //updated green colour

			if (r == 0) {
				y = ymax;
				x = x1 + (1 / m) * (y - y1);

				RC1 = getRegionCode(x, y);
				x1 = x; y1 = y;
				labelPoint(x1, y1, letter++);
			}
			else if (r == 1) {
				y = ymin;
				x = x1 + (1 / m) * (y - y1);

				RC2 = getRegionCode(x, y);
				x2 = x; y2 = y;
				labelPoint(x2, y2, letter++);
			}
			else if (r == 2) {
				x = xmax;
				y = y1 + m * (x - x1);

				RC2 = getRegionCode(x, y);
				x2 = x; y2 = y;
				labelPoint(x2, y2, letter++);
			}
			else if (r == 3) {
				x = xmin;
				y = y1 + m * (x - x1);

				RC1 = getRegionCode(x, y);
				x1 = x; y1 = y;
				labelPoint(x1, y1, letter++);
			}
```

```
                        //display updated line after intersection
                        cout << "\nP1: " << x1 << " " << y1 << endl;
                        cout << "P2: " << x2 << " " << y2 << endl;
                        displayLine(x1, y1, x2, y2);
                }

        } while (!algoEnd);
}

void plotChart() {
        glClear(GL_COLOR_BUFFER_BIT);

        drawPlane();

        cout << "COHEN SUTHERLAND LINE CLIPPING" << endl;

        //plot the clipping window
        cout << "\nEnter clipping window edges: " << endl;
        cout << "x_min: "; cin >> xmin;
        cout << "x_max: "; cin >> xmax;
        cout << "y_min: "; cin >> ymin;
        cout << "y_max: "; cin >> ymax;
        plotClippingWindow();

        //plot the input line
        float x1, y1, x2, y2;
        cout << "\nEnter line endpoints: " << endl;
        cout << "P1: "; cin >> x1 >> y1;
        cout << "P2: "; cin >> x2 >> y2;
        if (x2 < x1) {
                cout << "Swapping points so that P1 lies to the left of P2..." <<
endl;
                float tx = x1, ty = y1;
                x1 = x2; y1 = y2;
                x2 = tx; y2 = ty;
        }
        plotInputLine(x1, y1, x2, y2);

        labelPoint(x1, y1, letter++);
        labelPoint(x2, y2, letter++);

        //clip the line using
        applyCSLineClipping(x1, y1, x2, y2);

        glFlush();
}
```

```
int main(int argc, char* argv[]) {
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
	glutInitWindowSize(640, 480);
	glutCreateWindow("Cohen Sutherland Line Clipping");
	glutDisplayFunc(plotChart);
	myInit();
	glutMainLoop();
	return 1;
}
```

Output:

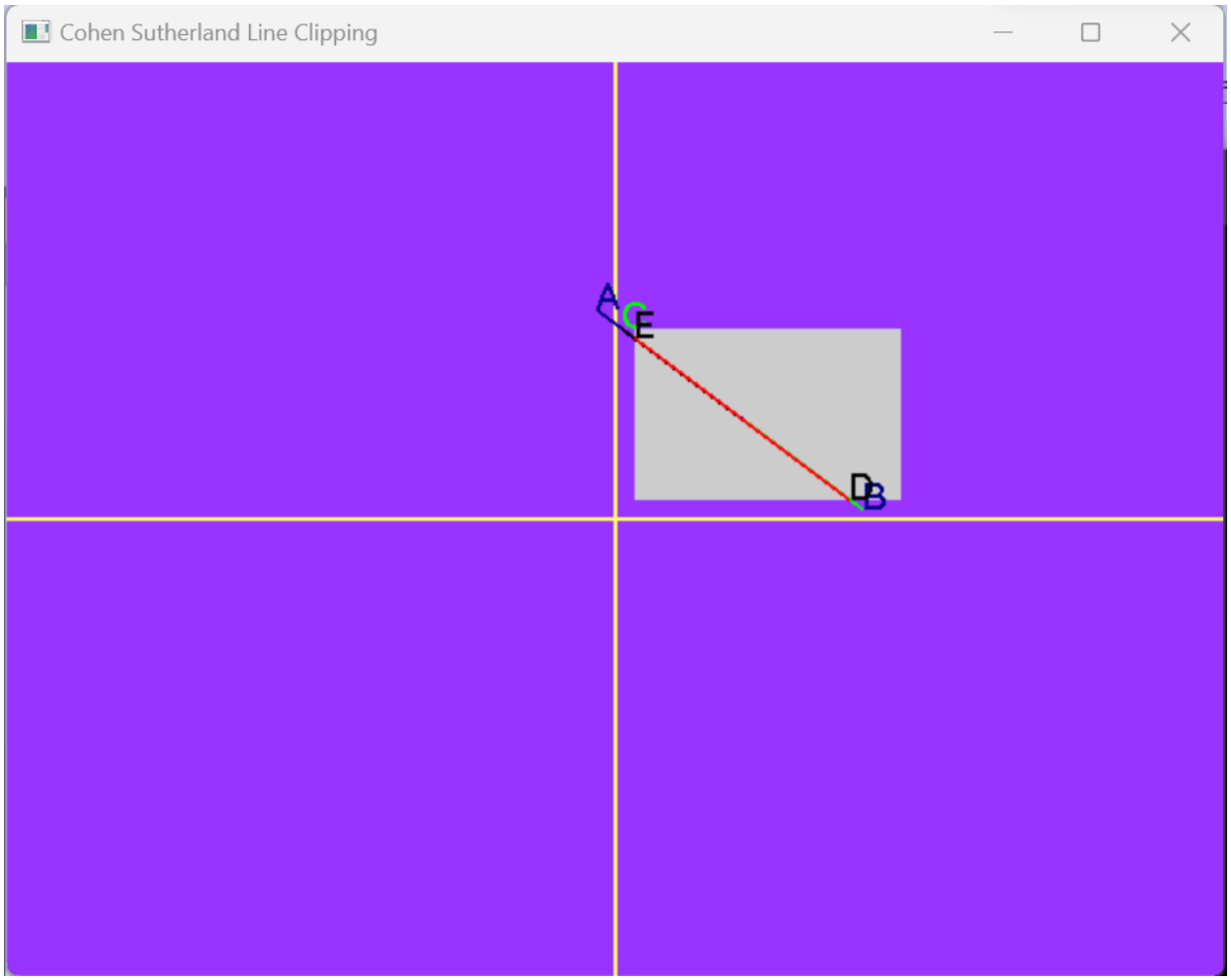
```
COHEN SUTHERLAND LINE CLIPPING

Enter clipping window edges:
x_min: 10
x_max: 150
y_min: 10
y_max: 100

Enter line endpoints:
P1: -10 110
P2: 130 5

Region code of P1:
1
0
0
1


Region code of P2:
0
1
0
0
```

```
OR:
1
1
0
1
AND:
0
0
0
0
not accept
not reject
r: 0

P1: 3.33333 100
P2: 130 5
```

```
OR:
0
1
0
1
AND:
0
0
0
0
not accept
not reject
r: 1

P1: 3.33333 100
P2: 123.333 10
```

```
OR:
0
0
0
1
AND:
0
0
0
0
not accept
not reject
r: 3

P1: 10 95
P2: 123.333 10
```

```
OR:
0
0
0
0
AND:
0
0
0
0
accept

end
```



Learning outcomes:

- The Cohen Sutherland Line Clipping Algorithm was implemented.
- The line was clipped iteratively and depicted using different colours.
- All the points were labeled.

Exercise 8 – 3-Dimensional Transformations in C++ using OpenGL

Date: 30/10/2023

Aim:

To perform the following basic 3D Transformations on any 3D Object.
1) Translation
2) Rotation
3) Scaling
Use only homogeneous coordinate representation and matrix multiplication to perform transformations.
Set the camera to any position on the 3D space. Have (0,0,0) at the center of the screen. Draw X, Y and Z axes.

Algorithm:

4. Get points of the object as input.
5. Draw the object.
6. Transform each vertex of the object using appropriate 3-D transformation matrices.
7. Draw the object with the transformed vertices.

Code:

```
#define GL_SILENCE_DEPRECATION

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

typedef float Matrix4x4 [4][4];
Matrix4x4 theMatrix;
// Initial Co-ordinates of the Cube to be Transformed
float cube[8][3]={{80,80,-100},{180,80,-100},{180,180,-100},{80,180,-100},
      {60,60,0},{160,60,0},{160,160,0},{60,160,0}};
float ptsFin[8][3];
float refptX,refptY,refptZ;                  //Reference points
float TransDistX,TransDistY,TransDistZ;      //Translations along Axes
float ScaleX,ScaleY,ScaleZ;                  //Scaling Factors along Axes
float Alpha,Beta,Gamma,Theta;                //Rotation angles about Axes
```

```
float A,B,C;                                    //Arbitrary Line Attributes
float aa,bb,cc;                                 //Arbitrary Line Attributes
float x1,y11,z1,x2,y2,z2;
int choice,choiceRot,choiceRef;

void matrixSetIdentity(Matrix4x4 m)
{
	int i, j;
	for (i=0; i<4; i++)
	for (j=0; j<4; j++)
		m[i][j] = (i == j);
}

void matrixPreMultiply(Matrix4x4 a , Matrix4x4 b){//putting result in b
	int i,j;
	Matrix4x4 tmp;
	for (i = 0; i < 4; i++)
	for (j = 0; j < 4; j++)

tmp[i][j]=a[i][0]*b[0][j]+a[i][1]*b[1][j]+a[i][2]*b[2][j]+a[i][3]*b[3][j];
	for (i = 0; i < 4; i++)
	for (j = 0; j < 4; j++)
		theMatrix[i][j] = tmp[i][j];
}
void Translate(int tx, int ty, int tz)
{
	Matrix4x4 m;
	matrixSetIdentity(m);
	m[0][3] = tx;
	m[1][3] = ty;
	m[2][3] = tz;
	matrixPreMultiply(m, theMatrix);
}
void Scale(float sx , float sy ,float sz)
{
	Matrix4x4 m;
	matrixSetIdentity(m);
	m[0][0] = sx;
	m[0][3] = (1 - sx)*refptX;
	m[1][1] = sy;
	m[1][3] = (1 - sy)*refptY;
	m[2][2] = sz;
	m[2][3] = (1 - sy)*refptZ;
	matrixPreMultiply(m , theMatrix);
}
void RotateX(float angle)
```

```
{
	Matrix4x4 m;
	matrixSetIdentity(m);
	angle = angle*22/1260;
	m[1][1] = cos(angle);
	m[1][2] = -sin(angle);
	m[2][1] = sin(angle);
	m[2][2] = cos(angle);
	matrixPreMultiply(m , theMatrix);
}
void RotateY(float angle)
{
	Matrix4x4 m;
	matrixSetIdentity(m);
	angle = angle*22/1260;
	m[0][0] = cos(angle);
	m[0][2] = sin(angle);
	m[2][0] = -sin(angle);
	m[2][2] = cos(angle);
	matrixPreMultiply(m , theMatrix);
}
void RotateZ(float angle)
{
	Matrix4x4 m;
	matrixSetIdentity(m);
	angle = angle*22/1260;
	m[0][0] = cos(angle);
	m[0][1] = -sin(angle);
	m[1][0] = sin(angle);
	m[1][1] = cos(angle);
	matrixPreMultiply(m , theMatrix);
}
void DrawRotLine(void)
{
	switch(choiceRot)
	{
	case 1:
		glBegin(GL_LINES);
		glVertex3s(-640 ,B,C);
		glVertex3s( 640 ,B,C);
		glEnd();
		break;
	case 2:
		glBegin(GL_LINES);
		glVertex3s(A ,-480 ,C);
		glVertex3s(A ,480 ,C);
```

```
			glEnd();
			break;
		case 3:
			glBegin(GL_LINES);
			glVertex3s(A ,B ,-300);
			glVertex3s(A ,B ,300);
			glEnd();
			break;
		case 4:
			glBegin(GL_LINES);
			glVertex3s(x1-aa*500 ,y11-bb*500 , z1-cc*500);
			glVertex3s(x2+aa*500 ,y2+bb*500 , z2+cc*500);
			glEnd();
			break;
		}
}
void TransformPoints(void)
{
		int i,k;
		for(k=0 ; k<8 ; k++)
		for (i=0 ; i<3 ; i++)
			ptsFin[k][i] = theMatrix[i][0]*cube[k][0] + theMatrix[i][1]*cube[k][1]
+ theMatrix[i][2]*cube[k][2] + theMatrix[i][3];
}
void Axes(void)
{
		glColor3f (0.0, 0.0, 0.0);			// Set the color to BLACK
		glBegin(GL_LINES);				// Plotting X-Axis
		glVertex2s(-640 ,0);
		glVertex2s(640,0);
		glEnd();
		glBegin(GL_LINES);				// Plotting Y-Axis
		glVertex2s(0,-480);
		glVertex2s(0 ,480);
		glEnd();
}
void Draw(float a[8][3],int original=1)			//Display the Figure
{
		int i;
		if(original)
		glColor3f (0.2, 0.4, 0.7);
		else
		glColor3f (0.2, 0.3, 0.2);
		glBegin(GL_POLYGON);
		glVertex3f(a[0][0],a[0][1],a[0][2]);
		glVertex3f(a[1][0],a[1][1],a[1][2]);
```

```
	glVertex3f(a[2][0],a[2][1],a[2][2]);
	glVertex3f(a[3][0],a[3][1],a[3][2]);
	glEnd();
	i=0;
	glBegin(GL_POLYGON);
	glVertex3s(a[0+i][0],a[0+i][1],a[0+i][2]);
	glVertex3s(a[1+i][0],a[1+i][1],a[1+i][2]);
	glVertex3s(a[5+i][0],a[5+i][1],a[5+i][2]);
	glVertex3s(a[4+i][0],a[4+i][1],a[4+i][2]);
	glEnd();
	glBegin(GL_POLYGON);
	glVertex3f(a[0][0],a[0][1],a[0][2]);
	glVertex3f(a[3][0],a[3][1],a[3][2]);
	glVertex3f(a[7][0],a[7][1],a[7][2]);
	glVertex3f(a[4][0],a[4][1],a[4][2]);
	glEnd();
	i=1;
	glBegin(GL_POLYGON);
	glVertex3s(a[0+i][0],a[0+i][1],a[0+i][2]);
	glVertex3s(a[1+i][0],a[1+i][1],a[1+i][2]);
	glVertex3s(a[5+i][0],a[5+i][1],a[5+i][2]);
	glVertex3s(a[4+i][0],a[4+i][1],a[4+i][2]);
	glEnd();
	i=2;
	glBegin(GL_POLYGON);
	glVertex3s(a[0+i][0],a[0+i][1],a[0+i][2]);
	glVertex3s(a[1+i][0],a[1+i][1],a[1+i][2]);
	glVertex3s(a[5+i][0],a[5+i][1],a[5+i][2]);
	glVertex3s(a[4+i][0],a[4+i][1],a[4+i][2]);
	glEnd();
	i=4;
	glBegin(GL_POLYGON);
	glVertex3f(a[0+i][0],a[0+i][1],a[0+i][2]);
	glVertex3f(a[1+i][0],a[1+i][1],a[1+i][2]);
	glVertex3f(a[2+i][0],a[2+i][1],a[2+i][2]);
	glVertex3f(a[3+i][0],a[3+i][1],a[3+i][2]);
	glEnd();
}

void display(void)
{
	glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	Axes();
	glColor3f (1.0, 0.0, 0.0);
	Draw(cube);
	matrixSetIdentity(theMatrix);
```

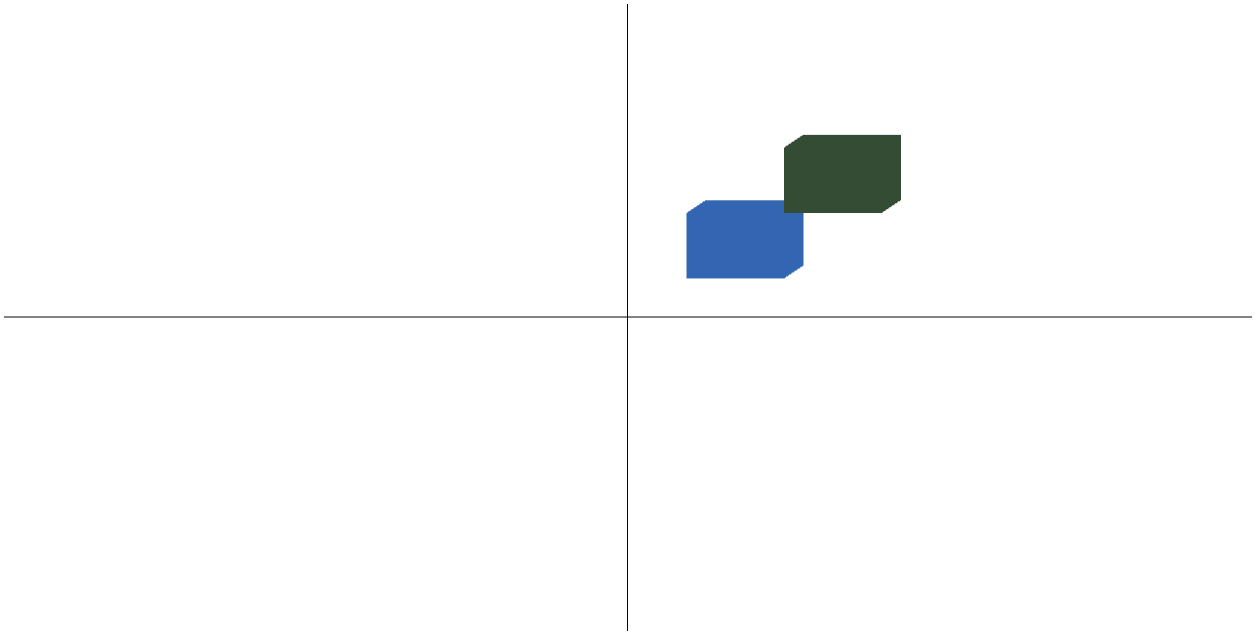
```
	switch(choice)
	{
	case 1:
		Translate(TransDistX , TransDistY ,TransDistZ);
		break;
	case 2:
		Scale(ScaleX, ScaleY, ScaleZ);
		break;
	case 3:
		switch(choiceRot)
		{
		case 1:
			DrawRotLine();
			Translate(0,-B,-C);
			RotateX(Alpha);
			Translate(0,B,C);
			break;
		case 2:
			DrawRotLine();
			Translate(-A,0,-C);
			RotateY(Beta);
			Translate(A,0,C);
			break;
		case 3:
			DrawRotLine();
			Translate(-A,-B,0);
			RotateZ(Gamma);
			Translate(A,B,0);
			break;
		case 4:
			DrawRotLine();
			float MOD =sqrt((x2-x1)*(x2-x1) + (y2-y11)*(y2-y11) +
(z2-z1)*(z2-z1));
			aa = (x2-x1)/MOD;
			bb = (y2-y11)/MOD;
			cc = (z2-z1)/MOD;
			Translate(-x1,-y11,-z1);
			float ThetaDash;
			ThetaDash = 1260*atan(bb/cc)/22;
			RotateX(ThetaDash);
			RotateY(1260*asin(-aa)/22);
			RotateZ(Theta);
			RotateY(1260*asin(aa)/22);
			RotateX(-ThetaDash);
			Translate(x1,y11,z1);
			break;
```

```
			}
		}
		TransformPoints();
		Draw(ptsFin,0);
		glFlush();
}
void init(void){
		glClearColor (1.0, 1.0, 1.0, 1.0);
		glOrtho(-640.0,640.0,-480.0,480.0,-400.0,400.0);
//https://stackoverflow.com/questions/2571402/how-to-use-glortho-in-opengl
		glEnable(GL_DEPTH_TEST);

 }
int main(int argc, char *argv[])
{
		glutInit(&argc, argv);
		glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
		glutInitWindowSize (640*2, 480*2);
		glutInitWindowPosition (0, 0);
		glutCreateWindow (" 3D Basic Transformations ");
		init ();

		printf("\n1.Translation\n2.Scaling\n3.Rotation\nEnter your choice:");
		scanf("%d",&choice);
		switch(choice)
		{
		case 1:
			printf("Enter Translation along X, Y & Z:");
			scanf("%f%f%f",&TransDistX , &TransDistY , &TransDistZ);
			break;
		case 2:
			printf("Enter Scaling ratios along X, Y & Z:");
			scanf("%f%f%f",&ScaleX , &ScaleY , &ScaleZ);
			break;
		case 3:
			printf("\n\t1.Parallel to X-axis.(y=B & z=C)\n\t2.Parallel to
Y-axis.(x=A & z=C)\n\t3.Parallel to Z-axis.(x=A & y=B)\n\t4.Arbitrary line passing
through (x1,y1,z1) &(x2,y2,z2)\n\n\tEnter your choice for Rotation about axis:");
			scanf("%d",&choiceRot);
			switch(choiceRot)
			{
			case 1:
				printf("\tEnter B & C: ");
				scanf("%f %f",&B,&C);
				printf("\tEnter Rotational Angle: ");
				scanf("%f",&Alpha);
```

```
                        break;
                case 2:
                        printf("\tEnter A & C: ");
                        scanf("%f %f",&A,&C);
                        printf("\tEnter Rotational Angle: ");
                        scanf("%f",&Beta);
                        break;
                case 3:
                        printf("\tEnter A & B: ");
                        scanf("%f %f",&A,&B);
                        printf("\tEnter Rotational Angle: ");
                        scanf("%f",&Gamma);
                        break;
                case 4:
                        printf("\tEnter values of x1 ,y1 & z1:\n");
                        scanf("%f %f %f",&x1,&y11,&z1);
                        printf("\tEnter values of x2 ,y2 & z2:\n");
                        scanf("%f %f %f",&x2,&y2,&z2);
                        printf("\tEnter Rotational Angle: ");
                        scanf("%f",&Theta);
                        break;
                }
        break;
        default:
                printf("Please enter a valid choice!!!\n");
        return 0;
        }
        glutDisplayFunc(display);
        glutMainLoop();
        return 0;
}
```

Output:

```
1.Translation
2.Scaling
3.Rotation
Enter your choice:1
Enter Translation along X, Y & Z:100 100 20
```



Learning outcomes:

- 3D functions in OpenGL were understood and implemented.
- 3D Transformations were implemented using the transformation matrices.

Exercise 9 – 3-Dimensional Projections in C++ using OpenGL

Date: 11/11/2023

Aim:

Write a menu driven program to perform Orthographic and Perspective projection on any 3D object.
Set the camera to any position on the 3D space. Have (0,0,0) at the center of the screen. Draw X , Y and Z
axis.
Use gluPerspective() to perform perspective projection. Also, can use inbuilt functions for 3D transformations.

Algorithm:

1. For orthographic projection, display the cube using `glOrtho()`
2. For perspective projection, use `gluPerspective()` and `gluLookAt()` to set the projection and camera position.
3. Enable depth testing, before entering the GLUT main loop for rendering and event handling.
4. Utilize OpenGL functions to draw a rotating cube with specified colors for edges and faces according to the chosen projection type.
5. Swap buffers to display the rendered scene.

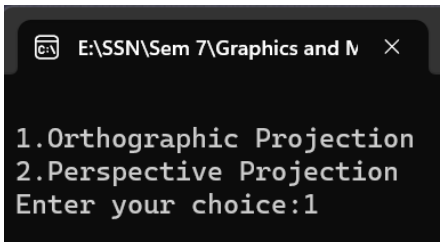
Code:

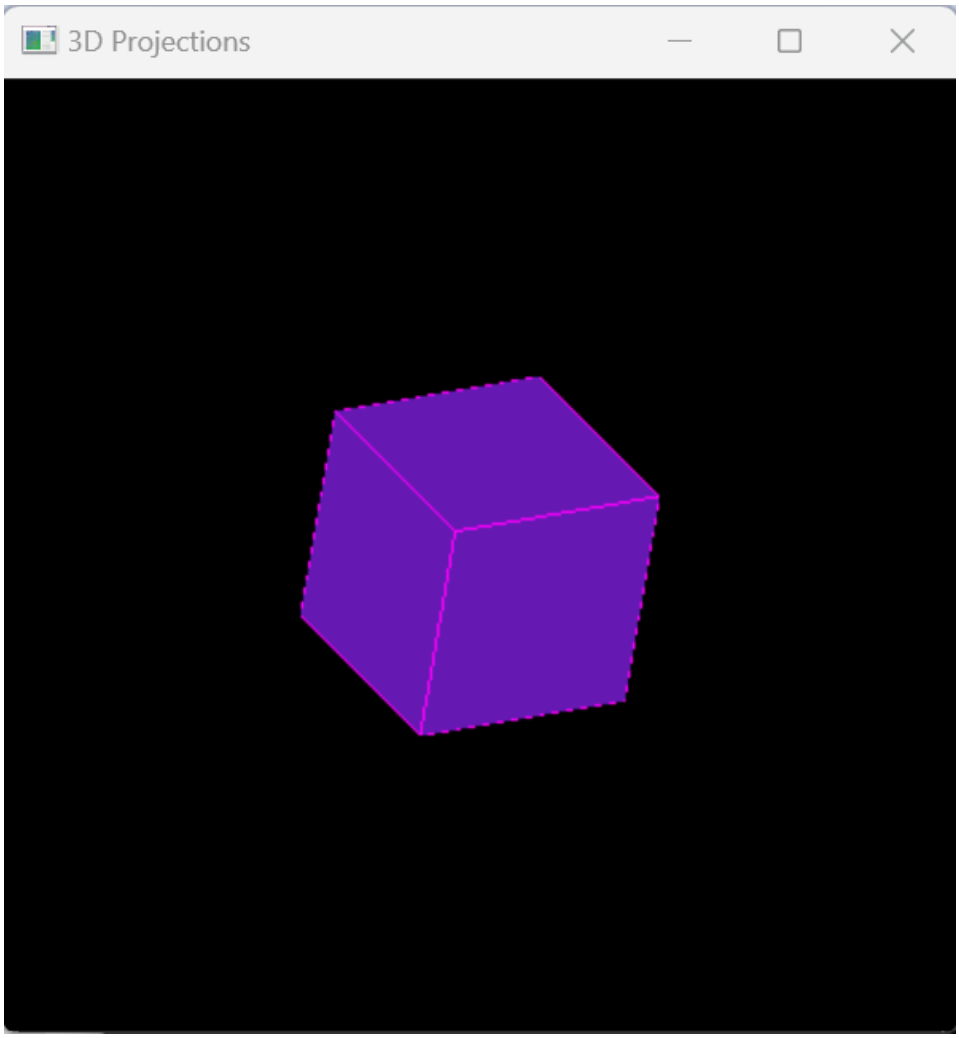
```
#define GL_SILENCE_DEPRECATION

#include <GL/glut.h>
#include <iostream>
using namespace std;

void display_ortho() {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	glOrtho(-200, 200, -200, 200, -200, 200); // Set up orthographic projection
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	// Draw the colored edges of the cube
```

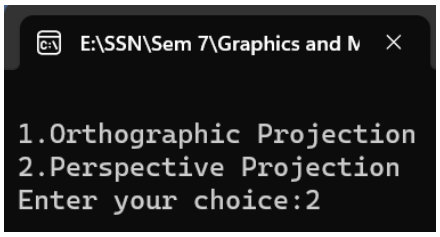
```
	glPushMatrix();
	glRotatef(45, 1, 1, 0);
	glColor3f(1.0f, 0.0f, 1.0f);
	glutWireCube(100);
	glPopMatrix();
	glPushMatrix();
	glRotatef(45, 1, 1, 0);
	glColor3f(0.4f, 0.1f, 0.7f);
	glutSolidCube(100);
	glPopMatrix();
	glutSwapBuffers();
}
void display_pers() {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(45.0f, 1.0f, 0.1f, 500.0f); // Set up perspective projection
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	gluLookAt(0, 0, 250, 0, 0, 0, 0, 1, 0); // Set the camera position
	// Draw the colored edges of the cube
	glPushMatrix();
	glRotatef(45, 1, 1, 0);
	glColor3f(1.0f, 0.0f, 1.0f);
	glutWireCube(100);
	glPopMatrix();
	// Draw the solid faces of the cube
	glPushMatrix();
	glRotatef(45, 1, 1, 0);
	glColor3f(0.4f, 0.1f, 0.7f);
	glutSolidCube(100);
	glPopMatrix();
	glutSwapBuffers();
}
int main(int argc, char** argv) {
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowSize(400, 400);
	glutCreateWindow("3D Projections");
	glEnable(GL_DEPTH_TEST);
	int choice;
	cout << "\n1.Orthographic Projection\n2.Perspective Projection\nEnter your
choice:";
	cin >> choice;
	switch (choice)
	{
```

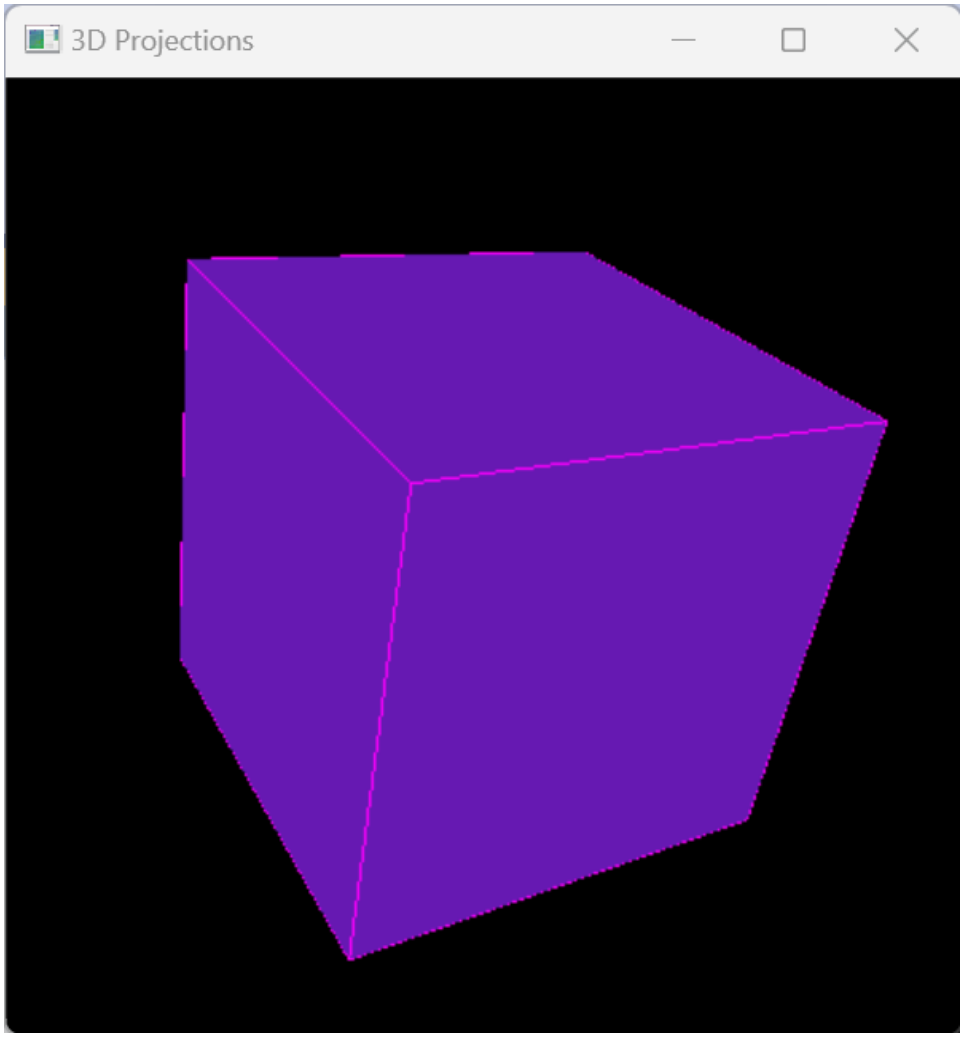
```
	case 1:
		glutDisplayFunc(display_ortho);
		break;
	case 2:
		glutDisplayFunc(display_pers);
		break;
	default:
		cout << "Please enter a valid choice!!!\n";
		return 0;
	}
	glutMainLoop();
	return 0;
}
```

Output:


E:\SSN\Sem 7\Graphics and M
1.Orthographic Projection
2.Perspective Projection
Enter your choice:1


3D Projections





Learning outcomes:

- 3D projections were understood and implemented in OpenGL.
- 3D orthographic and perspective projections were studied.

Exercise 10 – Creating a 3D Scene in C++ using OpenGL

Date: 11/11/2023

Aim:

Write a C++ program using Opengl to draw at least two 3D objects. Apply lighting and texture and render the scene.
OpenGL Functions to use:
glShadeModel()
glMaterialfv()
glLightfv()
glEnable()
glGenTextures()
glTexEnvf()
glBindTexture()
glTexParameteri()
glTexCoord2f()

Algorithm:

1. Initialise frame buffer using glutInit()
2. Set display mode as double buffer for 3D graphics with RGB colour and Depth Projections using glutInitDisplayMode()
3. Set output window size using glutWindowSize()
4. Create the output window using glutCreateWindow()
5. Call the reshape function using glutReshapeFunc()
6. Call the function to draw using glutDisplayFunc()
7. Call a function to resdisplay using glutTimerFunc() in period interval
8. Set Initial parameters of the screen
    - 8.1 Set background color using glClearColor()
    - 8.2 Setting material characteristic of objects using glMaterialfv() with parameters GL_AMBIENT_AND_DIFFUSE, GL_SPECULAR, GL_SHINENESS etc.
    - 8.3 Setting lighting characteristic of objects using glLightfv() with parameters GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_POSITION etc.
    - 8.4 Enable depth buffer, lighting, texture conditions etc using glEnable()
    - 8.5 Set the model to Flat using glShadeModel() with parameter GL_FLAT
    - 8.6 Set pixel storage mode using glPixelStorei
9. Refresh the screen repeatedly while calling the function to draw using glutMainLoop()

Function to Draw: display()

1. Clear frame buffer using glClear()
2. Set Matrix mode to ModelView using glMatrixMode()
3. Emable textre using glBindTexture() and glEnable()
4. Draw 4 different shapes using glutSolidTeacup(), glutSolidTeapot(), glutSolidCube(), glutSolidTeasSpoon() and also include glPushMatrix() and glPopMatrix() to map it on the
scene.
5. Use various other functions to define their characteristics and transformations within the scene
6. Swap Buffer using glutSwapBuffers()

Reshape Function: reshape()

1. Set viewport size using glViewport()
2. Set Matrix mode to GL_PROJECTION using glMatrixMode()
3. Load Identity Matrix using glLoadIdentity()
4. Set output window bound for all 3 axes using glOrtho()

Code:

```
#define GL_SILENCE_DEPRECATION

#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>

//Define colors
GLfloat black[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat white[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat gray[] = { 0.0, 0.0, 0.0, 0.72 };
GLfloat yellow[] = { 1.0, 1.0, 0.0, 1.0 };
GLfloat paleyellow[] = { 0.671, 0.671, 0.463, 0.871 };
GLfloat turqoise[] = { 0.063, 0.29, 0.29, 0.812 };
GLfloat direction[] = { 1.0, 1.0, 1.0, 0.0 };

//4 Components - Teapot, Tea cup, Tea Spoon, 2 sugar cubes
float ypos1 = 2.0, ypos2 = 2.5, ypos3 = 3;
float teapot_rotate = 0.2, teapot_rotate_direction = 1, teapot_posx = -1.0,
teapot_posy = 1.0, teapot_xplace = 0, teapot_yplace = 0;
float teaspoon_rotate = 0, teaspoon_posx = 0.75, teaspoon_posy = 2.5,
teaspoon_yplace = 0;
float sugar1_posx = 0.65, sugar1_posy = 2.5, sugar2_posx = 0.8, sugar2_posy =
2.75, sugar1_yplace = 0, sugar2_yplace = 0;
```

```
//Define Texture
#define blue {0x00, 0x00, 0xff}
#define cyan {0x00, 0xff, 0xff}
GLubyte texture[][3] = {
 blue, cyan,
 cyan, blue,
};

void display() {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glMatrixMode(GL_MODELVIEW);
	// GLuint texture;
	//glBindTexture(GL_TEXTURE_2D, texture);
	glPushMatrix();
	glEnable(GL_TEXTURE_2D);
	glDisable(GL_TEXTURE_2D);
	// Add a teacup to the scene.
	glPushMatrix();
	GLfloat teacup_color[] = { 0.929, 0.831, 0.055, 0.702 };
	GLfloat teacup_mat_shininess[] = { 90 };
	glMaterialfv(GL_FRONT, GL_DIFFUSE, teacup_color);
	glMaterialfv(GL_FRONT, GL_SHININESS, teacup_mat_shininess);
	glTranslatef(0.75, -0.25, 0.0);
	glutSolidTeacup(1.0);
	glPopMatrix();
	// Add a teapot to the scene.
	glPushMatrix();
	GLfloat teapot_color[] = { 0.243, 0.302, 0.639, 0.702 };
	GLfloat teapot_mat_shininess[] = { 80 };
	glMaterialfv(GL_FRONT, GL_DIFFUSE, teapot_color);
	glMaterialfv(GL_FRONT, GL_SHININESS, teapot_mat_shininess);
	glTranslatef(teapot_posx, teapot_posy, 0.0);
	glRotatef(teapot_rotate, 0, 0, 1);
	glutSolidTeapot(0.75);
	glPopMatrix();
	// Add a sugar cubes to the scene.
	glPushMatrix();
	GLfloat sugar_color[] = { 1.0, 1.0, 1.0, 1.0 };
	GLfloat sugar_mat_shininess[] = { 50 };
	glMaterialfv(GL_FRONT, GL_DIFFUSE, sugar_color);
	glMaterialfv(GL_FRONT, GL_SHININESS, sugar_mat_shininess);
	glTranslatef(sugar1_posx, sugar1_posy, 0.0);
	glRotatef(-45.0, 0, 0, 1);
	glutSolidCube(0.1);
	glPopMatrix();
```

```
	glPushMatrix();
	glTranslatef(sugar2_posx, sugar2_posy, 0.0);
	glRotatef(45.0, 0, 0, 1);
	glutSolidCube(0.1);
	glPopMatrix();
	// Add a teaspoon to the scene.
	glPushMatrix();
	GLfloat spoon_color[] = { 1.0, 1.0, 1.0, 1.0 };
	GLfloat spoon_mat_shininess[] = { 100 };
	glMaterialfv(GL_FRONT, GL_DIFFUSE, sugar_color);
	glMaterialfv(GL_FRONT, GL_SHININESS, sugar_mat_shininess);
	glTranslatef(teaspoon_posx, teaspoon_posy, 0.0);
	if (teaspoon_yplace == 0)
		glRotatef(135, 0, 1, 0);
	else
		glRotatef(teaspoon_rotate, 0, 1, 0);
	glRotatef(-60, 1, 0, 0);
	glutSolidTeaspoon(1.25);
	glPopMatrix();
	if (teapot_rotate_direction == 1 && teapot_rotate > -30.0) teapot_rotate -=
		0.5;
	if (teapot_rotate_direction == 1 && teapot_rotate <= -30.0)
		teapot_rotate_direction = -1;
	if (teapot_rotate_direction == -1 && teapot_rotate < 0) teapot_rotate +=
		0.5;
	if (teapot_rotate_direction == -1 && teapot_rotate >= 0)
		teapot_rotate_direction = 0;
	if (ypos3 >= 0.0)
		ypos3 -= 0.2;
	if (ypos2 >= -0.5)
		ypos2 -= 0.2;
	if (ypos1 >= -1.15)
		ypos1 -= 0.2;
	if (teapot_rotate_direction == 0) { //Drop sugar cubes and land tea pot only
after tea pot has become vertical
			if (teapot_posx > -1 && teapot_xplace == 0) teapot_posx -=
0.05;
		if (teapot_posx <= -1) teapot_xplace = 1;
		if (teapot_posy > 0 && teapot_yplace == 0) teapot_posy -= 0.05;
		if (teapot_posy <= -1) teapot_yplace = 1;

		if (sugar1_posy > -0.5 && sugar1_yplace == 0) sugar1_posy -= 0.05;
		if (sugar1_posy <= -0.5) sugar1_yplace = 1;
		if (sugar2_posy > -0.5 && sugar2_yplace == 0) sugar2_posy -= 0.05;
		if (sugar2_posy <= -0.5) sugar2_yplace = 1;
	}
```

```
	if (sugar1_yplace == 1 && sugar2_yplace == 1) { //Drop spoon only after sugar cubes are dropped
		if (teaspoon_posy > -0.5 && teaspoon_yplace == 0) teaspoon_posy -=
			0.05; //-0.5
		if (teaspoon_posy <= -0.5)
			teaspoon_yplace = 1;
	}
	if (teaspoon_yplace == 1) {
		teaspoon_rotate += 6.0;
		if (teaspoon_rotate == 360.0) teaspoon_yplace = 0;
	}
	glutSwapBuffers();
}
void reshape(GLint w, GLint h) {
	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION);
	GLfloat aspect = GLfloat(w) / GLfloat(h);
	glLoadIdentity();
	glOrtho(-2.5, 2.5, -2.5 / aspect, 2.5 / aspect, -10.0, 10.0);
}
void init() {
	glClearColor(0.871, 0.871, 0.871, 0);
	glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, white);
	glMaterialfv(GL_FRONT, GL_SPECULAR, white);
	glMaterialf(GL_FRONT, GL_SHININESS, 35);
	glLightfv(GL_LIGHT0, GL_AMBIENT, gray);
	glLightfv(GL_LIGHT0, GL_DIFFUSE, paleyellow);
	glLightfv(GL_LIGHT0, GL_SPECULAR, white);
	glLightfv(GL_LIGHT0, GL_POSITION, direction);
	glEnable(GL_LIGHTING); // so the renderer considers light
	glEnable(GL_LIGHT0); // turn LIGHT0 on
	glEnable(GL_DEPTH_TEST); // so the renderer considers depth
	glShadeModel(GL_FLAT);
	glEnable(GL_TEXTURE_2D);
	glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
	glTexImage2D(GL_TEXTURE_2D,
		0, // level 0
		3, // use only R, G, and B components
		2, 2, // texture has 2x2 texels
		0, // no border
		GL_RGB, // texels are in RGB format
		GL_UNSIGNED_BYTE, // color components are unsigned bytes
		texture);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

```
	glRotatef(20.0, 1.0, 0.0, 0.0);
}
void sceneDemo(int v)
{
	glutPostRedisplay();
	glutTimerFunc(1000 / 24, sceneDemo, 0);
}
int main(int argc, char** argv) {
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowPosition(80, 80);
	glutInitWindowSize(800, 600);
	glutCreateWindow("Exercise 10");
	glutReshapeFunc(reshape);
	glutDisplayFunc(display);
	glutTimerFunc(1000, sceneDemo, 0);
	init();
	glutMainLoop();
}
```

Output:

Learning outcomes:

- 3D functions were understood and implemented in OpenGL.
- 3D effects and textures were studied.

Exercise 11 – Image Editing and Manipulation

Date: 09/11/2023

Aim:

a) Using GIMP, include an image and apply filters, noise and masks.
b) Using GIMP, create a GIF animated image.

Algorithm:

1. Open an image in GIMP.
2. Apply filters, noise and masks of your choice.
3. Add layers for animation and optimise the image for animation.

Image:

Output:

Filters:

Lens blur

Pixelise

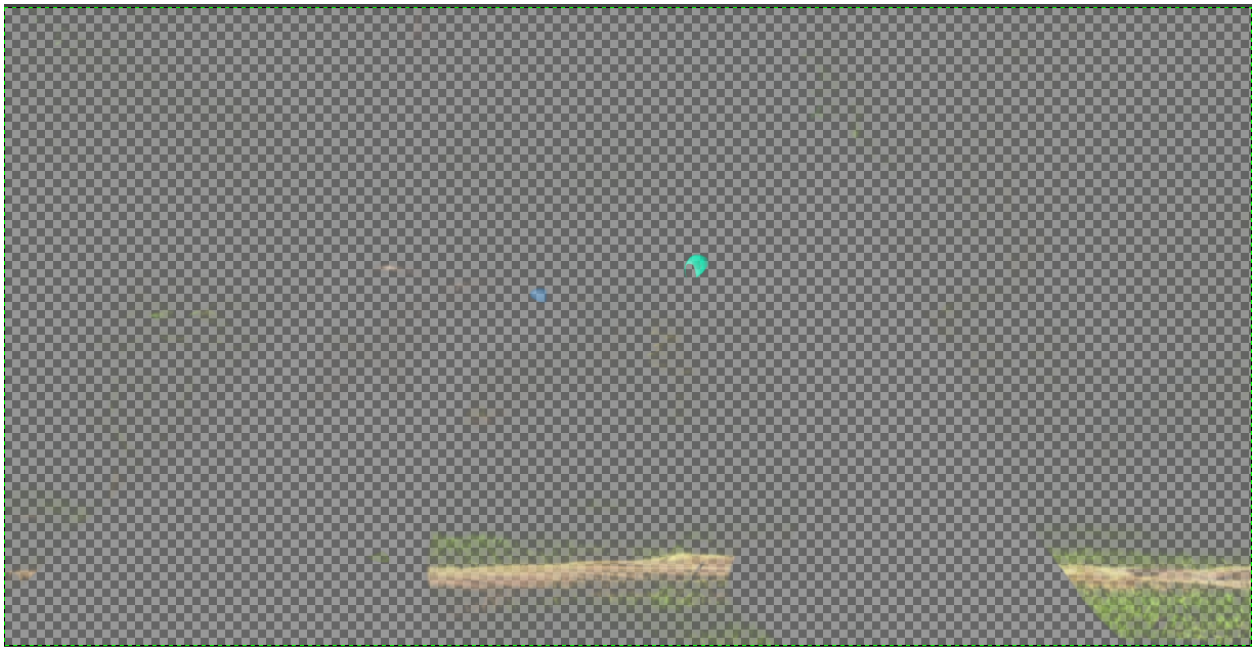
Noise:

RGB Noise

Spread

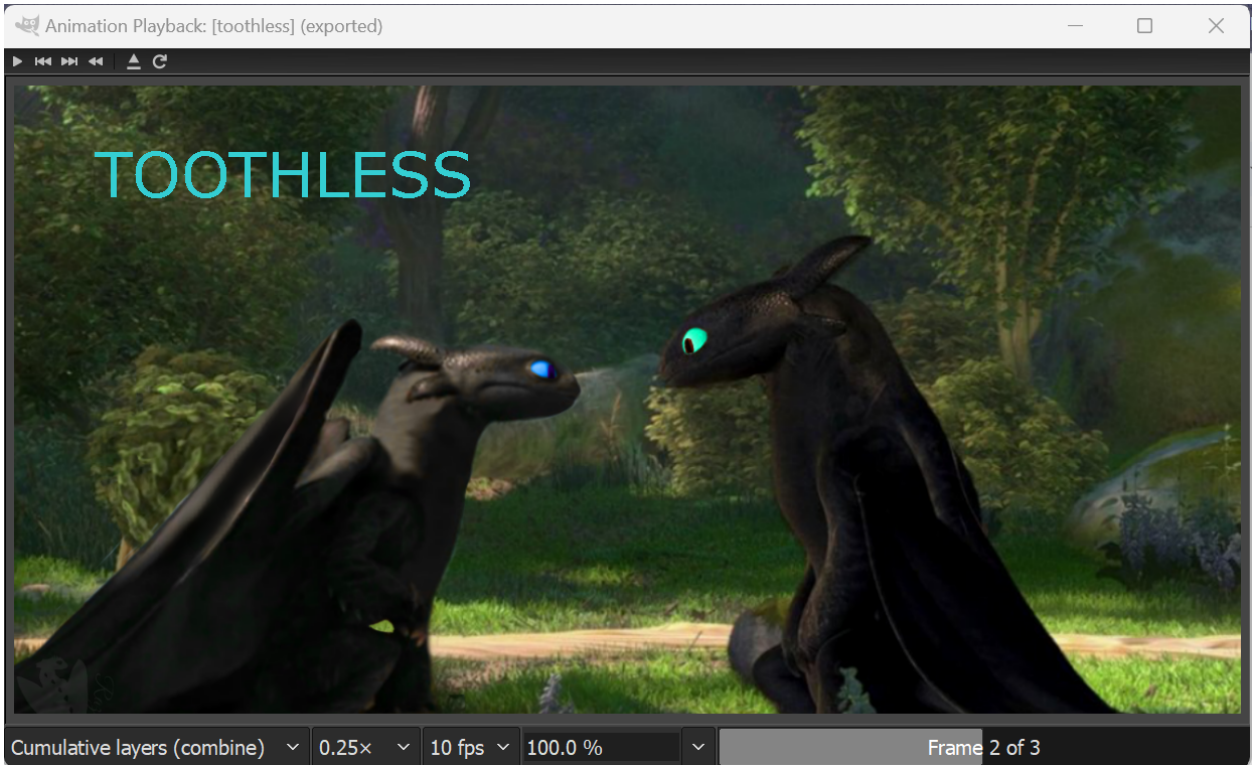
Masks:

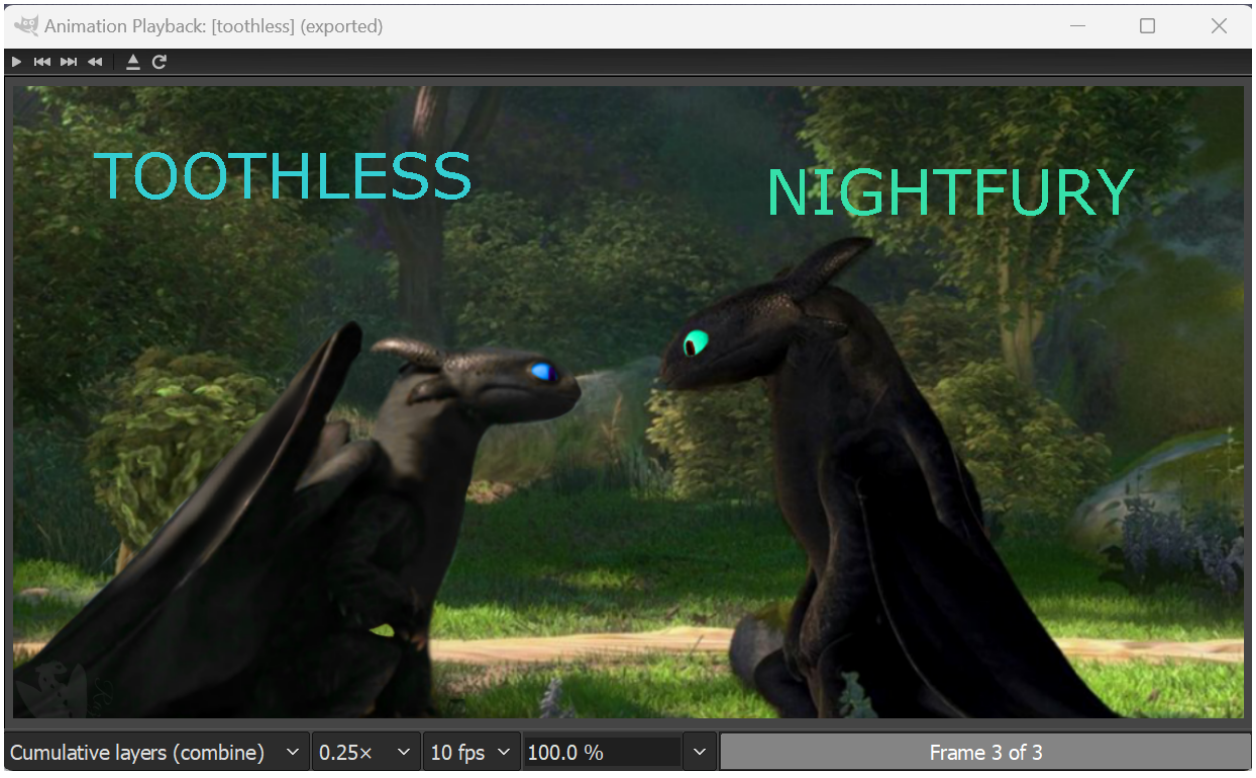
Grayscale Layer



Red Layer

Animation:


Animation Playback: [toothless] (exported)
TOOTHLESS
Cumulative layers (combine)
0.25×
10 fps
100.0 %
Frame 2 of 3


Animation Playback: [toothless] (exported)
TOOTHLESS
NIGHTFURY
Cumulative layers (combine)
0.25×
10 fps
100.0 %
Frame 3 of 3

Learning outcomes:

- GIMP software was explored.
- Filters, noise and masks were included in the image.
- A GIF animated image was created using GIMP.

Exercise 12 – Creating 2D Animation

Date: 11/11/2023

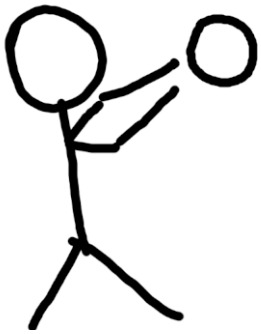
Aim:

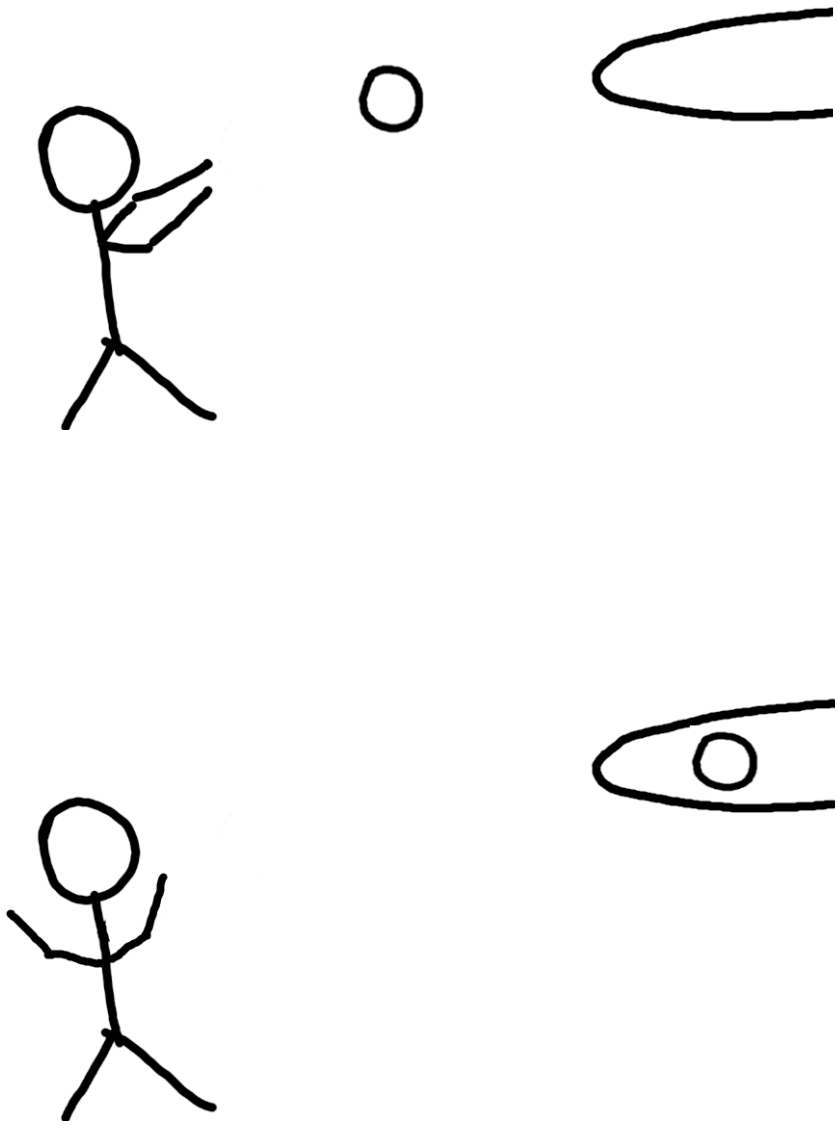
Using GIMP, include layers and create a simple animation of your choice.

Algorithm:

1. Draw an image in GIMP.
2. With that as background, draw more layers for scenes in the animation.
3. Optimise the image for animation.

Output:





Learning outcomes:

- A background image was drawn and scenes were included as layers in the image.
- A GIF animated image was created using GIMP.
