Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VII

25 August 2021

UCS 1712 - Graphics And Multimedia Lab

Exercise 5: 2D Transformations in C++ using OpenGL

Aim:

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

- Translation
- Rotation
  - About Origin
  - With Respect to a fixed point (x_r, y_r)
- Scaling with respect to
  - Origin - Uniform vs. Differential Scaling
  - Fixed Point (x_f, y_f)

- Reflection with respect to
  - X - Axis
  - Y - Axis
  - Origin
  - The Line $X = Y$
- Shearing
  - X - Direction Shear
  - Y - Direction Shear

Note: Use Homogeneous coordinate representations and matrix multiplication to perform transformations. Divide the output window into four quadrants.
(Use LINES primitive to draw X & Y Axes)

Code: 2D Transformations:

```
//To perform 2D Transformations on objects and to render the final output along with the original object
//Translation, Rotation, Reflection, Scaling, Shearing

#include <windows.h>
#include <stdio.h>
#include <math.h>
#include <GL/glut.h>

const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 800;
const int X_MIN = -400;
const int X_MAX = 400;
const int Y_MIN = -400;
const int Y_MAX = 400;

enum Axes {xAxis, yAxis};
enum Lines {XAxis, YAxis, Origin, XEqualsY};
enum Transformations {  Translation = 1, Rotation = 2, RotationAboutPivot = 3, ReflectAboutX = 4,
                        ReflectAboutY = 5, ReflectAboutO = 6, ReflectAboutXEqY = 7, UniformScale = 8,
                        DifferentialScale = 9, ScaleAboutFixed = 10, ShearAboutX = 11, ShearAboutY = 12,
                        ShearAboutXRef = 13, ShearAboutYRef = 14, ClearTransforms = 15, ChangePolygon = 16, Refresh = 17};

class Point{
private:
    GLdouble x, y, h;

public:
    Point(){
        x = y = 0;
        h = 1;
    }

    Point(GLint xCoord, GLint yCoord){
        x = xCoord;
        y = yCoord;
        h = 1;
    }

    Point(GLint xCoord, GLint yCoord, GLint H){
        x = xCoord;
        y = yCoord;
        h = H;
    }

    void setCoords(GLint xCoord, GLint yCoord){
        x = xCoord;
        y = yCoord;
    }

    GLdouble getX() const{
        return x;
    }

    GLdouble getY() const{
        return y;
    }

    GLdouble getH() const{
        return h;
    }

    GLdouble getHomogenousX() const{
        return x * h;

    }
    GLdouble getHomogenousY() const{
        return y * h;
    }

    Point getTranslatedPoint(Point translationVector){
        //For 2D Translation about a given translation vector

        double translationMatrix[3][3] = {  {1, 0, translationVector.getHomogenousX()},
                                            {0, 1, translationVector.getHomogenousY()},
                                            {0, 0, 1}};

        double values[3];

        for(int i = 0; i < 3; i++){
            values[i] = translationMatrix[i][0] * getHomogenousX() +
                        translationMatrix[i][1] * getHomogenousY() +
                        translationMatrix[i][2] * getH();
        }

        return Point(values[0]/h, values[1]/h, values[2]);
    }

    Point getRotatedPoint(int rotationAngle, Point pivot = Point(0, 0, 1)){
        //For 2D Rotation about a given pivot by a given rotation angle

        double rotationAngleInRadians = rotationAngle * 3.14159/180;
        double cosAngle = cos(rotationAngleInRadians);
        double sinAngle = sin(rotationAngleInRadians);

        double xPivotValue = (pivot.getX() * (1 - cosAngle)) + (pivot.getY() * sinAngle);
        double yPivotValue = (pivot.getY() * (1 - cosAngle)) - (pivot.getX() * sinAngle);

        double rotationMatrix[3][3] = { {cosAngle, -sinAngle, xPivotValue},
                                        {sinAngle, cosAngle, yPivotValue},
                                        {0, 0, 1}};

        double values[3];

        for(int i = 0; i < 3; i++){
            values[i] = rotationMatrix[i][0] * getHomogenousX() +
                        rotationMatrix[i][1] * getHomogenousY() +
                        rotationMatrix[i][2] * getH();
        }

        return Point(values[0]/h, values[1]/h, values[2]);
    }

    Point getReflectionAboutXAxis(){
        //For 2D Reflection about the X axis

        double reflectionMatrix[3][3] = {   {1, 0, 0},
                                            {0, -1, 0},
                                            {0, 0, 1}};

        double values[3];

        for(int i = 0; i < 3; i++){
            values[i] = reflectionMatrix[i][0] * getHomogenousX() +
                        reflectionMatrix[i][1] * getHomogenousY() +
                        reflectionMatrix[i][2] * getH();
        }

        return Point(values[0]/h, values[1]/h, values[2]);
    }

    Point getReflectionAboutYAxis(){
        //For 2D Reflection about the Y axis

        double reflectionMatrix[3][3] = {   {-1, 0, 0},
                                            {0, 1, 0},
                                            {0, 0, 1}};

        double values[3];

        for(int i = 0; i < 3; i++){
            values[i] = reflectionMatrix[i][0] * getHomogenousX() +
                        reflectionMatrix[i][1] * getHomogenousY() +
                        reflectionMatrix[i][2] * getH();
        }

        return Point(values[0]/h, values[1]/h, values[2]);
    }

    Point getReflectionAboutOrigin(){
        //For 2D Reflection about the Origin

        double reflectionMatrix[3][3] = {   {-1, 0, 0},
                                            {0, -1, 0},
                                            {0, 0, 1}};

        double values[3];

        for(int i = 0; i < 3; i++){
            values[i] = reflectionMatrix[i][0] * getHomogenousX() +
                        reflectionMatrix[i][1] * getHomogenousY() +
                        reflectionMatrix[i][2] * getH();
        }

        return Point(values[0]/h, values[1]/h, values[2]);
    }

    Point getReflectionAboutXEqualsY(){
        //For 2D Reflection about the line X=Y

        double reflectionMatrix[3][3] = {   {0, 1, 0},
                                            {1, 0, 0},
                                            {0, 0, 1}};

        double values[3];

        for(int i = 0; i < 3; i++){
            values[i] = reflectionMatrix[i][0] * getHomogenousX() +
                        reflectionMatrix[i][1] * getHomogenousY() +
                        reflectionMatrix[i][2] * getH();
        }

        return Point(values[0]/h, values[1]/h, values[2]);
    }

    Point getScaledPoint(double ScaleX, double ScaleY, Point fixed){
        //For 2D Scaling about a fixed point and scale factors for X & Y axes

        double xFixedValue = fixed.getX() * (1 - ScaleX);
        double yFixedValue = fixed.getY() * (1 - ScaleY);

        double scalingMatrix[3][3] = {  {ScaleX, 0, xFixedValue},
                                        {0, ScaleY, yFixedValue},
                                        {0, 0, 1}};

        double values[3];

        for(int i = 0; i < 3; i++){
            values[i] = scalingMatrix[i][0] * getHomogenousX() +
                        scalingMatrix[i][1] * getHomogenousY() +
                        scalingMatrix[i][2] * getH();
        }

        return Point(values[0]/h, values[1]/h, values[2]);
    }

    Point getShearAboutXAxis(double shearParam, double yRefLine = 0){
        //For shearing about X axis

        double shearMatrix[3][3] = {{1, shearParam, -shearParam * yRefLine},
                                    {0, 1, 0},
                                    {0, 0, 1}};

        double values[3];

        for(int i = 0; i < 3; i++){
            values[i] = shearMatrix[i][0] * getHomogenousX() +
                        shearMatrix[i][1] * getHomogenousY() +
                        shearMatrix[i][2] * getH();
        }

        return Point(values[0]/h, values[1]/h, values[2]);
    }

    Point getShearAboutYAxis(double shearParam, double xRefLine = 0){
        //For shearing about Y axis

        double shearMatrix[3][3] = {{1, 0, -shearParam * xRefLine},
                                    {shearParam, 1, 0},
                                    {0, 0, 1}};

        double values[3];

        for(int i = 0; i < 3; i++){
            values[i] = shearMatrix[i][0] * getHomogenousX() +
                        shearMatrix[i][1] * getHomogenousY() +
                        shearMatrix[i][2] * getH();
        }

        return Point(values[0]/h, values[1]/h, values[2]);
    }
};


class PolygonShape{
private:
    int numVertices;
    Point *points;

public:
    PolygonShape(){
        numVertices = 0;
    }

    PolygonShape(int noVertices){
        numVertices = noVertices;
        points = new Point[numVertices];
    }

    int getVertexCount() const{
        return numVertices;
    }

    Point getPoint(int i){
        return points[i];
    }

    void setVertices(int noVertices){
        numVertices = noVertices;
        points = new Point[numVertices];
    }

    void setPoint(int i, GLint x, GLint y){
        points[i].setCoords(x, y);
    }
};


void initializeDisplay();
void plotComponents();
void dummyFunction();
void dummyKeyFunction(unsigned char key, int x, int y);
void transformationMenu(int option);
void plotTransformation();
void drawAxes();
void drawPolygon(PolygonShape polygon);
void translatePolygon(PolygonShape polygon, Point translationVector);
void reflectPolygon(PolygonShape polygon, Lines line);
void rotatePolygon(PolygonShape polygon, int rotationAngle, Point pivot =
    Point(0, 0, 1));
```

```
void scalePolygon(PolygonShape polygon, double scaleX, double scaleY, Point fixed = Point(0, 0, 1));
void shearPolygon(PolygonShape polygon, Axes axis, double shearParam, double refLine = 0);

PolygonShape polygon;           //Global PolygonShape object to be plotted on the graph
int chosenTransformation = 0;   //Global variable to keep track of chosen transformation

int main(int argc, char **argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow("2D Transformations - Examples");

    printf("\n------[2D TRANSFORMATIONS]------\n");
    printf("\nUsage:\tRight-Click the GLUT Window to select a transformation.");
    printf("\n\tProvide input for the necessary parameters in this window.");
    printf("\n\tRefresh the output window if it becomes unresponsive during console I/O.");
    printf("\n\n------[2D TRANSFORMATIONS]------\n\n");

    //Set the initial default polygon for the graph
    polygon.setVertices(4);
    polygon.setPoint(0, 0, 0);
    polygon.setPoint(1, 0, 50);
    polygon.setPoint(2, 100, 50);
    polygon.setPoint(3, 100, 0);

    initializeDisplay();
    glutDisplayFunc(dummyFunction);
    plotComponents();

    glutCreateMenu(transformationMenu);
    glutAddMenuEntry("Translation", 1);
    glutAddMenuEntry("Rotation", 2);
    glutAddMenuEntry("Rotation About Pivot Point", 3);
    glutAddMenuEntry("Reflection About X Axis", 4);
    glutAddMenuEntry("Reflection About Y Axis", 5);
    glutAddMenuEntry("Reflection About Origin", 6);
    glutAddMenuEntry("Reflection About X = Y", 7);
    glutAddMenuEntry("Uniform Scaling", 8);
    glutAddMenuEntry("Differential Scaling", 9);
    glutAddMenuEntry("Scaling About Fixed Point", 10);
    glutAddMenuEntry("Shear About X Axis", 11);
    glutAddMenuEntry("Shear About Y Axis", 12);
    glutAddMenuEntry("Shear About X Axis About Y = y", 13);
    glutAddMenuEntry("Shear About Y Axis About X = x", 14);
    glutAddMenuEntry("Clear Transformations", 15);
    glutAddMenuEntry("Change Polygon", 16);
    glutAddMenuEntry("Refresh Screen", 17);
    glutAttachMenu(GLUT_RIGHT_BUTTON);

    glutMainLoop();

    return 1;
}

void transformationMenu(int option){
    chosenTransformation = option;
    plotTransformation();
}

void initializeDisplay(){
    //Initialize the display parameters

    glClearColor(1, 1, 1, 0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(X_MIN, X_MAX, Y_MIN, Y_MAX);
    glClear(GL_COLOR_BUFFER_BIT);   //Clear the display window

    glEnable(GL_BLEND);     //enable blending (translucent colors)
    glDepthMask(GL_FALSE);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);  //set the blend
   function for translucency
}

void plotComponents(){
    //Plot the axes and the base polygon

    glClear(GL_COLOR_BUFFER_BIT);   //Clear the display window
    drawAxes();
    drawPolygon(polygon);
    glFlush();
}

void dummyFunction(){
    //Placeholder function to be called in glutDisplayFunc
}

void plotTransformation(){
    //Plot the specified transformation

    switch(chosenTransformation){
        case Translation:{
            double x, y;
            printf("\n\n------[TRANSLATION]------\n");
            printf("\n\tEnter the Translation Vector Magnitudes: ");

            printf("\n\t\tX Component: ");
            scanf("%lf", &x);
            printf("\n\t\tY Component: ");
            scanf("%lf", &y);
            Point translationVector(x, y, 1);
            translatePolygon(polygon, translationVector);

            printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
            break;
        }

        case Rotation:{
            double rotationAngle;
            printf("\n\n------[ROTATION]------\n");
            printf("\n\tEnter the Rotation Angle: ");

            scanf("%lf", &rotationAngle);
            rotatePolygon(polygon, rotationAngle);

            printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
            break;
        }

        case RotationAboutPivot:{
            double rotationAngle, x, y;
            printf("\n\n------[ROTATION ABOUT PIVOT]------\n");
            printf("\n\tEnter the Rotation Angle: ");

            scanf("%lf", &rotationAngle);
            printf("\n\tEnter Pivot Point: ");
            printf("\n\t\tEnter X Coordinate: ");
            scanf("%lf", &x);
            printf("\n\t\tEnter Y Coordinate: ");
            scanf("%lf", &y);
            rotatePolygon(polygon, rotationAngle, Point(x, y, 1));

            printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
            break;
        }

        case ReflectAboutX:{
            printf("\n\n------[REFLECTION ABOUT X AXIS]------\n");

            reflectPolygon(polygon, XAxis);

            printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
            break;
        }

        case ReflectAboutY:{
```

```c
            printf("\n\n------[REFLECTION ABOUT Y AXIS]------\n");

            reflectPolygon(polygon, YAxis);

            printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
            break;
        }

        case ReflectAboutO:{
            printf("\n\n------[REFLECTION ABOUT ORIGIN]------\n");

            reflectPolygon(polygon, Origin);

            printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
            break;
        }

        case ReflectAboutXEqY:{
            printf("\n\n------[REFLECTION ABOUT X = Y]------\n");

            reflectPolygon(polygon, XEqualsY);

            printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
            break;
        }

        case UniformScale:{
            double scaleFactor;
            printf("\n\n------[UNIFORM SCALING]------\n");

            printf("\n\tEnter the Scaling Factors: ");
            scanf("%lf", &scaleFactor);

            scalePolygon(polygon, scaleFactor, scaleFactor);

            printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
            break;
        }

        case DifferentialScale:{
            double xScale, yScale;
            printf("\n\n------[DIFFERENTIAL SCALING]------\n");

            printf("\n\tEnter the Scaling Factors: ");

            printf("\n\t\tX Scale Factor: ");
            scanf("%lf", &xScale);
            printf("\n\t\tY Scale Factor: ");
            scanf("%lf", &yScale);

            scalePolygon(polygon, xScale, yScale);

            printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
            break;
        }

        case ScaleAboutFixed:{
            double xScale, yScale, xFixed, yFixed;
            printf("\n\n------[SCALING ABOUT FIXED POINT]------\n");

            printf("\n\tEnter the Scaling Factors: ");

            printf("\n\t\tX Scale Factor: ");
            scanf("%lf", &xScale);
            printf("\n\t\tY Scale Factor: ");
            scanf("%lf", &yScale);

            printf("\n\tEnter the Fixed Point: ");

            printf("\n\t\tX Coordinate: ");
            scanf("%lf", &xFixed);
            printf("\n\t\tY Coordinate: ");
            scanf("%lf", &yFixed);

            scalePolygon(polygon, xScale, yScale, Point(xFixed, yFixed, 1)
);

            printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
            break;
        }

        case ShearAboutX:{
            double shearParam;
            printf("\n\n------[SHEARING ABOUT X AXIS]------\n");

            printf("\n\tEnter the Shearing Parameter: ");
            scanf("%lf", &shearParam);

            shearPolygon(polygon, xAxis, shearParam);

            printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
            break;
        }

        case ShearAboutY:{
            double shearParam;
            printf("\n\n------[SHEARING ABOUT Y AXIS]------\n");

            printf("\n\tEnter the Shearing Parameter: ");
            scanf("%lf", &shearParam);

            shearPolygon(polygon, yAxis, shearParam);
```

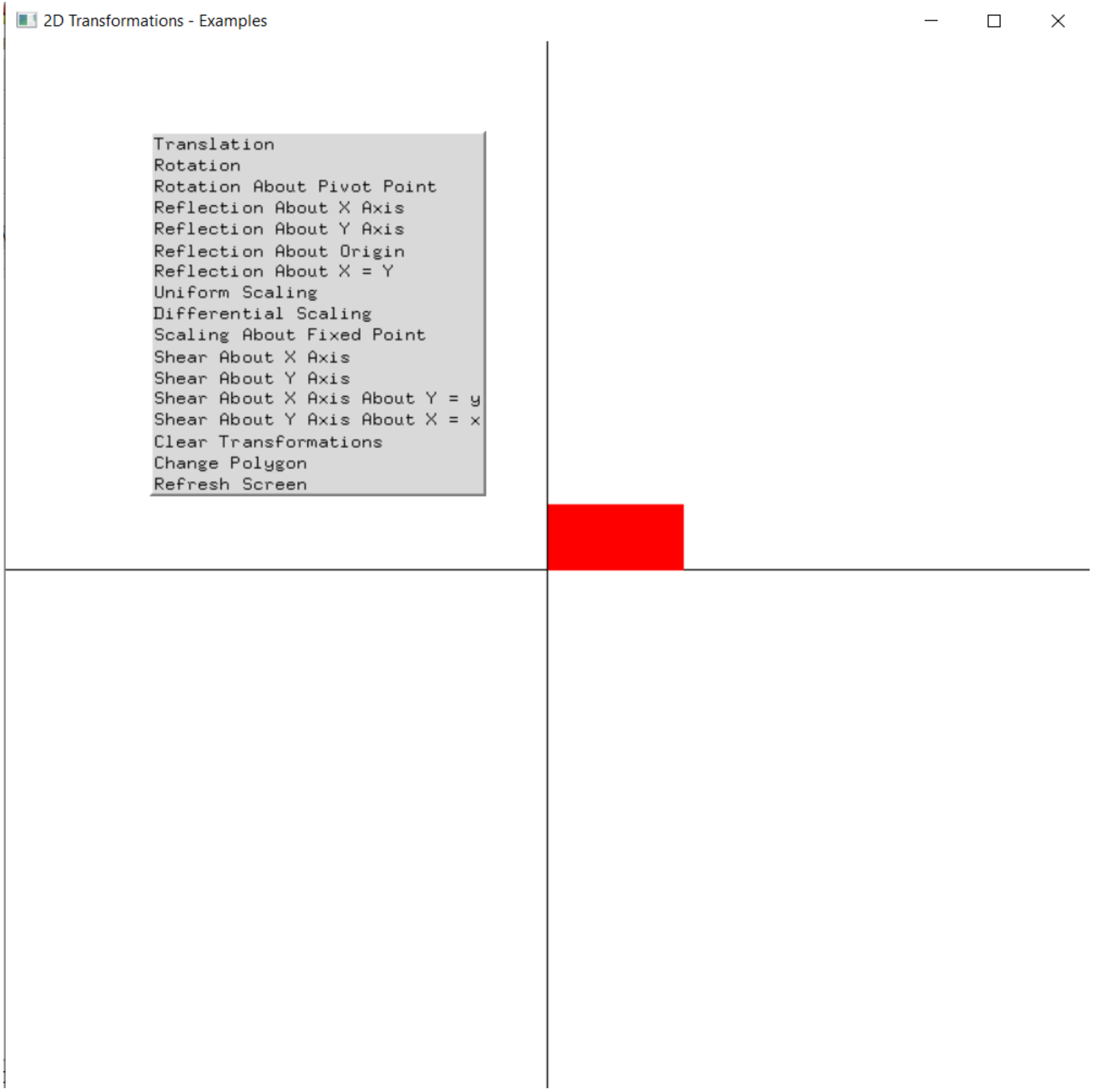
```c

        printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
        break;
    }

    case ShearAboutXRef:{
        double shearParam, yRef;
        printf("\n\n------[SHEARING ABOUT X AXIS ABOUT REF. LINE Y = y]------\n");

        printf("\n\tEnter the Shearing Parameter: ");
        scanf("%lf", &shearParam);

        printf("\n\tEnter the Reference Line Constant y (Y = y): ");
        scanf("%lf", &yRef);

        shearPolygon(polygon, xAxis, shearParam, yRef);

        printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
        break;
    }

    case ShearAboutYRef:{
        double shearParam, xRef;
        printf("\n\n------[SHEARING ABOUT Y AXIS ABOUT REF. LINE X = x]------\n");

        printf("\n\tEnter the Shearing Parameter: ");
        scanf("%lf", &shearParam);

        printf("\n\tEnter the Reference Line Constant x (X = x): ");
        scanf("%lf", &xRef);

        shearPolygon(polygon, yAxis, shearParam, xRef);

        printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
        break;
    }

    case ClearTransforms:{
        plotComponents();   //Re plot the base graph
        break;
    }

    case ChangePolygon:{
        int i = 0, vertices = 0;
        double x, y;
        printf("\n\n------[CHANGE POLYGON]------\n");

        printf("\n\tEnter the number of vertices: ");
        scanf("%d", &vertices);
```

```

            polygon.setVertices(vertices);

            while(i < vertices){
                printf("\n\tEnter Vertex %d Coordinates:", i+1);
                printf("\n\t\tX: ");
                scanf("%lf", &x);
                printf("\n\t\tY: ");
                scanf("%lf", &y);
                polygon.setPoint(i, x, y);
                i++;
            }

            plotComponents();   //Re plot the base graph

            printf("\n\n------[POLYGON CHANGED]------\n");
            break;
        }

        case Refresh:{
            //Draw an object off screen to refresh the display buffer
            glBegin(GL_LINES);
            glVertex2f(2000, 2000);
            glVertex2f(2001, 2001);

            glEnd();
            break;
        }
    }

    glFlush();
    glutPostRedisplay();    //IMPORTANT: To refresh the window with the new updated plots
}

void drawAxes(){
    //To draw the X and Y axes

    glColor3d(0, 0, 0); //Black color

    glBegin(GL_LINES);

    //X-axis
    glVertex2f(X_MIN, 0);
    glVertex2f(X_MAX, 0);

    //Y-axis
    glVertex2f(0, Y_MIN);
    glVertex2f(0, Y_MAX);

    glEnd();
```

```
}

void drawPolygon(PolygonShape polygon){
    //To draw a given polygon

    glColor3d(1, 0, 0); //Red color

    glBegin(GL_POLYGON);

    for(int i = 0; i < polygon.getVertexCount(); i++){
        Point p = polygon.getPoint(i);
        glVertex2f(p.getX(), p.getY());
    }

    glEnd();
}

void translatePolygon(PolygonShape polygon, Point translationVector){
    //To translate a given polygon using the translation vector

    glColor4f(0, 0, 1, 0.6); //Blue color

    glBegin(GL_POLYGON);

    for(int i = 0; i < polygon.getVertexCount(); i++){
        Point p = polygon.getPoint(i);
        Point pDash = p.getTranslatedPoint(translationVector);
        glVertex2f(pDash.getX(), pDash.getY()); //Plot the normal coordinates
    }

    glEnd();
}

void rotatePolygon(PolygonShape polygon, int rotationAngle, Point pivot){
    //To rotate a given polygon using the rotation angle and pivot point

    //Plot the pivot point
    glColor3d(1, 0, 1); //Purple Color
    glPointSize(5);

    glBegin(GL_POINTS);
    glVertex2f(pivot.getX(), pivot.getY());
    glEnd();

    glColor4f(0, 0, 1, 0.6); //Blue Color

    glBegin(GL_POLYGON);

    for(int i = 0; i < polygon.getVertexCount(); i++){
        Point p = polygon.getPoint(i);
        Point pDash = p.getRotatedPoint(rotationAngle, pivot);
        glVertex2f(pDash.getX(), pDash.getY()); //Plot the normal coordinates
    }

    glEnd();
}

void reflectPolygon(PolygonShape polygon, Lines line){
    //To reflect a polygon about a given line

    //Plot the given line
    glColor3f(1, 0, 1);

    glBegin(GL_LINES);

    switch(line){
        case XAxis:
            glVertex2f(X_MIN, 0);
            glVertex2f(X_MAX, 0);
            break;
        case YAxis:
            glVertex2f(0, Y_MIN);
            glVertex2f(0, Y_MAX);
            break;
        case Origin:
            glVertex2f(X_MIN, 0);
            glVertex2f(X_MAX, 0);
            glVertex2f(0, Y_MIN);
            glVertex2f(0, Y_MAX);
            break;
        case XEqualsY:
            glVertex2f(X_MIN, Y_MIN);
            glVertex2f(X_MAX, Y_MAX);
            break;
        default:
            return;
    }

    glEnd();

    glColor4f(0, 0, 1, 0.6); //Blue Color

    glBegin(GL_POLYGON);

    for(int i = 0; i < polygon.getVertexCount(); i++){
        Point p = polygon.getPoint(i);
        Point pDash;

        switch(line){
            case XAxis:
                pDash = p.getReflectionAboutXAxis();
                break;
            case YAxis:
                pDash = p.getReflectionAboutYAxis();
                break;
            case Origin:
                pDash = p.getReflectionAboutOrigin();
                break;
            case XEqualsY:
                pDash = p.getReflectionAboutXEqualsY();
                break;
            default:
                return;
        }

        glVertex2f(pDash.getX(), pDash.getY()); //Plot the normal coordinates
    }

    glEnd();
}

void scalePolygon(PolygonShape polygon, double scaleX, double scaleY, Point fixed){
    //To translate a given polygon using the scale factors and fixed point

    //Plot the fixed point
    glColor3d(1, 0, 1); //Purple Color
    glPointSize(5);

    glBegin(GL_POINTS);
    glVertex2f(fixed.getX(), fixed.getY());
    glEnd();

    glColor4f(0, 0, 1, 0.6); //Blue Color

    glBegin(GL_POLYGON);

    for(int i = 0; i < polygon.getVertexCount(); i++){
        Point p = polygon.getPoint(i);
        Point pDash = p.getScaledPoint(scaleX, scaleY, fixed);
        glVertex2f(pDash.getX(), pDash.getY()); //Plot the normal coordinates
    }

    glEnd();
}

void shearPolygon(PolygonShape polygon, Axes axis, double shearParam, double refLine){
    //To shear a polygon about axis and shear parameter
    //Plot the given line
    glColor3f(1, 0, 1);

    glBegin(GL_LINES);

    switch(axis){
        case xAxis:
            glVertex2f(X_MIN, 0);
            glVertex2f(X_MAX, 0);
            break;
        case yAxis:
            glVertex2f(0, Y_MIN);
            glVertex2f(0, Y_MAX);
            break;
        default:
            return;
    }

    glEnd();

    glColor4f(0, 0, 1, 0.6); //Blue Color, with alpha (transparency) factor as 0.6

    glBegin(GL_POLYGON);

    for(int i = 0; i < polygon.getVertexCount(); i++){
        Point p = polygon.getPoint(i);
        Point pDash;

        switch(axis){
            case xAxis:
                pDash = p.getShearAboutXAxis(shearParam, refLine);
                break;
            case yAxis:
                pDash = p.getShearAboutYAxis(shearParam, refLine);
                break;
            default:
                return;
        }

        glVertex2f(pDash.getX(), pDash.getY()); //Plot the normal coordinates
    }

    glEnd();
}
```

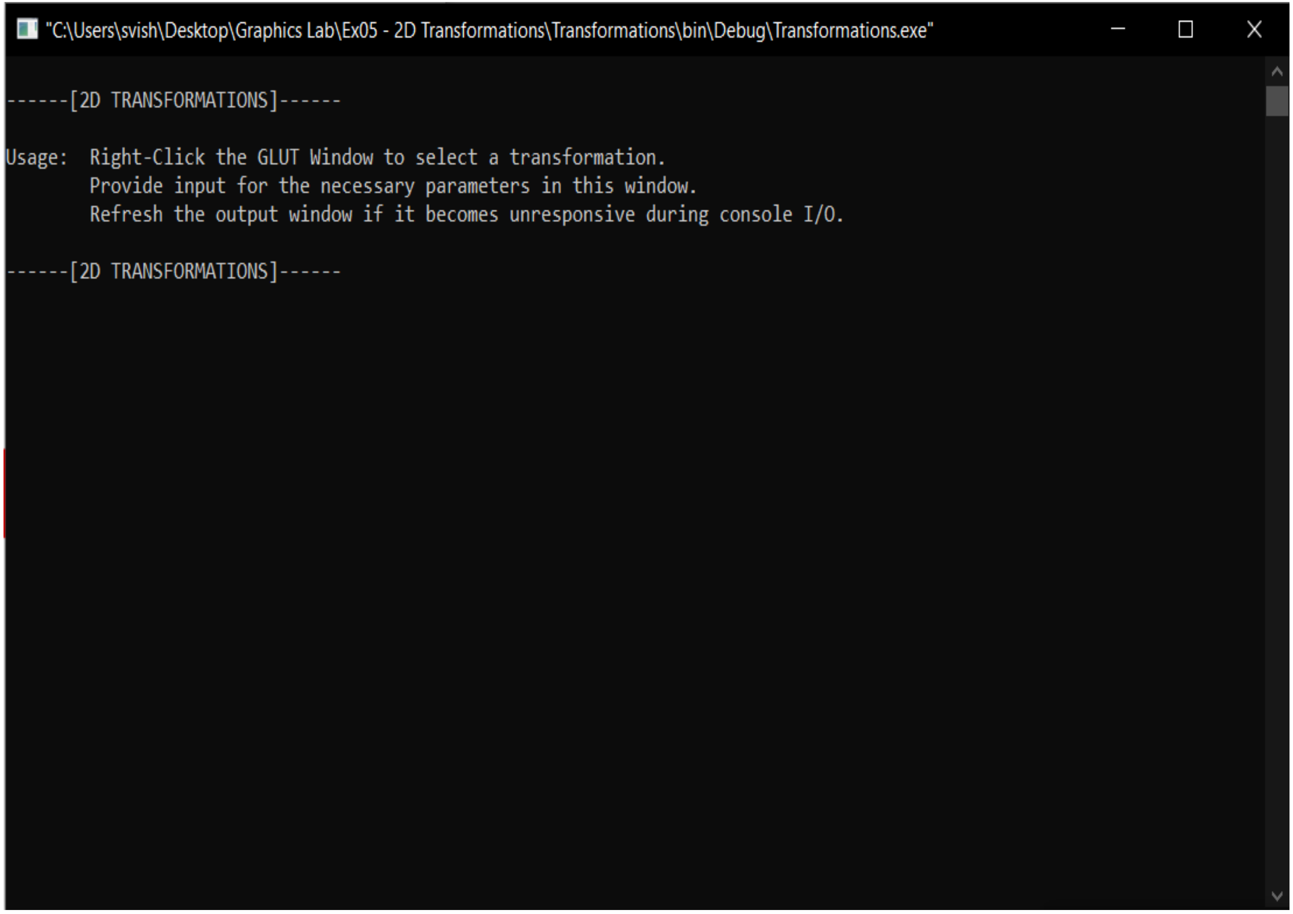
Output: Initial Plot With Transformations Menu

Figure 1: Initial Plot With Transformations Menu.



Output: Console

Figure 2: Output: Console.

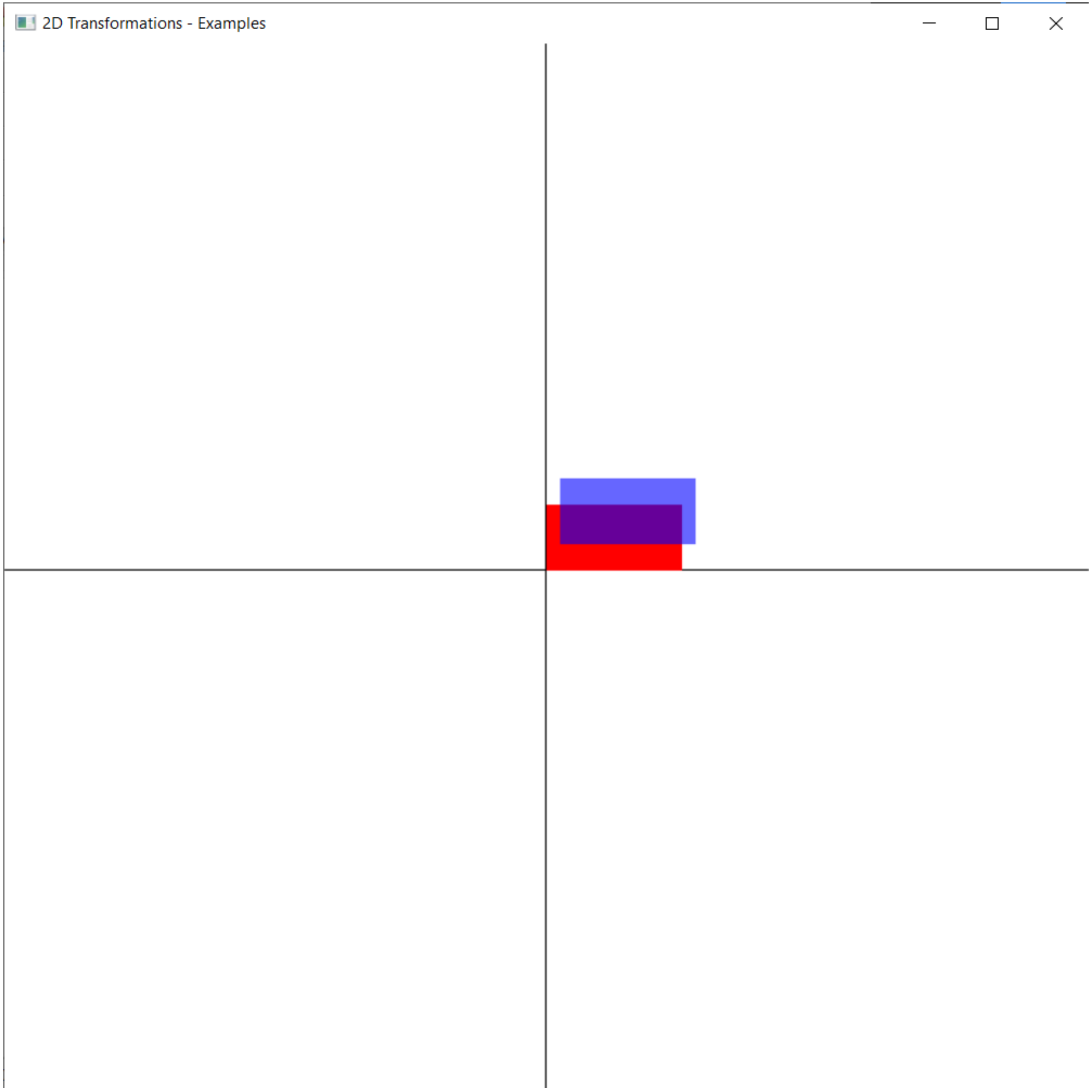


"C:\Users\svish\Desktop\Graphics Lab\Ex05 - 2D Transformations\Transformations\bin\Debug\Transformations.exe"

```
------[2D TRANSFORMATIONS]------

Usage:  Right-Click the GLUT Window to select a transformation.
        Provide input for the necessary parameters in this window.
        Refresh the output window if it becomes unresponsive during console I/O.

------[2D TRANSFORMATIONS]------
```

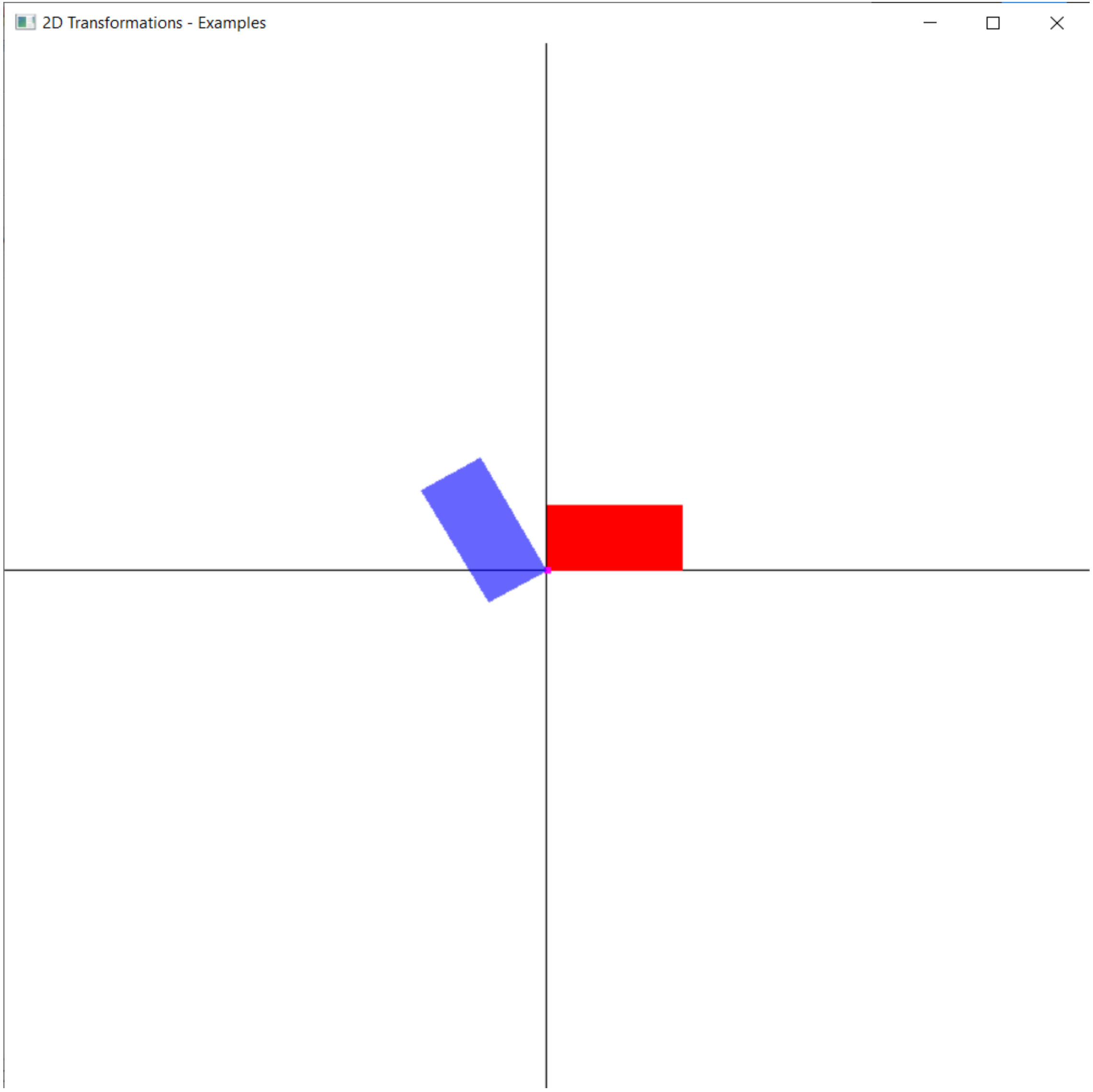
Output: Translation

Figure 3: Translation by $(10, 20)$.



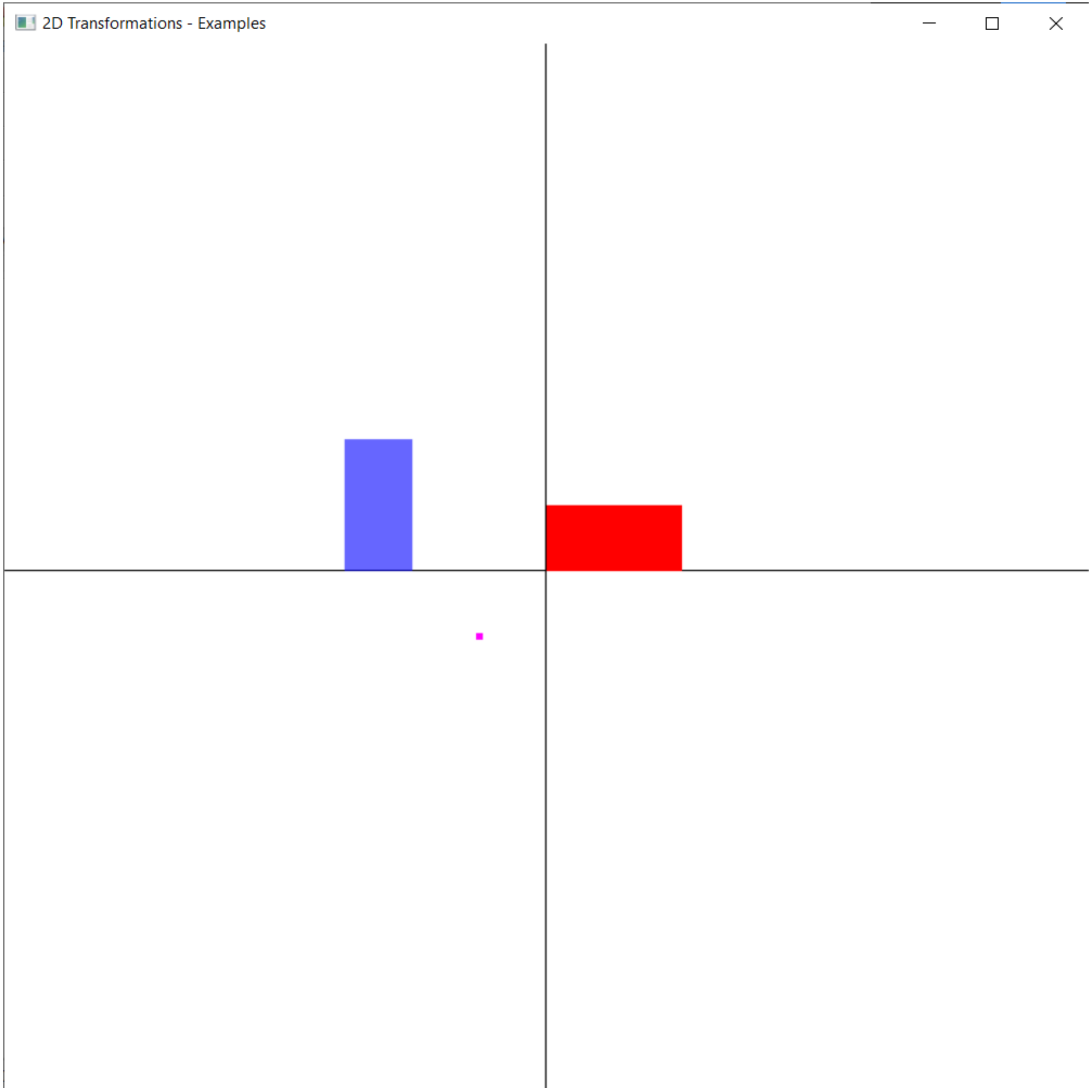
Output: Rotation

Figure 4: Rotation by 120°.



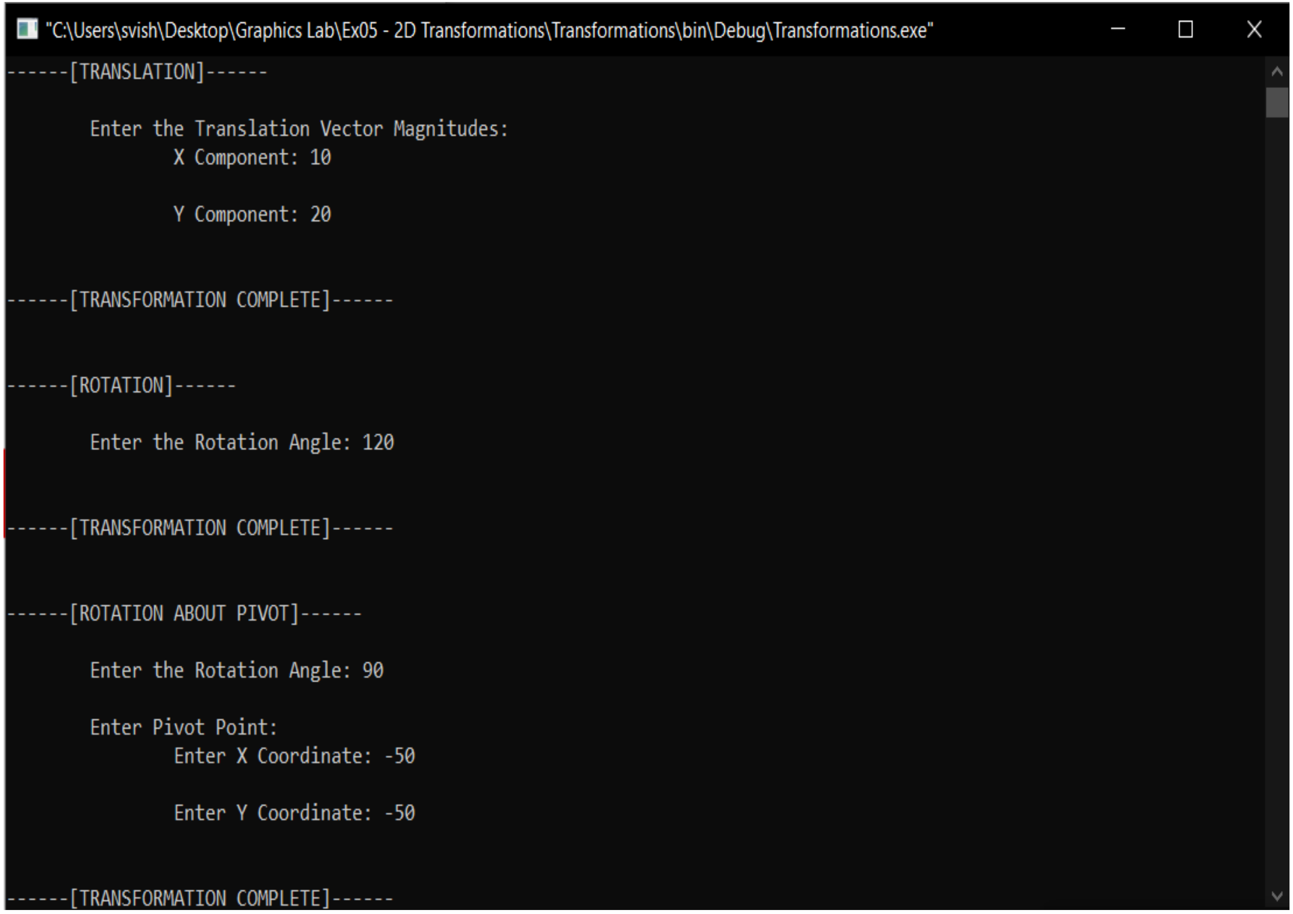
Output: Rotation About A Pivot Point

Figure 5: Rotation About A Pivot Point $(-50, -50)$ by 90°.



Output: Console

Figure 6: Console.

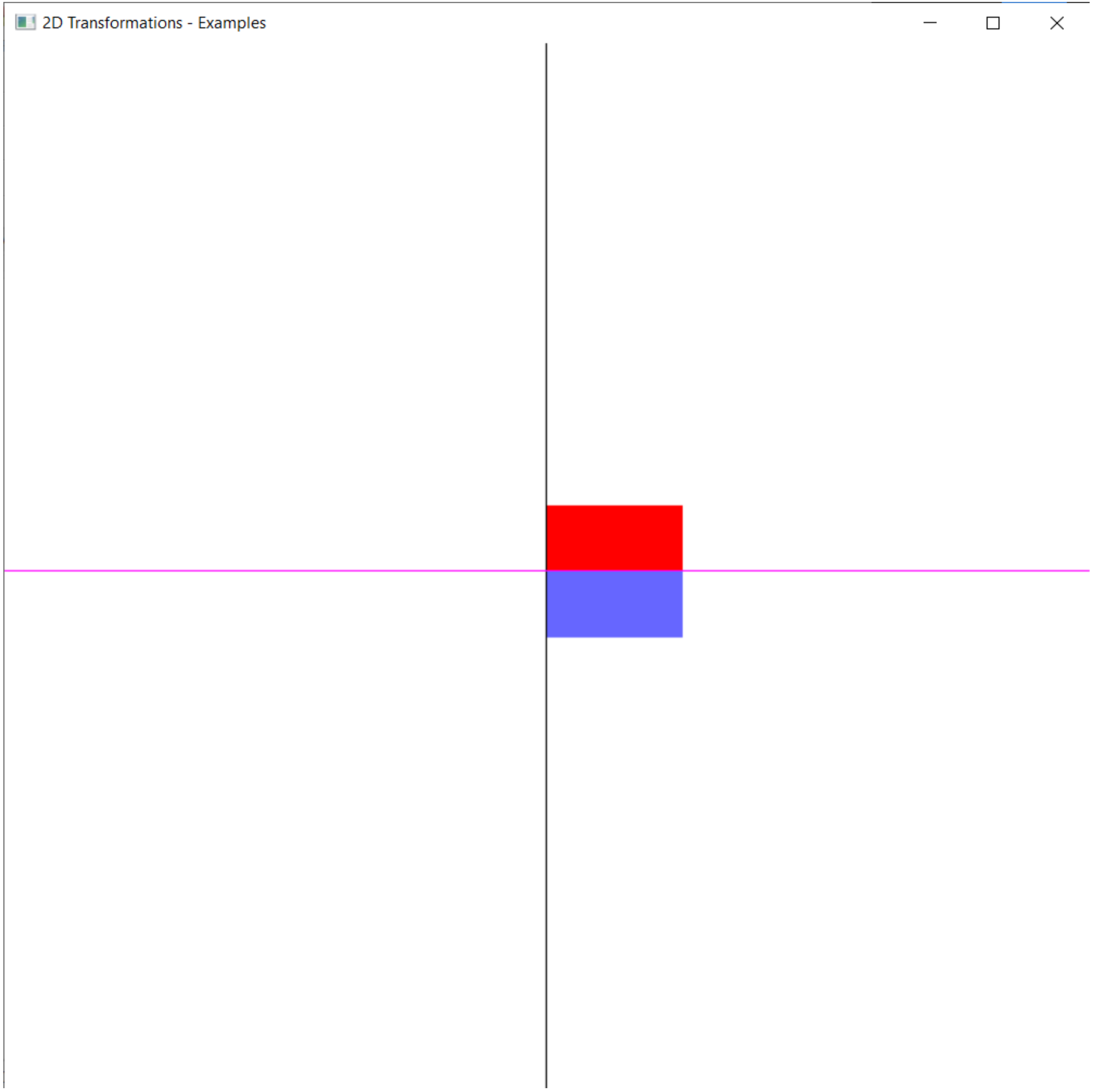


"C:\Users\svish\Desktop\Graphics Lab\Ex05 - 2D Transformations\Transformations\bin\Debug\Transformations.exe"

```
------[TRANSLATION]------

        Enter the Translation Vector Magnitudes:
                X Component: 10

                Y Component: 20


------[TRANSFORMATION COMPLETE]------


------[ROTATION]------

        Enter the Rotation Angle: 120


------[TRANSFORMATION COMPLETE]------


------[ROTATION ABOUT PIVOT]------

        Enter the Rotation Angle: 90

        Enter Pivot Point:
                Enter X Coordinate: -50

                Enter Y Coordinate: -50


------[TRANSFORMATION COMPLETE]------
```

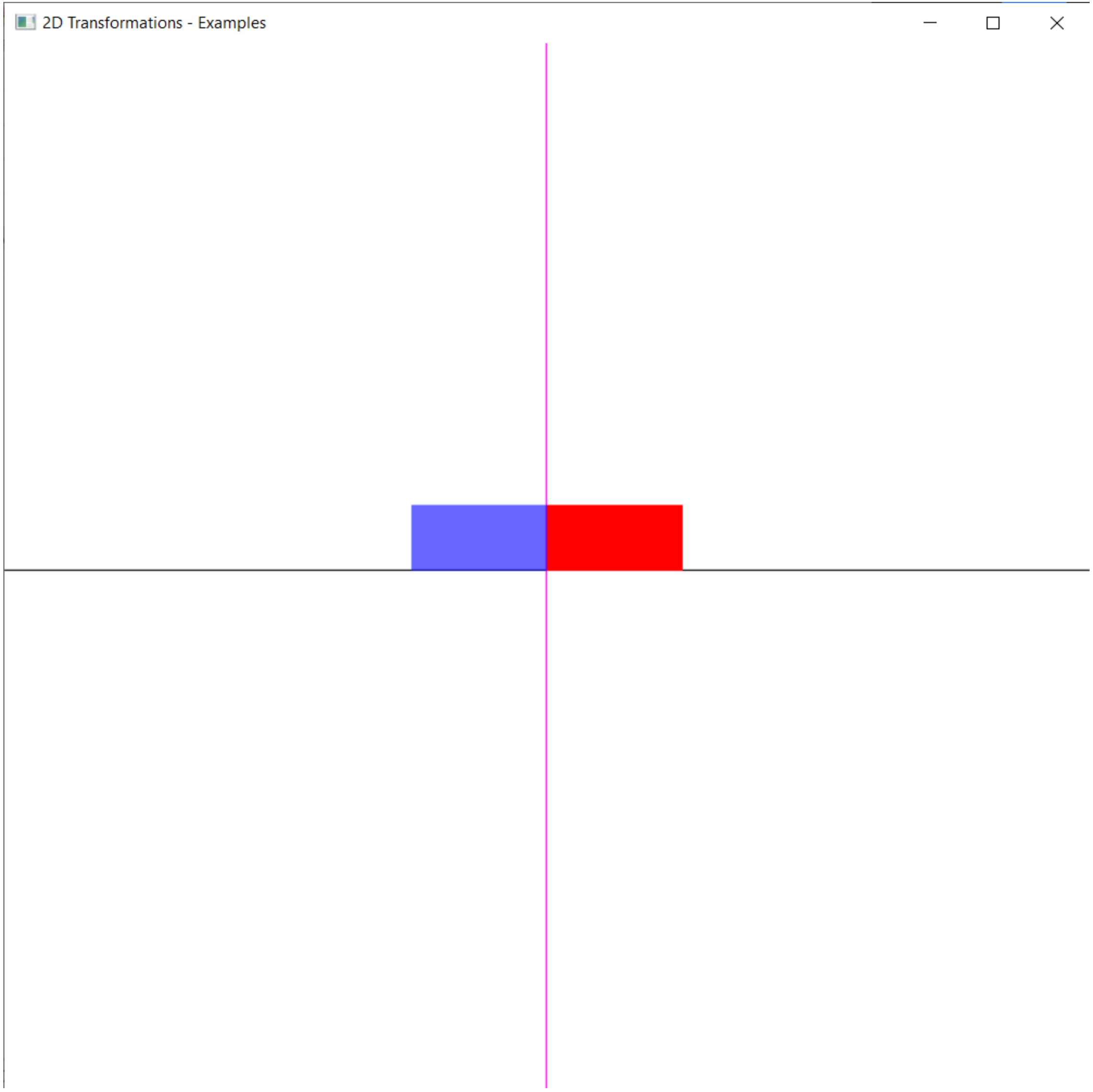
Output: Reflection About X Axis

Figure 7: Reflection About X Axis.



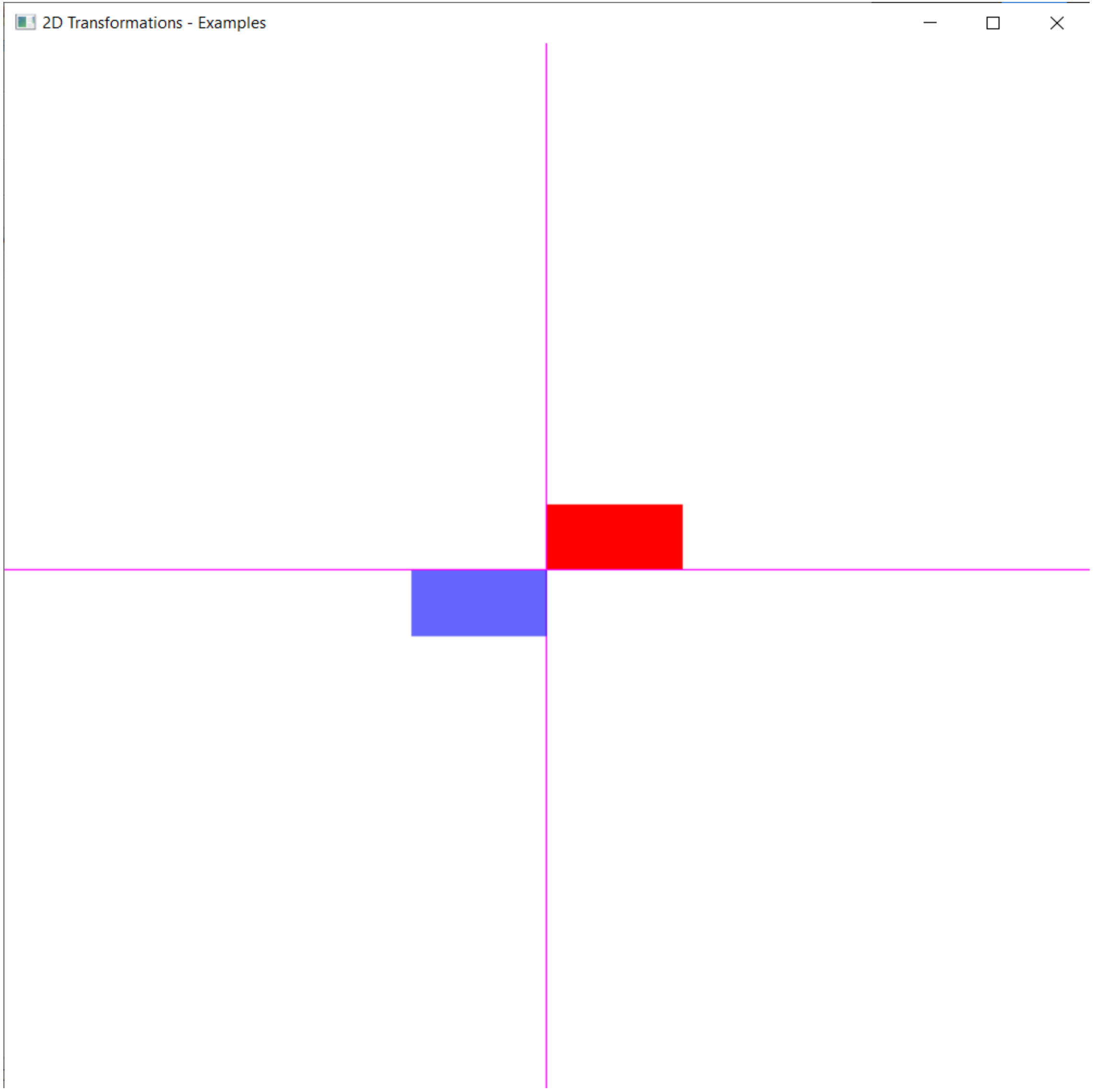
Output: Reflection About Y Axis

Figure 8: Reflection About Y Axis.



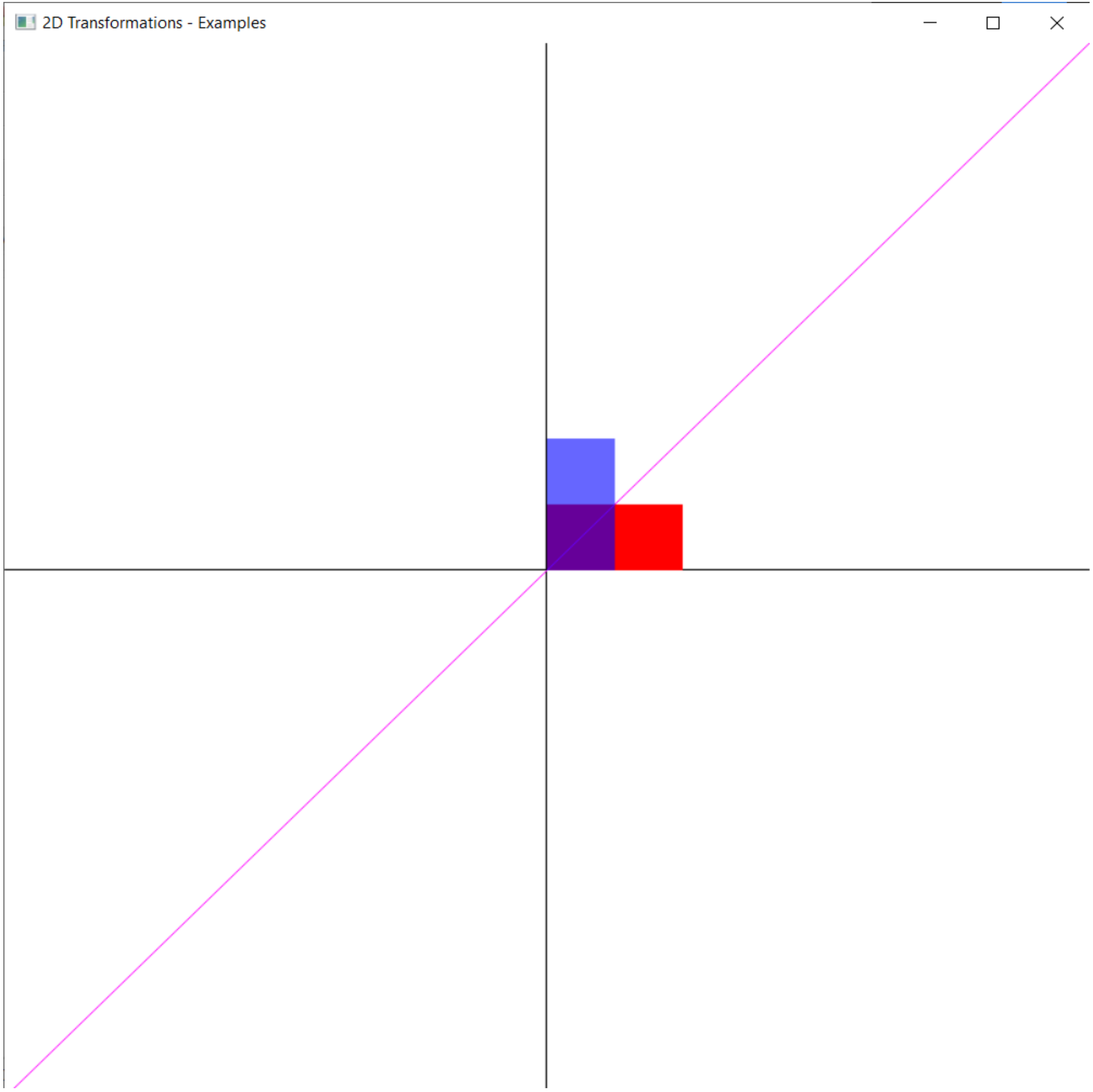
Output: Reflection About Origin

Figure 9: Reflection About Origin.



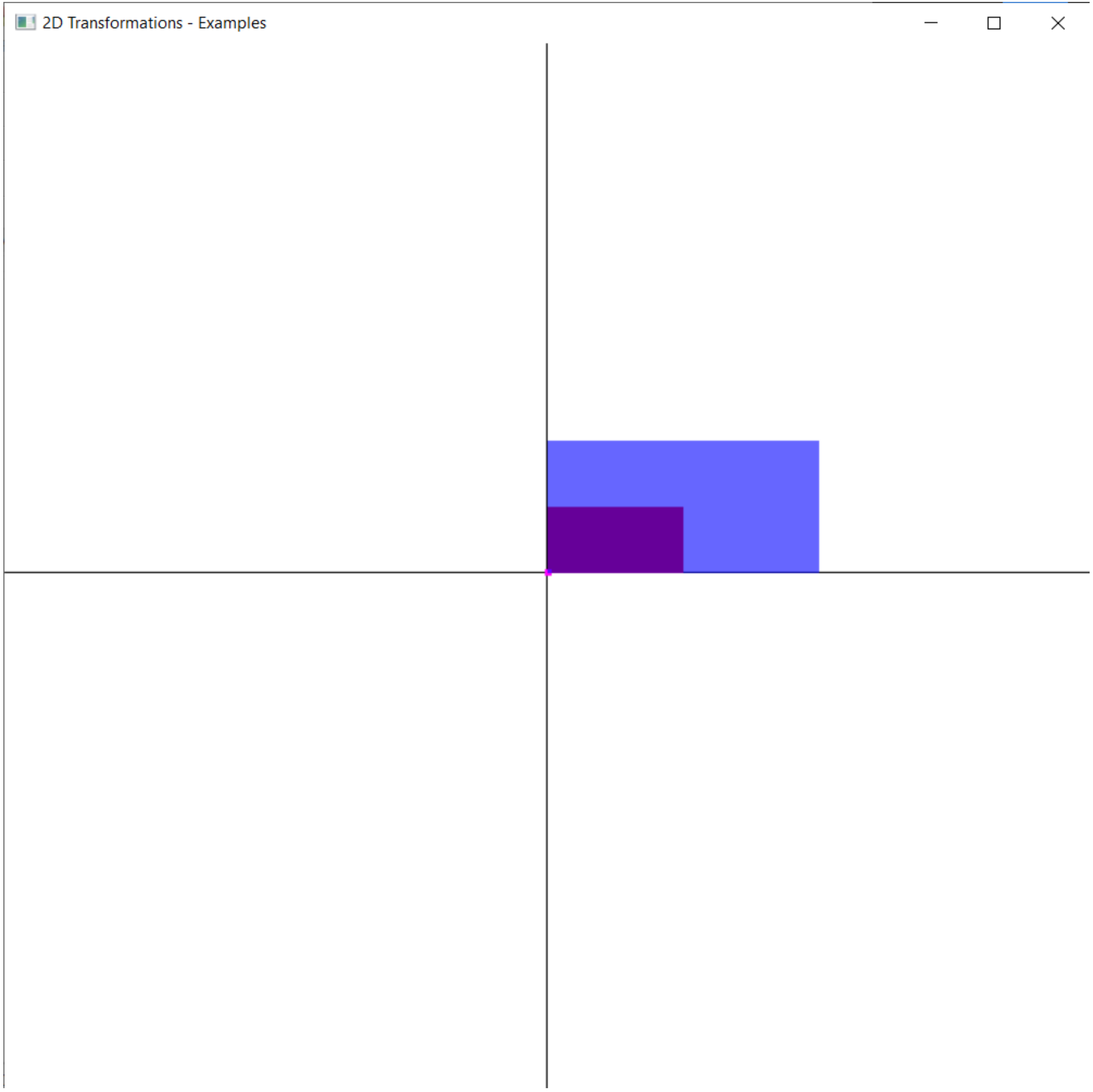
Output: Reflection About Line $X = Y$

Figure 10: Reflection About Line $X = Y$.



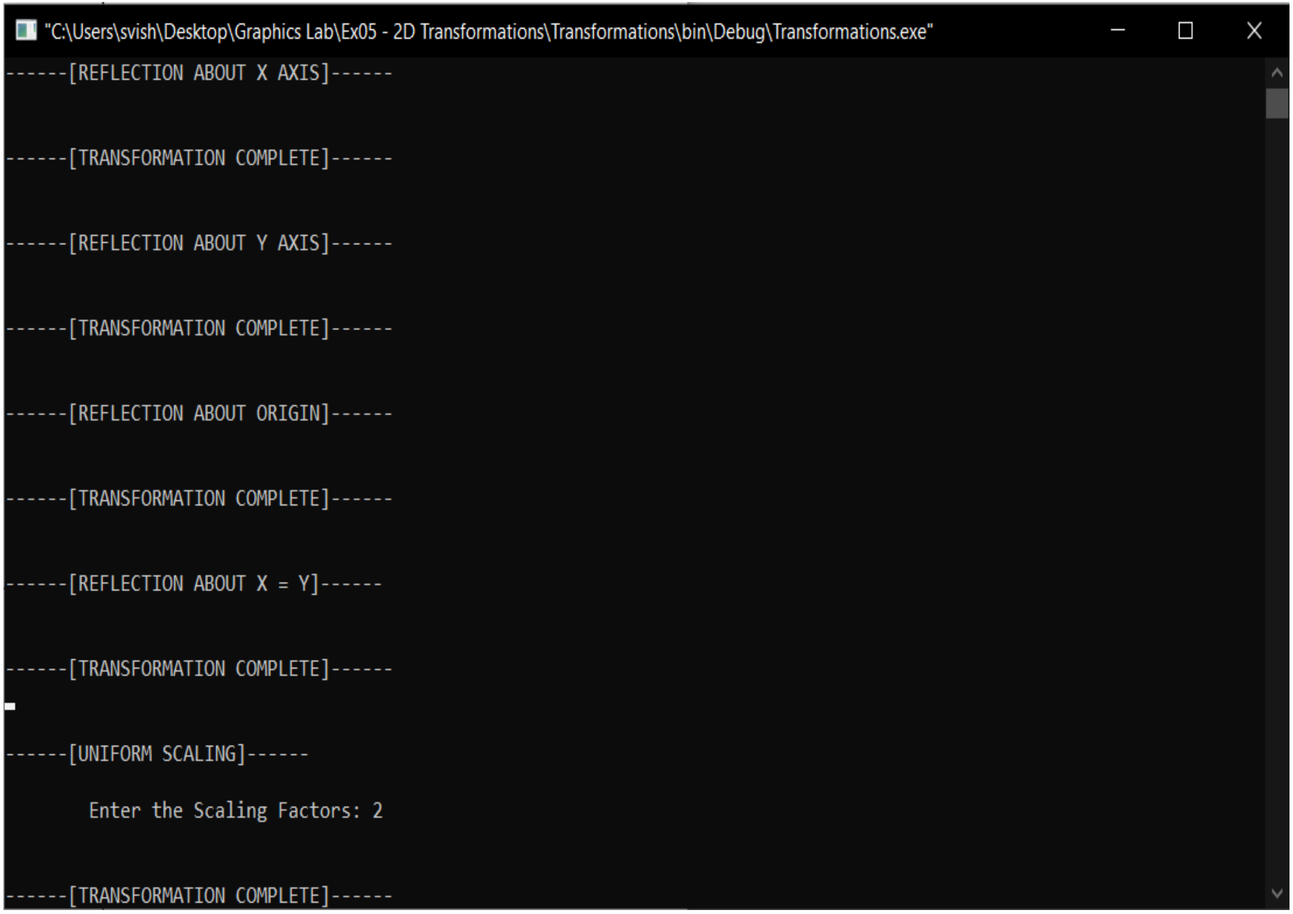
Output: Uniform Scaling

Figure 11: Uniform Scaling by a Factor of 2.



Output: Console

Figure 12: Console.

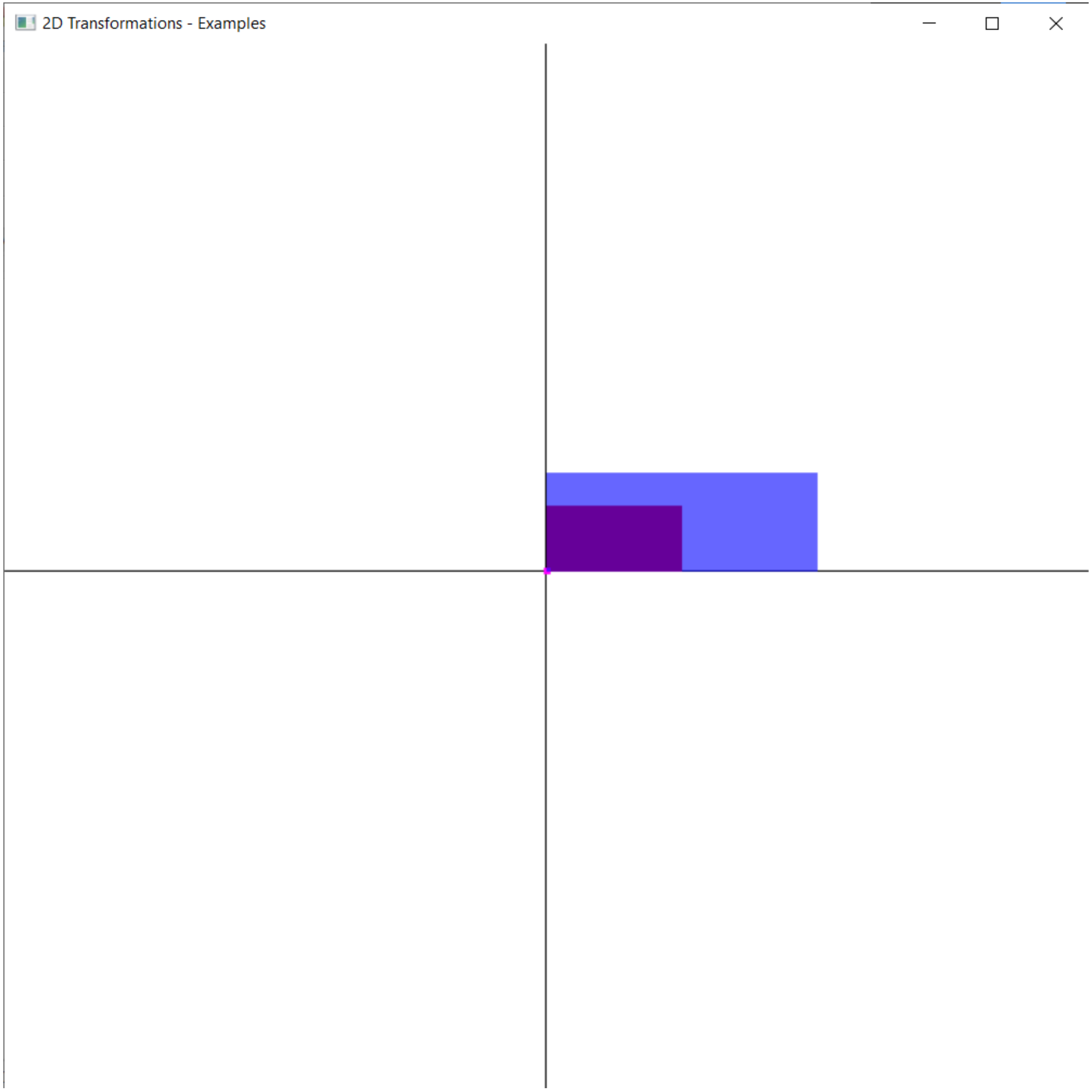


"C:\Users\svish\Desktop\Graphics Lab\Ex05 - 2D Transformations\Transformations\bin\Debug\Transformations.exe"

```
------[REFLECTION ABOUT X AXIS]------


------[TRANSFORMATION COMPLETE]------


------[REFLECTION ABOUT Y AXIS]------


------[TRANSFORMATION COMPLETE]------


------[REFLECTION ABOUT ORIGIN]------


------[TRANSFORMATION COMPLETE]------


------[REFLECTION ABOUT X = Y]------


------[TRANSFORMATION COMPLETE]------


------[UNIFORM SCALING]------

        Enter the Scaling Factors: 2


------[TRANSFORMATION COMPLETE]------
```

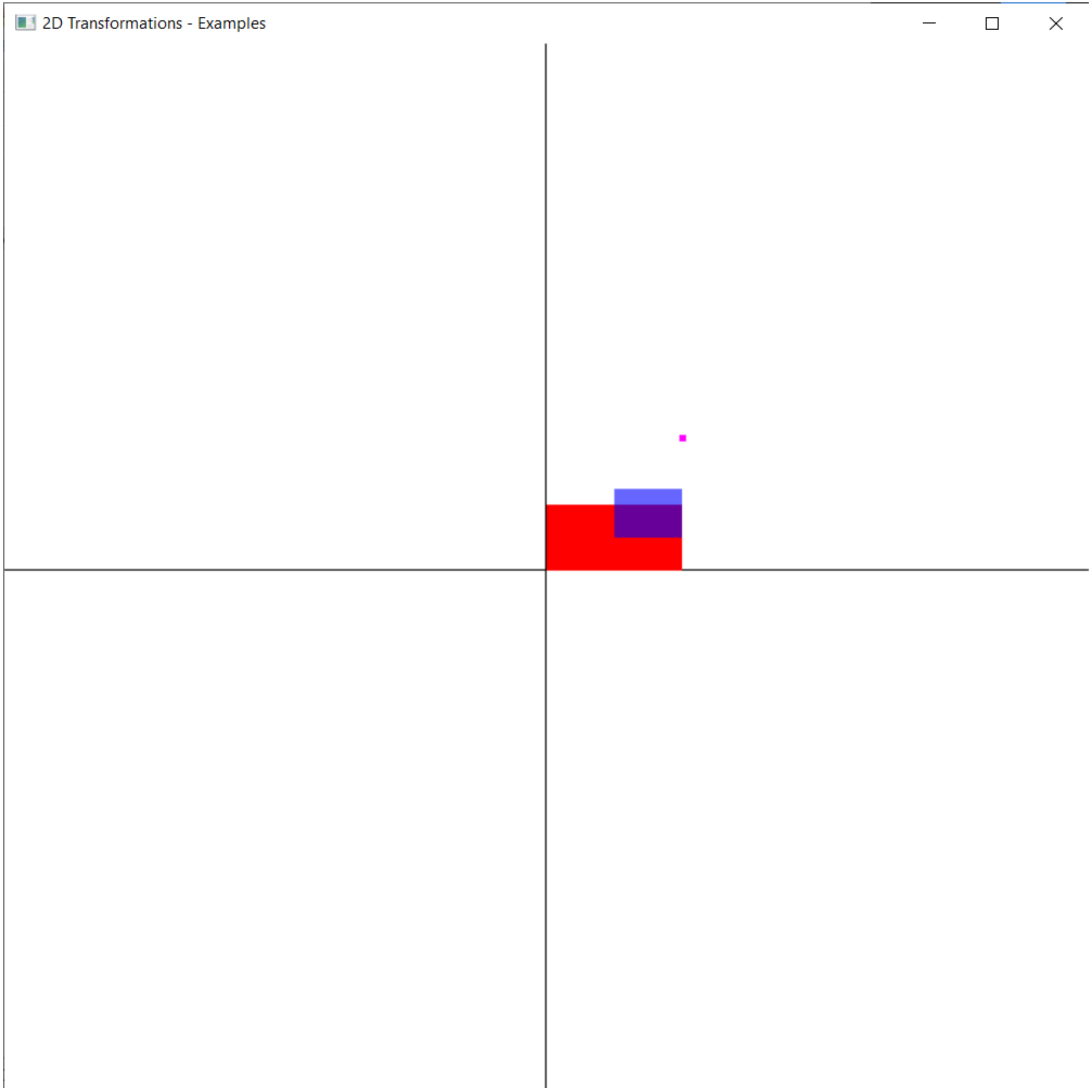
Output: Differential Scaling

Figure 13: Differential Scaling by a Factor of $(2, 1.5)$.



Output: Differential Scaling About A Fixed Point

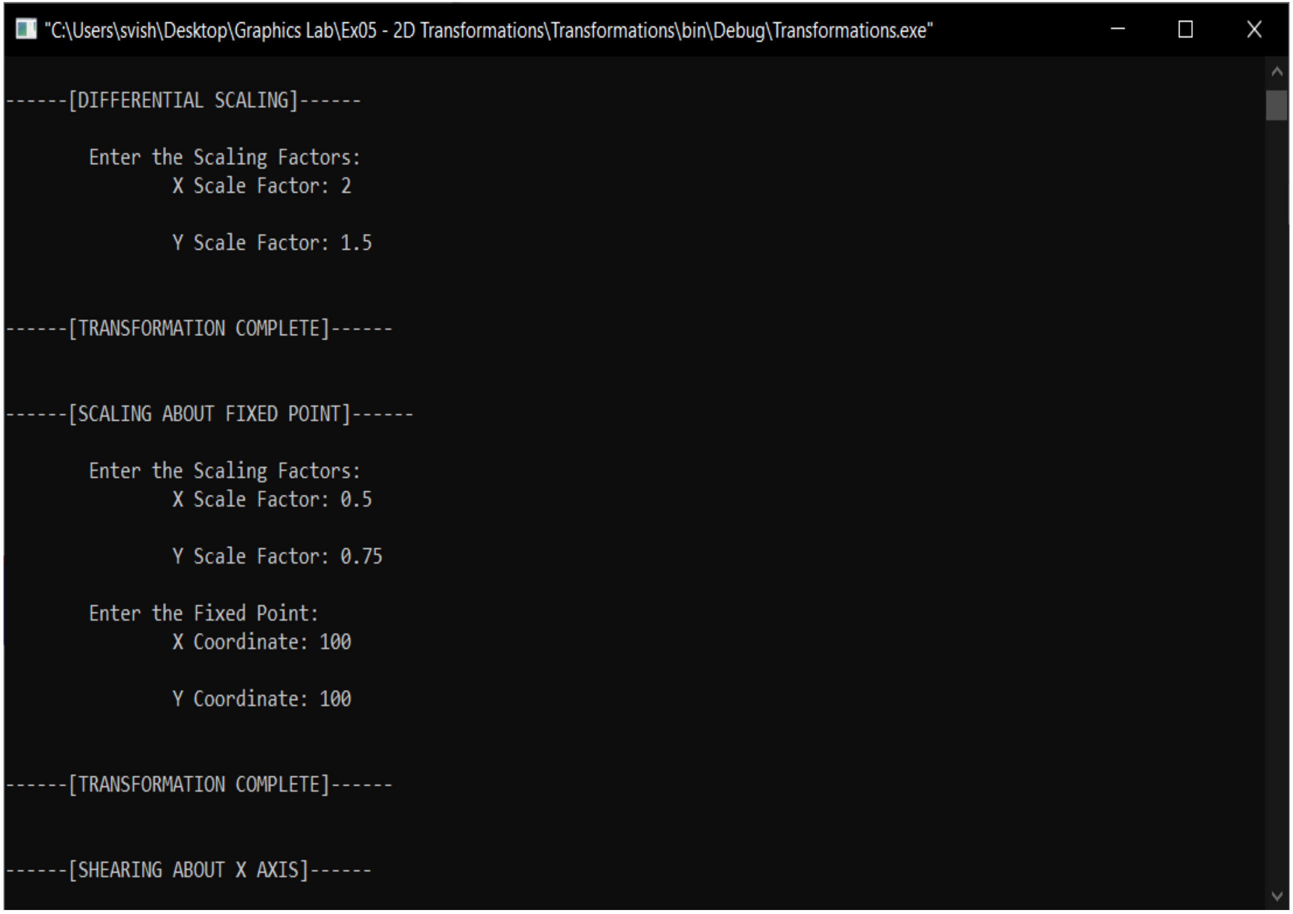
Figure 14: Differential Scaling About A Fixed Point (100, 100) by a Factor of (0.5, 0.75).



2D Transformations - Examples

Output: Console

Figure 15: Console.

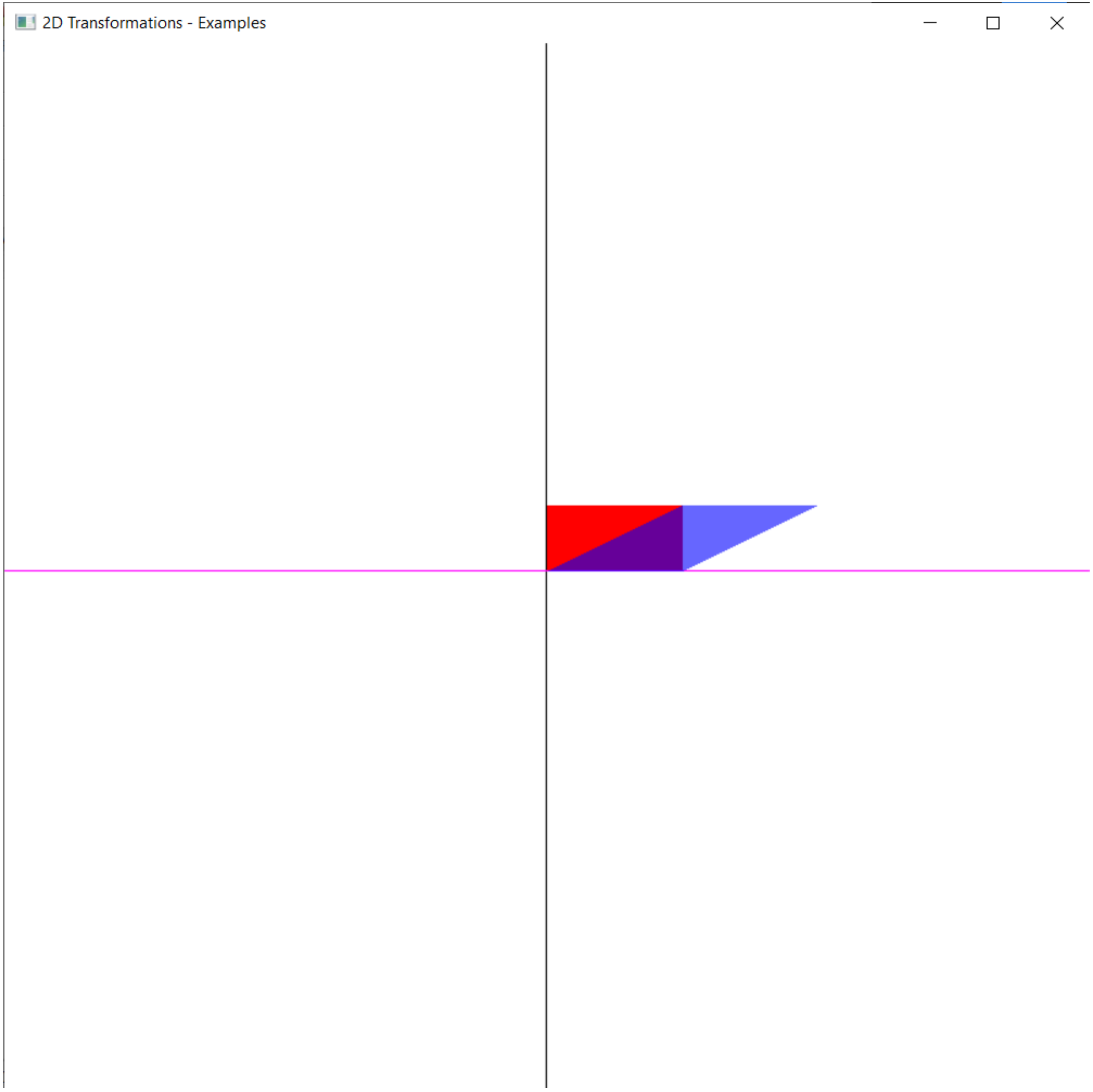


"C:\Users\svish\Desktop\Graphics Lab\Ex05 - 2D Transformations\Transformations\bin\Debug\Transformations.exe"

```
------[DIFFERENTIAL SCALING]------

        Enter the Scaling Factors:
                X Scale Factor: 2

                Y Scale Factor: 1.5


------[TRANSFORMATION COMPLETE]------


------[SCALING ABOUT FIXED POINT]------

        Enter the Scaling Factors:
                X Scale Factor: 0.5

                Y Scale Factor: 0.75

        Enter the Fixed Point:
                X Coordinate: 100

                Y Coordinate: 100


------[TRANSFORMATION COMPLETE]------


------[SHEARING ABOUT X AXIS]------
```

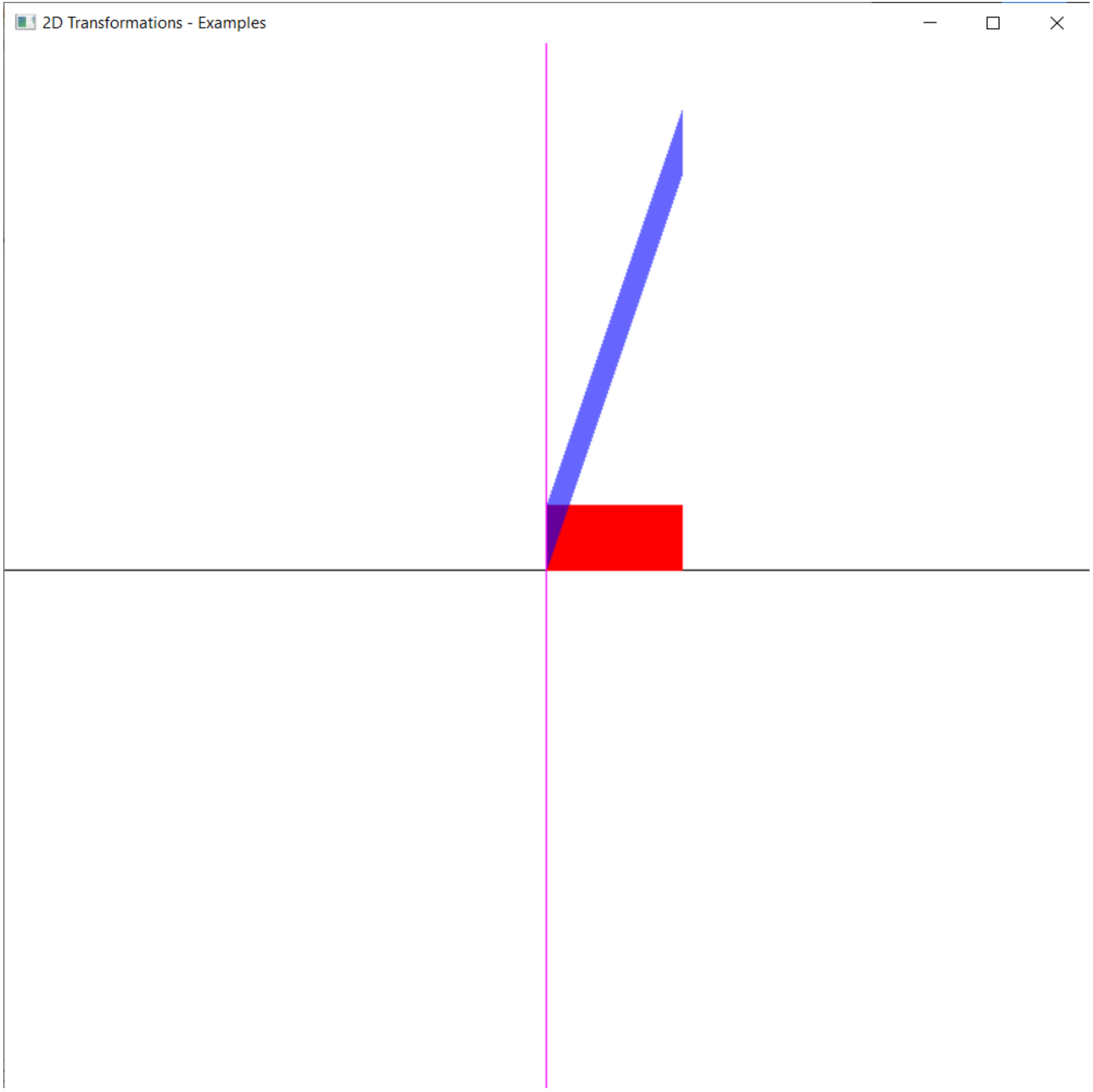
Output: Shearing About X Axis

Figure 16: Shearing About X Axis by a Parameter of 2.



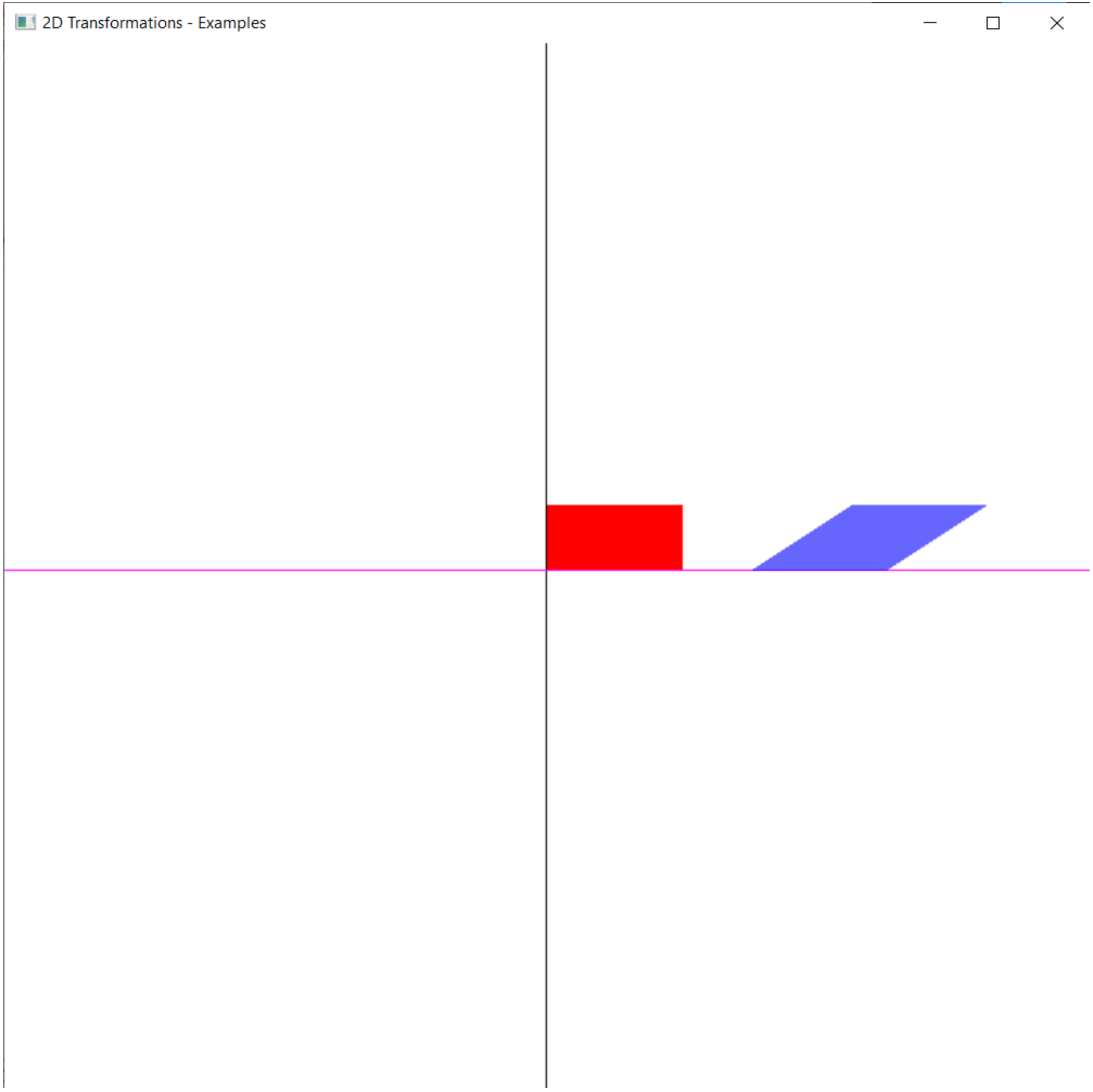
Output: Shearing About Y Axis

Figure 17: Shearing About Y Axis by a Parameter of 3.



Output: Shearing About X Axis & Ref. Line $Y = y$

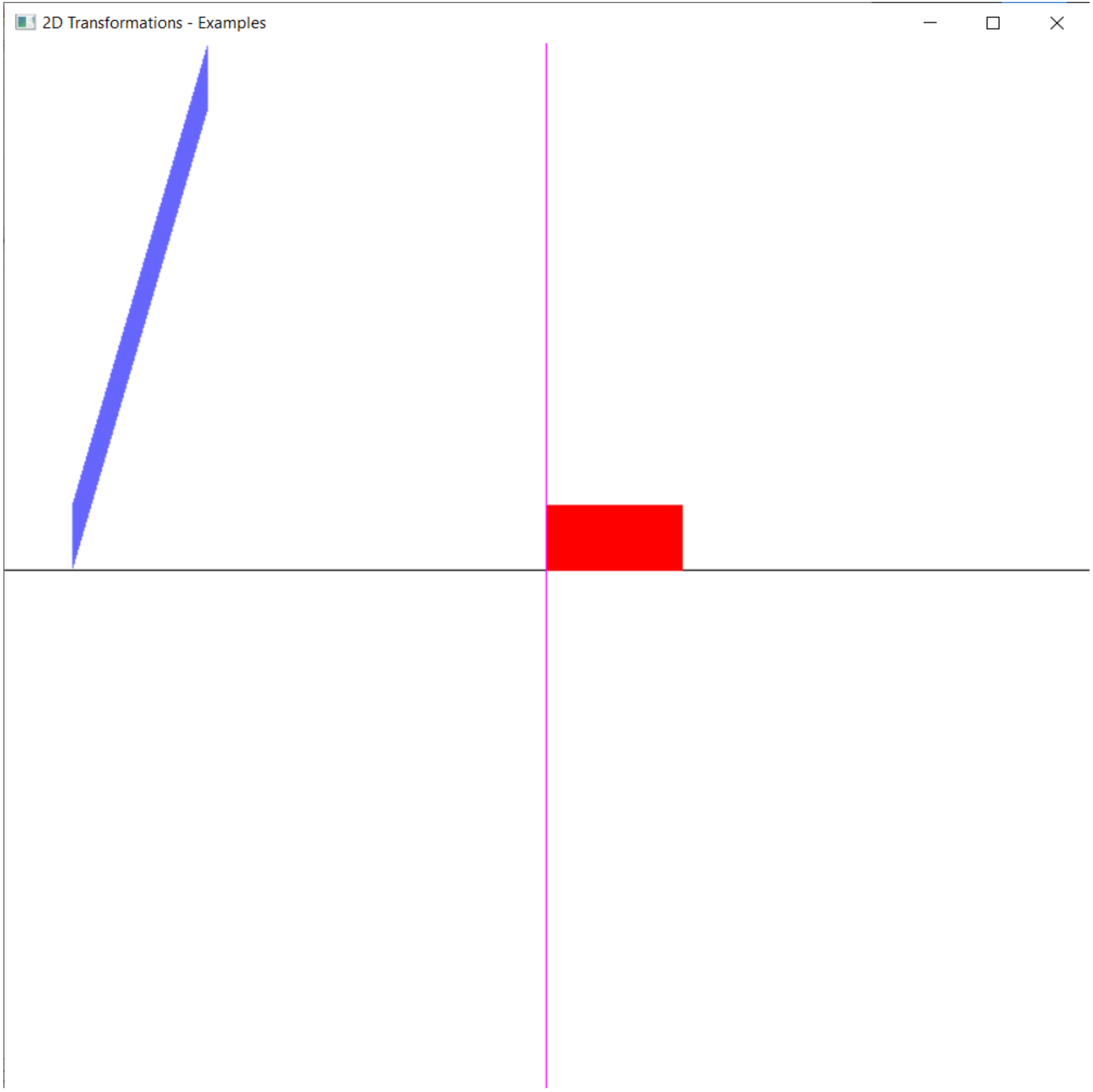
Figure 18: Shearing About X Axis & Ref. Line $Y = -100$ by a Parameter of 1.5.



2D Transformations - Examples

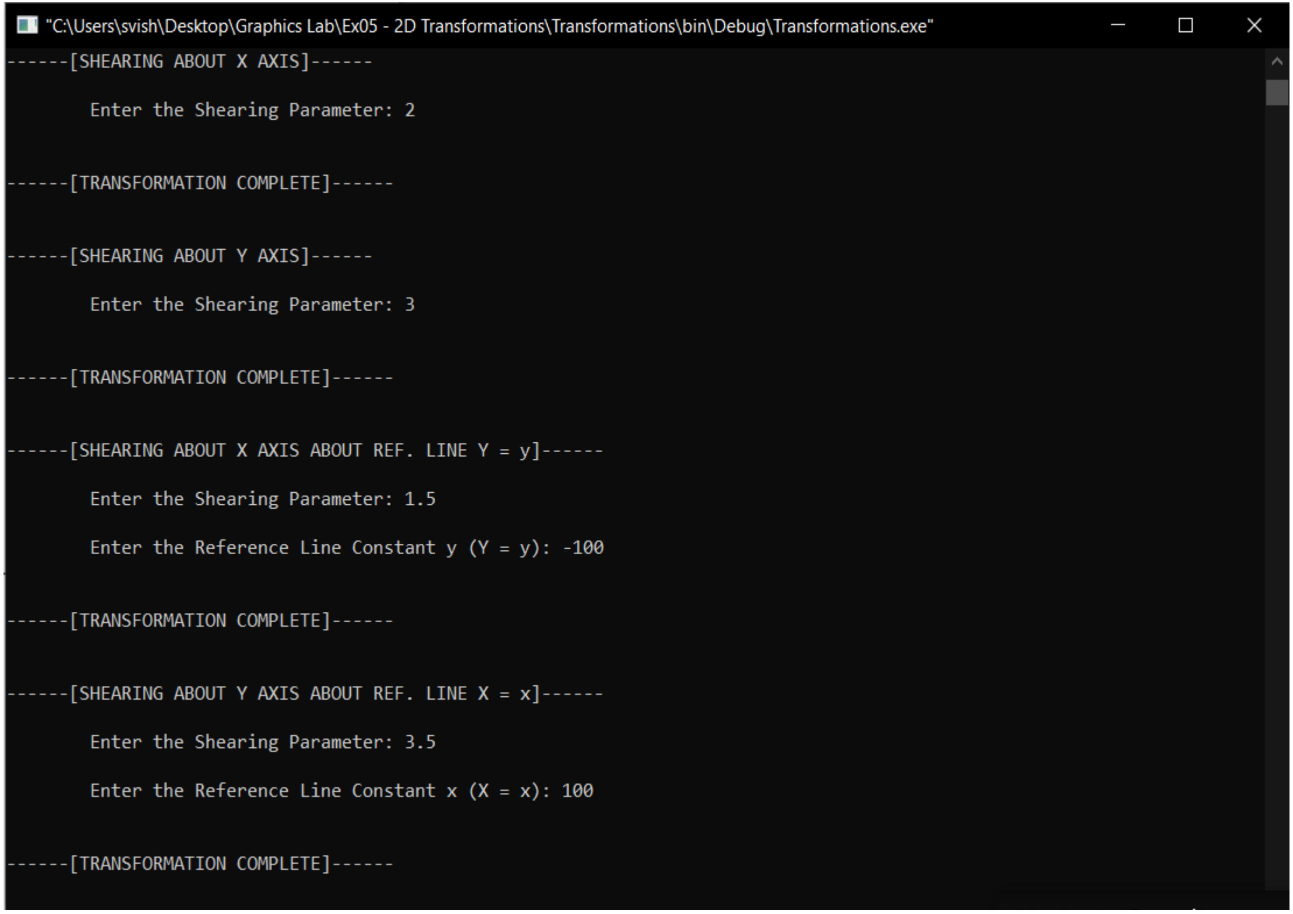
Output: Shearing About Y Axis & Ref. Line $X = x$

Figure 19: Shearing About Y Axis & Ref. Line $X = 100$ by a Parameter of 3.5.



Output: Console

Figure 20: Console.

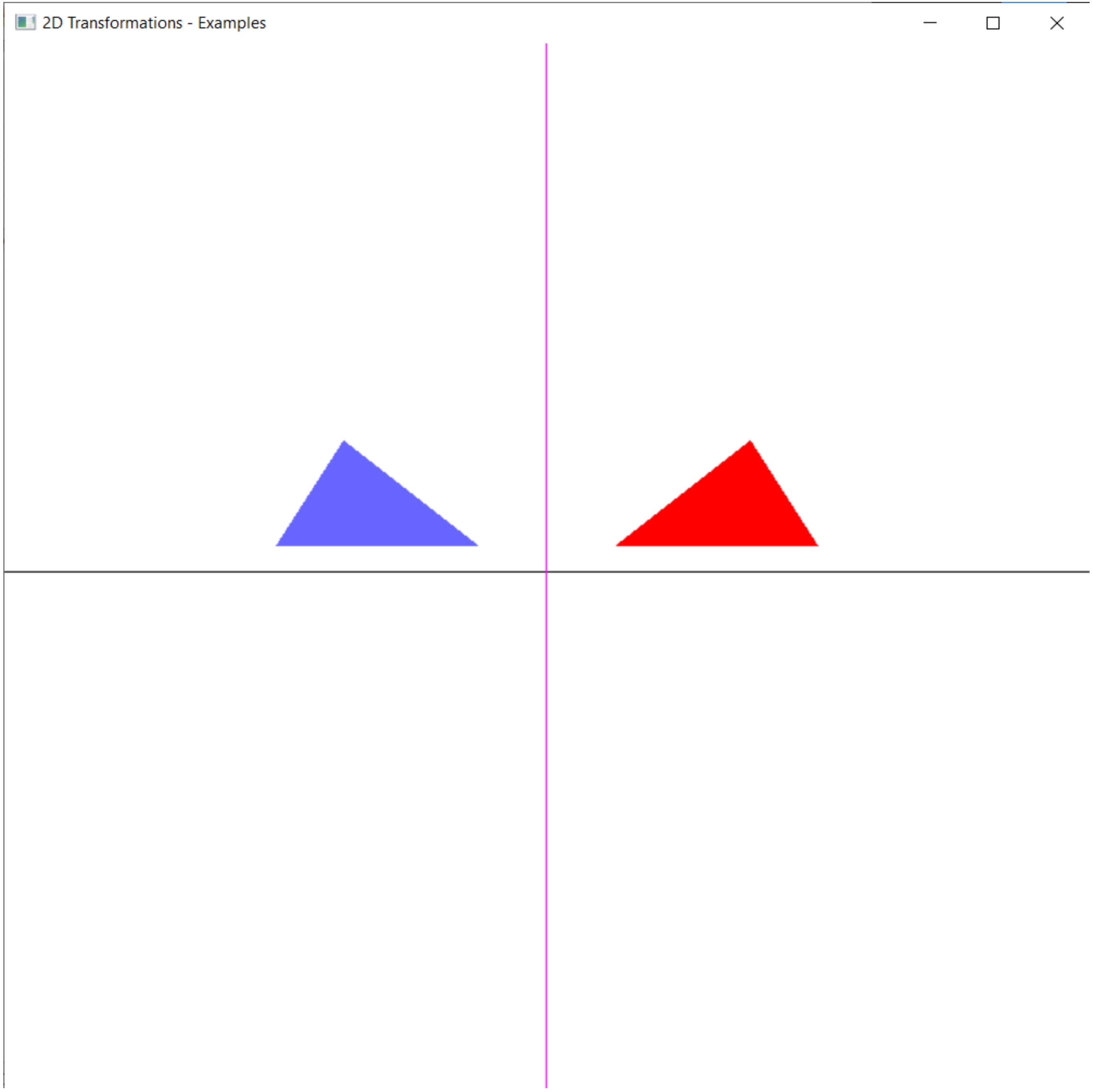


"C:\Users\svish\Desktop\Graphics Lab\Ex05 - 2D Transformations\Transformations\bin\Debug\Transformations.exe"

```
------[SHEARING ABOUT X AXIS]------

        Enter the Shearing Parameter: 2


------[TRANSFORMATION COMPLETE]------


------[SHEARING ABOUT Y AXIS]------

        Enter the Shearing Parameter: 3


------[TRANSFORMATION COMPLETE]------


------[SHEARING ABOUT X AXIS ABOUT REF. LINE Y = y]------

        Enter the Shearing Parameter: 1.5

        Enter the Reference Line Constant y (Y = y): -100


------[TRANSFORMATION COMPLETE]------


------[SHEARING ABOUT Y AXIS ABOUT REF. LINE X = x]------

        Enter the Shearing Parameter: 3.5

        Enter the Reference Line Constant x (X = x): 100


------[TRANSFORMATION COMPLETE]------
```

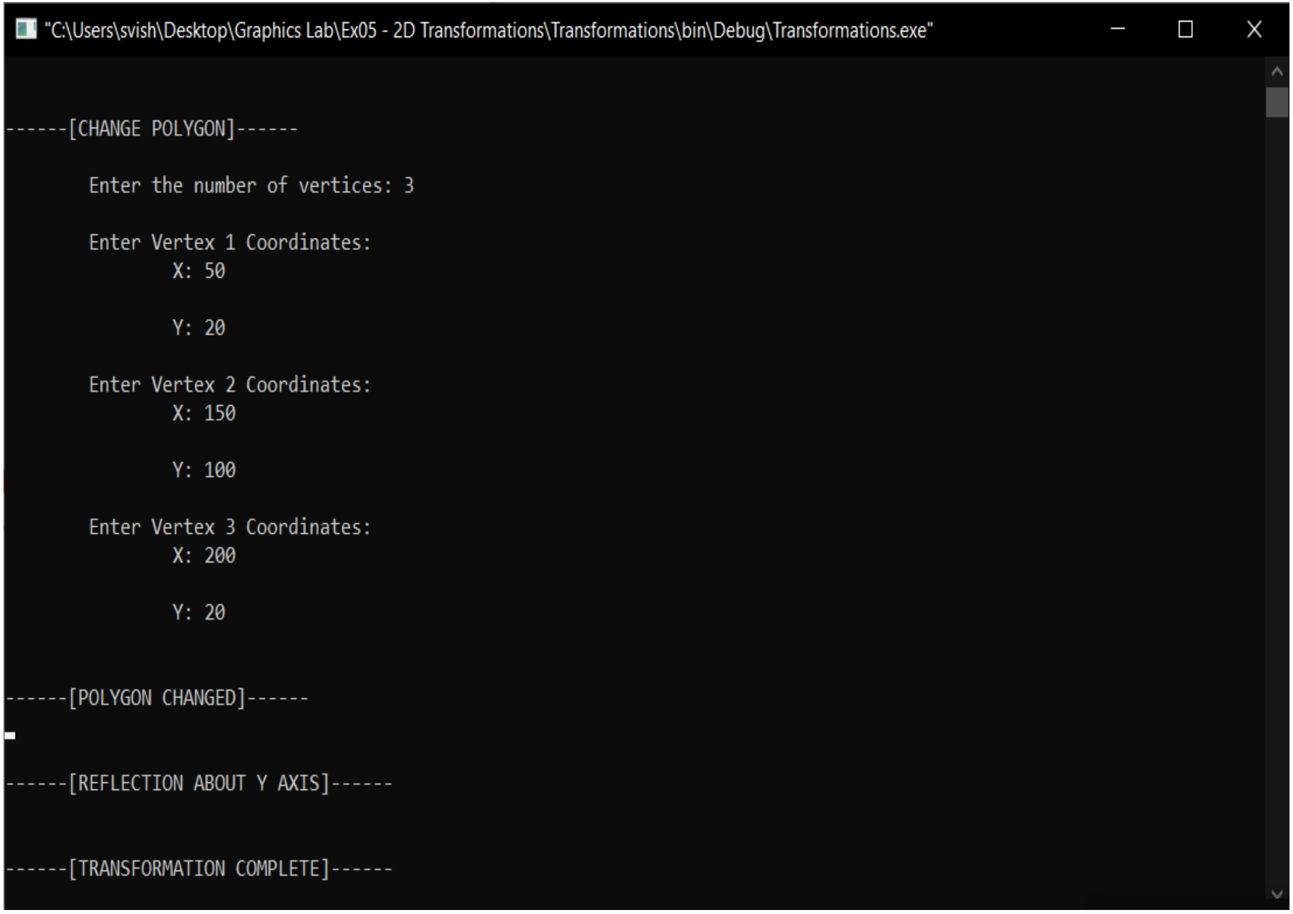
Output: Changed Base Polygon and its Y Axis Reflection

Figure 21: Changed Base Polygon and its Y Axis Reflection



Output: Console

Figure 22: Console.



"C:\Users\svish\Desktop\Graphics Lab\Ex05 - 2D Transformations\Transformations\bin\Debug\Transformations.exe"

```
------[CHANGE POLYGON]------

        Enter the number of vertices: 3

        Enter Vertex 1 Coordinates:
                X: 50

                Y: 20

        Enter Vertex 2 Coordinates:
                X: 150

                Y: 100

        Enter Vertex 3 Coordinates:
                X: 200

                Y: 20


------[POLYGON CHANGED]------


------[REFLECTION ABOUT Y AXIS]------


------[TRANSFORMATION COMPLETE]------
```

Learning Outcome:

- I understood how to convert (x, y) global coordinates into **homogeneous coordinates** and its relevance in applying transformations.
- I understood how to apply **matrix multiplication** operations to achieve various 2-D Transformations.
- I learnt about the transformation matrices for translation, rotation, reflection, scaling & shearing.
- I implemented separate classes for **PolygonShape** and **Point** for ease of use and modularizing the program.
- I understood how to implement a **GLUT Menu** for a menu-based approach to apply transformations.
- I understood how to draw translucent objects with the help of **glDepthMask(), glBlendFunc() and glColor4f()** with parameter **ALPHA**.
- I learnt how to project a **Cartesian Plane** with the use of **gluOrtho2D()**.
- I learnt to use **enum** to simplify and enhance readability for my menu-driven program.
- I learnt how to use default arguments in C++ to provide default variables.
- I implemented **translation** about a given translation vector.
- I implemented **rotation** about an angle θ and optionally about a pivot point (x_r, y_r).
- I implemented **reflection** about X-Axis, Y-Axis, Origin and the line $X = Y$.
- I implemented **scaling** uniformly, differentially and optionally about a fixed point (x_f, y_f).
- I implemented **shearing** about X-Axis and Y-Axis and optionally to include a reference line $Y = y$ and $X = x$ respectively.
- I created a function that allows the user to change the base polygon shape outputted in the window.
- I emphasized the use of **different colors** to highlight the transformed image, fixed points (if any) and reference lines (if any).
- I understood that OpenGL code executes in an **event-driven fashion**, thus while it waits for user-input, the output window might be stalled (unresponsive) and might need to be refreshed after the user I/O has finished.