

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:

```
1 //To perform 2D Transformations on objects and to render the final output
  along with the original object
2 //Translation, Rotation, Reflection, Scaling, Shearing
3
4 #include <windows.h>
5 #include <stdio.h>
6 #include <math.h>
7 #include <GL/glut.h>
8
9 const int WINDOW_WIDTH = 800;
10 const int WINDOW_HEIGHT = 800;
11 const int X_MIN = -400;
12 const int X_MAX = 400;
13 const int Y_MIN = -400;
14 const int Y_MAX = 400;
15
16 enum Axes {xAxis, yAxis};
17 enum Lines {XAxis, YAxis, Origin, XEqualsY};
18 enum Transformations { Translation = 1, Rotation = 2, RotationAboutPivot
  = 3, ReflectAboutX = 4,
19                        ReflectAboutY = 5, ReflectAboutO = 6,
  ReflectAboutXEqY = 7, UniformScale = 8,
20                        DifferentialScale = 9, ScaleAboutFixed = 10,
  ShearAboutX = 11, ShearAboutY = 12,
21                        ShearAboutXRef = 13, ShearAboutYRef = 14,
  ClearTransforms = 15, ChangePolygon = 16, Refresh = 17};
22
23 class Point{
24 private:
25     GLdouble x, y, h;
26
27 public:
28     Point(){
29         x = y = 0;
30         h = 1;
31     }
32
33     Point(GLint xCoord, GLint yCoord){
34         x = xCoord;
35         y = yCoord;
36         h = 1;
37     }
38
39     Point(GLint xCoord, GLint yCoord, GLint H){
40         x = xCoord;
41         y = yCoord;
42         h = H;
```

```

43     }
44
45     void setCoords(GLint xCoord, GLint yCoord){
46         x = xCoord;
47         y = yCoord;
48     }
49
50     GLdouble getX() const{
51         return x;
52     }
53
54     GLdouble getY() const{
55         return y;
56     }
57
58     GLdouble getH() const{
59         return h;
60     }
61
62     GLdouble getHomogenousX() const{
63         return x * h;
64     }
65
66     GLdouble getHomogenousY() const{
67         return y * h;
68     }
69
70     Point getTranslatedPoint(Point translationVector){
71         //For 2D Translation about a given translation vector
72
73         double translationMatrix[3][3] = { {1, 0, translationVector.
getHomogenousX()},
74                                             {0, 1, translationVector.
getHomogenousY()},
75                                             {0, 0, 1}};
76
77         double values[3];
78
79         for(int i = 0; i < 3; i++){
80             values[i] = translationMatrix[i][0] * getHomogenousX() +
81                       translationMatrix[i][1] * getHomogenousY() +
82                       translationMatrix[i][2] * getH();
83         }
84
85         return Point(values[0]/h, values[1]/h, values[2]);
86     }
87
88     Point getRotatedPoint(int rotationAngle, Point pivot = Point(0, 0, 1))
89     {
90         //For 2D Rotation about a given pivot by a given rotation angle

```

```

91     double rotationAngleInRadians = rotationAngle * 3.14159/180;
92     double cosAngle = cos(rotationAngleInRadians);
93     double sinAngle = sin(rotationAngleInRadians);
94
95     double xPivotValue = (pivot.getX() * (1 - cosAngle)) + (pivot.getY
() * sinAngle);
96     double yPivotValue = (pivot.getY() * (1 - cosAngle)) - (pivot.getX
() * sinAngle);
97
98     double rotationMatrix[3][3] = { {cosAngle, -sinAngle, xPivotValue
},
99                                     {sinAngle, cosAngle, yPivotValue},
100                                     {0, 0, 1}};
101
102     double values[3];
103
104     for(int i = 0; i < 3; i++){
105         values[i] = rotationMatrix[i][0] * getHomogenousX() +
106                     rotationMatrix[i][1] * getHomogenousY() +
107                     rotationMatrix[i][2] * getH();
108     }
109
110     return Point(values[0]/h, values[1]/h, values[2]);
111 }
112
113 Point getReflectionAboutXAxis(){
114     //For 2D Reflection about the X axis
115
116     double reflectionMatrix[3][3] = { {1, 0, 0},
117                                         {0, -1, 0},
118                                         {0, 0, 1}};
119
120     double values[3];
121
122     for(int i = 0; i < 3; i++){
123         values[i] = reflectionMatrix[i][0] * getHomogenousX() +
124                     reflectionMatrix[i][1] * getHomogenousY() +
125                     reflectionMatrix[i][2] * getH();
126     }
127
128     return Point(values[0]/h, values[1]/h, values[2]);
129 }
130
131 Point getReflectionAboutYAxis(){
132     //For 2D Reflection about the Y axis
133
134     double reflectionMatrix[3][3] = { {-1, 0, 0},
135                                         {0, 1, 0},
136                                         {0, 0, 1}};
137
138     double values[3];

```

```

139
140     for(int i = 0; i < 3; i++){
141         values[i] = reflectionMatrix[i][0] * getHomogenousX() +
142                 reflectionMatrix[i][1] * getHomogenousY() +
143                 reflectionMatrix[i][2] * getH();
144     }
145
146     return Point(values[0]/h, values[1]/h, values[2]);
147 }
148
149 Point getReflectionAboutOrigin(){
150     //For 2D Reflection about the Origin
151
152     double reflectionMatrix[3][3] = {    {-1, 0, 0},
153                                         {0, -1, 0},
154                                         {0, 0, 1}};
155
156     double values[3];
157
158     for(int i = 0; i < 3; i++){
159         values[i] = reflectionMatrix[i][0] * getHomogenousX() +
160                 reflectionMatrix[i][1] * getHomogenousY() +
161                 reflectionMatrix[i][2] * getH();
162     }
163
164     return Point(values[0]/h, values[1]/h, values[2]);
165 }
166
167 Point getReflectionAboutXEqualsY(){
168     //For 2D Reflection about the line X=Y
169
170     double reflectionMatrix[3][3] = {    {0, 1, 0},
171                                         {1, 0, 0},
172                                         {0, 0, 1}};
173
174     double values[3];
175
176     for(int i = 0; i < 3; i++){
177         values[i] = reflectionMatrix[i][0] * getHomogenousX() +
178                 reflectionMatrix[i][1] * getHomogenousY() +
179                 reflectionMatrix[i][2] * getH();
180     }
181
182     return Point(values[0]/h, values[1]/h, values[2]);
183 }
184
185 Point getScaledPoint(double ScaleX, double ScaleY, Point fixed){
186     //For 2D Scaling about a fixed point and scale factors for X & Y
187     axes
188     double xFixedValue = fixed.getX() * (1 - ScaleX);

```

```

189     double yFixedValue = fixed.getY() * (1 - ScaleY);
190
191     double scalingMatrix[3][3] = { {ScaleX, 0, xFixedValue},
192                                     {0, ScaleY, yFixedValue},
193                                     {0, 0, 1}};
194
195     double values[3];
196
197     for(int i = 0; i < 3; i++){
198         values[i] = scalingMatrix[i][0] * getHomogenousX() +
199                     scalingMatrix[i][1] * getHomogenousY() +
200                     scalingMatrix[i][2] * getH();
201     }
202
203     return Point(values[0]/h, values[1]/h, values[2]);
204 }
205
206 Point getShearAboutXAxis(double shearParam, double yRefLine = 0){
207     //For shearing about X axis
208
209     double shearMatrix[3][3] = {{1, shearParam, -shearParam * yRefLine
210
211                                     {0, 1, 0},
212                                     {0, 0, 1}};
213
214     double values[3];
215
216     for(int i = 0; i < 3; i++){
217         values[i] = shearMatrix[i][0] * getHomogenousX() +
218                     shearMatrix[i][1] * getHomogenousY() +
219                     shearMatrix[i][2] * getH();
220     }
221
222     return Point(values[0]/h, values[1]/h, values[2]);
223 }
224
225 Point getShearAboutYAxis(double shearParam, double xRefLine = 0){
226     //For shearing about Y axis
227
228     double shearMatrix[3][3] = {{1, 0, -shearParam * xRefLine},
229                                     {shearParam, 1, 0},
230                                     {0, 0, 1}};
231
232     double values[3];
233
234     for(int i = 0; i < 3; i++){
235         values[i] = shearMatrix[i][0] * getHomogenousX() +
236                     shearMatrix[i][1] * getHomogenousY() +
237                     shearMatrix[i][2] * getH();
238     }

```

```

239         return Point(values[0]/h, values[1]/h, values[2]);
240     }
241 };
242
243
244 class PolygonShape{
245 private:
246     int numVertices;
247     Point *points;
248
249 public:
250     PolygonShape(){
251         numVertices = 0;
252     }
253
254     PolygonShape(int noVertices){
255         numVertices = noVertices;
256         points = new Point[numVertices];
257     }
258
259     int getVertexCount() const{
260         return numVertices;
261     }
262
263     Point getPoint(int i){
264         return points[i];
265     }
266
267     void setVertices(int noVertices){
268         numVertices = noVertices;
269         points = new Point[numVertices];
270     }
271
272     void setPoint(int i, GLint x, GLint y){
273         points[i].setCoords(x, y);
274     }
275 };
276
277
278 void initializeDisplay();
279 void plotComponents();
280 void dummyFunction();
281 void dummyKeyFunction(unsigned char key, int x, int y);
282 void transformationMenu(int option);
283 void plotTransformation();
284 void drawAxes();
285 void drawPolygon(PolygonShape polygon);
286 void translatePolygon(PolygonShape polygon, Point translationVector);
287 void reflectPolygon(PolygonShape polygon, Lines line);
288 void rotatePolygon(PolygonShape polygon, int rotationAngle, Point pivot =
    Point(0, 0, 1));

```



```

289 void scalePolygon(PolygonShape polygon, double scaleX, double scaleY,
    Point fixed = Point(0, 0, 1));
290 void shearPolygon(PolygonShape polygon, Axes axis, double shearParam,
    double refLine = 0);
291
292 PolygonShape polygon;           //Global PolygonShape object to be plotted
    on the graph
293 int chosenTransformation = 0;    //Global variable to keep track of chosen
    transformation
294
295 int main(int argc, char **argv){
296     glutInit(&argc, argv);
297     glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
298     glutInitWindowPosition(0, 0);
299     glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
300     glutCreateWindow("2D Transformations - Examples");
301
302     printf("\n-----[2D TRANSFORMATIONS]-----\n");
303     printf("\nUsage:\tRight-Click the GLUT Window to select a
    transformation.");
304     printf("\n\tProvide input for the necessary parameters in this window.
    ");
305     printf("\n\tRefresh the output window if it becomes unresponsive
    during console I/O.");
306     printf("\n\n-----[2D TRANSFORMATIONS]-----\n\n");
307
308     //Set the initial default polygon for the graph
309     polygon.setVertices(4);
310     polygon.setPoint(0, 0, 0);
311     polygon.setPoint(1, 0, 50);
312     polygon.setPoint(2, 100, 50);
313     polygon.setPoint(3, 100, 0);
314
315     initializeDisplay();
316     glutDisplayFunc(dummyFunction);
317     plotComponents();
318
319     glutCreateMenu(transformationMenu);
320     glutAddMenuEntry("Translation", 1);
321     glutAddMenuEntry("Rotation", 2);
322     glutAddMenuEntry("Rotation About Pivot Point", 3);
323     glutAddMenuEntry("Reflection About X Axis", 4);
324     glutAddMenuEntry("Reflection About Y Axis", 5);
325     glutAddMenuEntry("Reflection About Origin", 6);
326     glutAddMenuEntry("Reflection About X = Y", 7);
327     glutAddMenuEntry("Uniform Scaling", 8);
328     glutAddMenuEntry("Differential Scaling", 9);
329     glutAddMenuEntry("Scaling About Fixed Point", 10);
330     glutAddMenuEntry("Shear About X Axis", 11);
331     glutAddMenuEntry("Shear About Y Axis", 12);
332     glutAddMenuEntry("Shear About X Axis About Y = y", 13);

```

```

333     glutAddMenuEntry("Shear About Y Axis About X = x", 14);
334     glutAddMenuEntry("Clear Transformations", 15);
335     glutAddMenuEntry("Change Polygon", 16);
336     glutAddMenuEntry("Refresh Screen", 17);
337     glutAttachMenu(GLUT_RIGHT_BUTTON);
338
339     glutMainLoop();
340
341     return 1;
342 }
343
344 void transformationMenu(int option){
345     chosenTransformation = option;
346     plotTransformation();
347 }
348
349 void initializeDisplay(){
350     //Initialize the display parameters
351
352     glClearColor(1, 1, 1, 0);
353     glMatrixMode(GL_PROJECTION);
354     gluOrtho2D(X_MIN, X_MAX, Y_MIN, Y_MAX);
355     glClear(GL_COLOR_BUFFER_BIT);    //Clear the display window
356
357     glEnable(GL_BLEND);    //enable blending (translucent colors)
358     glDepthMask(GL_FALSE);
359     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA); //set the blend
function for translucency
360 }
361
362 void plotComponents(){
363     //Plot the axes and the base polygon
364
365     glClear(GL_COLOR_BUFFER_BIT);    //Clear the display window
366     drawAxes();
367     drawPolygon(polygon);
368     glFlush();
369 }
370
371 void dummyFunction(){
372     //Placeholder function to be called in glutDisplayFunc
373 }
374
375 void plotTransformation(){
376     //Plot the specified transformation
377
378     switch(chosenTransformation){
379         case Translation:{
380             double x, y;
381             printf("\n\n-----[TRANSLATION]-----\n");
382             printf("\n\tEnter the Translation Vector Magnitudes: ");

```

```

383
384     printf("\n\t\tX Component: ");
385     scanf("%lf", &x);
386     printf("\n\t\tY Component: ");
387     scanf("%lf", &y);
388     Point translationVector(x, y, 1);
389     translatePolygon(polygon, translationVector);
390
391     printf("\n\n-----[TRANSFORMATION COMPLETE]-----\n");
392     break;
393 }
394
395 case Rotation:{
396     double rotationAngle;
397     printf("\n\n-----[ROTATION]-----\n");
398     printf("\n\tEnter the Rotation Angle: ");
399
400     scanf("%lf", &rotationAngle);
401     rotatePolygon(polygon, rotationAngle);
402
403     printf("\n\n-----[TRANSFORMATION COMPLETE]-----\n");
404     break;
405 }
406
407 case RotationAboutPivot:{
408     double rotationAngle, x, y;
409     printf("\n\n-----[ROTATION ABOUT PIVOT]-----\n");
410     printf("\n\tEnter the Rotation Angle: ");
411
412     scanf("%lf", &rotationAngle);
413     printf("\n\tEnter Pivot Point: ");
414     printf("\n\t\tEnter X Coordinate: ");
415     scanf("%lf", &x);
416     printf("\n\t\tEnter Y Coordinate: ");
417     scanf("%lf", &y);
418     rotatePolygon(polygon, rotationAngle, Point(x, y, 1));
419
420     printf("\n\n-----[TRANSFORMATION COMPLETE]-----\n");
421     break;
422 }
423
424 case ReflectAboutX:{
425     printf("\n\n-----[REFLECTION ABOUT X AXIS]-----\n");
426
427     reflectPolygon(polygon, XAxis);
428
429     printf("\n\n-----[TRANSFORMATION COMPLETE]-----\n");
430     break;
431 }
432
433 case ReflectAboutY:{

```

```

434         printf("\n\n-----[REFLECTION ABOUT Y AXIS]-----\n");
435
436         reflectPolygon(polygon, YAxis);
437
438         printf("\n\n-----[TRANSFORMATION COMPLETE]-----\n");
439         break;
440     }
441
442     case ReflectAboutO:{
443         printf("\n\n-----[REFLECTION ABOUT ORIGIN]-----\n");
444
445         reflectPolygon(polygon, Origin);
446
447         printf("\n\n-----[TRANSFORMATION COMPLETE]-----\n");
448         break;
449     }
450
451     case ReflectAboutXEqY:{
452         printf("\n\n-----[REFLECTION ABOUT X = Y]-----\n");
453
454         reflectPolygon(polygon, XEqualsY);
455
456         printf("\n\n-----[TRANSFORMATION COMPLETE]-----\n");
457         break;
458     }
459
460     case UniformScale:{
461         double scaleFactor;
462         printf("\n\n-----[UNIFORM SCALING]-----\n");
463
464         printf("\n\tEnter the Scaling Factors: ");
465         scanf("%lf", &scaleFactor);
466
467         scalePolygon(polygon, scaleFactor, scaleFactor);
468
469         printf("\n\n-----[TRANSFORMATION COMPLETE]-----\n");
470         break;
471     }
472
473     case DifferentialScale:{
474         double xScale, yScale;
475         printf("\n\n-----[DIFFERENTIAL SCALING]-----\n");
476
477         printf("\n\tEnter the Scaling Factors: ");
478
479         printf("\n\t\tX Scale Factor: ");
480         scanf("%lf", &xScale);
481         printf("\n\t\tY Scale Factor: ");
482         scanf("%lf", &yScale);
483
484         scalePolygon(polygon, xScale, yScale);

```

```

485
486     printf("\n\n-----[TRANSFORMATION COMPLETE]-----\n");
487     break;
488 }
489
490 case ScaleAboutFixed:{
491     double xScale, yScale, xFixed, yFixed;
492     printf("\n\n-----[SCALING ABOUT FIXED POINT]-----\n");
493
494     printf("\n\tEnter the Scaling Factors: ");
495
496     printf("\n\t\tX Scale Factor: ");
497     scanf("%lf", &xScale);
498     printf("\n\t\tY Scale Factor: ");
499     scanf("%lf", &yScale);
500
501     printf("\n\tEnter the Fixed Point: ");
502
503     printf("\n\t\tX Coordinate: ");
504     scanf("%lf", &xFixed);
505     printf("\n\t\tY Coordinate: ");
506     scanf("%lf", &yFixed);
507
508     scalePolygon(polygon, xScale, yScale, Point(xFixed, yFixed, 1)
509 );
510
511     printf("\n\n-----[TRANSFORMATION COMPLETE]-----\n");
512     break;
513 }
514 case ShearAboutX:{
515     double shearParam;
516     printf("\n\n-----[SHEARING ABOUT X AXIS]-----\n");
517
518     printf("\n\tEnter the Shearing Parameter: ");
519     scanf("%lf", &shearParam);
520
521     shearPolygon(polygon, xAxis, shearParam);
522
523     printf("\n\n-----[TRANSFORMATION COMPLETE]-----\n");
524     break;
525 }
526
527 case ShearAboutY:{
528     double shearParam;
529     printf("\n\n-----[SHEARING ABOUT Y AXIS]-----\n");
530
531     printf("\n\tEnter the Shearing Parameter: ");
532     scanf("%lf", &shearParam);
533
534     shearPolygon(polygon, yAxis, shearParam);

```

```

535
536     printf("\n\n-----[TRANSFORMATION COMPLETE]-----\n");
537     break;
538 }
539
540 case ShearAboutXRef:{
541     double shearParam, yRef;
542     printf("\n\n-----[SHEARING ABOUT X AXIS ABOUT REF. LINE Y = y
]-----\n");
543
544     printf("\n\tEnter the Shearing Parameter: ");
545     scanf("%lf", &shearParam);
546
547     printf("\n\tEnter the Reference Line Constant y (Y = y): ");
548     scanf("%lf", &yRef);
549
550     shearPolygon(polygon, xAxis, shearParam, yRef);
551
552     printf("\n\n-----[TRANSFORMATION COMPLETE]-----\n");
553     break;
554 }
555
556 case ShearAboutYRef:{
557     double shearParam, xRef;
558     printf("\n\n-----[SHEARING ABOUT Y AXIS ABOUT REF. LINE X = x
]-----\n");
559
560     printf("\n\tEnter the Shearing Parameter: ");
561     scanf("%lf", &shearParam);
562
563     printf("\n\tEnter the Reference Line Constant x (X = x): ");
564     scanf("%lf", &xRef);
565
566     shearPolygon(polygon, yAxis, shearParam, xRef);
567
568     printf("\n\n-----[TRANSFORMATION COMPLETE]-----\n");
569     break;
570 }
571
572 case ClearTransforms:{
573     plotComponents();    //Re plot the base graph
574     break;
575 }
576
577 case ChangePolygon:{
578     int i = 0, vertices = 0;
579     double x, y;
580     printf("\n\n-----[CHANGE POLYGON]-----\n");
581
582     printf("\n\tEnter the number of vertices: ");
583     scanf("%d", &vertices);

```

```

584
585     polygon.setVertices(vertices);
586
587     while(i < vertices){
588         printf("\n\tEnter Vertex %d Coordinates:", i+1);
589         printf("\n\t\tX: ");
590         scanf("%lf", &x);
591         printf("\n\t\tY: ");
592         scanf("%lf", &y);
593         polygon.setPoint(i, x, y);
594         i++;
595     }
596
597     plotComponents();    //Re plot the base graph
598
599     printf("\n\n-----[POLYGON CHANGED]-----\n");
600     break;
601 }
602
603 case Refresh:{
604     //Draw an object off screen to refresh the display buffer
605     glBegin(GL_LINES);
606     glVertex2f(2000, 2000);
607     glVertex2f(2001, 2001);
608
609     glEnd();
610     break;
611 }
612 }
613
614 glFlush();
615 glutPostRedisplay();    //IMPORTANT: To refresh the window with the
new updated plots
616 }
617
618 void drawAxes(){
619     //To draw the X and Y axes
620
621     glColor3d(0, 0, 0); //Black color
622
623     glBegin(GL_LINES);
624
625     //X-axis
626     glVertex2f(X_MIN, 0);
627     glVertex2f(X_MAX, 0);
628
629     //Y-axis
630     glVertex2f(0, Y_MIN);
631     glVertex2f(0, Y_MAX);
632
633     glEnd();

```

```

634 }
635
636 void drawPolygon(PolygonShape polygon){
637     //To draw a given polygon
638
639     glColor3d(1, 0, 0); //Red color
640
641     glBegin(GL_POLYGON);
642
643     for(int i = 0; i < polygon.getVertexCount(); i++){
644         Point p = polygon.getPoint(i);
645         glVertex2f(p.getX(), p.getY());
646     }
647
648     glEnd();
649 }
650
651 void translatePolygon(PolygonShape polygon, Point translationVector){
652     //To translate a given polygon using the translation vector
653
654     glColor4f(0, 0, 1, 0.6); //Blue color
655
656     glBegin(GL_POLYGON);
657
658     for(int i = 0; i < polygon.getVertexCount(); i++){
659         Point p = polygon.getPoint(i);
660         Point pDash = p.getTranslatedPoint(translationVector);
661         glVertex2f(pDash.getX(), pDash.getY()); //Plot the normal
coordinates
662     }
663
664     glEnd();
665 }
666
667 void rotatePolygon(PolygonShape polygon, int rotationAngle, Point pivot){
668     //To rotate a given polygon using the rotation angle and pivot point
669
670     //Plot the pivot point
671     glColor3d(1, 0, 1); //Purple Color
672     glPointSize(5);
673
674     glBegin(GL_POINTS);
675     glVertex2f(pivot.getX(), pivot.getY());
676     glEnd();
677
678     glColor4f(0, 0, 1, 0.6); //Blue Color
679
680     glBegin(GL_POLYGON);
681
682     for(int i = 0; i < polygon.getVertexCount(); i++){
683         Point p = polygon.getPoint(i);

```



```

684         Point pDash = p.getRotatedPoint(rotationAngle, pivot);
685         glVertex2f(pDash.getX(), pDash.getY()); //Plot the normal
coordinates
686     }
687
688     glEnd();
689 }
690
691 void reflectPolygon(PolygonShape polygon, Lines line){
692     //To reflect a polygon about a given line
693
694     //Plot the given line
695     glColor3f(1, 0, 1);
696
697     glBegin(GL_LINES);
698
699     switch(line){
700         case XAxis:
701             glVertex2f(X_MIN, 0);
702             glVertex2f(X_MAX, 0);
703             break;
704         case YAxis:
705             glVertex2f(0, Y_MIN);
706             glVertex2f(0, Y_MAX);
707             break;
708         case Origin:
709             glVertex2f(X_MIN, 0);
710             glVertex2f(X_MAX, 0);
711             glVertex2f(0, Y_MIN);
712             glVertex2f(0, Y_MAX);
713             break;
714         case XEqualsY:
715             glVertex2f(X_MIN, Y_MIN);
716             glVertex2f(X_MAX, Y_MAX);
717             break;
718         default:
719             return;
720     }
721
722     glEnd();
723
724     glColor4f(0, 0, 1, 0.6); //Blue Color
725
726     glBegin(GL_POLYGON);
727
728     for(int i = 0; i < polygon.getVertexCount(); i++){
729         Point p = polygon.getPoint(i);
730         Point pDash;
731
732         switch(line){
733             case XAxis:

```

```

734         pDash = p.getReflectionAboutXAxis();
735         break;
736     case YAxis:
737         pDash = p.getReflectionAboutYAxis();
738         break;
739     case Origin:
740         pDash = p.getReflectionAboutOrigin();
741         break;
742     case XEqualsY:
743         pDash = p.getReflectionAboutXEqualsY();
744         break;
745     default:
746         return;
747 }
748
749     glVertex2f(pDash.getX(), pDash.getY()); //Plot the normal
coordinates
750 }
751
752 glEnd();
753 }
754
755 void scalePolygon(PolygonShape polygon, double scaleX, double scaleY,
Point fixed){
756     //To translate a given polygon using the scale factors and fixed point
757
758     //Plot the fixed point
759     glColor3d(1, 0, 1); //Purple Color
760     glPointSize(5);
761
762     glBegin(GL_POINTS);
763     glVertex2f(fixed.getX(), fixed.getY());
764     glEnd();
765
766     glColor4f(0, 0, 1, 0.6); //Blue Color
767
768     glBegin(GL_POLYGON);
769
770     for(int i = 0; i < polygon.getVertexCount(); i++){
771         Point p = polygon.getPoint(i);
772         Point pDash = p.getScaledPoint(scaleX, scaleY, fixed);
773         glVertex2f(pDash.getX(), pDash.getY()); //Plot the normal
coordinates
774     }
775
776     glEnd();
777 }
778
779 void shearPolygon(PolygonShape polygon, Axes axis, double shearParam,
double refLine){
780     //To shear a polygon about axis and shear parameter

```

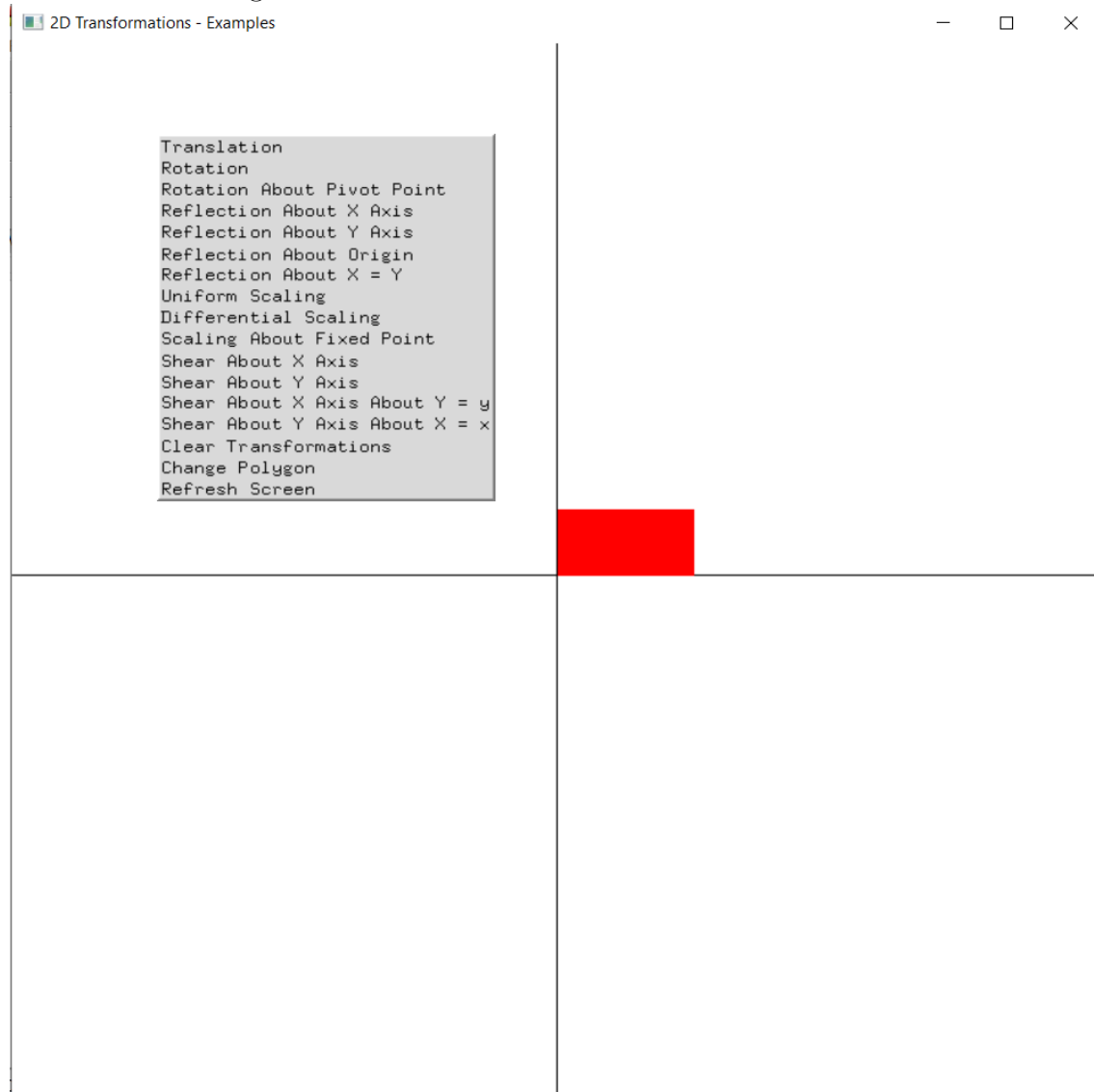
```

781
782 //Plot the given line
783 glColor3f(1, 0, 1);
784
785 glBegin(GL_LINES);
786
787 switch(axis){
788     case xAxis:
789         glVertex2f(X_MIN, 0);
790         glVertex2f(X_MAX, 0);
791         break;
792     case yAxis:
793         glVertex2f(0, Y_MIN);
794         glVertex2f(0, Y_MAX);
795         break;
796     default:
797         return;
798 }
799
800 glEnd();
801
802 glColor4f(0, 0, 1, 0.6); //Blue Color, with alpha (transparency)
803 factor as 0.6
804
805 glBegin(GL_POLYGON);
806
807 for(int i = 0; i < polygon.getVertexCount(); i++){
808     Point p = polygon.getPoint(i);
809     Point pDash;
810
811     switch(axis){
812         case xAxis:
813             pDash = p.getShearAboutXAxis(shearParam, refLine);
814             break;
815         case yAxis:
816             pDash = p.getShearAboutYAxis(shearParam, refLine);
817             break;
818         default:
819             return;
820     }
821
822     glVertex2f(pDash.getX(), pDash.getY()); //Plot the normal
823 coordinates
824 }
825 glEnd();
826 }

```

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]-----
```

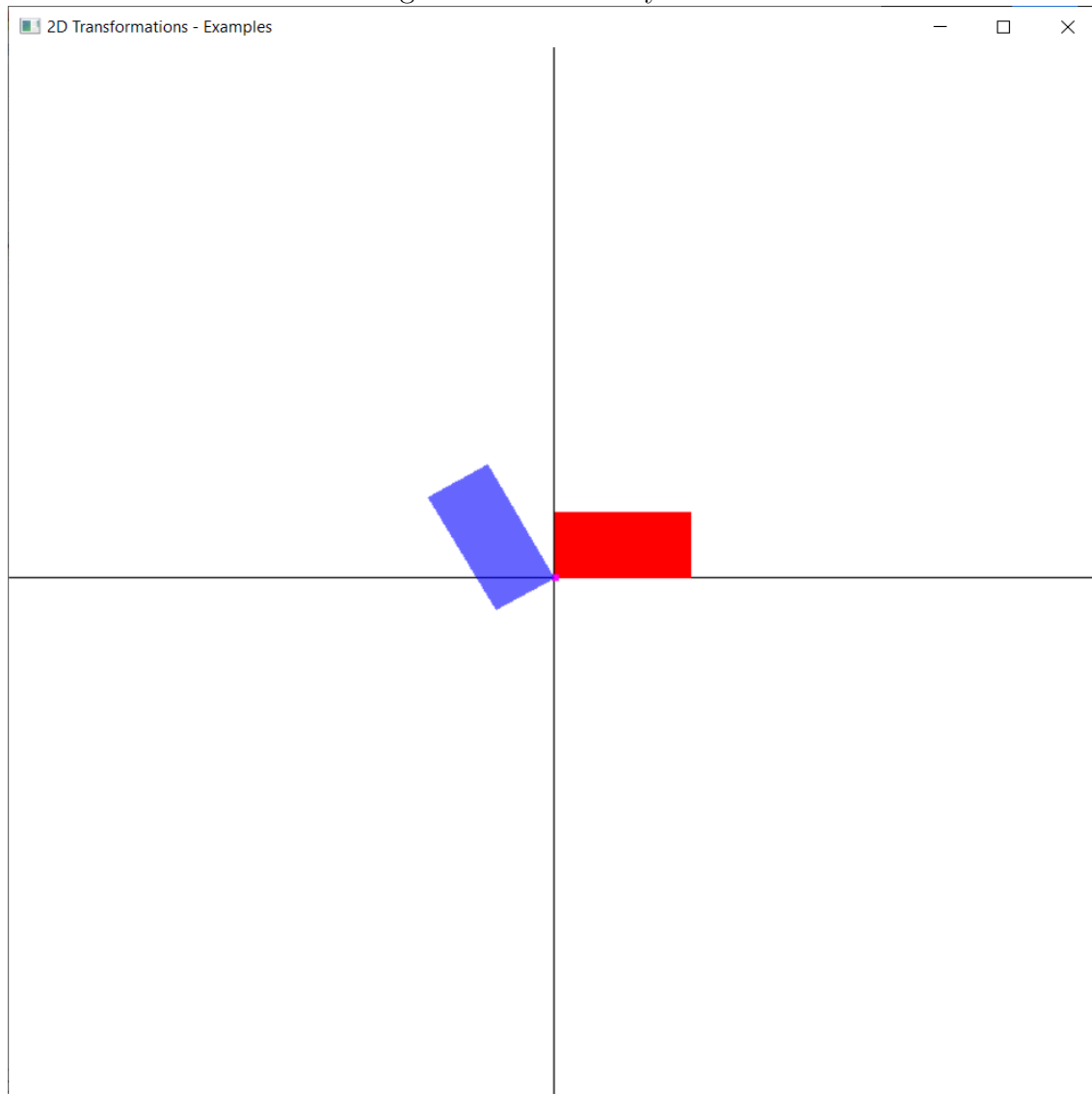
Output: Translation

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



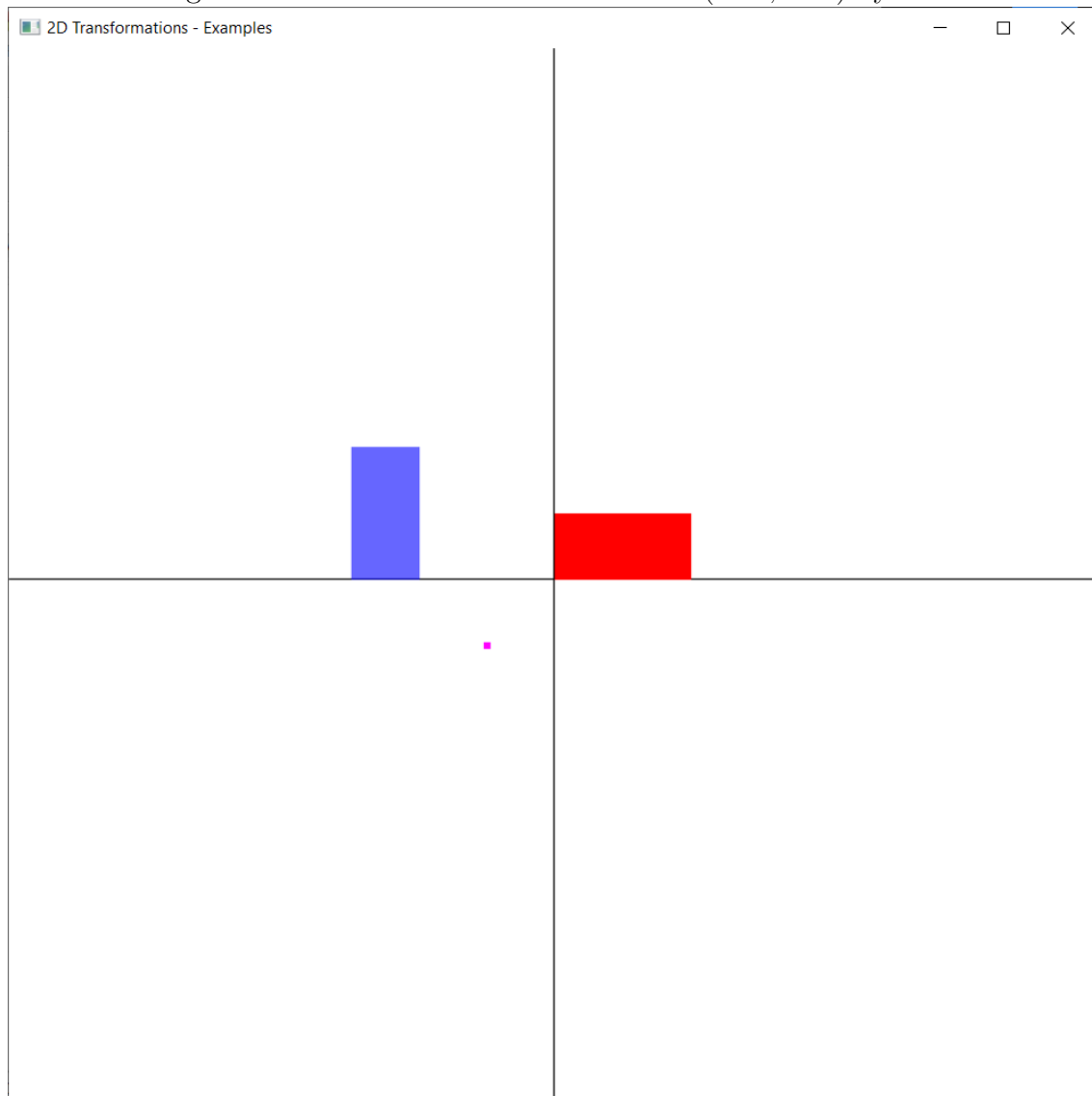
Output: Rotation

Figure 4: Rotation by 120° .



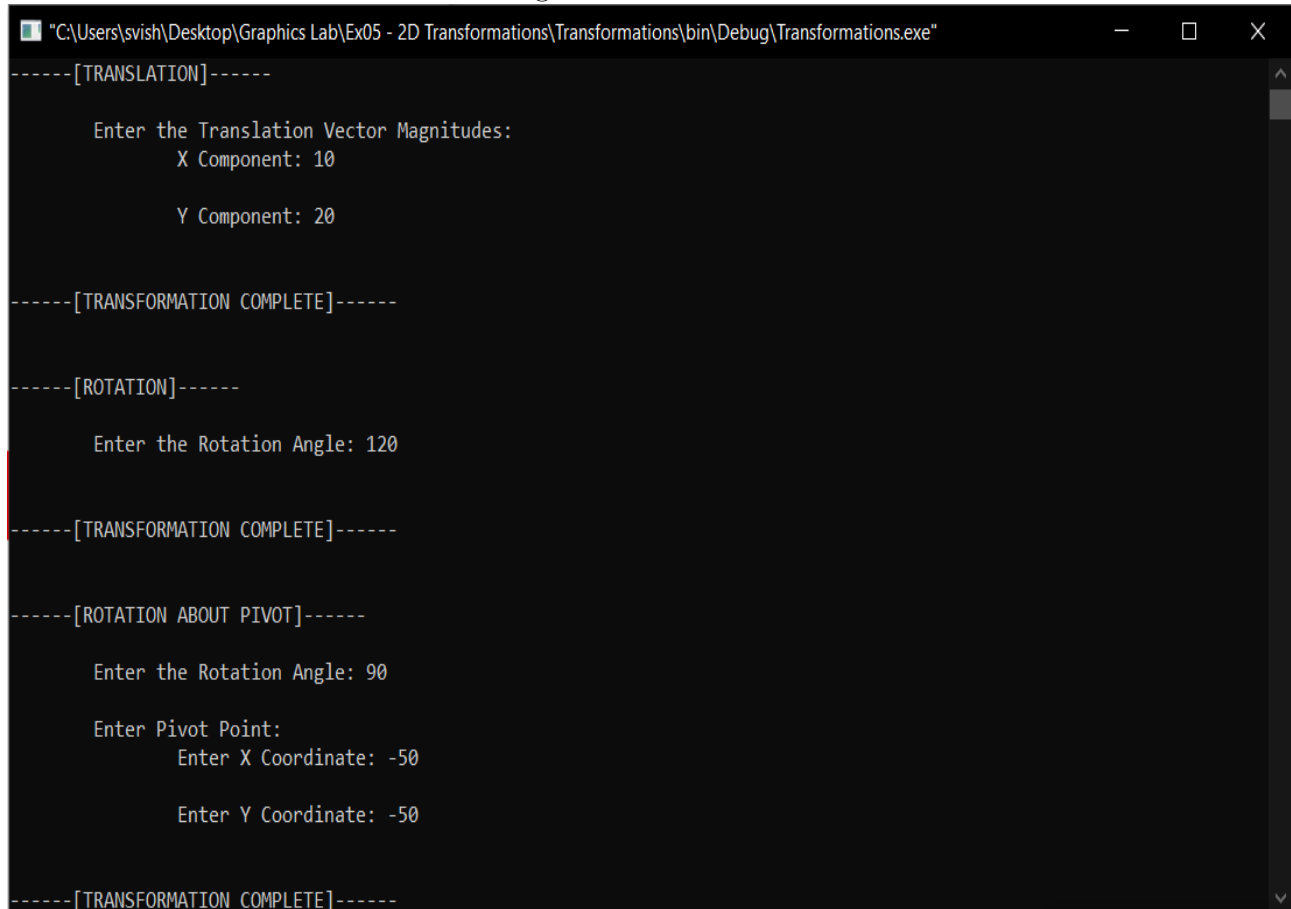
Output: Rotation About A Pivot Point

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

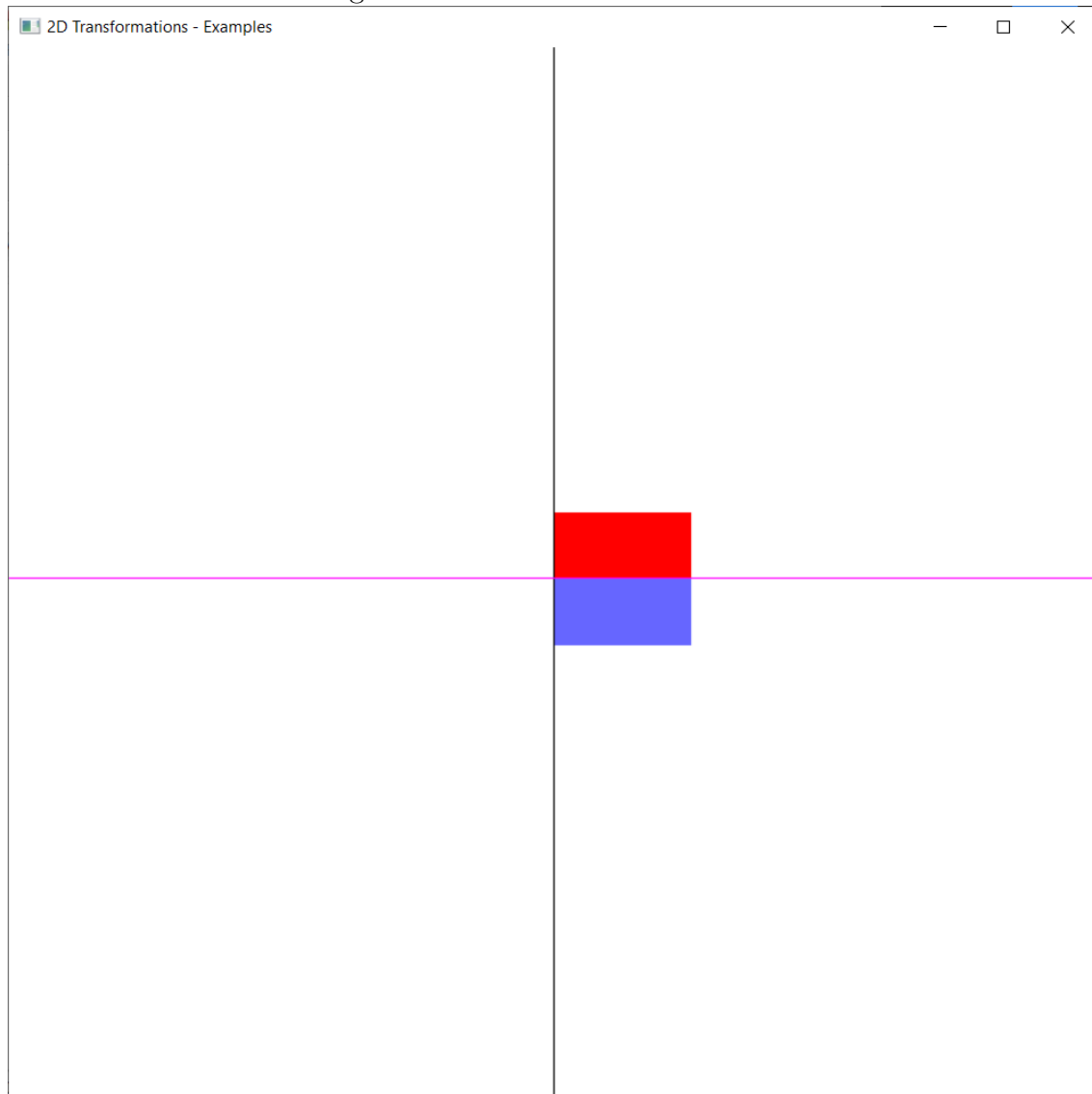
    Enter Pivot Point:
      Enter X Coordinate: -50

      Enter Y Coordinate: -50

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

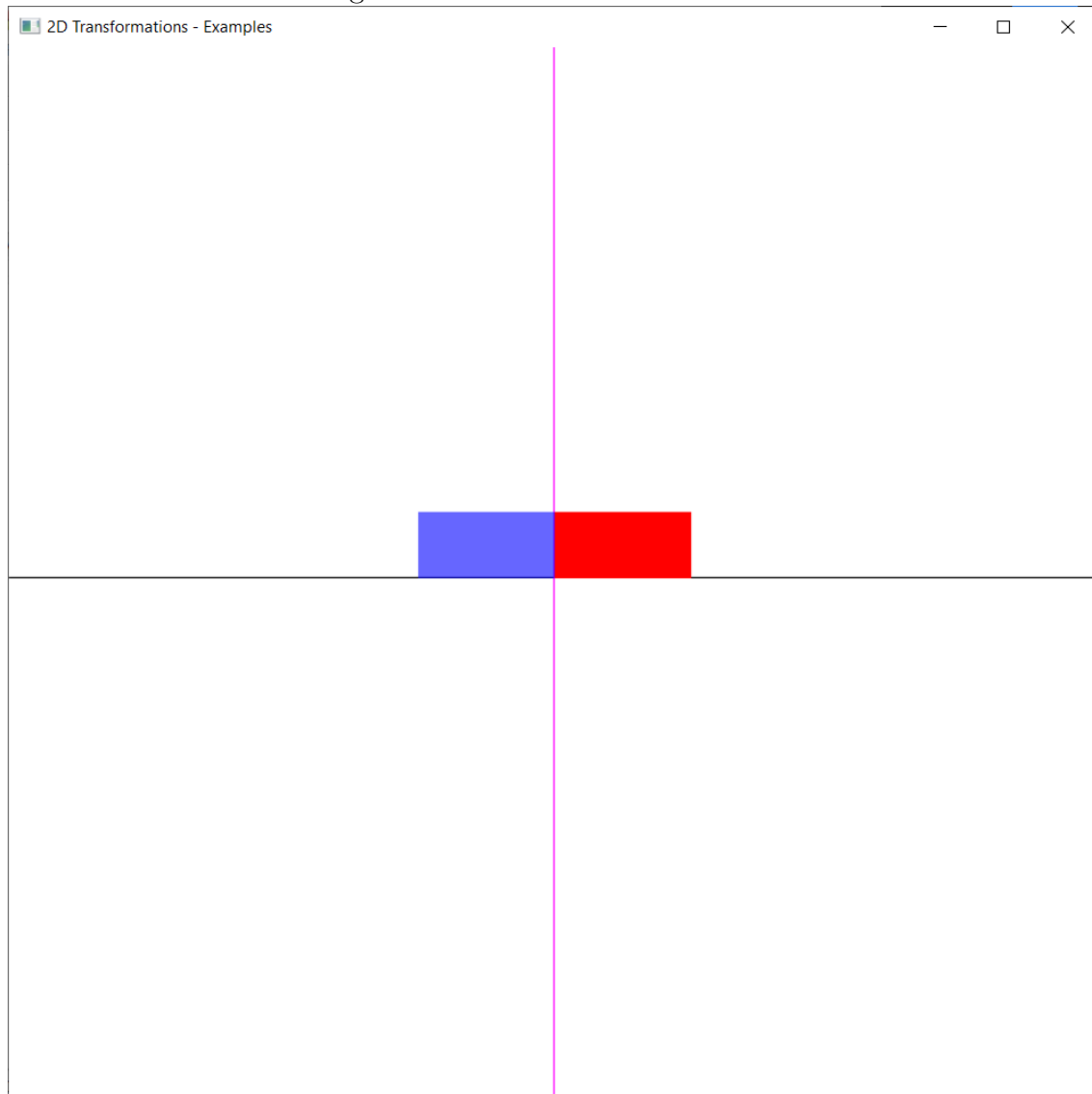
Output: Reflection About X Axis

Figure 7: Reflection About X Axis.



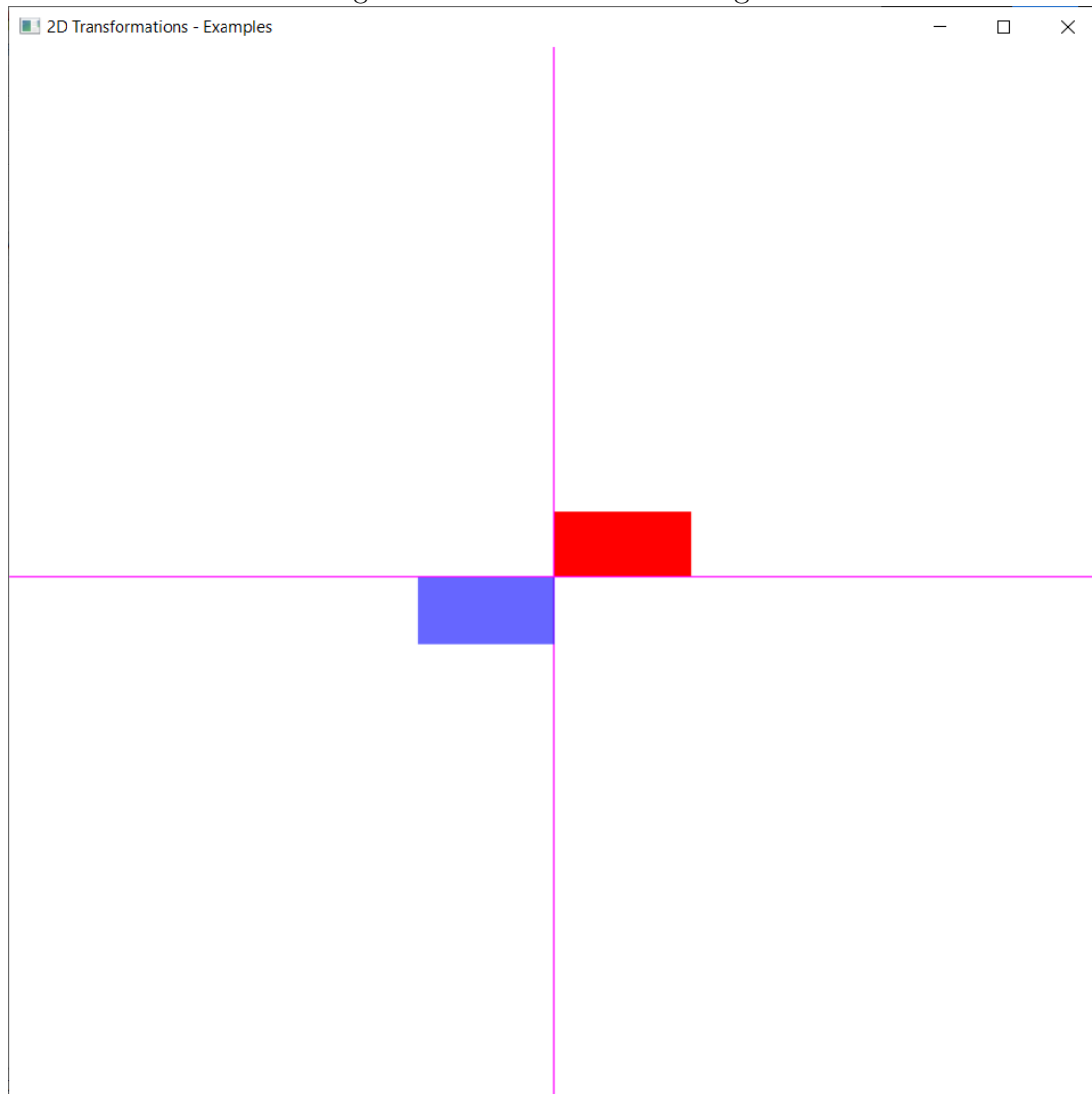
Output: Reflection About Y Axis

Figure 8: Reflection About Y Axis.



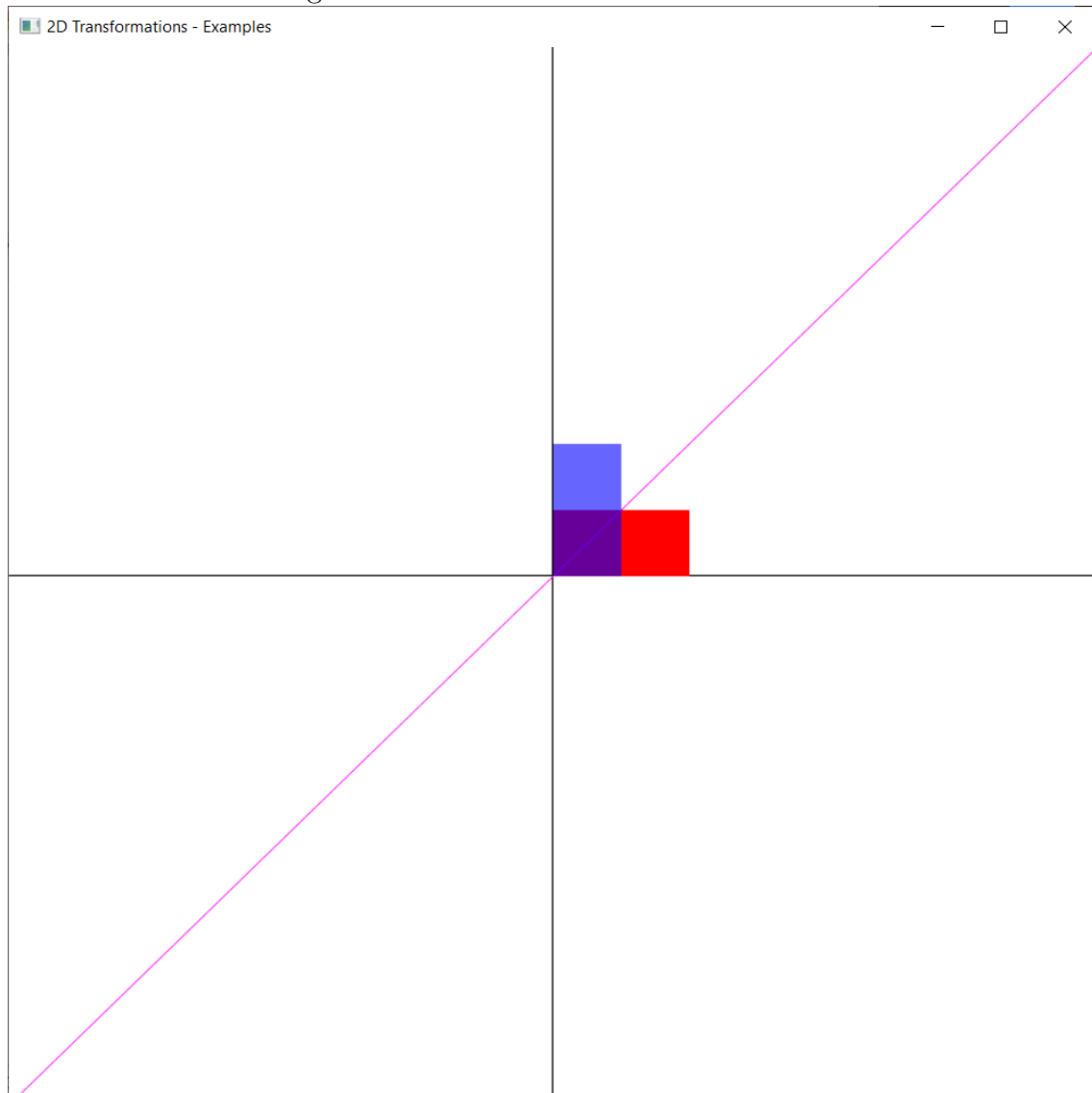
Output: Reflection About Origin

Figure 9: Reflection About Origin.



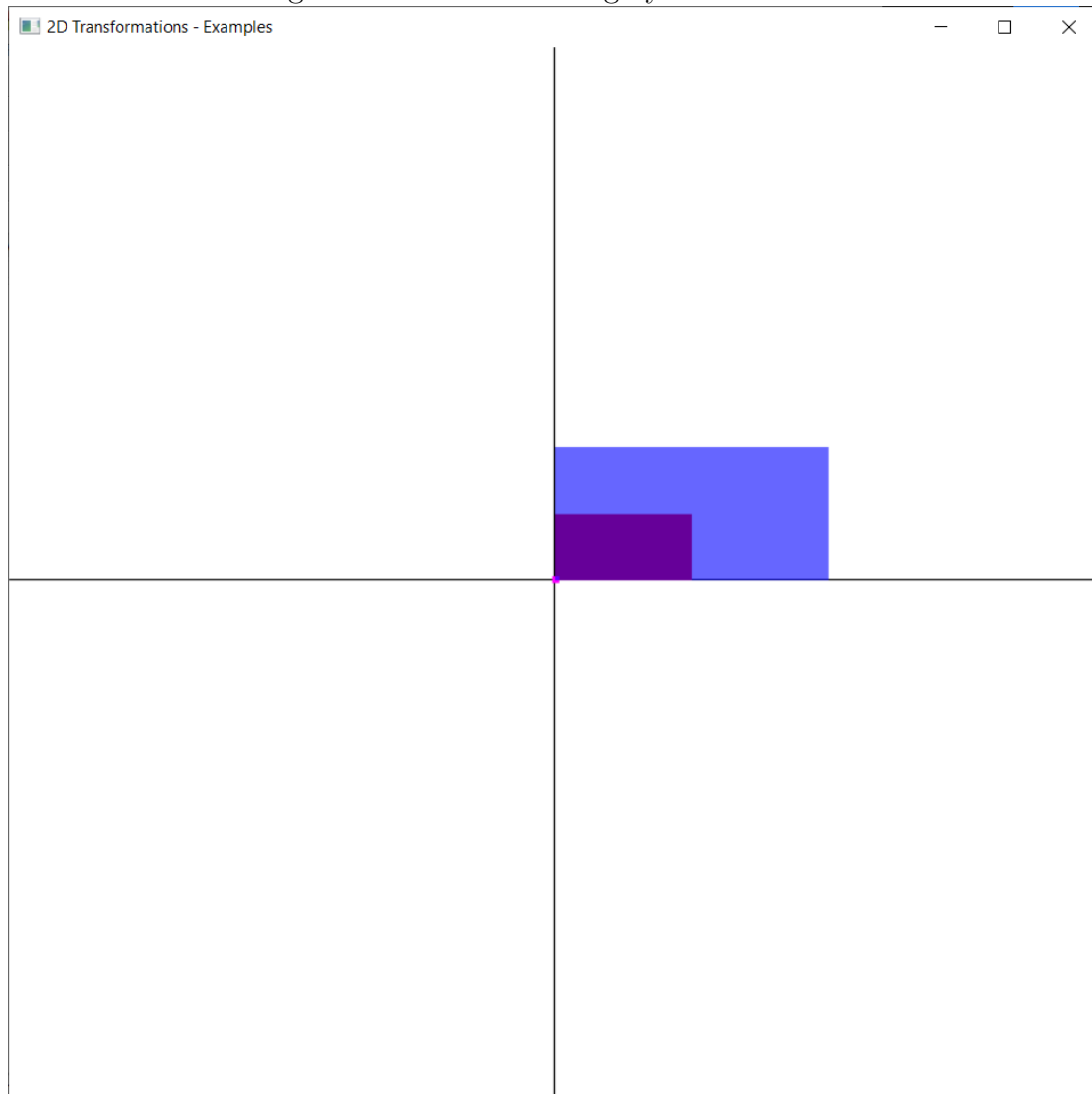
Output: Reflection About Line $X = Y$

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



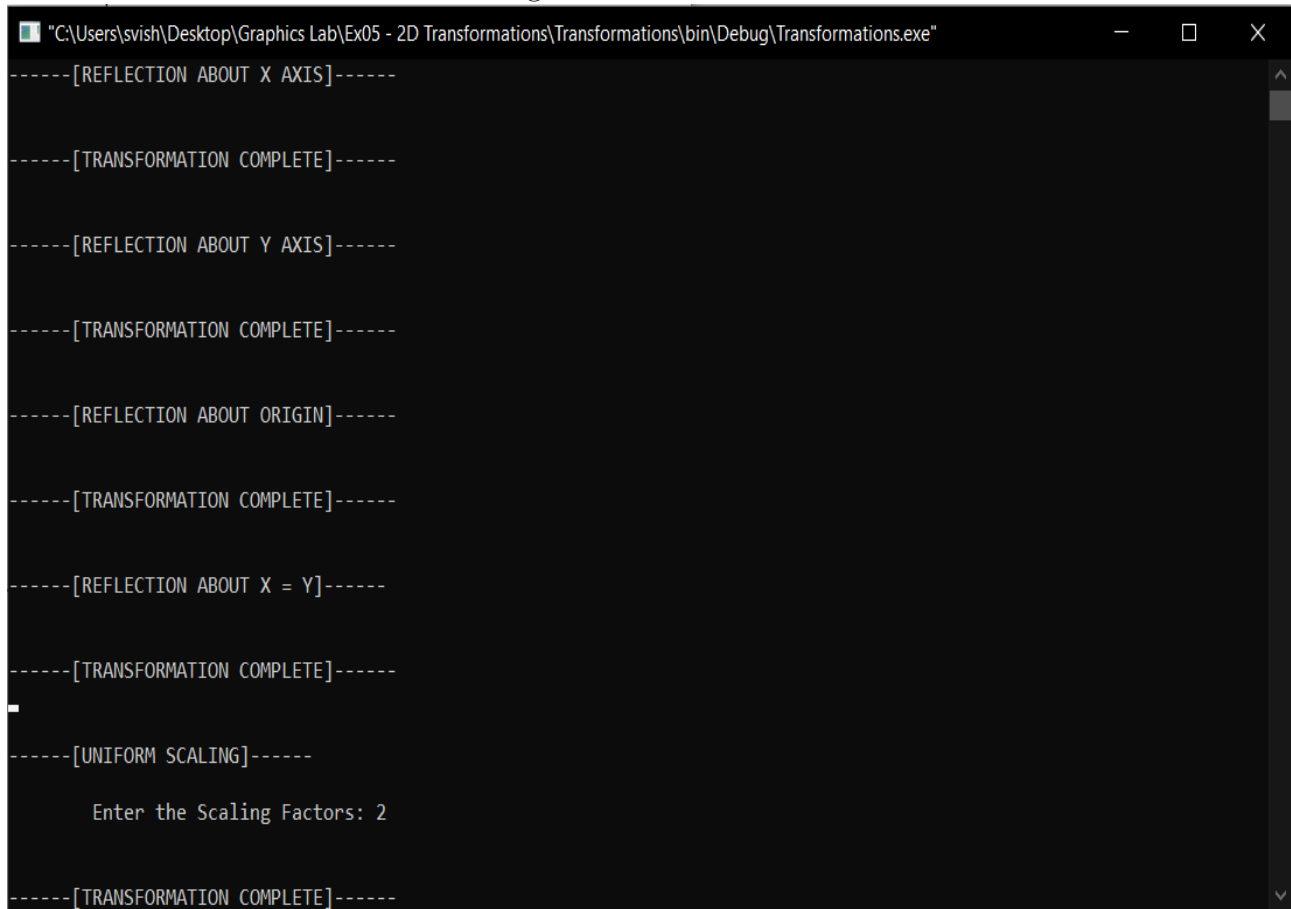
Output: Uniform Scaling

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]-----
```

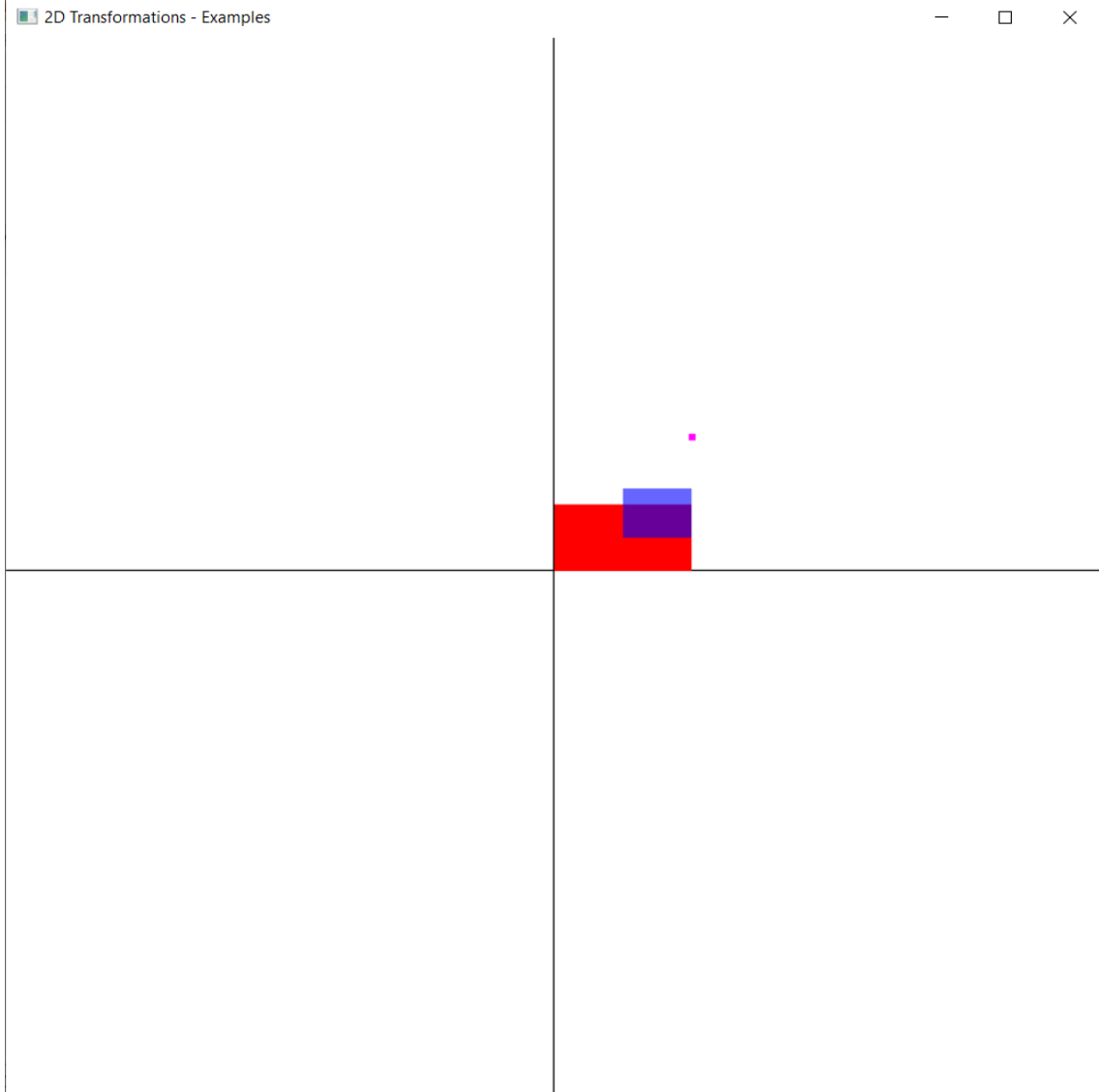
Output: Differential Scaling

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



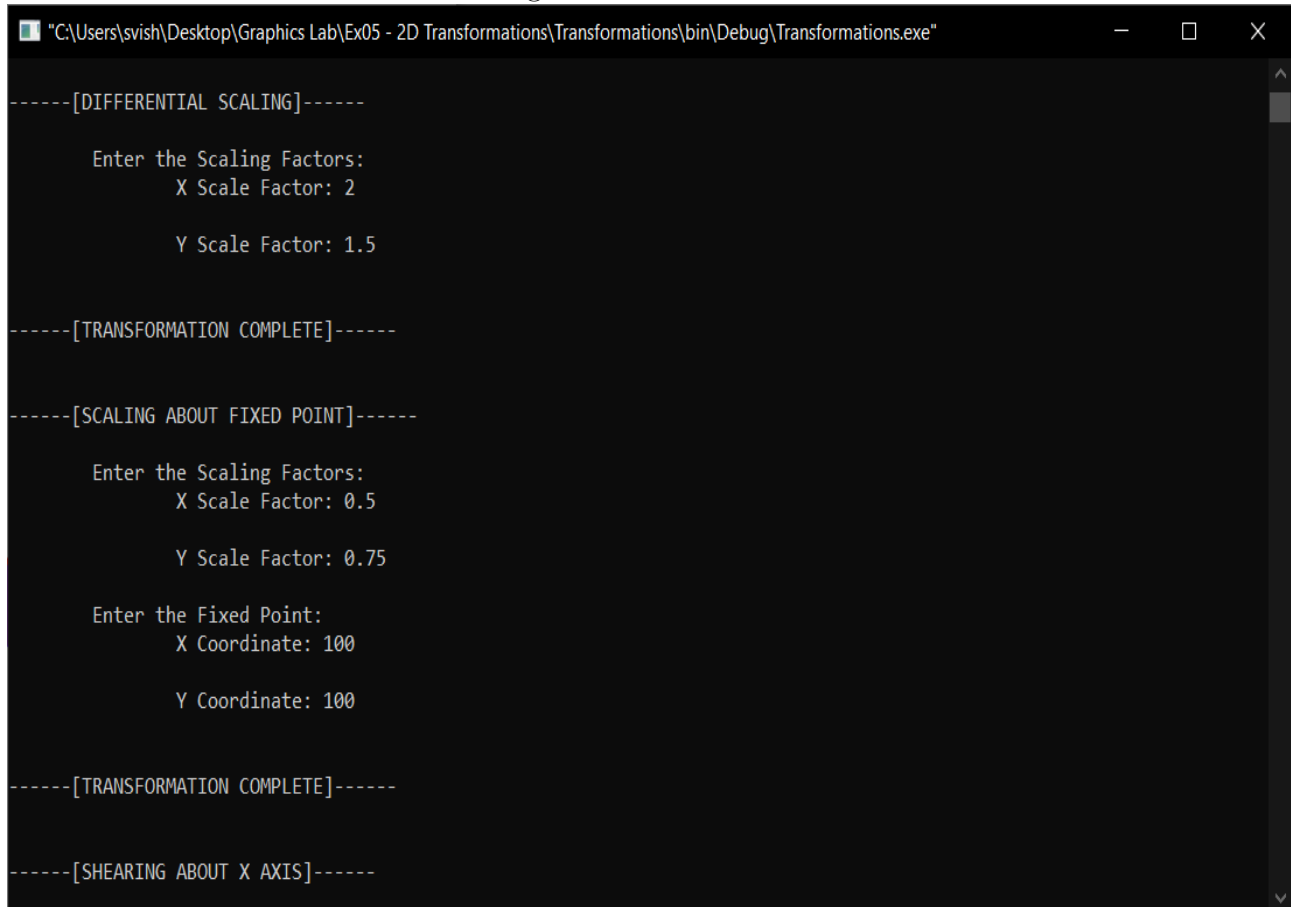
Output: Differential Scaling About A Fixed Point

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



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

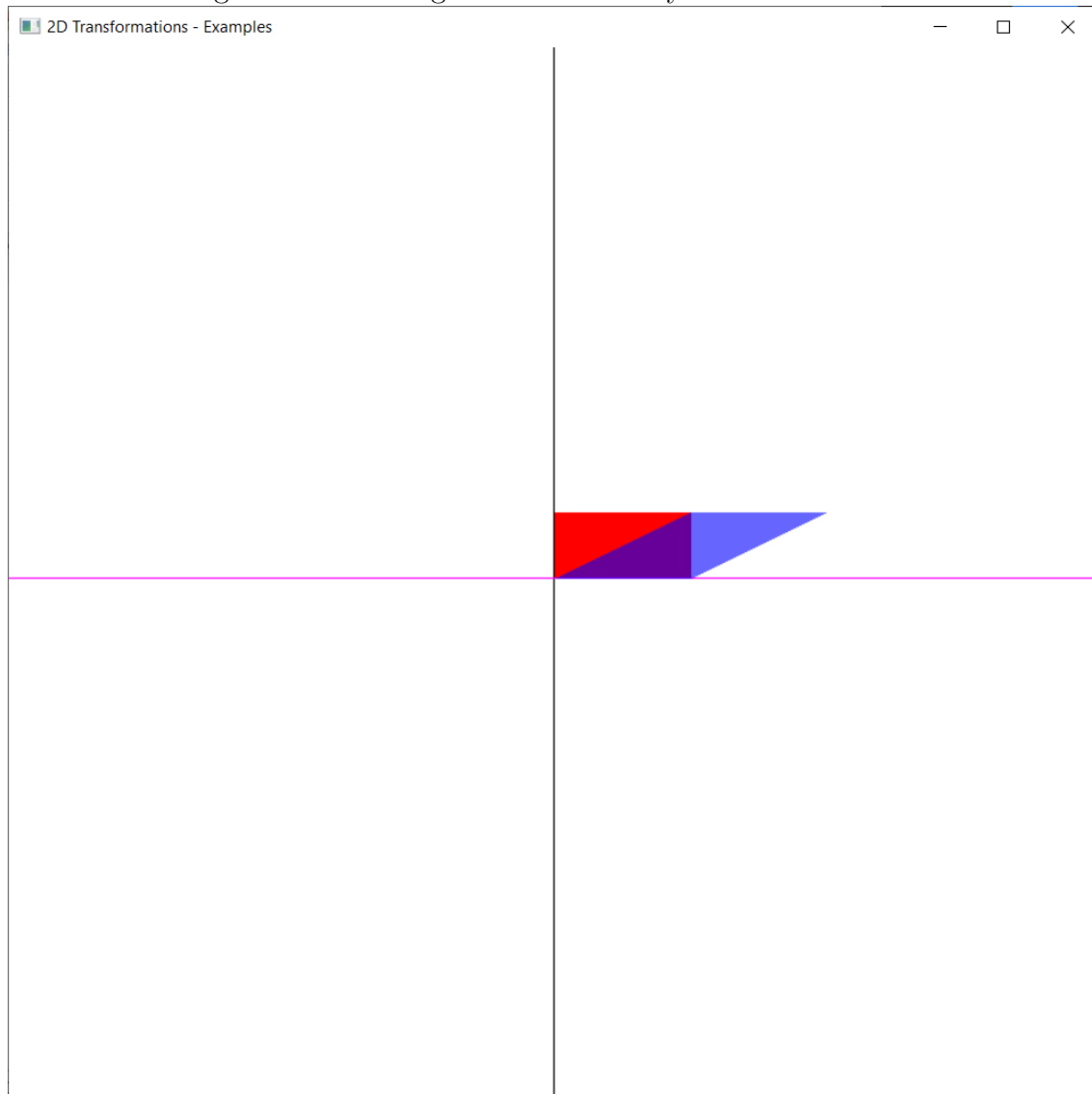
Y Coordinate: 100

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

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

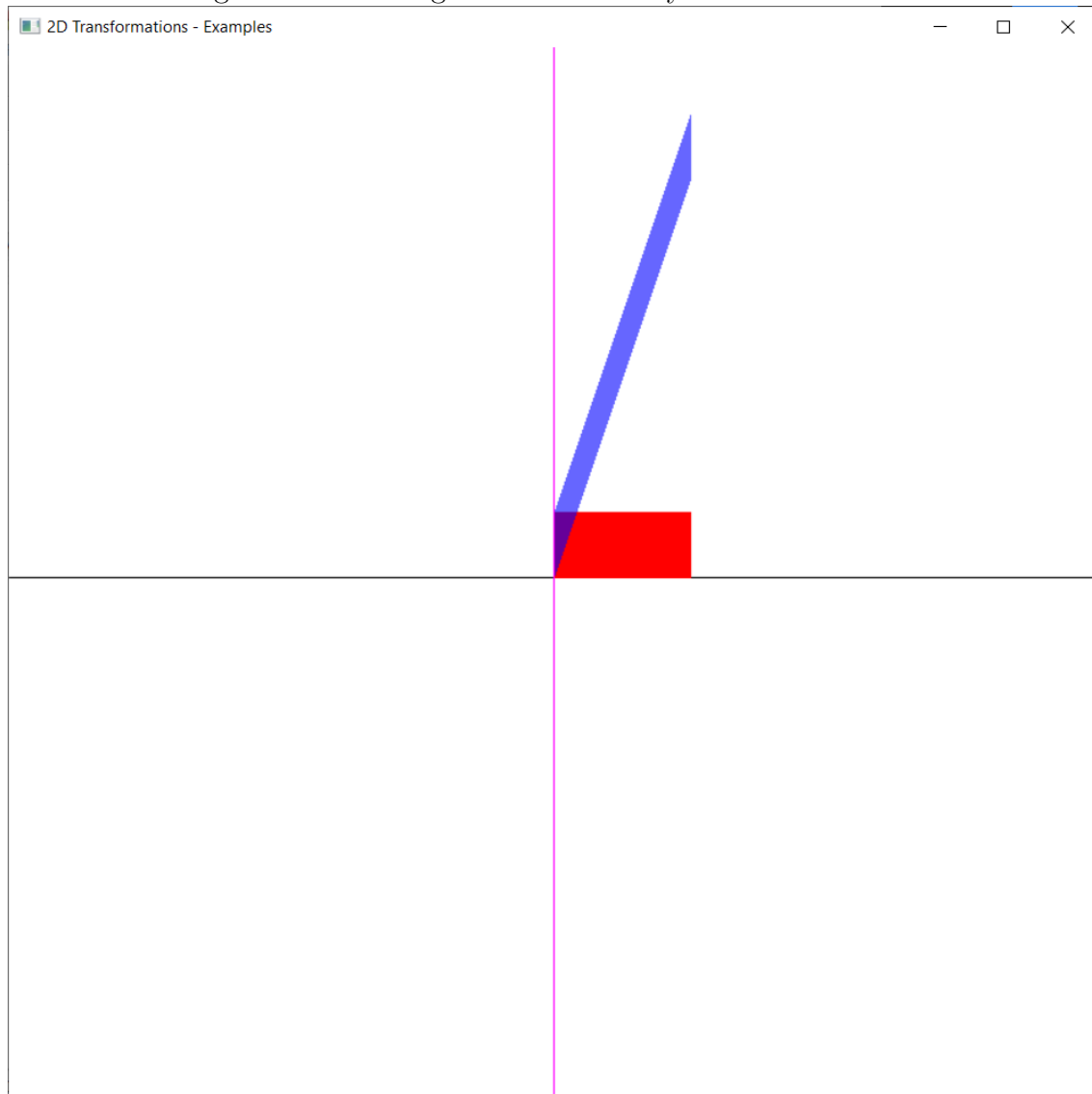
Output: Shearing About X Axis

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



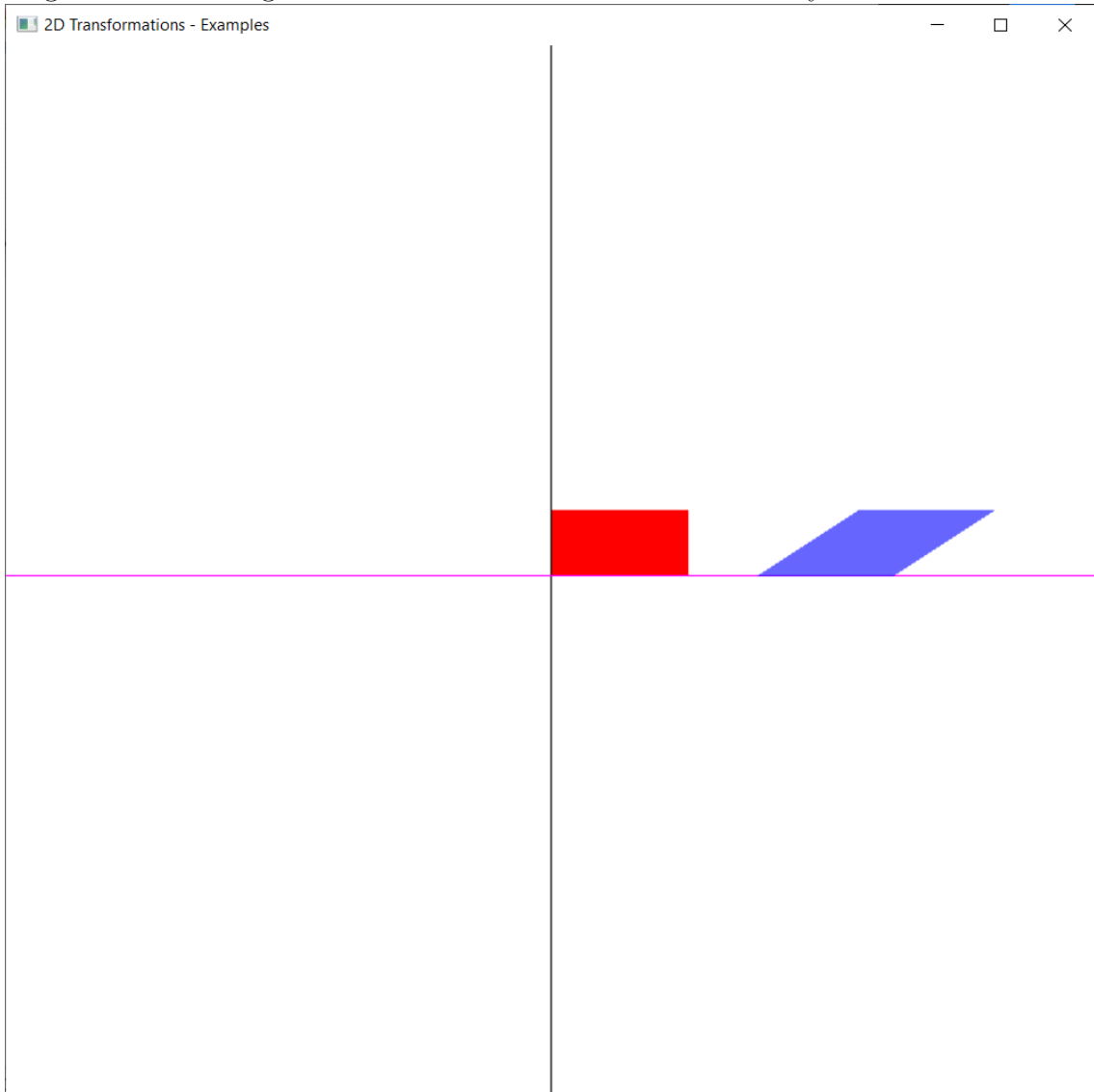
Output: Shearing About Y Axis

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



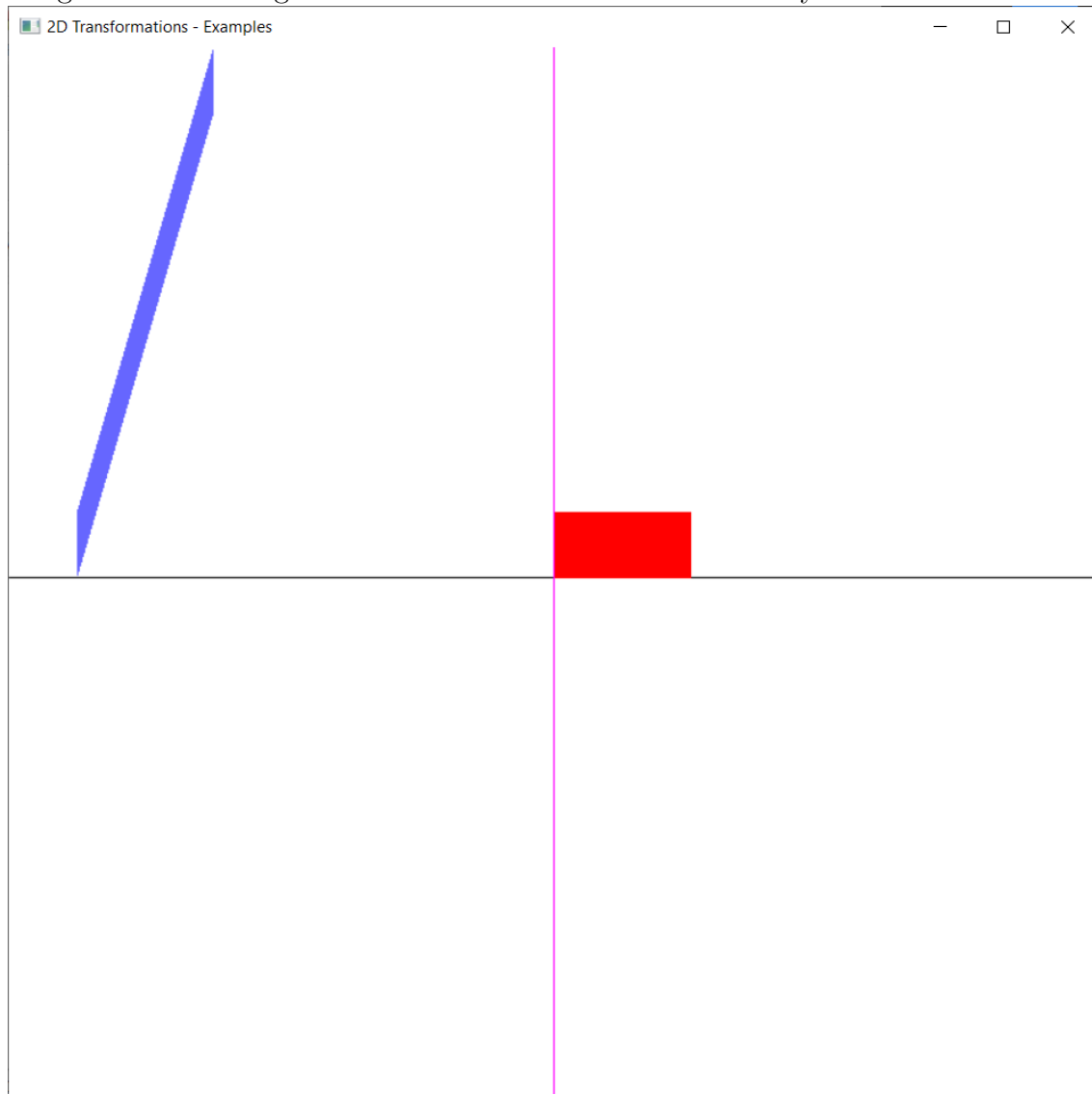
Output: Shearing About X Axis & Ref. Line $Y = y$

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



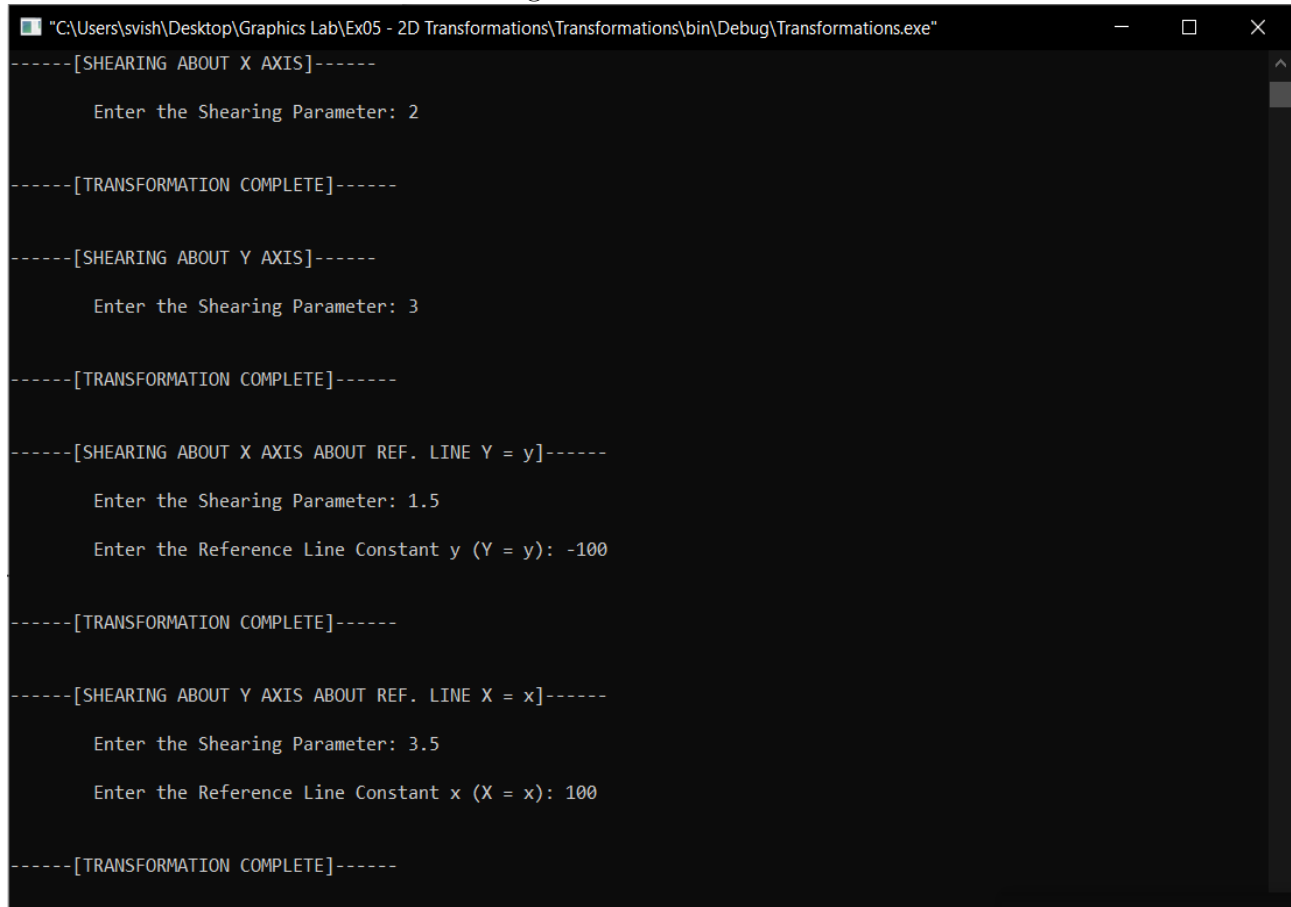
Output: Shearing About Y Axis & Ref. Line $X = x$

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]-----

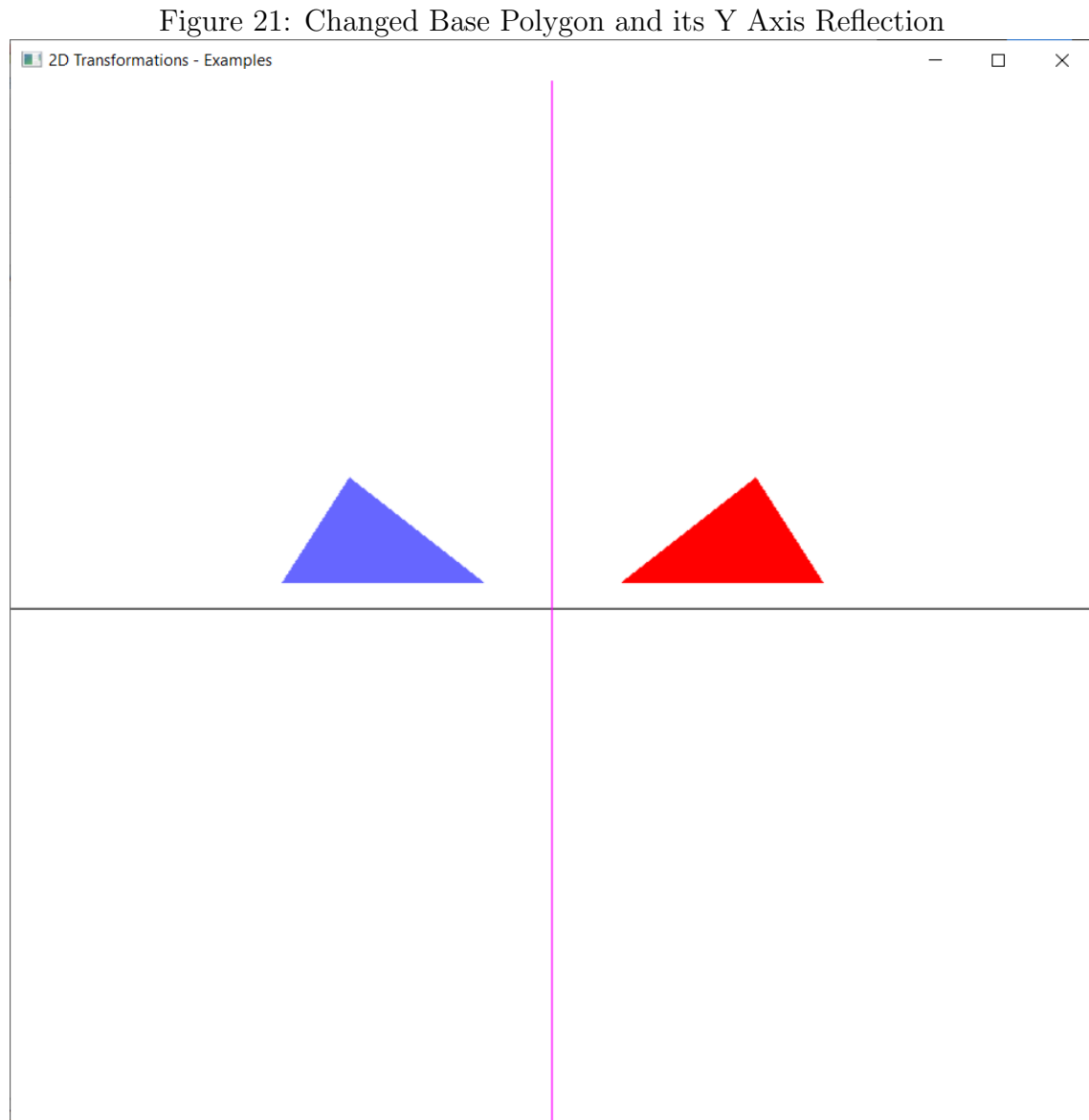
-----[SHEARING ABOUT X AXIS ABOUT REF. LINE Y = y]-----
    Enter the Shearing Parameter: 1.5
    Enter the Reference Line Constant y (Y = y): -100

-----[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

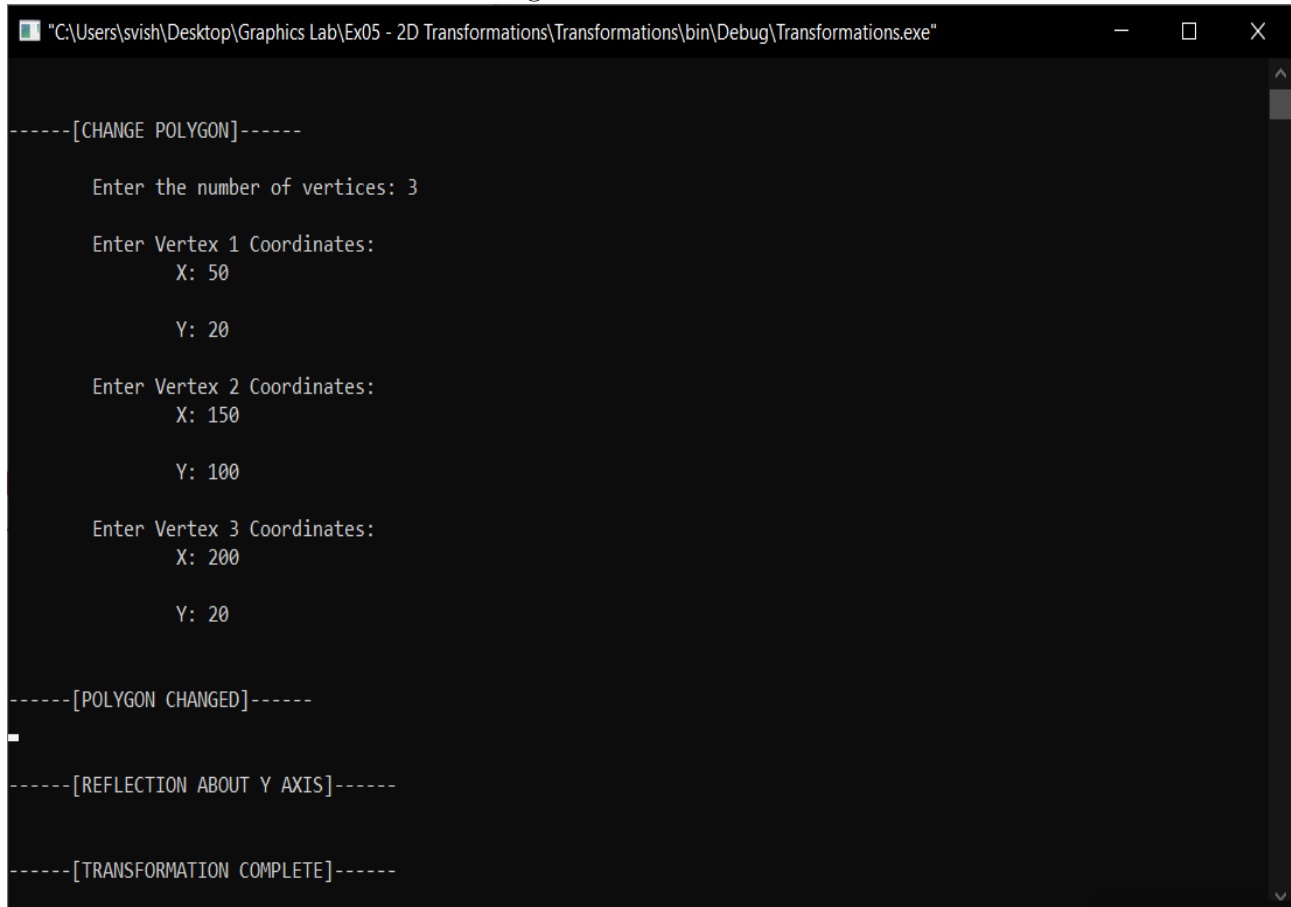
-----[TRANSFORMATION COMPLETE]-----
```

Output: Changed Base Polygon and its Y Axis Reflection



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.