

Department of CSE

SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VII

05 September 2021

UCS 1712 - Graphics And Multimedia Lab

Exercise 6: 2D Composite Transformations and Windowing in C++ using OpenGL

Aim:

- To compute the composite transformation matrix for any 2 transformations given as input by the user and applying it on the object. The transformation can be any combination of the following:
 - Translation
 - Rotation
 - Scaling
 - Reflection
 - Shearing

Display the original and the transformed object. Calculate the final transformation matrix by multiplying the two individual transformation matrices and then apply it to the object.

Note: Use Homogeneous coordinate representations and matrix multiplication to perform transformations. Divide the output window into four quadrants. (Use LINES primitive to draw x and y axis)

- 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.

Code: 2D Composite Transformations:

```
1 /*
2 To compute the composite transformation matrix for any 2 transformations
   given as input by
3 the user and applying it on the object. The transformation can be any
   combination of the following:
4 Translation, Rotation, Scaling, Reflection & Shearing.
5 */
6
7 #include <stdio.h>
8 #include <math.h>
9 #include <GL/glut.h>
10 #include <iostream>           //for cin, cout
11 #include <cstring>           //for memcpy
12
13 using namespace std;
14
15 const int WINDOW_WIDTH = 800;
16 const int WINDOW_HEIGHT = 800;
17 const int X_MIN = -400;
18 const int X_MAX = 400;
19 const int Y_MIN = -400;
20 const int Y_MAX = 400;
21 const int FPS = 60;
22
23 enum Axes {xAxis = 0, yAxis = 1};
24 enum Lines {XAxis = 0, YAxis = 1, Origin = 2, XEqualsY = 3};
25 enum Transforms {Translation = 1, Rotation = 2, Scaling = 3, Reflection =
   4, Shearing = 5};
26
27 class Point{
28 private:
29     GLdouble x, y, h;
30
31 public:
32     Point(){
33         x = y = 0;
34         h = 1;
35     }
36
37     Point(GLint xCoord, GLint yCoord){
38         x = xCoord;
39         y = yCoord;
40         h = 1;
41     }
42
43     Point(GLint xCoord, GLint yCoord, GLint H){
44         x = xCoord;
```

```

45         y = yCoord;
46         h = H;
47     }
48
49     void setCoords(GLdouble xCoord, GLdouble yCoord){
50         x = xCoord;
51         y = yCoord;
52     }
53
54     void setHomogeneousCoords(GLdouble xCoord, GLdouble yCoord, GLdouble H
55 ){
56         x = xCoord;
57         y = yCoord;
58         h = H;
59     }
60
61     GLdouble getX() const{
62         return x;
63     }
64
65     GLdouble getY() const{
66         return y;
67     }
68
69     GLdouble getH() const{
70         return h;
71     }
72
73     GLdouble getHomogenousX() const{
74         return x * h;
75     }
76
77     GLdouble getHomogenousY() const{
78         return y * h;
79     }
80 };
81
82
83 class PolygonShape{
84 private:
85     int numVertices;
86     Point *points;
87     bool matrix1Flag, matrix2Flag;
88     double matrix1[3][3], matrix2[3][3], compositeMatrix[3][3];
89
90 public:
91     PolygonShape(){
92         numVertices = 0;
93         matrix1Flag = false;
94         matrix2Flag = false;

```

```

95     }
96
97     PolygonShape(int noVertices){
98         numVertices = noVertices;
99         points = new Point[numVertices];
100         matrix1Flag = false;
101         matrix2Flag = false;
102     }
103
104     int getVertexCount() const{
105         return numVertices;
106     }
107
108     Point getPoint(int i){
109         return points[i];
110     }
111
112     void setVertices(int noVertices){
113         numVertices = noVertices;
114         points = new Point[numVertices];
115     }
116
117     void setPoint(int i, GLdouble x, GLdouble y, GLdouble h = 1){
118         points[i].setHomogeneousCoords(x, y, h);
119     }
120
121     void clearMatrices(){
122         //Clear the transformation matrices to identity matrices
123
124         matrix1Flag = false;
125         matrix2Flag = false;
126
127         for(int i = 0; i < 3; i++){
128             for(int j = 0; j < 3; j++){
129                 if(i == j){
130                     matrix1[i][j] = 1;
131                     matrix2[i][j] = 1;
132                 } else{
133                     matrix1[i][j] = 0;
134                     matrix2[i][j] = 0;
135                 }
136             }
137         }
138     }
139
140     void setTranslationMatrix(Point translationVector){
141         //Sets a translation matrix to one of the transformation matrices
142
143         double translationMatrix[3][3] = { {1, 0, translationVector.
getHomogenousX()},

```

```

144                                     {0, 1, translationVector.
getHomogenousY()}},
145                                     {0, 0, 1}};
146
147     if(!matrix1Flag){
148         memcpy(matrix1, translationMatrix, sizeof(translationMatrix));
149         matrix1Flag = true;
150     } else{
151         memcpy(matrix2, translationMatrix, sizeof(translationMatrix));
152         matrix2Flag = true;
153     }
154 }
155
156 void setRotationMatrix(int rotationAngle, Point pivot = Point(0, 0, 1)
){
157     //Sets a rotation matrix to one of the transformation matrices
158
159     double rotationAngleInRadians = rotationAngle * 3.14159/180;
160     double cosAngle = cos(rotationAngleInRadians);
161     double sinAngle = sin(rotationAngleInRadians);
162
163     double xPivotValue = (pivot.getX() * (1 - cosAngle)) + (pivot.getY
() * sinAngle);
164     double yPivotValue = (pivot.getY() * (1 - cosAngle)) - (pivot.getX
() * sinAngle);
165
166     double rotationMatrix[3][3] = { {cosAngle, -sinAngle, xPivotValue
},
167                                     {sinAngle, cosAngle, yPivotValue},
168                                     {0, 0, 1}};
169
170     if(!matrix1Flag){
171         memcpy(matrix1, rotationMatrix, sizeof(rotationMatrix));
172         matrix1Flag = true;
173     } else{
174         memcpy(matrix2, rotationMatrix, sizeof(rotationMatrix));
175         matrix2Flag = true;
176     }
177 }
178
179 void setReflectionMatrix(int line){
180     //Sets a reflection matrix to one of the transformation matrices
181
182     double reflectionMatrix[3][3];
183
184     switch(line){
185         case XAxis:{
186             double temp[3][3] = { {1, 0, 0},
187                                   {0, -1, 0},
188                                   {0, 0, 1}};
189

```

```

190         memcpy(reflectionMatrix, temp, sizeof(temp));
191         break;
192     }
193
194
195     case YAxis:{
196         double temp[3][3] = {    {-1, 0, 0},
197                                 {0, 1, 0},
198                                 {0, 0, 1}};
199
200         memcpy(reflectionMatrix, temp, sizeof(temp));
201         break;
202     }
203
204
205     case Origin:{
206         double temp[3][3] = {    {-1, 0, 0},
207                                 {0, -1, 0},
208                                 {0, 0, 1}};
209
210         memcpy(reflectionMatrix, temp, sizeof(temp));
211         break;
212     }
213
214
215     case XEqualsY:{
216         double temp[3][3] = {    {0, 1, 0},
217                                 {1, 0, 0},
218                                 {0, 0, 1}};
219
220         memcpy(reflectionMatrix, temp, sizeof(temp));
221         break;
222     }
223 }
224
225 if(!matrix1Flag){
226     memcpy(matrix1, reflectionMatrix, sizeof(reflectionMatrix));
227     matrix1Flag = true;
228 } else{
229     memcpy(matrix2, reflectionMatrix, sizeof(reflectionMatrix));
230     matrix2Flag = true;
231 }
232 }
233
234 void setScaleMatrix(double ScaleX, double ScaleY, Point fixed = Point
(0, 0, 1)){
235     //Sets a scale matrix to one of the transformation matrices
236
237     double xFixedValue = fixed.getX() * (1 - ScaleX);
238     double yFixedValue = fixed.getY() * (1 - ScaleY);
239

```

```

240     double scaleMatrix[3][3] = { {ScaleX, 0, xFixedValue},
241                                   {0, ScaleY, yFixedValue},
242                                   {0, 0, 1}};
243
244     if(!matrix1Flag){
245         memcpy(matrix1, scaleMatrix, sizeof(scaleMatrix));
246         matrix1Flag = true;
247     } else{
248         memcpy(matrix2, scaleMatrix, sizeof(scaleMatrix));
249         matrix2Flag = true;
250     }
251 }
252
253 void setShearMatrix(double shearParam, int axis, double refConst = 0){
254     //Sets a shear matrix to one of the transformation matrices
255
256     double shearMatrix[3][3];
257
258     switch(axis){
259         case xAxis:{
260             double temp[3][3] = { {1, shearParam, -shearParam *
refConst},
261                                   {0, 1, 0},
262                                   {0, 0, 1}};
263
264             memcpy(shearMatrix, temp, sizeof(temp));
265             break;
266         }
267
268         case yAxis:{
269             double temp[3][3] = { {1, 0, -shearParam * refConst},
270                                   {shearParam, 1, 0},
271                                   {0, 0, 1}};
272
273             memcpy(shearMatrix, temp, sizeof(temp));
274             break;
275         }
276     }
277
278     if(!matrix1Flag){
279         memcpy(matrix1, shearMatrix, sizeof(shearMatrix));
280         matrix1Flag = true;
281     } else{
282         memcpy(matrix2, shearMatrix, sizeof(shearMatrix));
283         matrix2Flag = true;
284     }
285 }
286
287 void setCompositeMatrix(){
288     //Sets the composite matrix based on matrix multiplication
289

```

```

290     //of the two transformation matrices
291
292     if(!matrix1Flag || !matrix2Flag){
293         //if any one matrix is not set, don't multiply
294         return;
295     }
296
297     for(int i = 0; i < 3; i++){
298         for(int j = 0; j < 3; j++){
299             double tempSum = 0;
300
301             for(int k = 0; k < 3; k++){
302                 tempSum += matrix1[i][k] * matrix2[k][j];
303             }
304
305             compositeMatrix[i][j] = tempSum;
306         }
307     }
308 }
309
310 PolygonShape getTransformedPolygon(){
311     //Obtain the transformed polygon based upon the composite
312     transformation
313
314     PolygonShape polyDash(numVertices);
315     double values[3];
316
317     for(int i = 0; i < numVertices; i++){
318         Point p = getPoint(i);
319
320         //[[3 x 3] x [3 x 1] = [3 x 1] matrix
321         for(int j = 0; j < 3; j++){
322             values[j] = compositeMatrix[j][0] * p.getHomogenousX() +
323                 compositeMatrix[j][1] * p.getHomogenousY() +
324                 compositeMatrix[j][2] * p.getH();
325         }
326         polyDash.setPoint(i, values[0]/p.getH(), values[1]/p.getH(),
327         values[2]);
328     }
329     return polyDash;
330 }
331 };
332
333
334 void initializeDisplay();
335 void plotComponents();
336 void dummyFunction();
337 void renderContents();
338 void mainLoop(int val);

```



```

339 void setTransformMatrices();
340 void plotTransformation();
341 void drawAxes();
342 void drawPolygon(PolygonShape polygon, bool transformed = false);
343
344 PolygonShape polygon;                                //Global PolygonShape object to be
    plotted on the graph
345 int transform1 = 0, transform2 = 0;                    //Global variable to keep track of
    chosen transformation
346
347 int main(int argc, char **argv){
348     glutInit(&argc, argv);
349     glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
350     glutInitWindowPosition(0, 0);
351     glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
352     glutCreateWindow("2D Composite Transformations - Examples");
353
354     printf("\n-----[2D COMPOSITE TRANSFORMATIONS]-----\n");
355     printf("\nUsage:\tSelect the required transformations in the console."
356 );
357     printf("\n\tEnter the parameters for the specified transformations.");
358     printf("\n\tView the output in the GLUT window.");
359     printf("\n\n-----[2D COMPOSITE TRANSFORMATIONS]-----\n\n");
360
361     //Set the initial default polygon for the graph
362     polygon.setVertices(4);
363     polygon.setPoint(0, 0, 0);
364     polygon.setPoint(1, 0, 50);
365     polygon.setPoint(2, 100, 50);
366     polygon.setPoint(3, 100, 0);
367
368     initializeDisplay();
369
370     glutDisplayFunc(dummyFunction);
371
372     //important - to refresh screen periodically
373     glutTimerFunc(1000/FPS, mainLoop, 0);
374
375     glutMainLoop();
376
377     return 1;
378 }
379
380 void initializeDisplay(){
381     //Initialize the display parameters
382
383     glClearColor(1, 1, 1, 0);
384     glMatrixMode(GL_PROJECTION);
385     gluOrtho2D(X_MIN, X_MAX, Y_MIN, Y_MAX);
386     glClear(GL_COLOR_BUFFER_BIT);    //Clear the display window

```

```

387     glEnable(GL_BLEND);           //enable blending (translucent colors)
388     glDepthMask(GL_FALSE);
389     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA); //set the blend
function for translucency
390 }
391
392 void plotComponents(){
393     //Plot the axes and the base polygon
394
395     glClear(GL_COLOR_BUFFER_BIT); //Clear the display window
396     drawAxes();
397     drawPolygon(polygon);
398     glFlush();
399 }
400
401 void dummyFunction(){
402     //Placeholder function to be called in glutDisplayFunc
403 }
404
405 void mainLoop(int val){
406     //Function to be called within the glutTimerFunc periodically to
refresh screen at 60FPS
407     renderContents();
408 }
409
410
411 void renderContents(){
412     //to render the graph along with a user-defined composite
transformation
413
414     plotComponents();
415
416     while(true){
417         //Await user input
418
419         cout << "\nChoose Transformation 1: " << endl;
420         cout << "\t1 for Translation" << endl;
421         cout << "\t2 for Rotation" << endl;
422         cout << "\t3 for Scaling" << endl;
423         cout << "\t4 for Reflection" << endl;
424         cout << "\t5 for Shearing" << endl;
425         cout << "\t0 to Exit" << endl;
426         cout << "\tYour Option -> ";
427         cin >> transform1;
428
429         if(!transform1){ //user chooses to exit
430             exit(0);
431         }
432
433         cout << "\nChoose Transformation 2: " << endl;
434         cout << "\t1 for Translation" << endl;

```

```

435     cout << "\t2 for Rotation" << endl;
436     cout << "\t3 for Scaling" << endl;
437     cout << "\t4 for Reflection" << endl;
438     cout << "\t5 for Shearing" << endl;
439     cout << "\tYour Option -> ";
440     cin >> transform2;
441
442     plotComponents();
443     polygon.clearMatrices();           //clear previous transformations
444     setTransformMatrices();           //set the new transform matrices
445     polygon.setCompositeMatrix();     //multiply to form composite
matrix
446
447     PolygonShape polygonDash;
448     polygonDash = polygon.getTransformedPolygon();
449
450     //To print the transformed polygon's coordinates
451     // for(int i = 0; i < polygonDash.getVertexCount(); i++){
452     //     Point pDash = polygonDash.getPoint(i);
453     //     cout << "(" << pDash.getX() << ", " << pDash.getY() << ")"
<< endl;
454     // }
455
456     drawPolygon(polygonDash, true);
457     glFlush();
458     //glutPostRedisplay();
459 }
460 }
461
462 void setTransformMatrices(){
463     int i = 0;
464
465     while(i < 2){
466         int currentTransform = (i == 0) ? transform1 : transform2;
467         i++;
468
469         switch(currentTransform){
470             case Translation:{
471                 double x, y;
472                 cout << "\n\n-----[TRANSLATION]-----" << endl;
473                 cout << "\n\tEnter the Translation Vector Magnitude: ";
474                 cout << "\n\t\tX Component: "; cin >> x;
475                 cout << "\n\t\tY Component: "; cin >> y;
476
477                 polygon.setTranslationMatrix(Point(x, y, 1));
478                 cout << "\n\n-----[TRANSLATION NOTED]-----" << endl;
479                 break;
480             }
481
482             case Rotation:{
483                 double rotationAngle, x = 0, y = 0;

```

```

484     int pivot = 0;
485     cout << "\n\n-----[ROTATION]-----" << endl;
486     cout << "\n\tEnter the Rotation Angle: ";
487     cin >> rotationAngle;
488
489     cout << "\n\tEnter 1 for Rotating about Pivot, else enter
0.";
490
491     cout << "\n\t\tYour Choice -> "; cin >> pivot;
492
493     if(pivot){
494         cout << "\n\tEnter Pivot Point: ";
495         cout << "\n\t\tEnter X Coordinate: "; cin >> x;
496         cout << "\n\t\tEnter Y Coordinate: "; cin >> y;
497     }
498     polygon.setRotationMatrix(rotationAngle, Point(x, y, 1));
499     cout << "\n\n-----[ROTATION NOTED]-----" << endl;
500     break;
501 }
502
503 case Scaling:{
504     double xScale, yScale, xFixed = 0, yFixed = 0;
505     int uniform = 0, fixed = 0;
506     cout << "\n\n-----[SCALING]-----" << endl;
507     cout << "\n\tEnter an option:";
508     cout << "\n\t\t0 for Uniform Scaling";
509     cout << "\n\t\t1 for Differential Scaling";
510     cout << "\n\t\tYour Choice -> "; cin >> uniform;
511
512     if(uniform){
513         cout << "\n\tEnter the Scaling Factors: ";
514         cout << "\n\t\tX Scale Factor: "; cin >> xScale;
515         cout << "\n\t\tY Scale Factor: "; cin >> yScale;
516     } else{
517         cout << "\n\tEnter the Scaling Factor: "; cin >>
xScale;
518         yScale = xScale;
519     }
520
521     cout << "\n\tEnter 1 for Scaling about Fixed Point, else
enter 0.";
522     cout << "\n\t\tYour Choice -> "; cin >> fixed;
523
524     if(fixed){
525         cout << "\n\tEnter Fixed Point: ";
526         cout << "\n\t\tEnter X Coordinate: "; cin >> xFixed;
527         cout << "\n\t\tEnter Y Coordinate: "; cin >> yFixed;
528     }
529     polygon.setScaleMatrix(xScale, yScale, Point(xFixed,
yFixed, 1));
530

```

```

531         cout << "\n\n-----[SCALING NOTED]-----" << endl;
532         break;
533     }
534
535     case Reflection:{
536         int reflectionOption = 4;
537         cout << "\n\n-----[REFLECTION]-----" << endl;
538
539         while(reflectionOption < 0 || reflectionOption > 3){
540             cout << "\n\tEnter an option:";
541             cout << "\n\t\t0 for Reflection About X Axis";
542             cout << "\n\t\t1 for Reflection About Y Axis.";
543             cout << "\n\t\t2 for Reflection About Origin.";
544             cout << "\n\t\t3 for Reflection About Line X = Y.";
545             cout << "\n\t\tYour Choice -> "; cin >>
reflectionOption;
546         }
547
548         polygon.setReflectionMatrix(reflectionOption);
549
550         cout << "\n\n-----[REFLECTION NOTED]-----" << endl;
551         break;
552     }
553
554     case Shearing:{
555         double shearParam, refConst = 0;
556         int axis = 0, refLine = 0;
557         cout << "\n\n-----[SHEARING]-----" << endl;
558
559         cout << "\n\tEnter an option:";
560         cout << "\n\t\t0 for Shearing About X Axis";
561         cout << "\n\t\t1 for Shearing About Y Axis";
562         cout << "\n\t\tYour Choice -> "; cin >> axis;
563
564         cout << "\n\tEnter the Shearing Parameter: "; cin >>
shearParam;
565
566         cout << "\n\tEnter 1 for Shearing About Reference Line,
else enter 0.";
567         cout << "\n\t\tYour Choice -> "; cin >> refLine;
568
569         if(refLine){
570             if(!axis){
571                 cout << "\n\tEnter c for Ref. Line Y = c: ";
572                 cin >> refConst;
573             } else{
574                 cout << "\n\tEnter c for Ref. Line X = c: ";
575                 cin >> refConst;
576             }
577         }
578     }

```

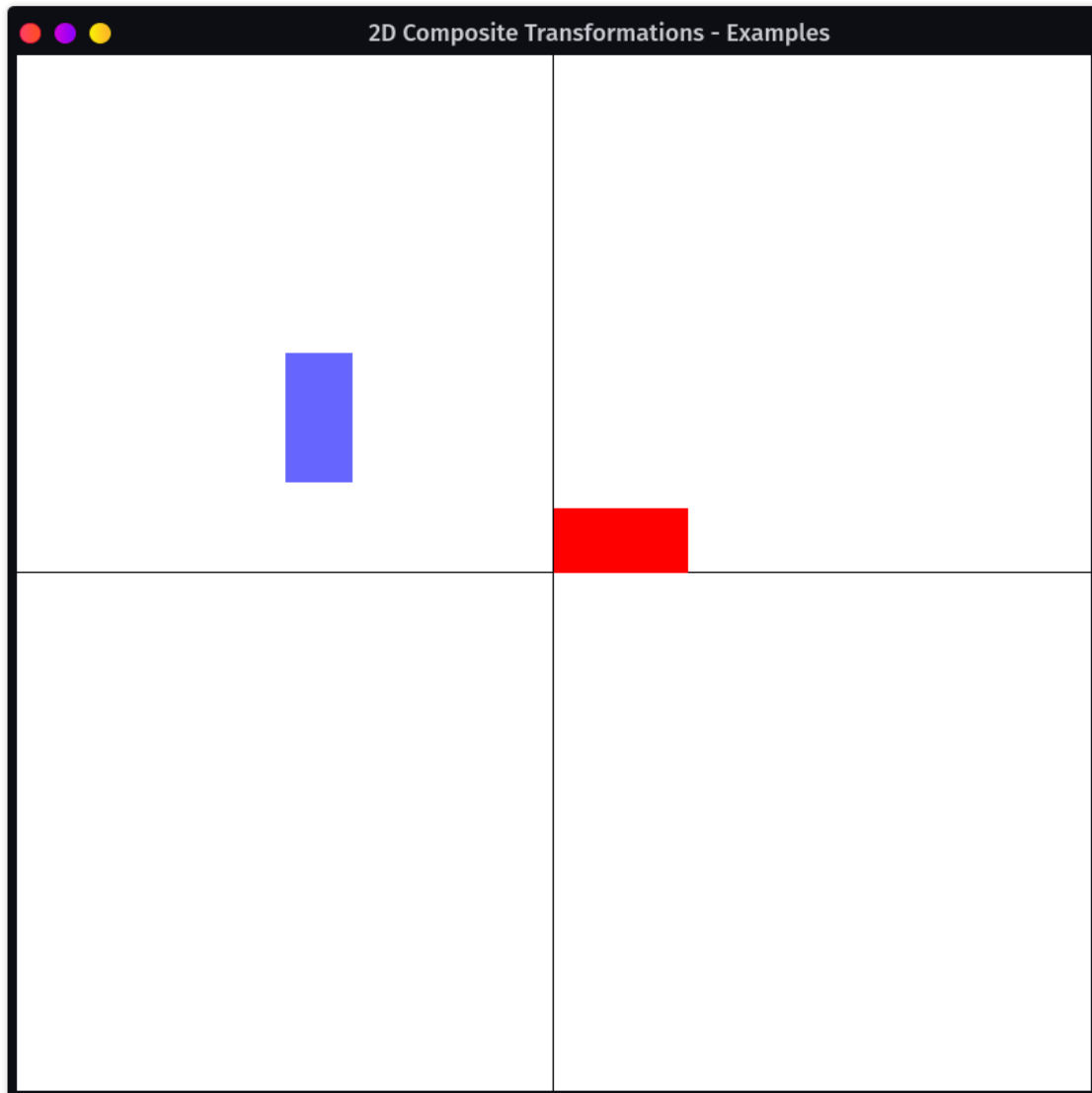
```

579         polygon.setShearMatrix(shearParam, axis, refConst);
580
581         cout << "\n\n-----[SHEARING NOTED]-----" << endl;
582         break;
583     }
584 }
585
586     if(i == 1){
587         cout << "\n\n-----[TRANSFORMATION 1 NOTED]-----" << endl;
588     } else{
589         cout << "\n\n-----[TRANSFORMATION 2 NOTED]-----" << endl;
590     }
591 }
592 }
593
594 void drawAxes(){
595     //To draw the X and Y axes
596
597     glColor3d(0, 0, 0); //Black color
598     glBegin(GL_LINES);
599
600     //X-axis
601     glVertex2f(X_MIN, 0);
602     glVertex2f(X_MAX, 0);
603
604     //Y-axis
605     glVertex2f(0, Y_MIN);
606     glVertex2f(0, Y_MAX);
607
608     glEnd();
609 }
610
611 void drawPolygon(PolygonShape polygon, bool transformed){
612     //To draw a given polygon
613
614     if(!transformed){
615         glColor3d(1, 0, 0); //Red color
616     } else{
617         glColor4f(0, 0, 1, 0.6); //Blue Color
618     }
619
620     glBegin(GL_POLYGON);
621
622     for(int i = 0; i < polygon.getVertexCount(); i++){
623         Point p = polygon.getPoint(i);
624         glVertex2f(p.getX(), p.getY());
625     }
626
627     glEnd();
628 }

```

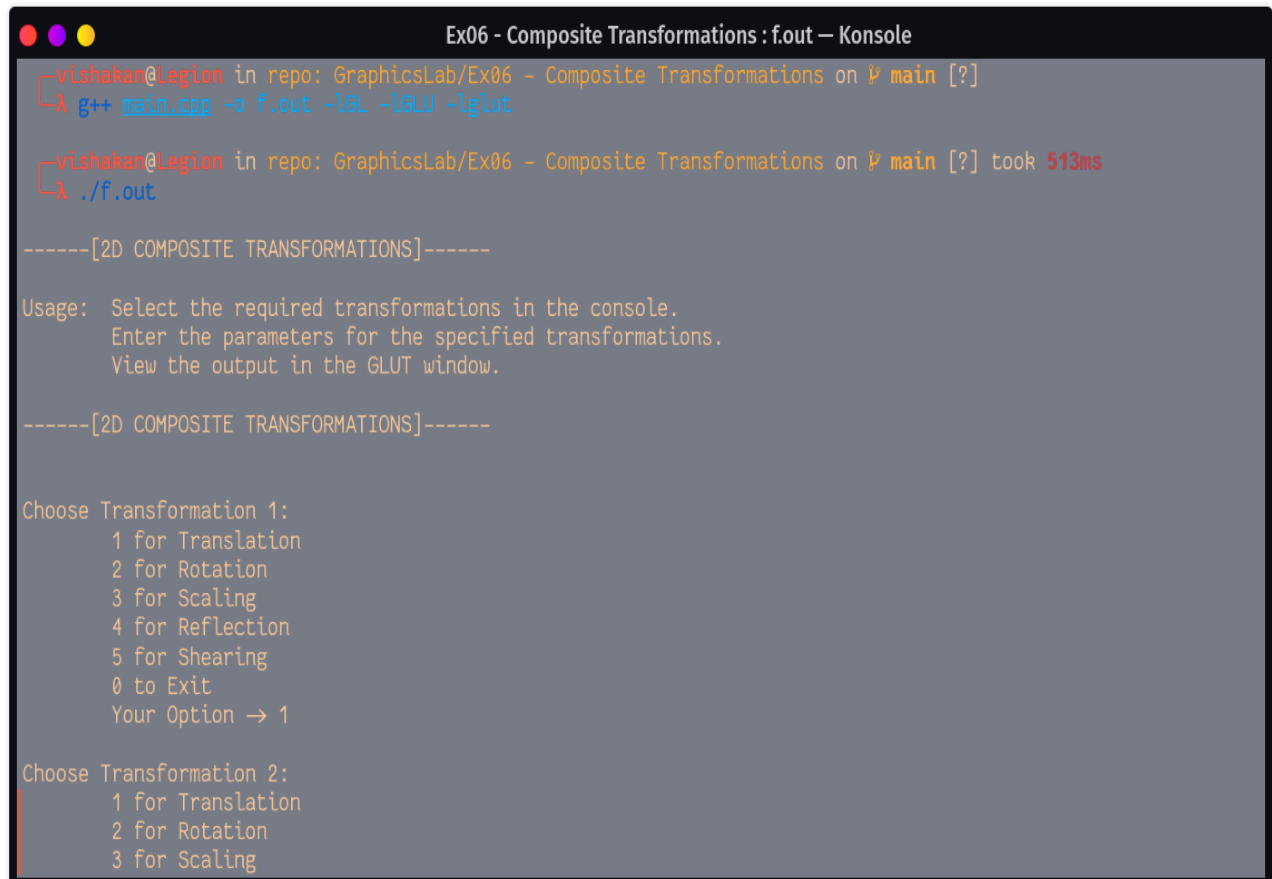
Output: Plot With Translation and Rotation

Figure 1: Plot With Translation and Rotation.



Output: Console

Figure 2: Output: Console.



```
Ex06 - Composite Transformations : f.out - Konsole
vishakan@Legion in repo: GraphicsLab/Ex06 - Composite Transformations on P main [?]
λ g++ main.cpp -o f.out -lGL -lGLU -lglut
vishakan@Legion in repo: GraphicsLab/Ex06 - Composite Transformations on P main [?] took 513ms
λ ./f.out

-----[2D COMPOSITE TRANSFORMATIONS]-----

Usage:  Select the required transformations in the console.
        Enter the parameters for the specified transformations.
        View the output in the GLUT window.

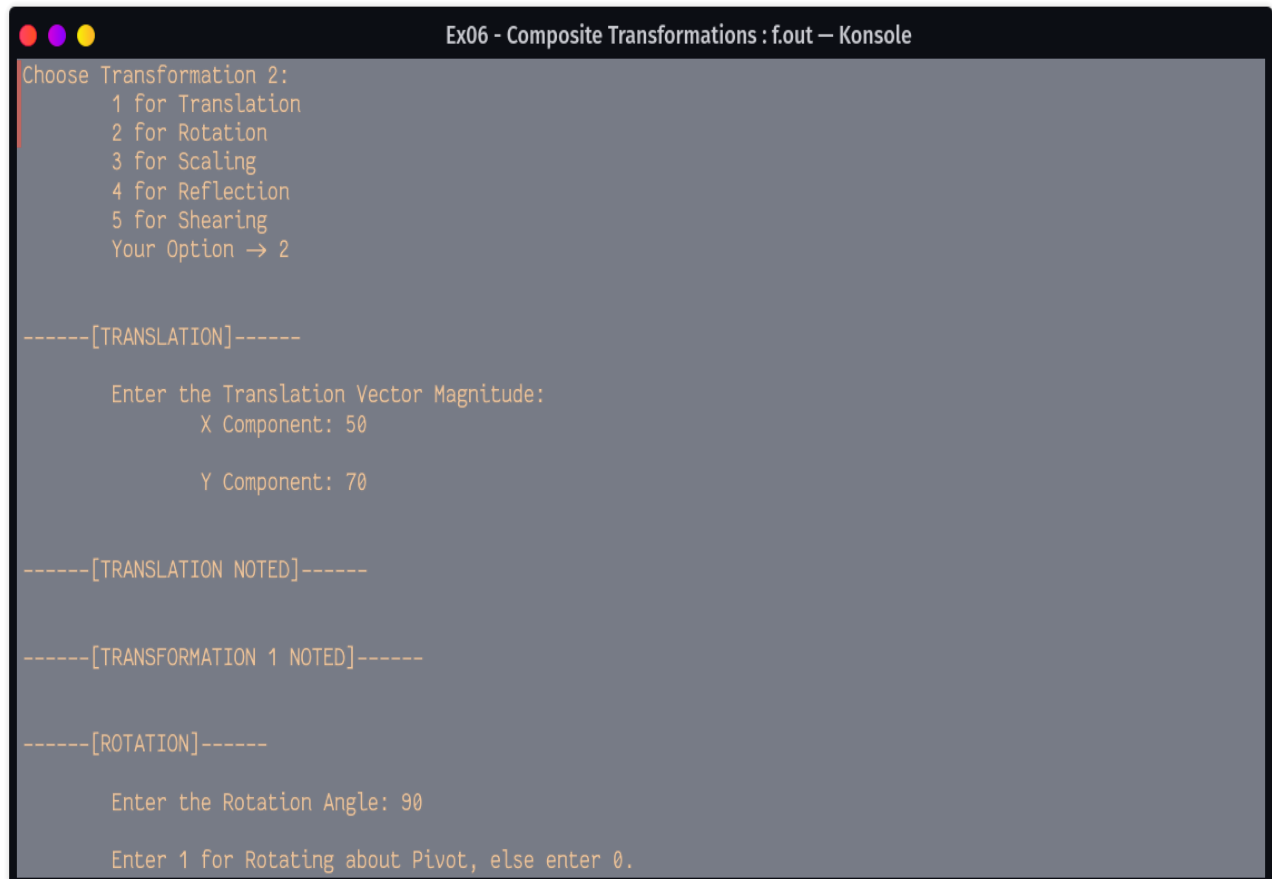
-----[2D COMPOSITE TRANSFORMATIONS]-----

Choose Transformation 1:
    1 for Translation
    2 for Rotation
    3 for Scaling
    4 for Reflection
    5 for Shearing
    0 to Exit
    Your Option → 1

Choose Transformation 2:
    1 for Translation
    2 for Rotation
    3 for Scaling
```


Output: Console

Figure 3: Output: Console.



```
Choose Transformation 2:
  1 for Translation
  2 for Rotation
  3 for Scaling
  4 for Reflection
  5 for Shearing
Your Option → 2

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

    Enter the Translation Vector Magnitude:
      X Component: 50
      Y Component: 70

-----[TRANSLATION NOTED]-----

-----[TRANSFORMATION 1 NOTED]-----

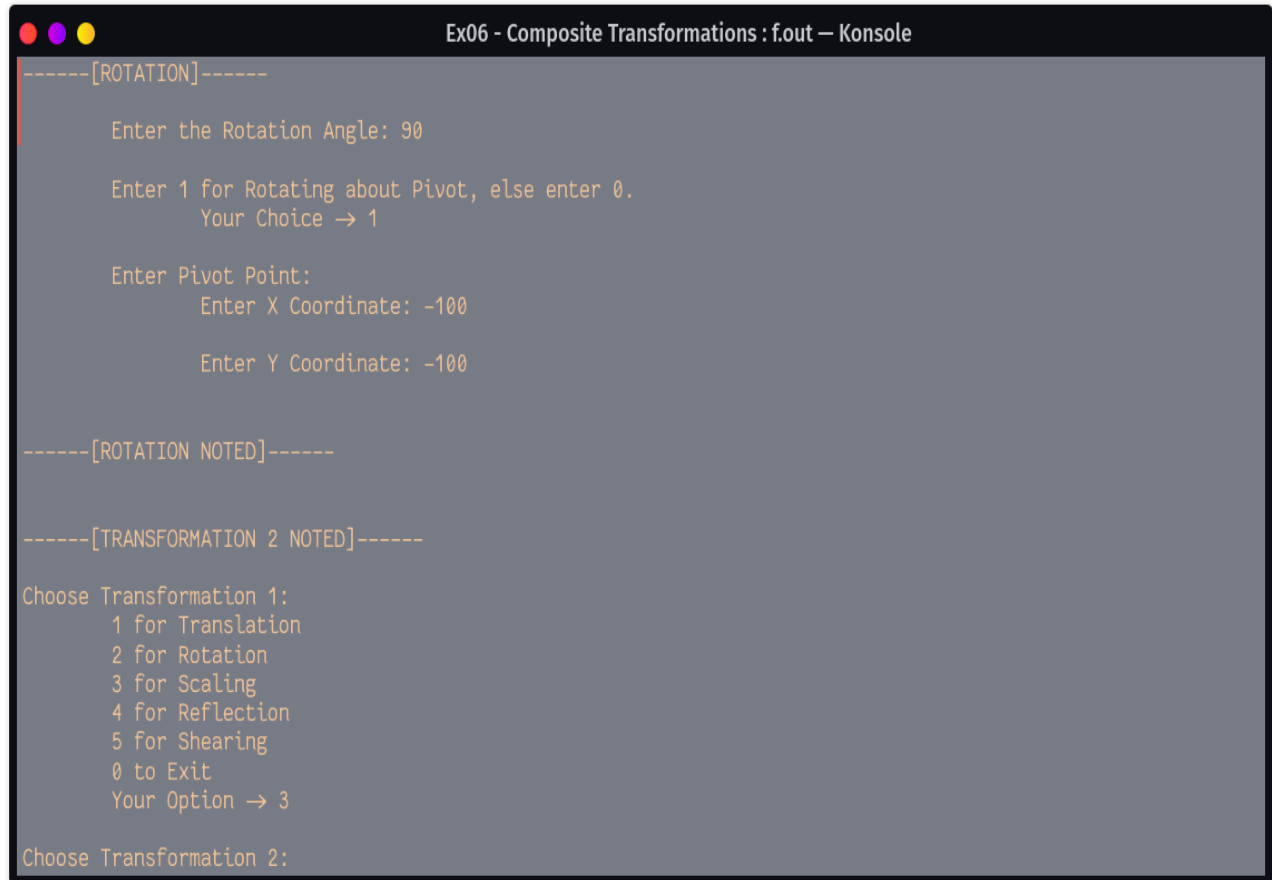
-----[ROTATION]-----

    Enter the Rotation Angle: 90

    Enter 1 for Rotating about Pivot, else enter 0.
```

Output: Console

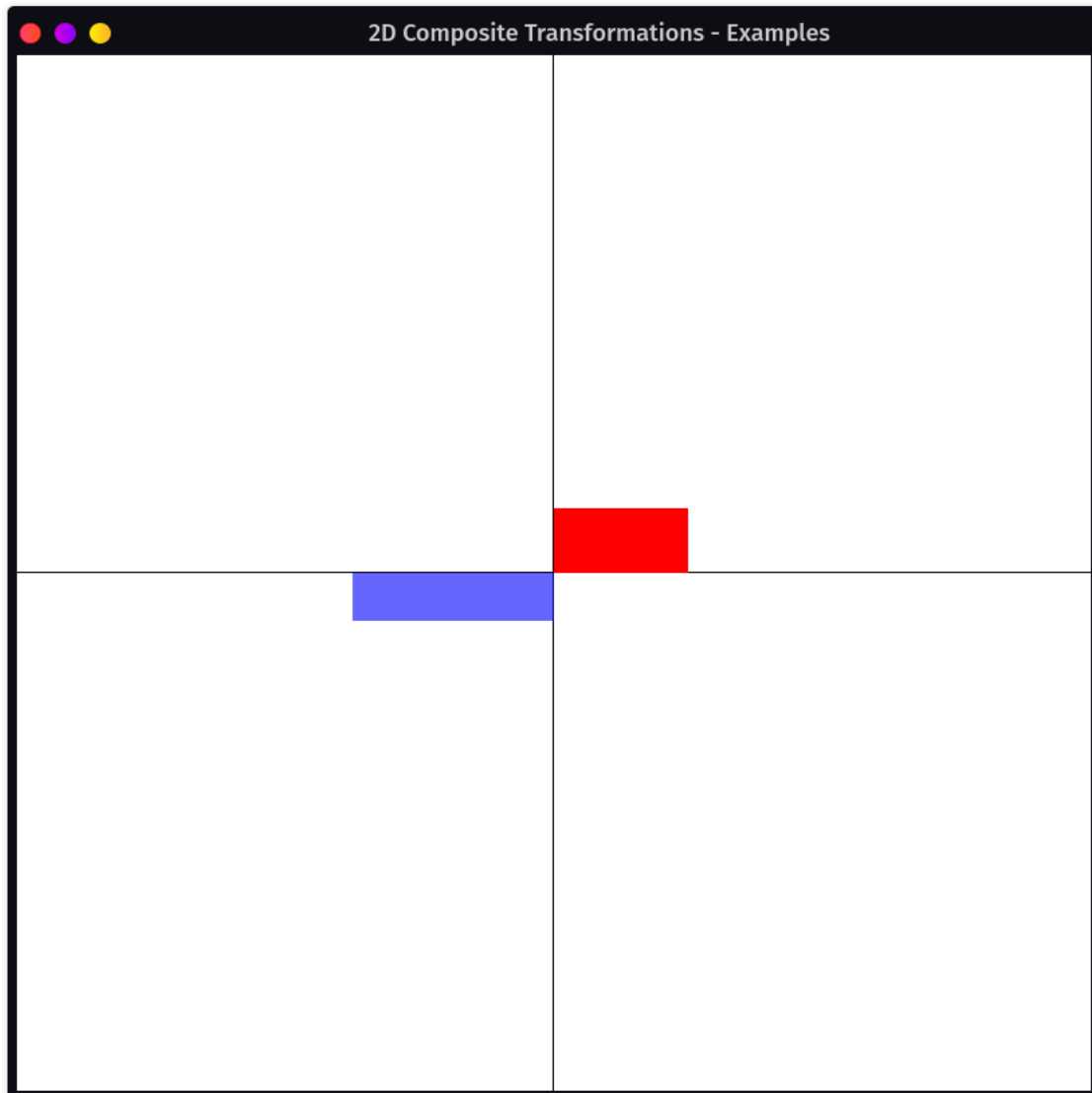
Figure 4: Output: Console.



```
-----[ROTATION]-----  
  
Enter the Rotation Angle: 90  
  
Enter 1 for Rotating about Pivot, else enter 0.  
Your Choice → 1  
  
Enter Pivot Point:  
Enter X Coordinate: -100  
  
Enter Y Coordinate: -100  
  
-----[ROTATION NOTED]-----  
  
-----[TRANSFORMATION 2 NOTED]-----  
  
Choose Transformation 1:  
1 for Translation  
2 for Rotation  
3 for Scaling  
4 for Reflection  
5 for Shearing  
0 to Exit  
Your Option → 3  
  
Choose Transformation 2:
```

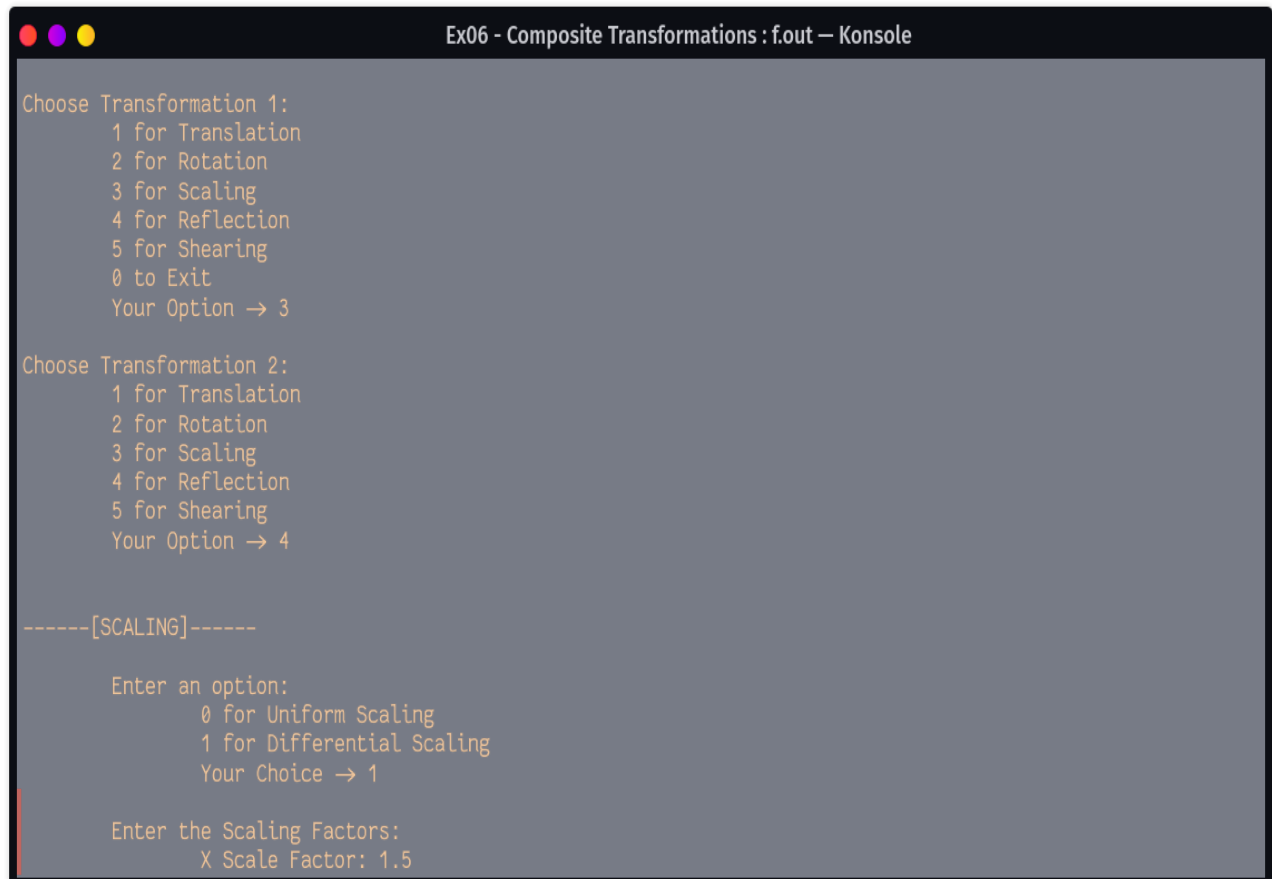
Output: Plot With Scaling and Reflection

Figure 5: Plot With Scaling and Reflection.



Output: Console

Figure 6: Console.



```
Choose Transformation 1:
  1 for Translation
  2 for Rotation
  3 for Scaling
  4 for Reflection
  5 for Shearing
  0 to Exit
Your Option → 3

Choose Transformation 2:
  1 for Translation
  2 for Rotation
  3 for Scaling
  4 for Reflection
  5 for Shearing
Your Option → 4

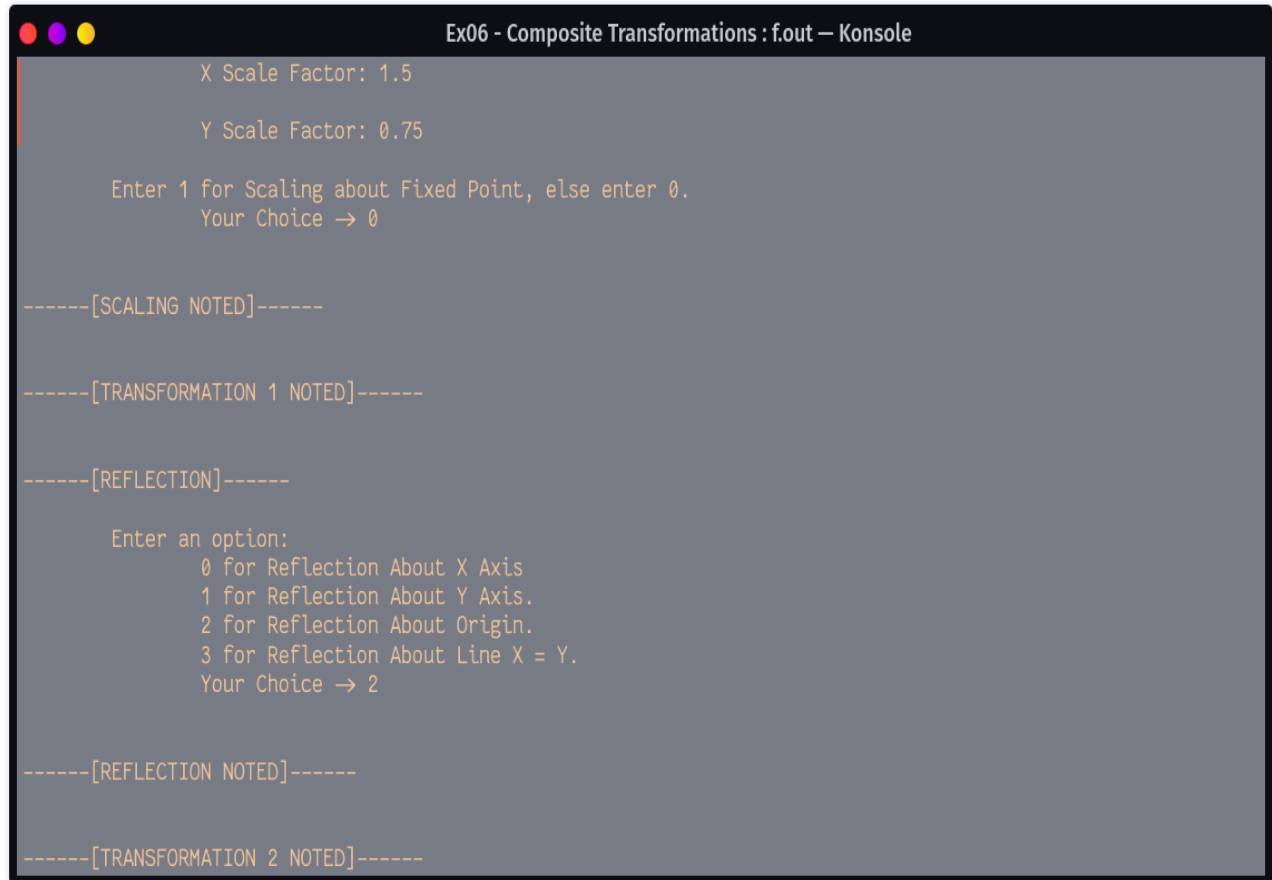
-----[SCALING]-----

Enter an option:
  0 for Uniform Scaling
  1 for Differential Scaling
Your Choice → 1

Enter the Scaling Factors:
  X Scale Factor: 1.5
```

Output: Console

Figure 7: Console.



```
Ex06 - Composite Transformations : f.out — Konsole

X Scale Factor: 1.5
Y Scale Factor: 0.75

Enter 1 for Scaling about Fixed Point, else enter 0.
Your Choice → 0

-----[SCALING NOTED]-----

-----[TRANSFORMATION 1 NOTED]-----

-----[REFLECTION]-----

Enter an option:
  0 for Reflection About X Axis
  1 for Reflection About Y Axis.
  2 for Reflection About Origin.
  3 for Reflection About Line X = Y.
Your Choice → 2

-----[REFLECTION NOTED]-----

-----[TRANSFORMATION 2 NOTED]-----
```

Code: Window To Viewport Transformation:

```
1 /*
2 Create a window with any 2D object and a different sized viewport. Apply
   window to viewport
3 transformation on the object. Display both window and viewport.
4 */
5
6 #include <stdio.h>
7 #include <math.h>
8 #include <GL/glut.h>
9 #include <iostream>           //for cin, cout
10 #include <cstring>           //for memcpy
11
12 using namespace std;
13
14 const int HEIGHT = 870;
15 const int WIDTH = 1010;
16 const int WINDOW_XMIN = -500;
17 const int WINDOW_XMAX = 0;
18 const int WINDOW_YMIN = -400;
19 const int WINDOW_YMAX = 400;
20 const int VIEWPORT_XMIN = 100;
21 const int VIEWPORT_XMAX = 500;
22 const int VIEWPORT_YMIN = -300;
23 const int VIEWPORT_YMAX = 300;
24
25 class Point{
26 private:
27     GLdouble x, y, h;
28
29 public:
30     Point(){
31         x = y = 0;
32         h = 1;
33     }
34
35     Point(GLint xCoord, GLint yCoord){
36         x = xCoord;
37         y = yCoord;
38         h = 1;
39     }
40
41     Point(GLint xCoord, GLint yCoord, GLint H){
42         x = xCoord;
43         y = yCoord;
44         h = H;
45     }
46 }
```

```

47     void setCoords(GLdouble xCoord, GLdouble yCoord){
48         x = xCoord;
49         y = yCoord;
50     }
51
52     void setHomogeneousCoords(GLdouble xCoord, GLdouble yCoord, GLdouble H
53 ){
54         x = xCoord;
55         y = yCoord;
56         h = H;
57     }
58
59     GLdouble getX() const{
60         return x;
61     }
62
63     GLdouble getY() const{
64         return y;
65     }
66
67     GLdouble getH() const{
68         return h;
69     }
70
71     GLdouble getHomogenousX() const{
72         return x * h;
73     }
74
75     GLdouble getHomogenousY() const{
76         return y * h;
77     }
78 };
79
80 class PolygonShape{
81 private:
82     int numVertices;
83     Point *points;
84     double transformMatrix[3][3];
85
86 public:
87     PolygonShape(){
88         numVertices = 0;
89     }
90
91     PolygonShape(int noVertices){
92         numVertices = noVertices;
93         points = new Point[numVertices];
94     }
95
96     int getVertexCount() const{

```

```

97         return numVertices;
98     }
99
100     Point getPoint(int i){
101         return points[i];
102     }
103
104     void setVertices(int noVertices){
105         numVertices = noVertices;
106         points = new Point[numVertices];
107     }
108
109     void setPoint(int i, GLdouble x, GLdouble y, GLdouble h = 1){
110         points[i].setHomogeneousCoords(x, y, h);
111     }
112
113     void setTransformMatrix(){
114         //Perform Window->Viewport Transformation using Translation and
Scaling
115
116         double xShift = VIEWPORT_XMIN - WINDOW_XMIN;
117         double yShift = VIEWPORT_YMIN - WINDOW_YMIN;
118
119         double translateMatrix[3][3] = {    {1, 0, xShift},
120                                             {0, 1, yShift},
121                                             {0, 0, 1}};
122
123         double xScale = (double) (VIEWPORT_XMAX - VIEWPORT_XMIN) / (
WINDOW_XMAX - WINDOW_XMIN);
124         double yScale = (double) (VIEWPORT_YMAX - VIEWPORT_YMIN) / (
WINDOW_YMAX - WINDOW_YMIN);
125
126         Point pivot(VIEWPORT_XMIN, VIEWPORT_YMIN);
127
128         double scaleMatrix[3][3] = {    {xScale, 0, pivot.getHomogenousX()
* (1 - xScale)},
129
130                                         {0, yScale, pivot.getHomogenousY()
* (1 - yScale)},
131
132                                         {0, 0, 1}};
133
134         double product = 0;
135
136         //Composite Transformation = Scaling * Translation
137         for(int i = 0; i < 3; i++){
138             for(int j = 0; j < 3; j++){
139                 product = 0;
140
141                 for(int k = 0; k < 3; k++){
142                     product += scaleMatrix[i][k] * translateMatrix[k][j];

```



```

143         transformMatrix[i][j] = product;
144     }
145 }
146 }
147
148 PolygonShape getViewPortPolygon(){
149
150     PolygonShape polyDash(numVertices);
151     double values[3];
152
153     for(int i = 0; i < numVertices; i++){
154         Point p = getPoint(i);
155
156         //[3 x 3] x [3 x 1] = [3 x 1] matrix
157         for(int j = 0; j < 3; j++){
158             values[j] = transformMatrix[j][0] * p.getHomogenousX() +
159                       transformMatrix[j][1] * p.getHomogenousY() +
160                       transformMatrix[j][2] * p.getH();
161         }
162
163         polyDash.setPoint(i, values[0]/p.getH(), values[1]/p.getH(),
164 values[2]);
165     }
166
167     return polyDash;
168 };
169
170
171 void initializeDisplay();
172 void plotComponents();
173 void plotBoundaries();
174 void plotPolygon(PolygonShape polygon, bool transformed = false);
175
176 PolygonShape polygon;
177
178 int main(int argc, char **argv){
179     glutInit(&argc, argv);
180     glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
181     glutInitWindowPosition(0, 0);
182     glutInitWindowSize(WIDTH, HEIGHT);
183     glutCreateWindow("Window To Viewport Transformation");
184
185     polygon.setVertices(3);
186     polygon.setPoint(0, -300, -300);
187     polygon.setPoint(1, -100, -300);
188     polygon.setPoint(2, -100, 300);
189     //polygon.setPoint(3, -300, 300);
190
191     initializeDisplay();
192     glutDisplayFunc(plotComponents);

```

```

193
194     glutMainLoop();
195
196     return 1;
197 }
198
199 void initializeDisplay(){
200     //Initialize the display parameters
201
202     glClearColor(1, 1, 1, 0);
203     glMatrixMode(GL_PROJECTION);
204     gluOrtho2D(-WIDTH/2, WIDTH/2, -HEIGHT/2, HEIGHT/2);
205     glClear(GL_COLOR_BUFFER_BIT);
206 }
207
208 void plotComponents(){
209     //Plot the window, viewport and transformations
210
211     plotBoundaries();
212     plotPolygon(polygon);
213     polygon.setTransformMatrix();
214     PolygonShape polygonDash = polygon.getViewportPolygon();
215     plotPolygon(polygonDash, true);
216 }
217
218 void plotBoundaries(){
219     //Plot the window and viewport boundaries
220
221     glLineWidth(3);
222
223     //Title of window area
224     glColor3d(0, 0, 1); //Blue color
225     unsigned char windowString[] = "Window Area";
226     glutBitmapLength(GLUT_BITMAP_HELVETICA_18, windowString);
227     glRasterPos2d(-320, 410);
228
229     for(int i = 0; i < strlen((const char *)windowString); i++) {
230         glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, windowString[i]);
231     }
232
233     //Plot the window area
234     glBegin(GL_LINE_LOOP);
235     glVertex2f(WINDOW_XMIN, WINDOW_YMIN);
236     glVertex2f(WINDOW_XMAX, WINDOW_YMIN);
237     glVertex2f(WINDOW_XMAX, WINDOW_YMAX);
238     glVertex2f(WINDOW_XMIN, WINDOW_YMAX);
239     glEnd();
240
241     //Title of viewport area
242     glColor3d(1, 0, 0); //Red color
243     unsigned char viewportString[] = "Viewport Area";

```

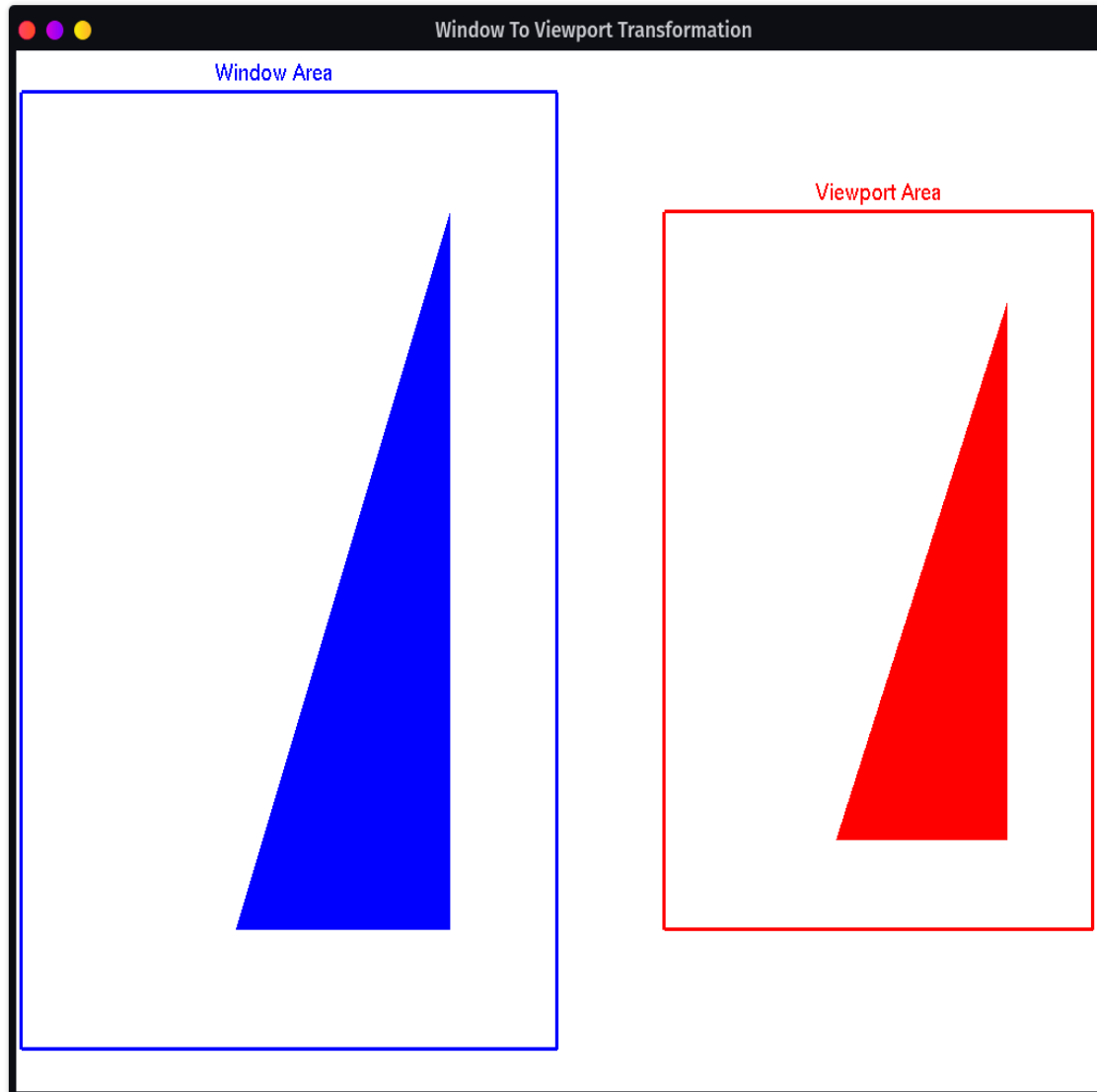
```

244     glutBitmapLength(GLUT_BITMAP_HELVETICA_18, viewportString);
245     glRasterPos2d(240, 310);
246
247     for(int i = 0; i < strlen((const char *)viewportString); i++) {
248         glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, viewportString[i]);
249     }
250
251     //Plot the viewport area
252     glBegin(GL_LINE_LOOP);
253     glVertex2f(VIEWPORT_XMIN, VIEWPORT_YMIN);
254     glVertex2f(VIEWPORT_XMAX, VIEWPORT_YMIN);
255     glVertex2f(VIEWPORT_XMAX, VIEWPORT_YMAX);
256     glVertex2f(VIEWPORT_XMIN, VIEWPORT_YMAX);
257     glEnd();
258
259     glFlush();
260 }
261
262 void plotPolygon(PolygonShape polygon, bool transformed){
263     //To draw a given polygon
264
265     if(!transformed){
266         glColor3d(0, 0, 1); //Blue color
267     } else{
268         glColor3d(1, 0, 0); //Red Color
269     }
270
271     glBegin(GL_POLYGON);
272
273     for(int i = 0; i < polygon.getVertexCount(); i++){
274         Point p = polygon.getPoint(i);
275         glVertex2f(p.getX(), p.getY());
276     }
277
278     glEnd();
279 }

```

Output: Window To Viewport Transformation

Figure 8: Window To Viewport Transformation.



Learning Outcome:

- I understood how to perform **composite transformations** using OpenGL and C++ programming.
- I learnt to perform proper matrix multiplication of the base transformation matrices to get the appropriate composite transformation matrix.
- I applied the composite transformation on the base polygon and displayed the transformed polygon on the graph window.
- I learnt how to overcome the screen refreshing issue/asynchronous event handling while getting user I/O with the help of **glutTimerFunc()** and setting a **60 FPS** refresh rate.
- I was able to perform composite transformations based on translation, rotation, scaling, reflection and shearing.
- I learnt about **Window to Viewport Transformation**.
- I learnt how to set up 2 windows, each to simulate a window and a viewport output window for the purpose of demonstrating the transformation.
- I learnt to perform the transformation from window to viewport using **Translation & Scaling** transformations.
- I learnt the formula for performing the specific translation and scaling required for viewport transformation.
- I understood how to display raster text using the **glutBitmapCharacter()** and **glRasterPos2d()** methods.
- I refreshed my C/C++ concepts regarding **cin, cout and memcpy()** methods.
- I learnt how to configure **OpenGL** in **Linux**.