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.
7 #include <stdio.h>
8 #include <math.h>
9 #include <GL/glut.h>
10 #include <iostream>
                           //for cin, cout
11 #include <cstring>
                          //for memcpy
13 using namespace std;
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;
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};
27 class Point{
28 private:
      GLdouble x, y, h;
31 public:
      Point(){
          x = y = 0;
34
          h = 1;
      }
35
36
      Point(GLint xCoord, GLint yCoord){
          x = xCoord;
38
          y = yCoord;
          h = 1;
40
42
      Point(GLint xCoord, GLint yCoord, GLint H){
          x = xCoord;
```

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

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

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

189

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

```
double scaleMatrix[3][3] = {
                                          {ScaleX, 0, xFixedValue},
240
                                             {0, ScaleY, yFixedValue},
241
                                             {0, 0, 1}};
242
243
           if(!matrix1Flag){
244
               memcpy(matrix1, scaleMatrix, sizeof(scaleMatrix));
               matrix1Flag = true;
246
           } else{
247
               memcpy(matrix2, scaleMatrix, sizeof(scaleMatrix));
248
               matrix2Flag = true;
249
           }
250
       }
251
252
       void setShearMatrix(double shearParam, int axis, double refConst = 0){
253
           //Sets a shear matrix to one of the transformation matrices
254
255
           double shearMatrix[3][3];
256
           switch(axis){
258
               case xAxis:{
259
                   260
      refConst},
                                             {0, 1, 0},
261
                                             {0, 0, 1}};
262
263
                   memcpy(shearMatrix, temp, sizeof(temp));
264
265
                   break;
               }
266
268
               case yAxis:{
269
                   double temp[3][3] = {
                                             {1, 0, -shearParam * refConst},
270
                                             {shearParam, 1, 0},
                                             {0, 0, 1}};
272
273
                   memcpy(shearMatrix, temp, sizeof(temp));
274
                   break;
275
               }
           }
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
```

```
//of the two transformation matrices
291
            if (!matrix1Flag || !matrix2Flag){
292
                //if any one matrix is not set, don't multiply
293
                return;
294
            }
295
296
            for(int i = 0; i < 3; i++){</pre>
297
                for(int j = 0; j < 3; j++){
298
                     double tempSum = 0;
300
                     for (int k = 0; k < 3; k++) {
301
                          tempSum += matrix1[i][k] * matrix2[k][j];
302
                     }
304
                     compositeMatrix[i][j] = tempSum;
                }
306
            }
307
       }
308
309
       PolygonShape getTransformedPolygon(){
310
            //Obtain the transformed polygon based upon the composite
311
      transformation
312
            PolygonShape polyDash(numVertices);
313
            double values[3];
314
315
            for(int i = 0; i < numVertices; i++){</pre>
316
                Point p = getPoint(i);
318
                //[3 \times 3] \times [3 \times 1] = [3 \times 1] \text{ matrix}
319
                for(int j = 0; j < 3; j++){
320
                     values[j] = compositeMatrix[j][0] * p.getHomogenousX() +
                                   compositeMatrix[j][1] * p.getHomogenousY() +
322
                                   compositeMatrix[j][2] * p.getH();
                }
324
325
                polyDash.setPoint(i, values[0]/p.getH(), values[1]/p.getH(),
326
      values[2]);
            }
327
328
            return polyDash;
329
       }
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);
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
  int main(int argc, char **argv){
347
       glutInit(&argc, argv);
348
       glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
349
       glutInitWindowPosition(0, 0);
       glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
351
       glutCreateWindow("2D Composite Transformations - Examples");
353
       printf("\n----[2D COMPOSITE TRANSFORMATIONS]-----\n");
       printf("\nUsage:\tSelect the required transformations in the console."
355
       printf("\n\tEnter the parameters for the specified transformations.");
356
       printf("\n\tView the output in the GLUT window.");
357
       printf("\n\n-----[2D COMPOSITE TRANSFORMATIONS]-----\n\n");
358
359
       //Set the initial default polygon for the graph
360
       polygon.setVertices(4);
361
       polygon.setPoint(0, 0, 0);
362
       polygon.setPoint(1, 0, 50);
363
       polygon.setPoint(2, 100, 50);
       polygon.setPoint(3, 100, 0);
365
366
       initializeDisplay();
367
       glutDisplayFunc(dummyFunction);
369
       //important - to refresh screen periodically
371
       glutTimerFunc(1000/FPS, mainLoop, 0);
372
373
       glutMainLoop();
374
375
       return 1;
376
377 }
378
  void initializeDisplay(){
       //Initialize the display parameters
380
381
       glClearColor(1, 1, 1, 0);
382
       glMatrixMode(GL_PROJECTION);
       gluOrtho2D(X_MIN, X_MAX, Y_MIN, Y_MAX);
384
       glClear(GL_COLOR_BUFFER_BIT);
                                         //Clear the display window
385
386
```

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

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

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

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

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

Output: Plot With Translation and Rotation

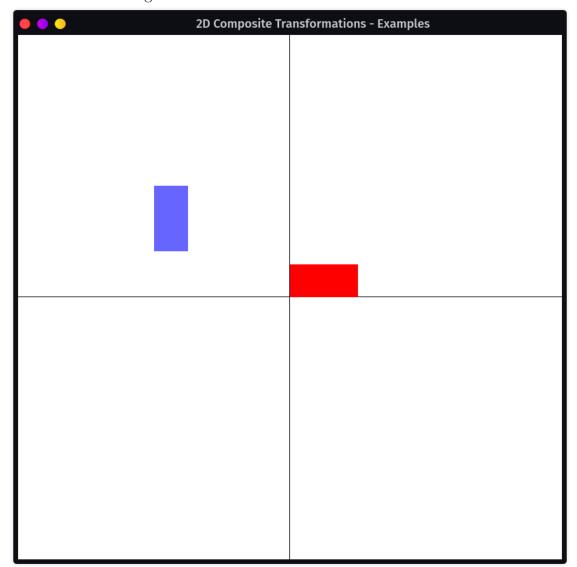


Figure 1: Plot With Translation and Rotation.

Figure 2: Output: Console.

```
EXO6 - Composite Transformations: fout — Konsole

- VLSHakun@ Legion in repo: GraphicsLab/EXO6 - Composite Transformations on P main [?]

- Ret main.cpp = o f.out -lGL -lGLU -lglut

- Vishakun@ Legion in repo: GraphicsLab/EXO6 - Composite Transformations on P main [?] took 513ms

- A ./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 to Exit
Your Option → 1

Choose Transformation 2:

1 for Translation
2 for Rotation
3 for Scaling
```

Figure 3: Output: Console.

```
Ex06 - Composite Transformations: fout — Konsole

Choose Translation 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: 58
    Y Component: 70

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

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

Enter the Rotation Angle: 90

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

Figure 4: Output: Console.

```
Ex06 - Composite Transformations: fout — Konsole

------[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

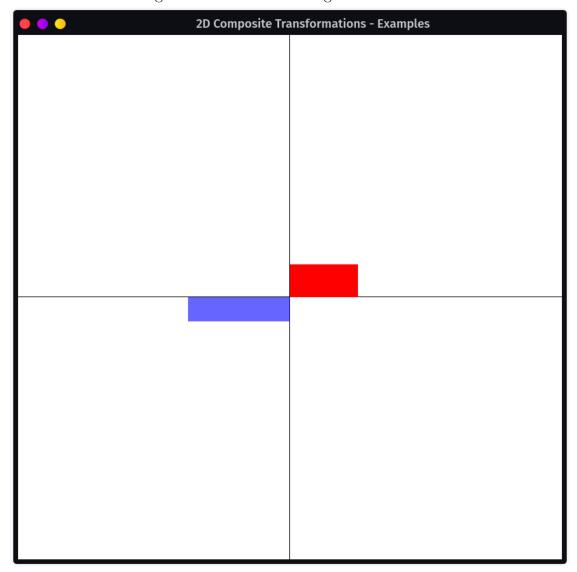


Figure 5: Plot With Scaling and Reflection.

Figure 6: Console.

```
Ex06 - Composite Transformations: fout — Konsole

Choose Transformation 1:
    1 for Translation
    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
```

Figure 7: Console.

```
EXO6 - Composite Transformations: fout — 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]-----

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

------[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.
6 #include <stdio.h>
7 #include <math.h>
8 #include <GL/glut.h>
9 #include <iostream>
                           //for cin, cout
10 #include <cstring>
                           //for memcpy
12 using namespace std;
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;
25 class Point{
26 private:
      GLdouble x, y, h;
29 public:
      Point(){
          x = y = 0;
          h = 1;
32
33
34
      Point(GLint xCoord, GLint yCoord){
36
          x = xCoord;
          y = yCoord;
37
          h = 1;
38
40
      Point(GLint xCoord, GLint yCoord, GLint H){
          x = xCoord;
42
          y = yCoord;
          h = H;
44
      }
```

```
void setCoords(GLdouble xCoord, GLdouble yCoord){
           x = xCoord;
48
           y = yCoord;
      }
50
51
      void setHomogeneousCoords(GLdouble xCoord, GLdouble yCoord, GLdouble H
     ) {
           x = xCoord;
           y = yCoord;
54
           h = H;
55
56
57
58
       GLdouble getX() const{
           return x;
59
60
61
      GLdouble getY() const{
62
           return y;
64
       GLdouble getH() const{
66
           return h;
68
69
      GLdouble getHomogenousX() const{
70
           return x * h;
71
72
      }
73
       GLdouble getHomogenousY() const{
75
           return y * h;
77
<sub>78</sub> };
80 class PolygonShape{
81 private:
      int numVertices;
      Point *points;
83
      double transformMatrix[3][3];
86 public:
      PolygonShape(){
87
           numVertices = 0;
88
      }
89
90
      PolygonShape(int noVertices){
91
92
           numVertices = noVertices;
           points = new Point[numVertices];
93
      }
94
      int getVertexCount() const{
```

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

```
transformMatrix[i][j] = product;
143
                }
144
           }
145
       }
146
147
       PolygonShape getViewportPolygon(){
148
149
            PolygonShape polyDash(numVertices);
           double values[3];
           for(int i = 0; i < numVertices; i++){</pre>
                Point p = getPoint(i);
154
                //[3 \times 3] \times [3 \times 1] = [3 \times 1] \text{ matrix}
                for(int j = 0; j < 3; j++){
                     values[j] = transformMatrix[j][0] * p.getHomogenousX() +
                                  transformMatrix[j][1] * p.getHomogenousY() +
159
                                  transformMatrix[j][2] * p.getH();
                }
161
162
                polyDash.setPoint(i, values[0]/p.getH(), values[1]/p.getH(),
      values[2]);
           }
164
165
           return polyDash;
166
       }
167
168 };
171 void initializeDisplay();
172 void plotComponents();
173 Void plotBoundaries();
   void plotPolygon(PolygonShape polygon, bool transformed = false);
175
  PolygonShape polygon;
177
  int main(int argc, char **argv){
       glutInit(&argc, argv);
179
       glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
180
       glutInitWindowPosition(0, 0);
181
       glutInitWindowSize(WIDTH, HEIGHT);
182
       glutCreateWindow("Window To Viewport Transformation");
183
184
185
       polygon.setVertices(3);
       polygon.setPoint(0, -300, -300);
186
       polygon.setPoint(1, -100, -300);
187
       polygon.setPoint(2, -100, 300);
188
       //polygon.setPoint(3, -300, 300);
190
       initializeDisplay();
191
       glutDisplayFunc(plotComponents);
```

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

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

Output: Window To Viewport Transformation

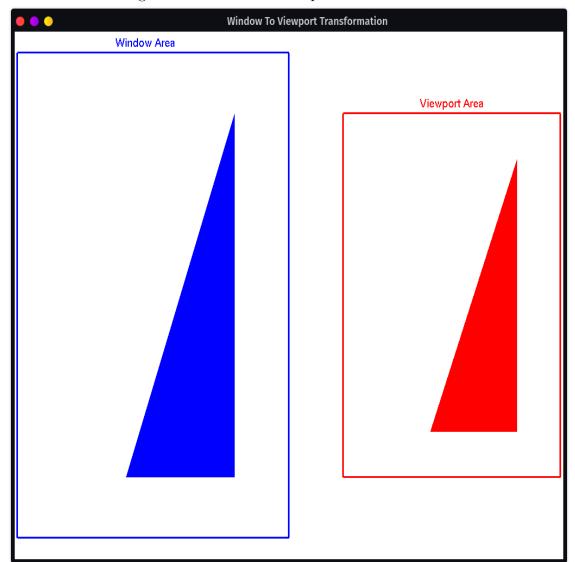


Figure 8: Window To Viewport Transformation.

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**.