Ex. No. 5	**2D Transformations**	**Karthik D**
September 25, 2022	**UCS1712 - Graphics and Multimedia Lab**	**195001047**

Aim

To develop a C++ program using the OpenGL framework to implement the 2D transformation algorithms, and demonstrate all its output cases.

Question

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

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

2D Transformation Algorithms

// assume transformations for a triangle
Procedure plot2DTransformations(x1, x2, x3, y1, y2, y3);

```
var
tx, ty, theta, xr, yr, xref, yref: integer;
shx, shy, sx, sy: float;
(accept parameters from user)

Begin

    triangle_mat := [ [x1, x2, x3], [y1, y2, y3], [1, 1, 1] ]

    // translation
    translation_mat := [ [ 1 0 tx ], [0, 1, ty], [0, 0, 1] ]
    translated_triangle := translation_mat * triangle_mat

    // rotation
    rotation_mat := [ [ cos(theta) -sin(theta) 0 ], [sin(theta), cos(theta), 0], [0, 0, 1] ]
    pivotpt_mat := [ [ 1 0 xr ], [0, 1, yr], [0, 0, 1] ]
    pivotpt_rev_mat := [ [ 1 0 -xr ], [0, 1, -yr], [0, 0, 1] ]
    rotated_triangle := pitvotpt_mat * rotation_mat * pivotpt_rev_mat * triangle_mat

    // scaling
    scaling_mat := [ [ sx 0 0 ], [0, sy, 0], [0, 0, 1] ]
    fixedpt_mat := [ [ 1 0 xr ], [0, 1, yr], [0, 0, 1] ]
    fixedpt_rev_mat := [ [ 1 0 -xr ], [0, 1, -yr], [0, 0, 1] ]
    scaled_triangle := fixedpt_mat * scaling_mat * fixedpt_rev_mat * triangle_mat

    // reflection
    xref_mat := [ [ 1 0 0 ], [0, -1, 0], [0, 0, 1] ]
    yref_mat := [ [ -1 0 0 ], [0, 1, 0], [0, 0, 1] ]
    xyref_mat := [ [ -1 0 0 ], [0, -1, 0], [0, 0, 1] ]
    xeqyref_mat := [ [ 0 1 0 ], [1, 0, 0], [0, 0, 1] ]
    xref_triangle := xref_mat * triangle_mat
    yref_triangle := yref_mat * triangle_mat
    xyref_triangle := xyref_mat * triangle_mat
    xeqyref_triangle := xeqyref_mat * triangle_mat

    // shearing
    xshearing_mat := [ [ 1 shx -shx.yref ], [0, -1, 0], [0, 0, 1] ]
    yshearing_mat := [ [ 1 0 0 ], [shy 1 -shy.xref], [0, 0, 1] ]
```

xshear_triangle := xshearing_mat * triangle_mat
yshear_triangle := yshearing_mat * triangle_mat

End {plot2DTransformations}

Implementation using C++ Program Code

1. main.cpp - Driver and Handler to render all 2D tranformations

```
#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define PI 3.141592654


void renderSpacedBitmapString(float x, float y, void *font, char
*string) {
   char *c;
   int x1 = x;
   for (c = string; *c != '\0'; c++) {
       glRasterPos2f(x1, y);
       glutBitmapCharacter(font, *c);
       x1 = x1 + glutBitmapWidth(font, *c);
   }
}


void markString(char *string, int x, int y, int x_offset, int y_offset)
{
   glColor3f(255.0, 0, 0.0); // red color
   renderSpacedBitmapString(x+x_offset, y+y_offset,
GLUT_BITMAP_HELVETICA_12, string);
   glFlush();
}
```

```
float** multiplyMatrices(float **m1, float **m2, int r1, int c1, int c2)
{
   // assume compatible matrices
   float **res = (float**)malloc(sizeof(float*)*r1);
   for(int i=0; i<r1; i++) {
       *(res+i) = (float*)malloc(sizeof(float)*c2);
       for(int j=0; j<c2; j++) {
           res[i][j] = 0;
           for(int k=0; k<c1; k++)   {
               res[i][j] += m1[i][k] * m2[k][j];
           }
       }
   }
   return res;
}


void displayMatrix(float **matrix, int r, int c)  {
   printf("\n");
   for(int i=0;i<r;i++)    {
       for(int j=0; j<c; j++)  {
           printf("%f ", matrix[i][j]);
       }
       printf("\n");
   }
}


void plotDivisionLines()    {
   glBegin(GL_LINES);
   glVertex2d(-320, 0);
   glVertex2d(320, 0);
   glVertex2d(0, -240);
   glVertex2d(0, 240);
   glEnd();
}
```

```
void plotPoint(int x, int y, int x_offset, int y_offset)    {
   glBegin(GL_POINTS);
   glVertex2d(x + x_offset, y + y_offset);
   glEnd();
}


void plotTriangle(int *xs, int *ys)   {
   glBegin(GL_TRIANGLES);
   for(int i=0; i<3; i++)  {
       glVertex2d(xs[i], ys[i]);
   }
   glEnd();
}


float** makeTriangleMatrix(int *xs, int *ys)  {
   float **res = (float**)malloc(sizeof(float*)*3);
   for(int i=0; i<3; i++) {
       *(res+i) = (float*)malloc(sizeof(float)*3);
   }
   for(int i=0; i<3; i++) {
       res[0][i] = xs[i];
       res[1][i] = ys[i];
       res[2][i] = 1;
   }
   return res;
}


float** makeTranslationMatrix(int tx, int ty) {
   float **res = (float**)malloc(sizeof(float*)*3);
   for(int i=0; i<3; i++) {
       *(res+i) = (float*)malloc(sizeof(float)*3);
       for(int j=0; j<3; j++) {
           if(i==j){
               res[i][j] = 1;
           }
```

```
            else{
                res[i][j] = 0;
            }
        }
    }
    res[0][2] = tx;
    res[1][2] = ty;
    return res;
}


float** makeRotationMatrix(int theta) {
    float **res = (float**)malloc(sizeof(float*)*3);
    for(int i=0; i<3; i++) {
        *(res+i) = (float*)malloc(sizeof(float*)*3);
        for(int j=0; j<3; j++) {
            if(i==j){
                res[i][j] = 1;
            }
            else{
                res[i][j] = 0;
            }
        }
    }
    res[0][0] = cos(theta*PI/180);
    res[1][1] = res[0][0];
    res[0][1] = -sin(theta*PI/180);
    res[1][0] = -res[0][1];
    return res;
}


float** makeScalingMatrix(float sx, float sy)   {
    float **res = (float**)malloc(sizeof(float*)*3);
    for(int i=0; i<3; i++) {
        *(res+i) = (float*)malloc(sizeof(float*)*3);
        for(int j=0; j<3; j++) {
            if(i==j){
```

```
            res[i][j] = 1;
        }
        else{
            res[i][j] = 0;
        }
    }
  }
  res[0][0] = sx;
  res[1][1] = sy;
  return res;
}


float** makeReflectionMatrix(short along_x, short along_y, short
along_xeqy) {
  // truth of the first two arguments overrides the last

  float **res = (float**)malloc(sizeof(float*)*3);
  for(int i=0; i<3; i++) {
      *(res+i) = (float*)malloc(sizeof(float)*3);
      for(int j=0; j<3; j++) {
          if(i==j){
              res[i][j] = 1;
          }
          else{
              res[i][j] = 0;
          }
      }
  }
  short override_xeqy = 0;
  if(along_x) {
      res[1][1] = -1;
      override_xeqy = 1;
  }
  if(along_y) {
      res[0][0] = -1;
      override_xeqy = 1;
  }
```

```
   if(!override_xeqy && along_xeqy)    {
       float *temp = res[0];
       res[0] = res[1];
       res[1] = temp;
   }
   return res;
}


float **makeShearingMatrix(float xshear, float yshear, int yref, int
xref)  {
   float **res = (float**)malloc(sizeof(float*)*3);
   for(int i=0; i<3; i++) {
       *(res+i) = (float*)malloc(sizeof(float)*3);
       for(int j=0; j<3; j++) {
           if(i==j){
               res[i][j] = 1;
           }
           else{
               res[i][j] = 0;
           }
       }
   }
   res[0][1] = xshear;
   res[0][2] = -(xshear*yref);
   res[1][0] = yshear;
   res[1][2] = -(yshear*xref);
   return res;
}


void plotTranslatedTriangle(int *xs, int *ys, int tx, int ty)    {
   glBegin(GL_TRIANGLES);
   float **tr_matrix = makeTranslationMatrix(tx, ty);
   float **triangle_matrix = makeTriangleMatrix(xs, ys);
   float **translated_triangle = multiplyMatrices(tr_matrix,
triangle_matrix, 3, 3, 3);
   for(int i=0; i<3; i++)  {
```

```
       glVertex2d((int)translated_triangle[0][i],
(int)translated_triangle[1][i]);
   }
   glEnd();

   char *string = (char*)malloc(sizeof(char)*100);
   for(int i=0; i<3; i++)  {
       sprintf(string, "(%d, %d)", (int)translated_triangle[0][i],
(int)translated_triangle[1][i]);
       markString(string, (int)translated_triangle[0][i],
(int)translated_triangle[1][i], 0, -10);
   }
}


void plotRotatedTriangle(int *xs, int *ys, int xr, int yr, int theta)
{
   // (xr, yr) --> Pivot Point
   glBegin(GL_TRIANGLES);

   float **triangle_matrix = makeTriangleMatrix(xs, ys);

   float **rotated_triangle = multiplyMatrices(
       makeTranslationMatrix(xr, yr),
       multiplyMatrices(
           makeRotationMatrix(theta),
           multiplyMatrices(
               makeTranslationMatrix(-xr, -yr),
               triangle_matrix,
               3, 3, 3
           ),
           3, 3, 3
       ),
       3, 3, 3);

   for(int i=0; i<3; i++)  {
       glVertex2d((int)rotated_triangle[0][i],
(int)rotated_triangle[1][i]);
```

```
   }
   glEnd();

   char *string = (char*)malloc(sizeof(char)*100);
   for(int i=0; i<3; i++)  {
       sprintf(string, "(%d, %d)", (int)rotated_triangle[0][i],
(int)rotated_triangle[1][i]);
       markString(string, (int)rotated_triangle[0][i],
(int)rotated_triangle[1][i], 0, -10);
   }
}


void plotScaledTriangle(int *xs, int *ys, int xf, int yf, float sx,
float sy, int x_offset, int y_offset)    {
   // (xf, yf) --> Fixed Point
   glBegin(GL_TRIANGLES);

   float **triangle_matrix = makeTriangleMatrix(xs, ys);

   float **actual_result = multiplyMatrices(
   makeTranslationMatrix(xf, yf),
   multiplyMatrices(
       makeScalingMatrix(sx, sy),
       multiplyMatrices(
           makeTranslationMatrix(-xf, -yf),
           triangle_matrix,
           3, 3, 3
       ),
       3, 3, 3
   ),
   3, 3, 3);

   float **scaled_triangle = multiplyMatrices(
       makeTranslationMatrix(x_offset, y_offset),
       actual_result,
       3, 3, 3);
```

```
   for(int i=0; i<3; i++)  {
       glVertex2d((int)scaled_triangle[0][i],
(int)scaled_triangle[1][i]);
   }
   glEnd();

   char *string = (char*)malloc(sizeof(char)*100);
   for(int i=0; i<3; i++)  {
       sprintf(string, "(%d, %d)", (int)actual_result[0][i],
(int)actual_result[1][i]);
       markString(string, (int)scaled_triangle[0][i],
(int)scaled_triangle[1][i], 0, -10);
   }
   sprintf(string, "Xscale: %.2f, Yscale: %.2f, Ref: (%d, %d)", sx, sy,
xf, yf);
   markString(string, 100, 180, x_offset, y_offset);
}


void plotReflectedTriangle(int *xs, int *ys)    {
   glBegin(GL_TRIANGLES);

   float **xref_matrix = makeReflectionMatrix(1, 0, 0);
   float **yref_matrix = makeReflectionMatrix(0, 1, 0);
   float **xyref_matrix = makeReflectionMatrix(1, 1, 0);
   float **xeqyref_matrix = makeReflectionMatrix(0, 0, 1);

   float **triangle_matrix = makeTriangleMatrix(xs, ys);

   float **xref_triangle = multiplyMatrices(xref_matrix,
triangle_matrix, 3, 3, 3);
   float **yref_triangle = multiplyMatrices(yref_matrix,
triangle_matrix, 3, 3, 3);
   float **xyref_triangle = multiplyMatrices(xyref_matrix,
triangle_matrix, 3, 3, 3);
   float **xeqyref_triangle = multiplyMatrices(xeqyref_matrix,
triangle_matrix, 3, 3, 3);
```

```
   for(int i=0; i<3; i++)  {
       glVertex2d((int)xref_triangle[0][i], (int)xref_triangle[1][i]);
   }
   for(int i=0; i<3; i++)  {
       glVertex2d((int)yref_triangle[0][i], (int)yref_triangle[1][i]);
   }
   for(int i=0; i<3; i++)  {
       glVertex2d((int)xyref_triangle[0][i], (int)xyref_triangle[1][i]);
   }
   for(int i=0; i<3; i++)  {
       glVertex2d((int)xeqyref_triangle[0][i],
(int)xeqyref_triangle[1][i]);
   }
   glEnd();

   char *string = (char*)malloc(sizeof(char)*100);
   for(int i=0; i<3; i++)  {
       // plot line
       glBegin(GL_LINES);
       glVertex2d(0, 0);
       glVertex2d(200, 200);
       glEnd();

       sprintf(string, "(%d, %d)", (int)xref_triangle[0][i],
(int)xref_triangle[1][i]);
       markString(string, (int)xref_triangle[0][i],
(int)xref_triangle[1][i], 0, 0);

       sprintf(string, "(%d, %d)", (int)yref_triangle[0][i],
(int)yref_triangle[1][i]);
       markString(string, (int)yref_triangle[0][i],
(int)yref_triangle[1][i], -40, -10);

       sprintf(string, "(%d, %d)", (int)xyref_triangle[0][i],
(int)xyref_triangle[1][i]);
       markString(string, (int)xyref_triangle[0][i],
(int)xyref_triangle[1][i], -60, -5);
```

```
      sprintf(string, "(%d, %d)", (int)xeqyref_triangle[0][i],
(int)xeqyref_triangle[1][i]);
      markString(string, (int)xeqyref_triangle[0][i],
(int)xeqyref_triangle[1][i], 0, -10);
   }
}


void plotShearedTriangle(int *xs, int *ys, float xshear, float yshear,
int yref, int xref, int x_offset, int y_offset)  {
   glBegin(GL_TRIANGLES);

   float **triangle_matrix = makeTriangleMatrix(xs, ys);

   // without translating for display
   float **actual_result = multiplyMatrices(
       makeShearingMatrix(xshear, yshear, yref, xref),
       triangle_matrix,
       3, 3, 3
   );

   float **sheared_triangle = multiplyMatrices(
       makeTranslationMatrix(x_offset, y_offset),
       multiplyMatrices(
           makeShearingMatrix(xshear, yshear, yref, xref),
           triangle_matrix,
           3, 3, 3
       ), 3, 3, 3
   );

   for(int i=0; i<3; i++)  {
       glVertex2d((int)sheared_triangle[0][i],
(int)sheared_triangle[1][i]);
   }
   glEnd();

   char *string = (char*)malloc(sizeof(char)*100);
   for(int i=0; i<3; i++)  {
```

```
      sprintf(string, "(%d, %d)", (int)actual_result[0][i],
(int)actual_result[1][i]);
      markString(string, (int)sheared_triangle[0][i],
(int)sheared_triangle[1][i], 0, 0);
   }

}



void display_transforms()   {
   glClear(GL_COLOR_BUFFER_BIT);
   plotDivisionLines();

   // int xs[3], ys[3];
   int xs[] = {10, 80, 40};
   int ys[] = {70, 100, 180};

   /* GET USER INPUTS */
   // printf("\nVertex 1: ");
   // scanf("%d %d", &xs[0], &ys[0]);
   // printf("\nVertex 2: ");
   // scanf("%d %d", &xs[1], &ys[1]);
   // printf("\nVertex 3: ");
   // scanf("%d %d", &xs[2], &ys[2]);

   /* PLOT MAIN FIGURE -- A TRIANGLE */
   glColor3f(0.0, 0.0, 1.0);
   plotTriangle(xs, ys);

   /* PLOT TRANSFORMATIONS */
   glColor3f(1.0, 0.0, 0.0);

   /* TRANSLATION */
   int tx = -100;
   int ty = -50;
   plotTranslatedTriangle(xs, ys, -100, -50);
   // label translation
```

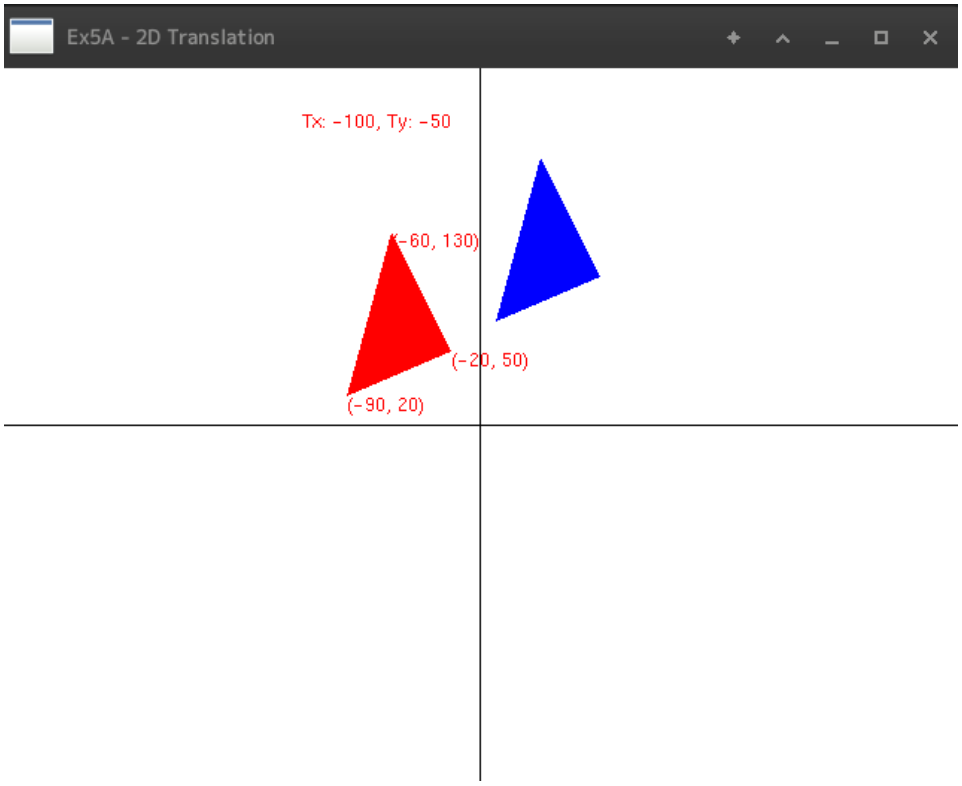
```
  char *string = (char*)malloc(sizeof(char)*100);
  sprintf(string, "Tx: %d, Ty: %d", tx, ty);
  markString(string, 200, 200, -320, 0);

  /* ROTATIONS */
  // int x0 = -100;
  // int y0 = -100;
  // int theta1 = -30;
  // int theta2 = 45;
  // plotRotatedTriangle(xs, ys, 0, 0, theta1);
  // plotRotatedTriangle(xs, ys, x0, y0, theta2);
  // // label rotations
  // char *string = (char*)malloc(sizeof(char)*100);
  // sprintf(string, "Theta: %d, Pivot: (%d, %d)", theta1, 0, 0);
  // markString(string, 120, 10, 0, 0);
  // sprintf(string, "Theta: %d, Pivot: (%d, %d)", theta2, x0, y0);
  // markString(string, 120, 10, -320, 0);

  /* SCALING */
  // int x0 = -100;
  // int y0 = -100;
  // float sx1 = 0.75;
  // float sy1 = 0.75;
  // float sx2 = 1.8;
  // float sy2 = 1.2;
  // plotScaledTriangle(xs, ys, 0, 0, sx1, sy1, 0, -240);
  // plotScaledTriangle(xs, ys, 0, 0, sx2, sy2, -320, 0);
  // plotScaledTriangle(xs, ys, -40, -60, sx1, sy2, -320, -240);

  /* REFLECTION */
  // plotReflectedTriangle(xs, ys);

  /* SHEARING */
  // int yref = -1;
  // int xref = -2;
  // float xshear = 0.9;
  // float yshear = 1.2;
  // plotShearedTriangle(xs, ys, xshear, 0, yref, 0, 0, -240);
```

```
   // plotShearedTriangle(xs, ys, 0, yshear, 0, xref, -320, 0);
   // // label shears
   // char *string = (char*)malloc(sizeof(char)*100);
   // sprintf(string, "Xshear: %.2f, Yref: %d", xshear, yref);
   // markString(string, 50, 50, 0, -240);
   // sprintf(string, "Yshear: %.2f, Xref: %d", yshear, xref);
   // markString(string, 50, 50, -320, 0);

   /* MODIFIED ORIGIN LABELS */
   // markString("(0, 0)", 5, 5, 0, 0);
   // markString("(0, 0)", 5, 5, 0, -240);
   // markString("(0, 0)", 5, 5, -320, 0);
   // markString("(0, 0)", 5, 5, -320, -240);

   glFlush();
}


void init() {
   glClearColor(1.0, 1.0, 1.0, 0.0);
   glColor3f(0.0f, 0.0f, 0.0f);
   glPointSize(4);
   glLineWidth(1);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-320.0, 320.0, -240.0, 240.0);
}


int main(int argc, char **argv)  {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(640, 480);

   glutCreateWindow("Ex5A - 2D Translation");
   // glutCreateWindow("Ex5B - 2D Rotation");
   // glutCreateWindow("Ex5C - 2D Scaling");
   // glutCreateWindow("Ex5D - 2D Reflection");
```

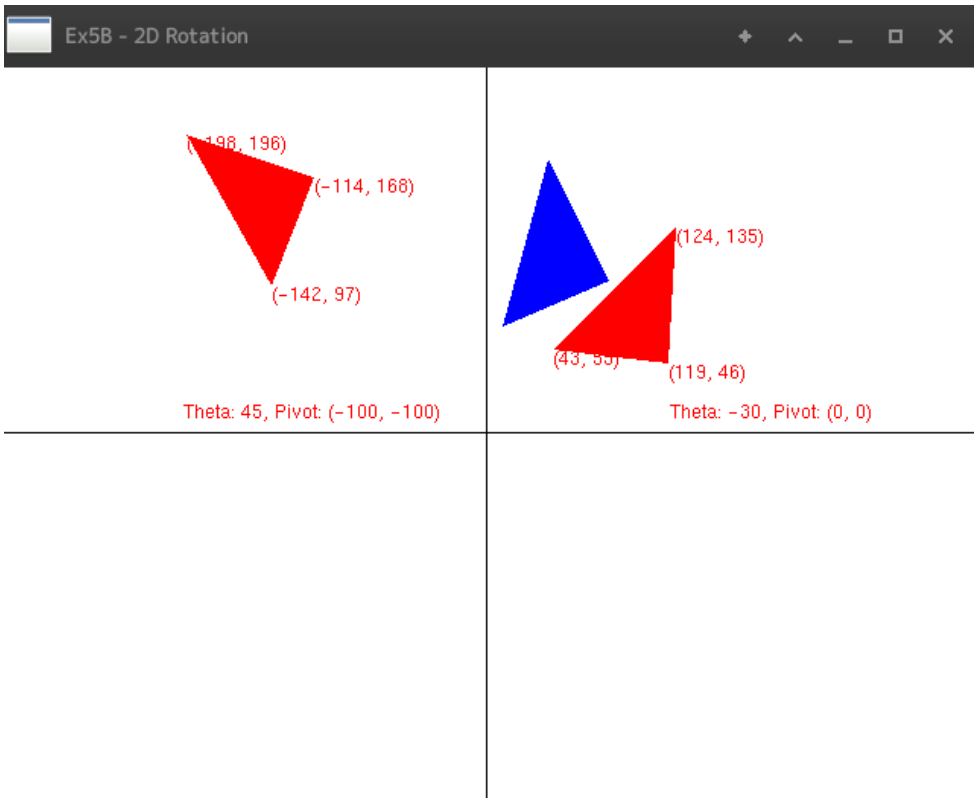
```
   // glutCreateWindow("Ex5E - 2D Shearing");
   glutDisplayFunc(display_transforms);

   init();
   glutMainLoop();
   return 1;
}
```

Sample Output

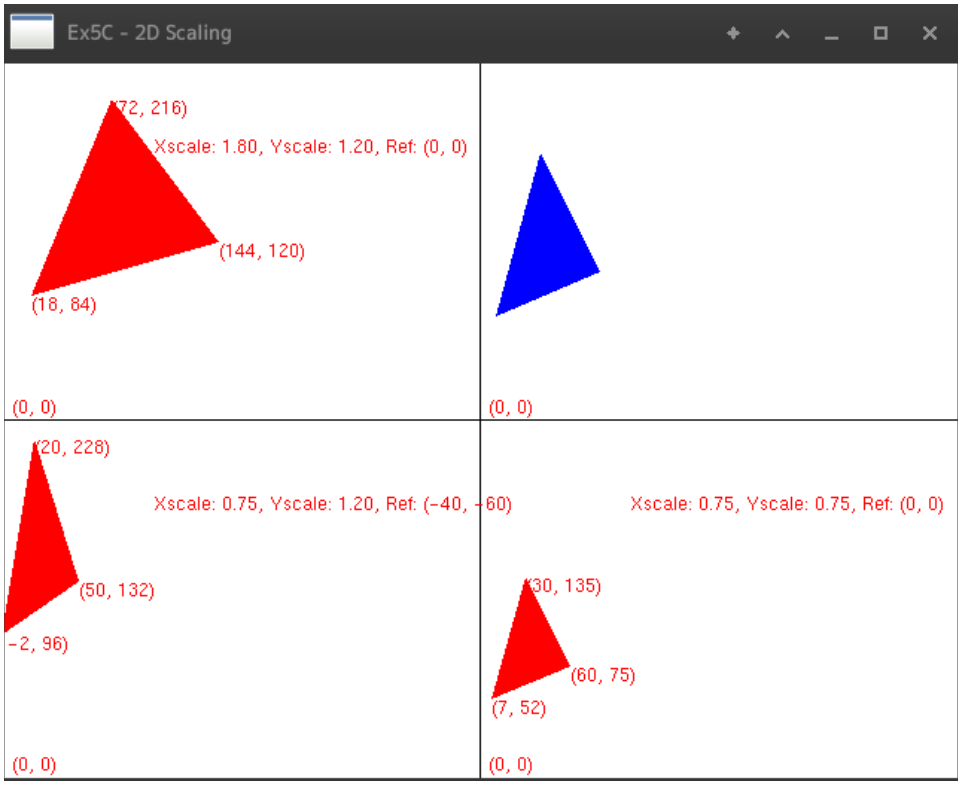
- **All cases for 2D Translation**



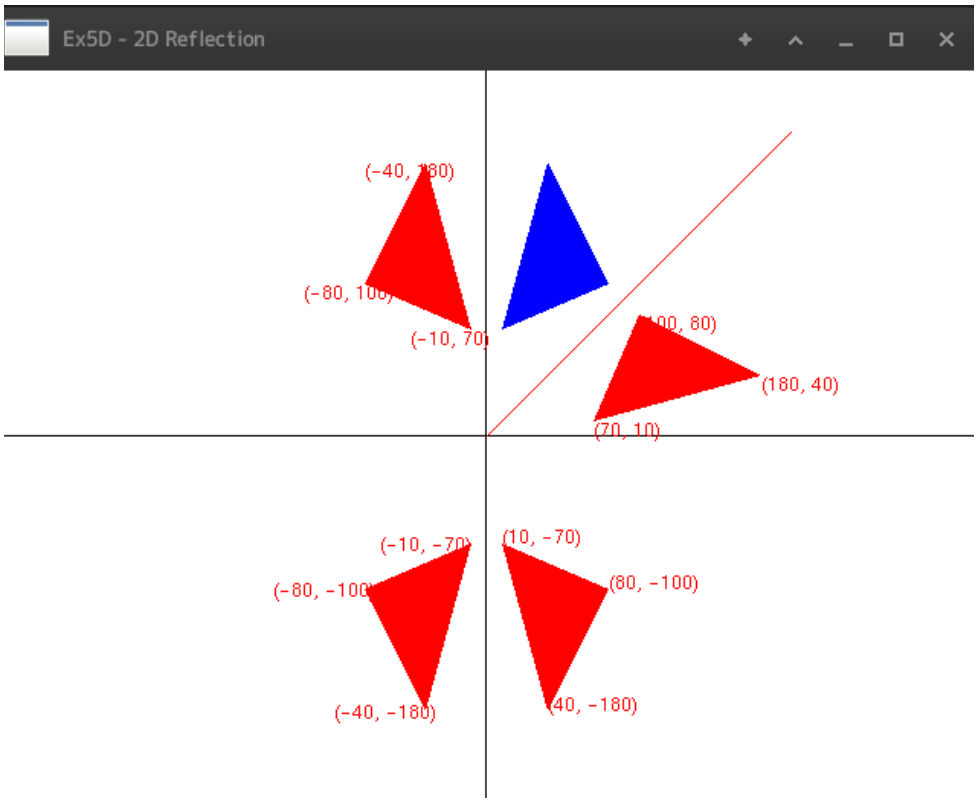
- **All cases for 2D Rotation**



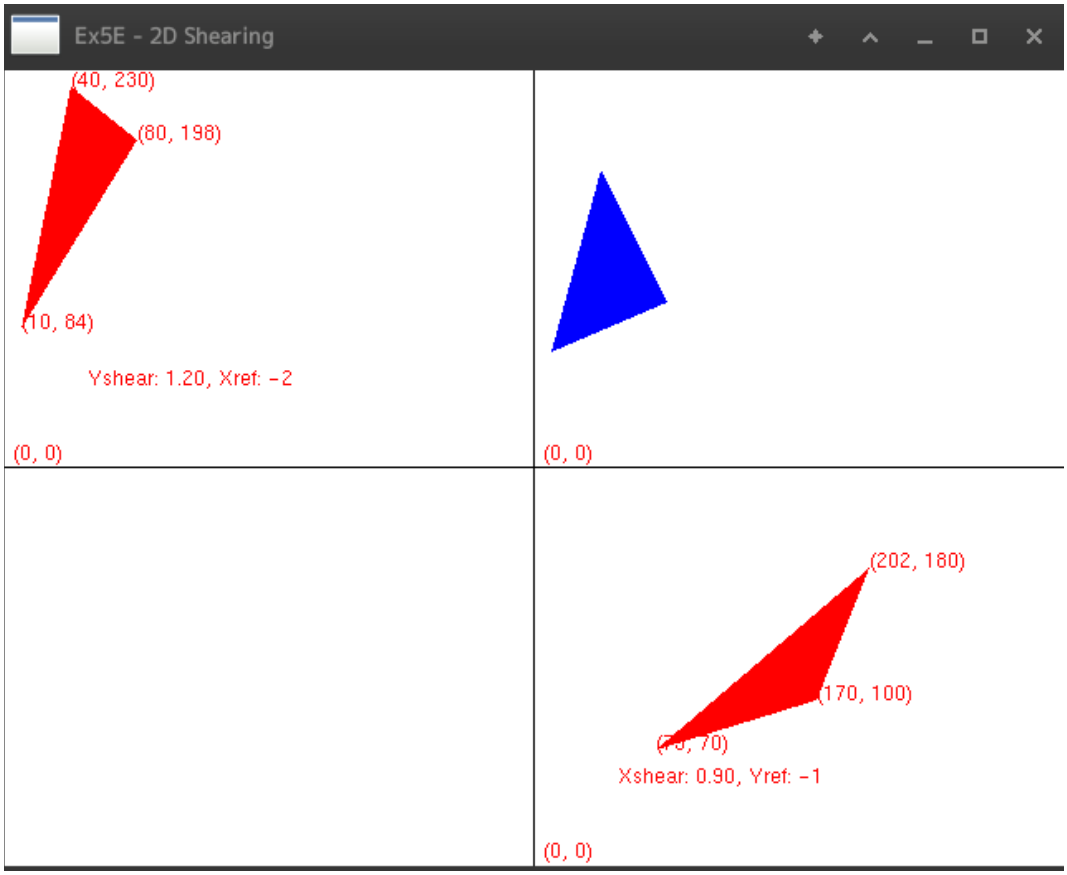
- **All cases for 2D Scaling**



- **All case for 2D Reflection**



- **All case for 2D Shearing**



Learning Outcomes

Through this implementation of 2D transformation algorithms using the OpenGL framework and C++ programming language, the following concepts were learnt:

1. The working of various 2D transformations — reflection, rotation, scaling, shearing and translation.
2. The use and application of homogeneous coordinates when rendering transformations using matrices.
3. General understanding of the OpenGL framework and its APIs.