**Ex. No. 6** **2D Composite Transformations and Windowing** **Karthik D**

**October 1, 2022** **UCS1712 - Graphics and Multimedia Lab** **195001047**

## Aim

To develop a C++ program using the OpenGL framework to implement the 2D composite transformation algorithms and perform window to viewport transformations, and demonstrate all their output cases.

## Question

A) To compute the composite transformation matrix for any 2 transformations input by the user and apply it on the object

   a) Translation

   b) Rotation

   c) Scaling

   d) Reflection

   e) Shearing

B) 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.

**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 axes).

## 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}
```

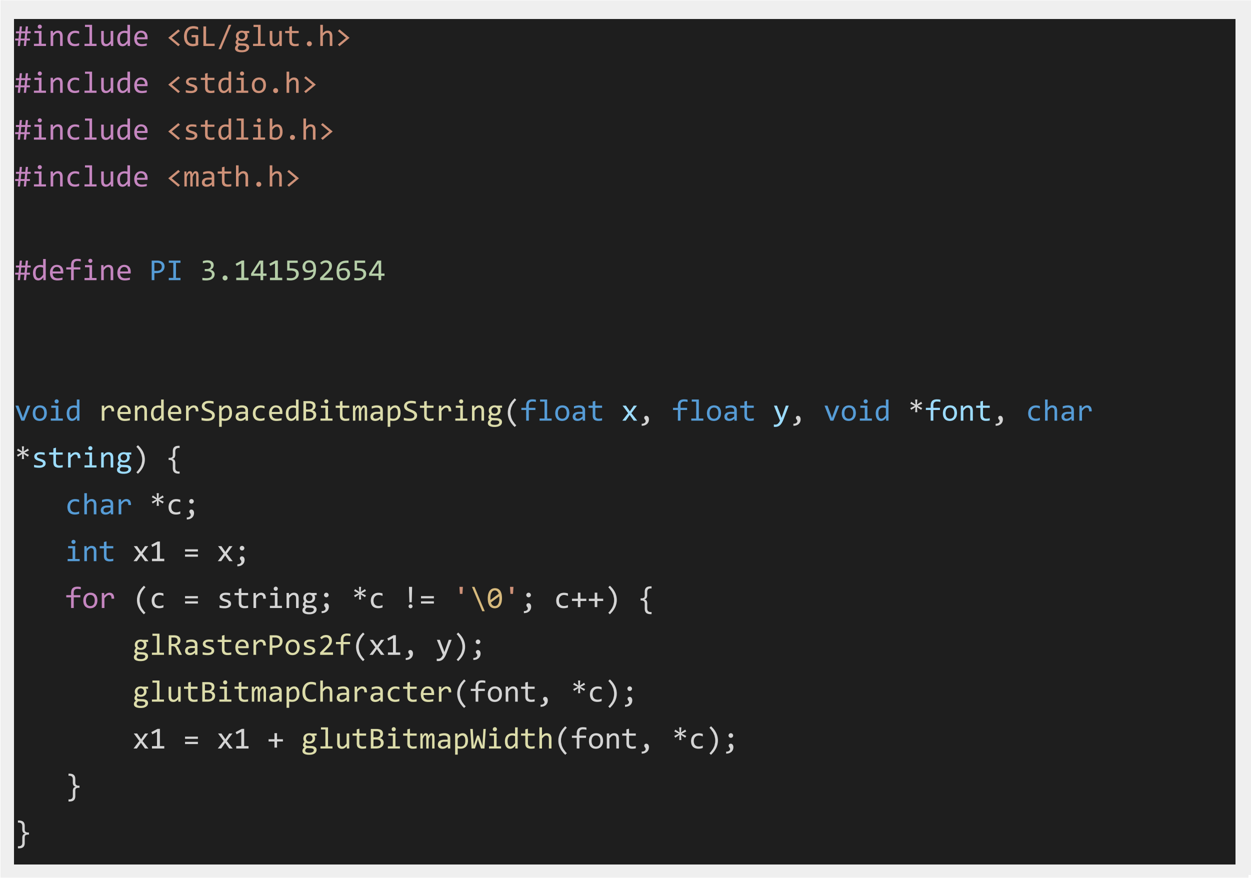
## **World to Viewport Transformation**

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

    var
    sx, sy, xv_min, xv_max, xw_min, xw_max: integer;
```

```
        yv_min, yv_max, yw_min, yw_max: integer;
        (accept parameters from user)

        sx = (xv_max-xv_min)/(xw_max-xw_min);
        sy = (yv_max-yv_min)/(yw_max-yw_min);
        triangle := [[x1, x2, x3], [y1, y2, y3]]
        trans_triangle := [[0, 0, 0], [0, 0, 0]]

        // apply coordinate transformation
        forEach i=1:3
        Begin
                trans_triangle[1][i] = xv_min + (triangle[1][i] -  xw_min)*sx
                trans_triangle[2][i] = yv_min + (triangle[2][i] -  yw_min)*sy
        End

End {plotViewportTransformations}
```

## Implementation using C++ Program Code

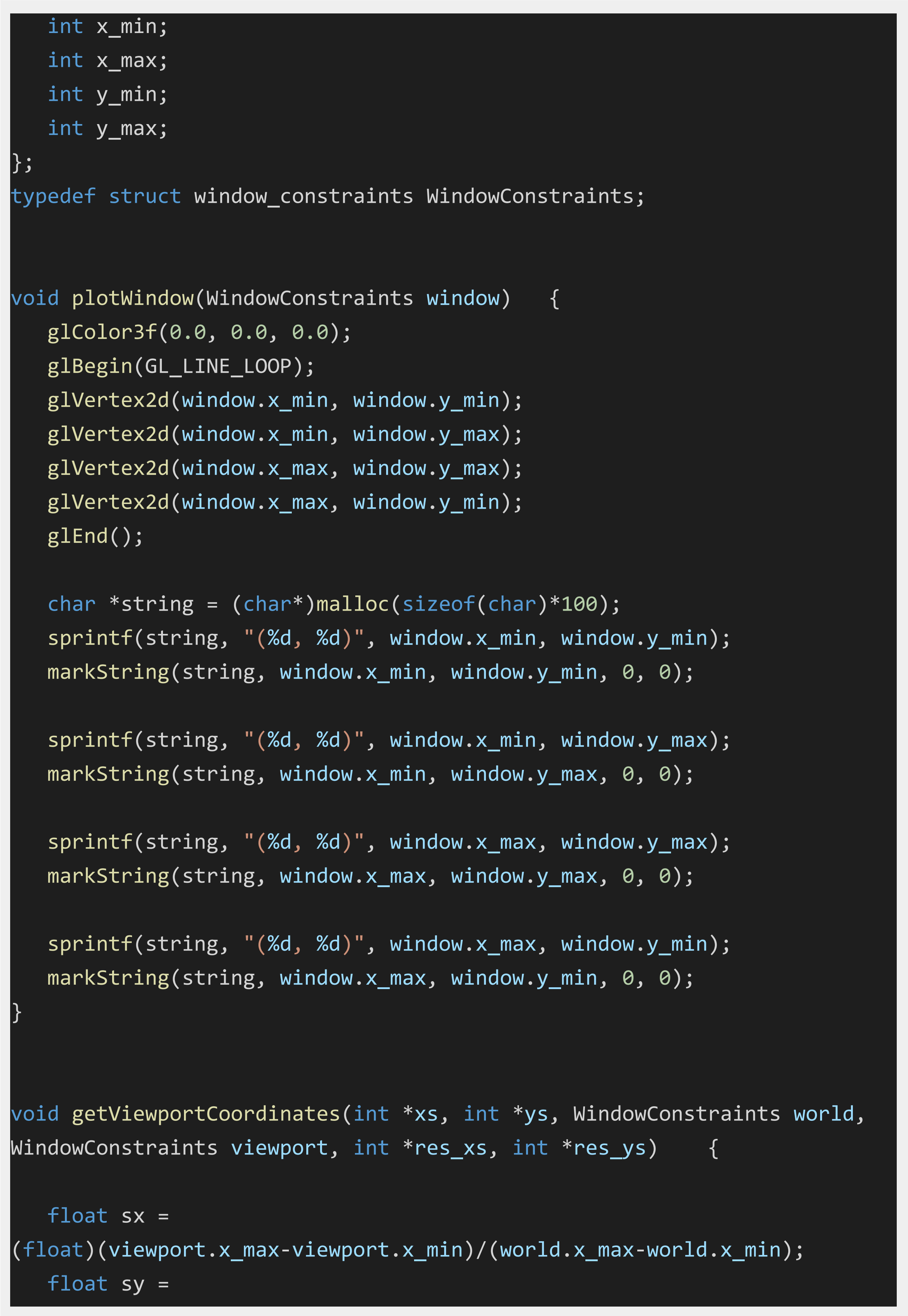
1. main.cpp - Driver and Handler to render all 2D tranformations



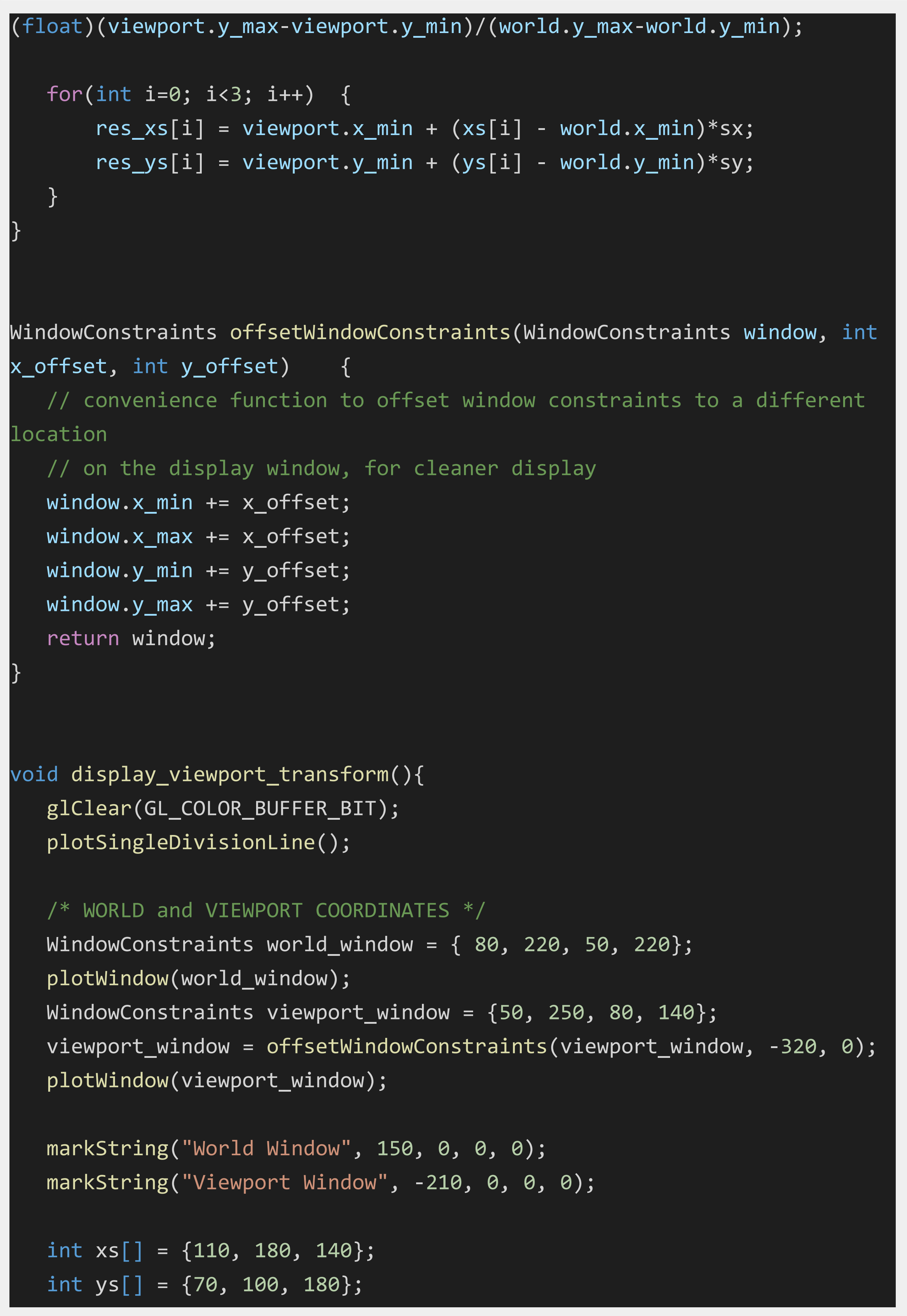
```
#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#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 plotSingleDivisionLine()    {
   glBegin(GL_LINES);
   glVertex2d(0, -240);
   glVertex2d(0, 240);
   glEnd();
}


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();
}


/* WINDOWING ----------------------------------------------------- */

struct window_constraints   {
```

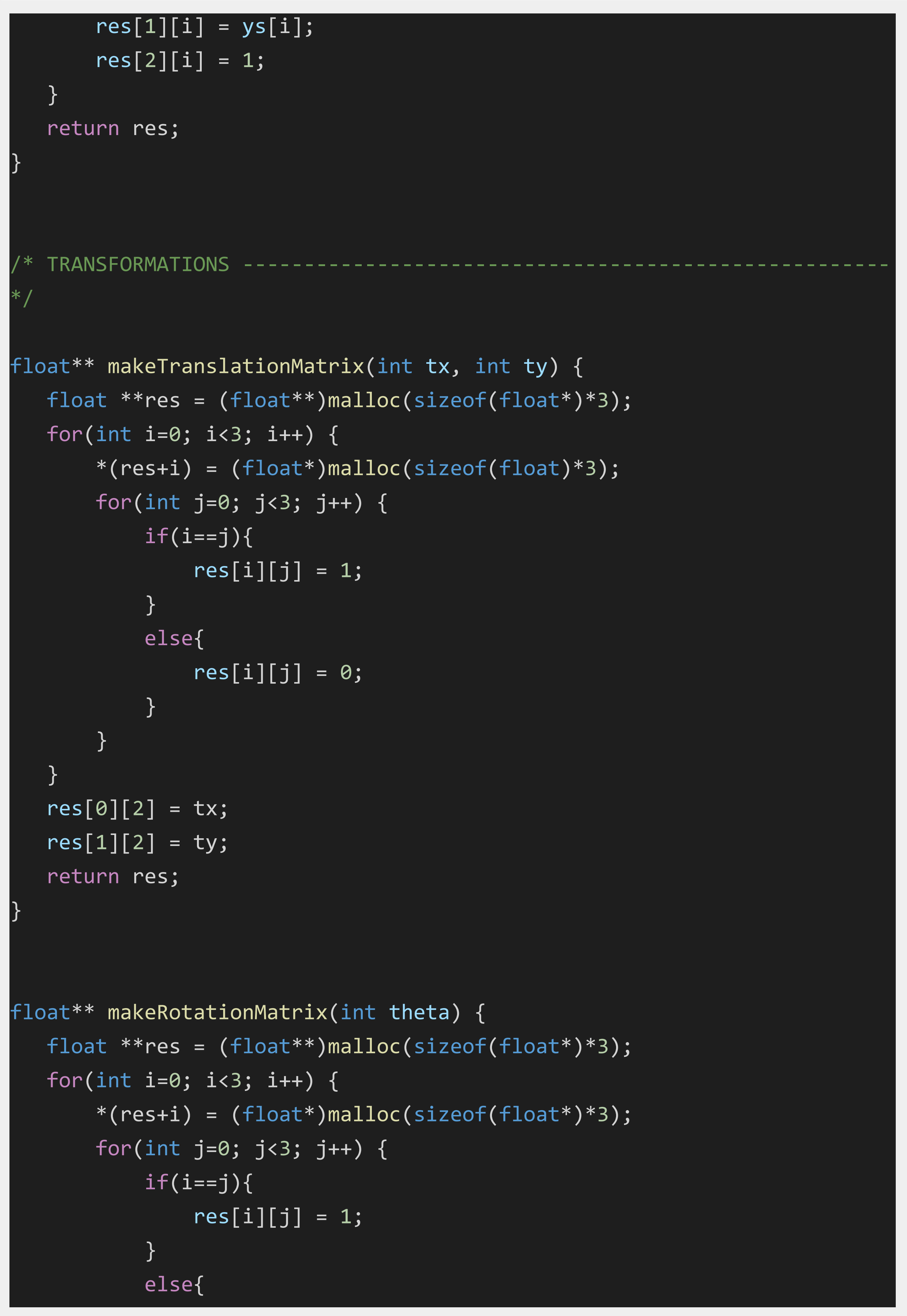


```
   int x_min;
   int x_max;
   int y_min;
   int y_max;
};
typedef struct window_constraints WindowConstraints;


void plotWindow(WindowConstraints window)   {
   glColor3f(0.0, 0.0, 0.0);
   glBegin(GL_LINE_LOOP);
   glVertex2d(window.x_min, window.y_min);
   glVertex2d(window.x_min, window.y_max);
   glVertex2d(window.x_max, window.y_max);
   glVertex2d(window.x_max, window.y_min);
   glEnd();

   char *string = (char*)malloc(sizeof(char)*100);
   sprintf(string, "(%d, %d)", window.x_min, window.y_min);
   markString(string, window.x_min, window.y_min, 0, 0);

   sprintf(string, "(%d, %d)", window.x_min, window.y_max);
   markString(string, window.x_min, window.y_max, 0, 0);

   sprintf(string, "(%d, %d)", window.x_max, window.y_max);
   markString(string, window.x_max, window.y_max, 0, 0);

   sprintf(string, "(%d, %d)", window.x_max, window.y_min);
   markString(string, window.x_max, window.y_min, 0, 0);
}


void getViewportCoordinates(int *xs, int *ys, WindowConstraints world,
WindowConstraints viewport, int *res_xs, int *res_ys)    {

   float sx =
(float)(viewport.x_max-viewport.x_min)/(world.x_max-world.x_min);
   float sy =
```

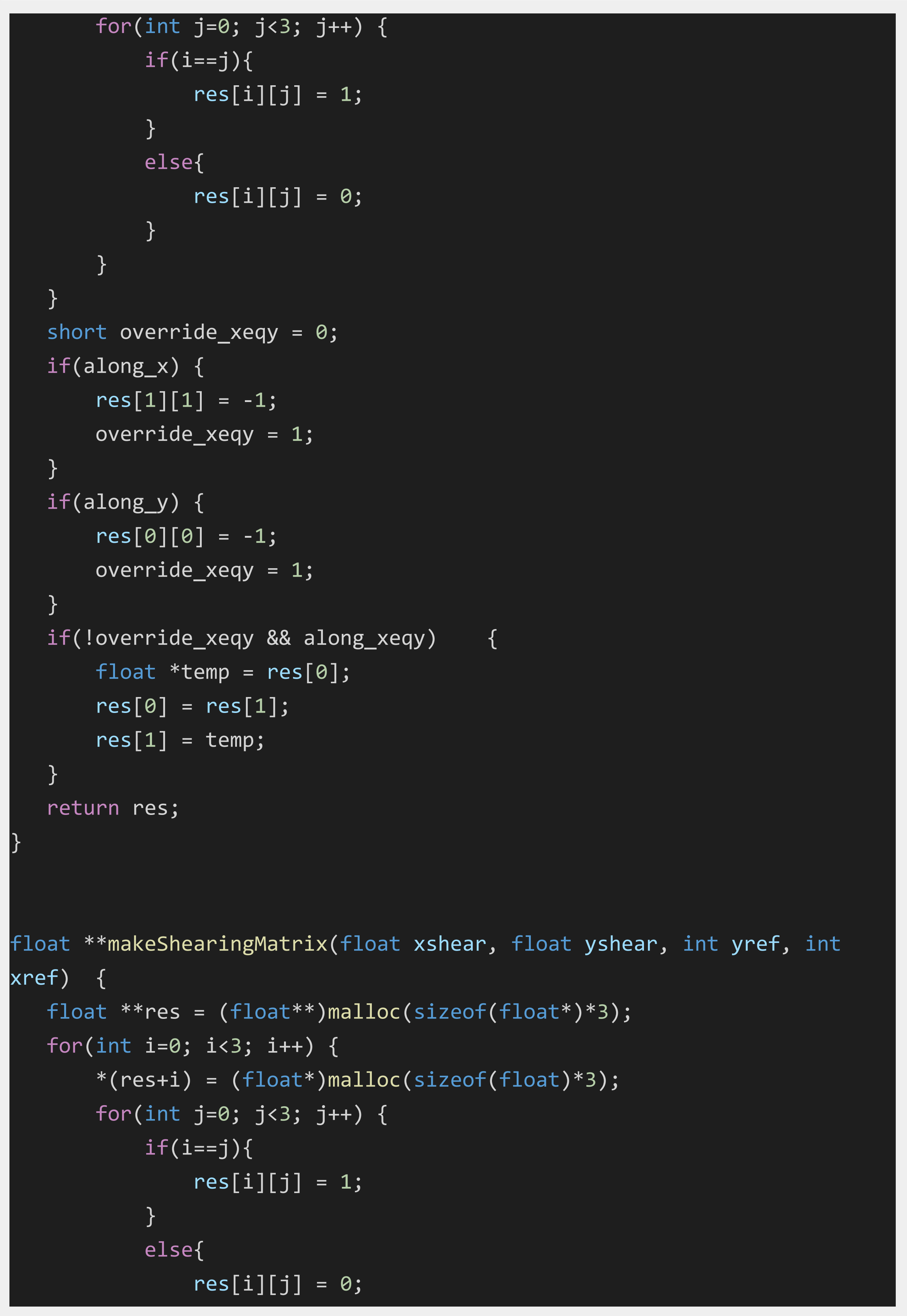


```
(float)(viewport.y_max-viewport.y_min)/(world.y_max-world.y_min);

    for(int i=0; i<3; i++)  {
        res_xs[i] = viewport.x_min + (xs[i] - world.x_min)*sx;
        res_ys[i] = viewport.y_min + (ys[i] - world.y_min)*sy;
    }
}


WindowConstraints offsetWindowConstraints(WindowConstraints window, int x_offset, int y_offset)    {
    // convenience function to offset window constraints to a different location
    // on the display window, for cleaner display
    window.x_min += x_offset;
    window.x_max += x_offset;
    window.y_min += y_offset;
    window.y_max += y_offset;
    return window;
}


void display_viewport_transform(){
    glClear(GL_COLOR_BUFFER_BIT);
    plotSingleDivisionLine();

    /* WORLD and VIEWPORT COORDINATES */
    WindowConstraints world_window = { 80, 220, 50, 220};
    plotWindow(world_window);
    WindowConstraints viewport_window = {50, 250, 80, 140};
    viewport_window = offsetWindowConstraints(viewport_window, -320, 0);
    plotWindow(viewport_window);

    markString("World Window", 150, 0, 0, 0);
    markString("Viewport Window", -210, 0, 0, 0);

    int xs[] = {110, 180, 140};
    int ys[] = {70, 100, 180};
```

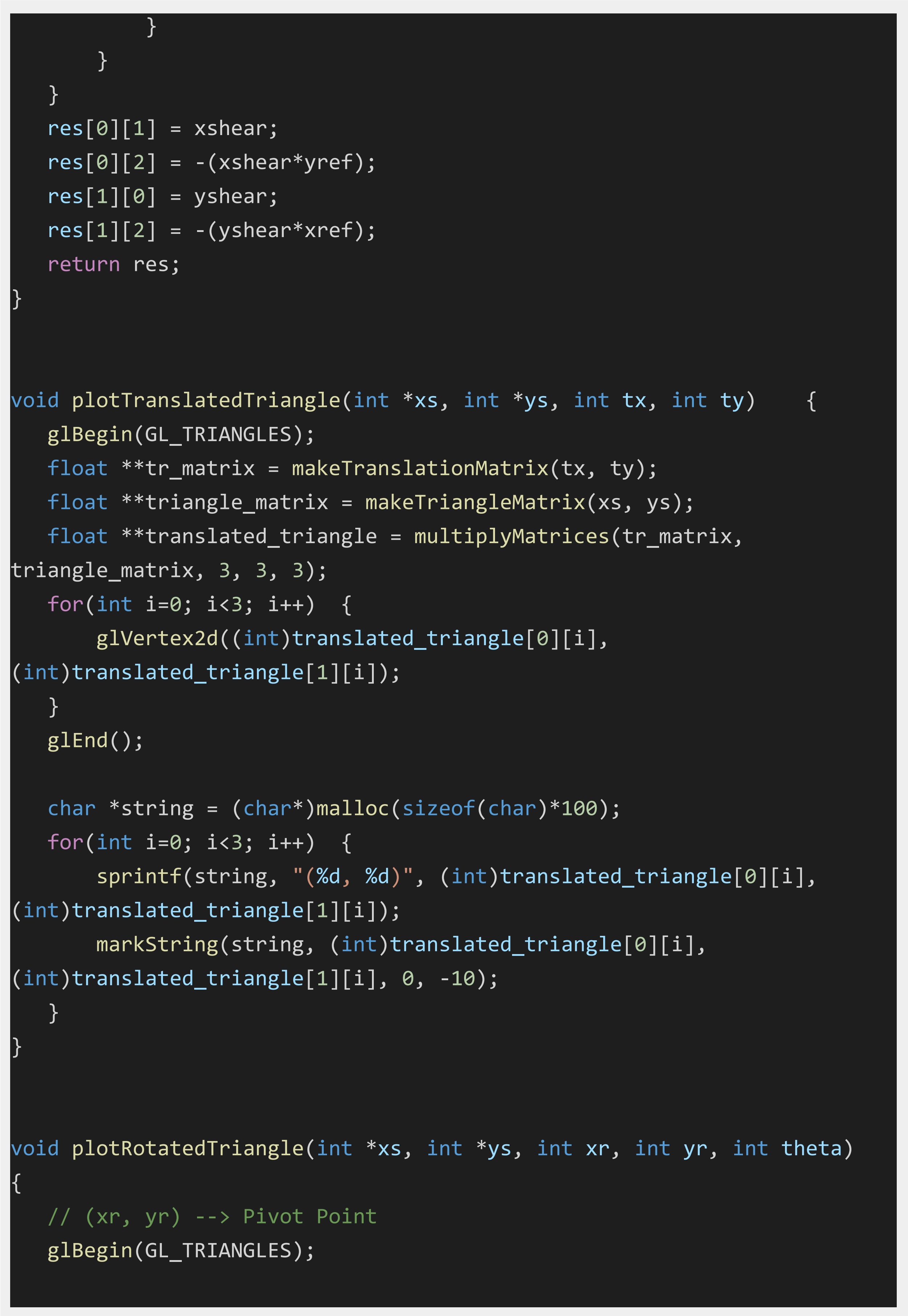
```
   int *viewport_xs = (int*)malloc(sizeof(int)*3);
   int *viewport_ys = (int*)malloc(sizeof(int)*3);
   getViewportCoordinates(xs, ys, world_window, viewport_window,
viewport_xs, viewport_ys);

   /* 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);
   for(int i=0; i<3; i++)  {
       printf("(%d, %d)\n", viewport_xs[i], viewport_ys[i]);
   }
   plotTriangle(viewport_xs, viewport_ys);

   glFlush();
}

/* ---------------------------------------------------------- */


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];
```



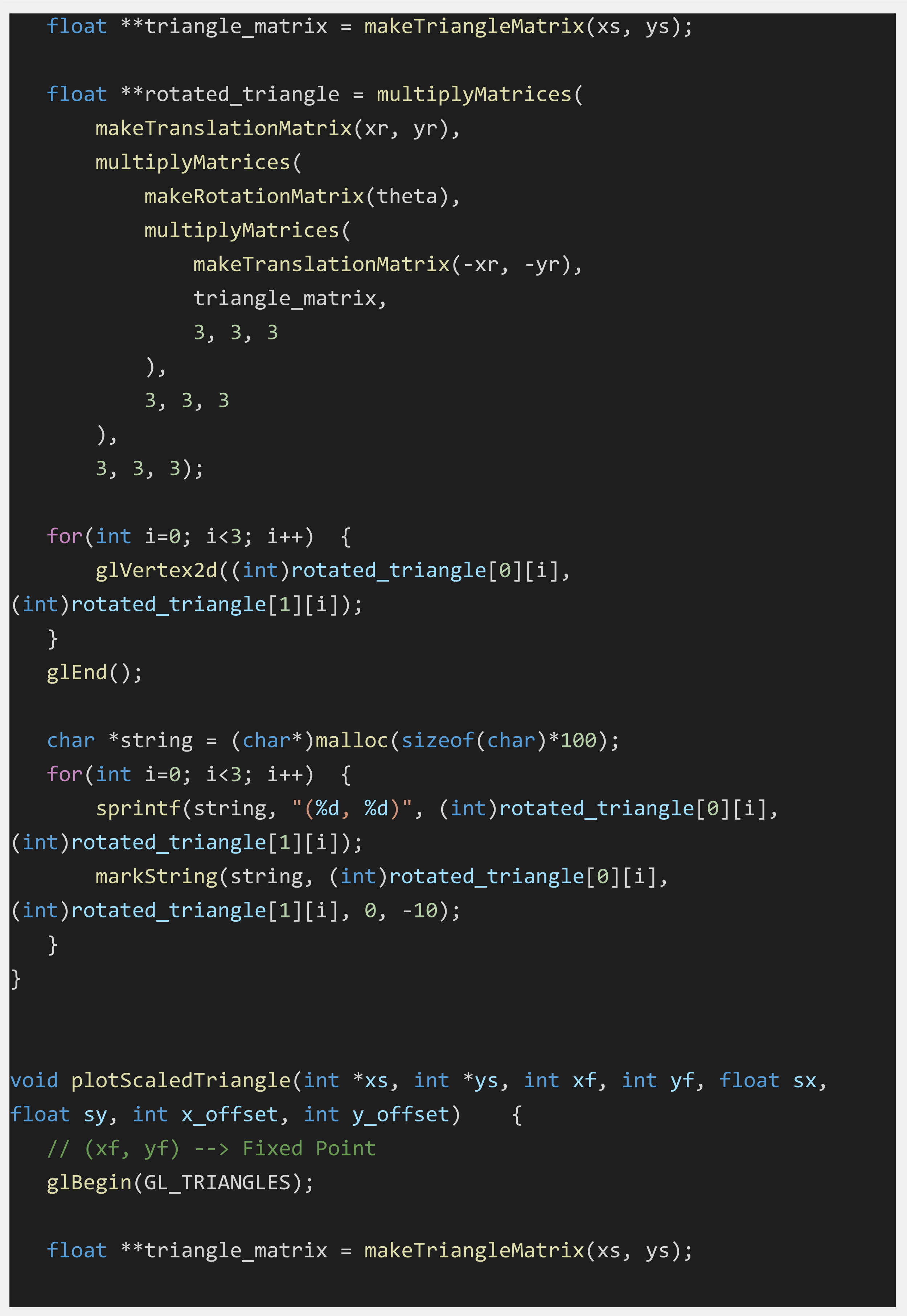
```
        res[1][i] = ys[i];
        res[2][i] = 1;
    }
    return res;
}


/* TRANSFORMATIONS ------------------------------------------------------
*/

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);
```



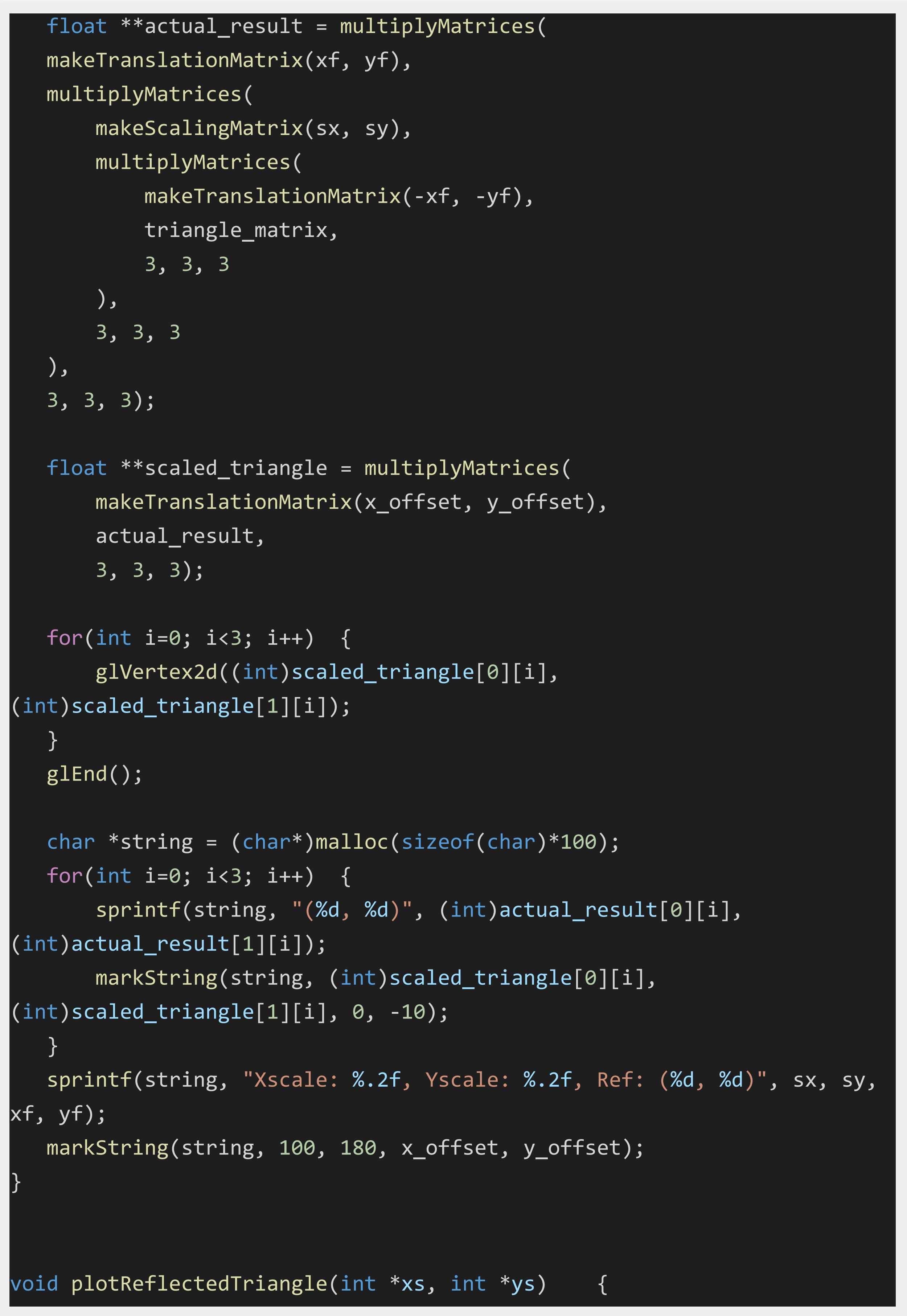
```
       for(int j=0; j<3; j++) {
           if(i==j){
               res[i][j] = 1;
           }
           else{
               res[i][j] = 0;
           }
       }
   }
   short override_xeqy = 0;
   if(along_x) {
       res[1][1] = -1;
       override_xeqy = 1;
   }
   if(along_y) {
       res[0][0]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



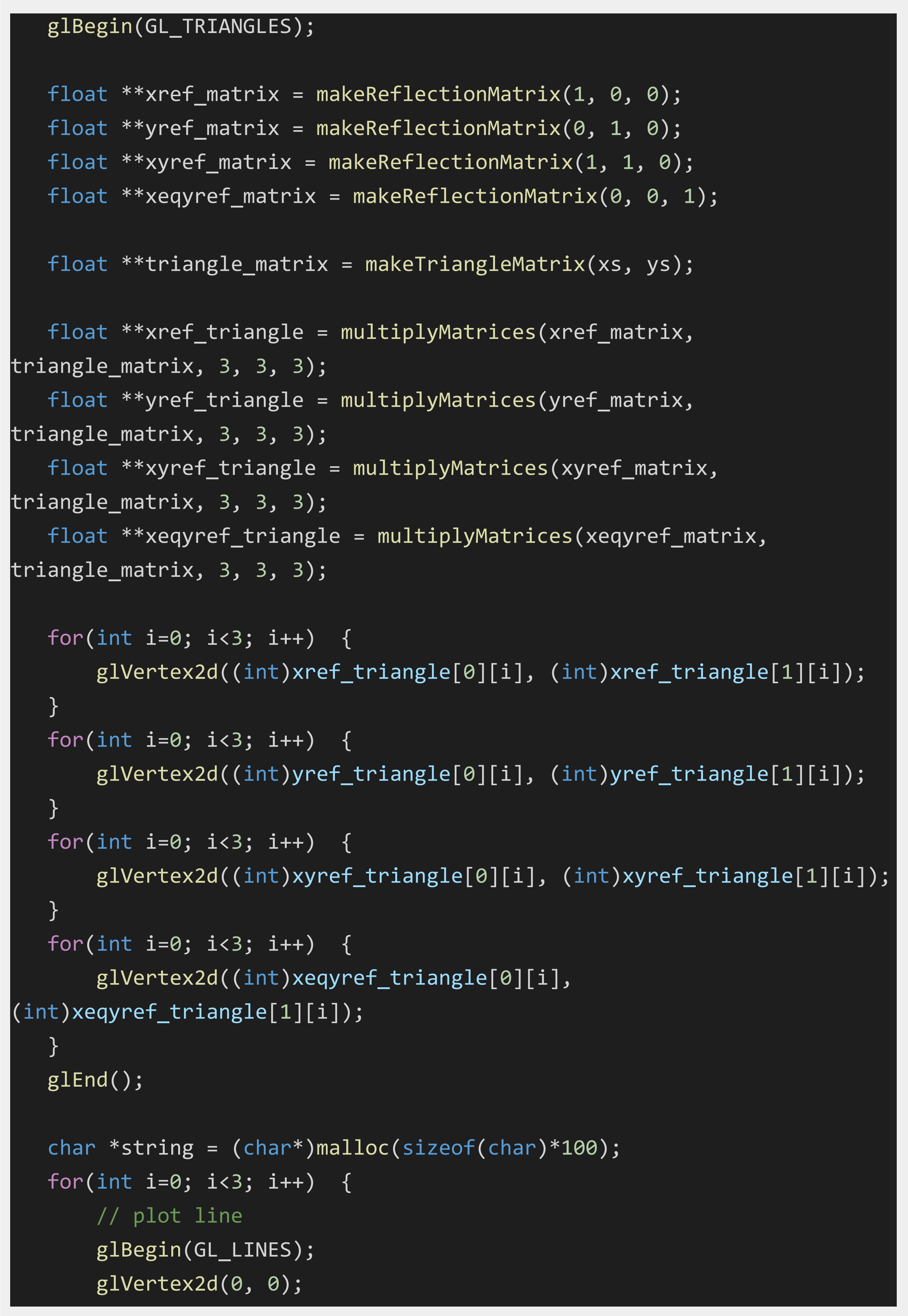
```
            }
        }
    }
    res[0][1] = xshear;
    res[0][2] = -(xshear*yref);
    res[1][0] = yshear;
    res[1][2] = -(yshear*xref);
    return res;
}


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);
```



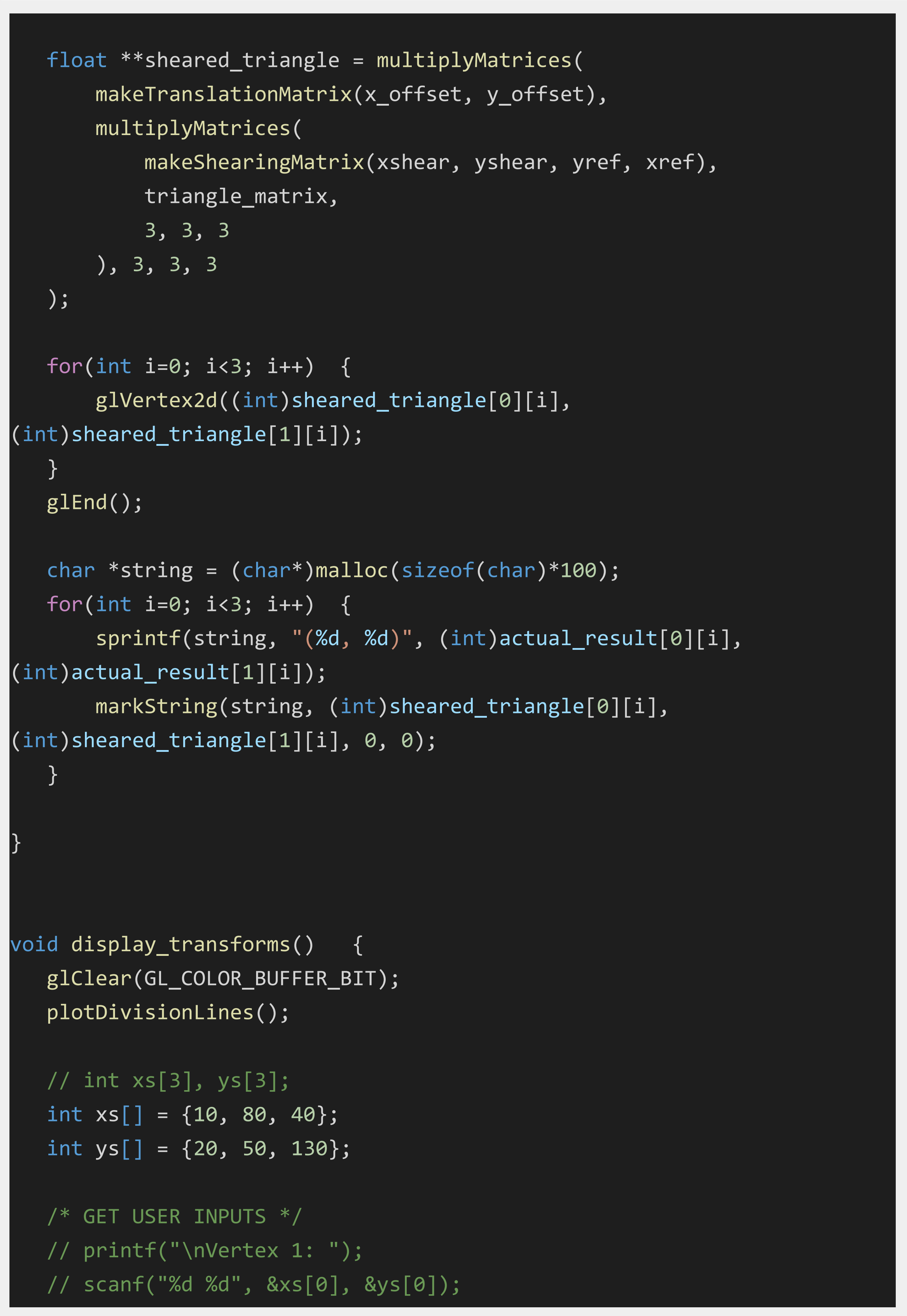
```
   float **triangle_matrix = makeTriangleMatrix(xs, ys);

   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);
```



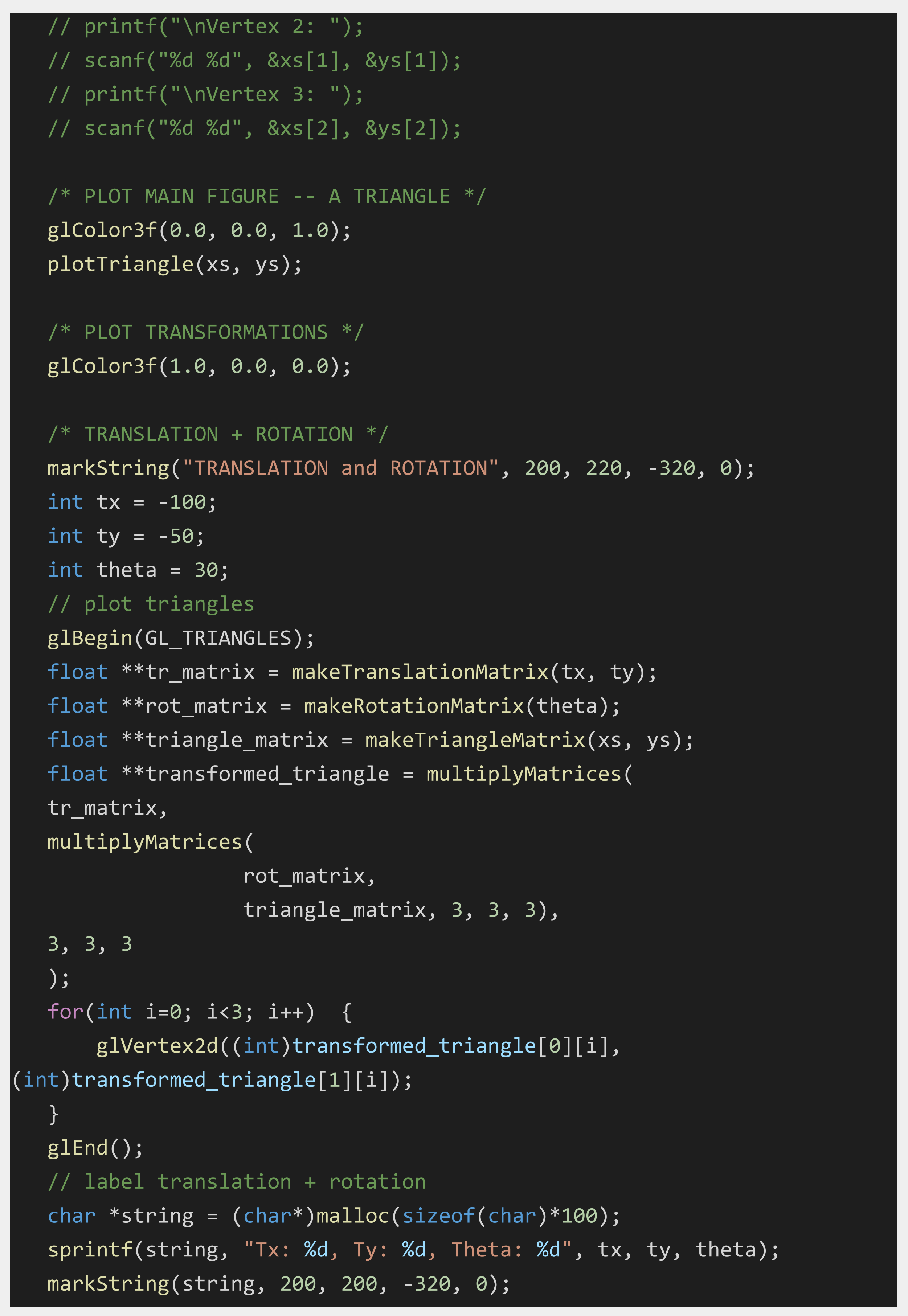
```
   float **actual_result = multiplyMatrices(
   makeTranslationMatrix(xf, yf),
   multiplyMatrices(
       makeScalingMatrix(sx, sy),
       multiplyMatrices(
           makeTranslationMatrix(-xf, -yf),
           triangle_matrix,
           3, 3, 3
       ),
       3, 3, 3
   ),
   3, 3, 3);

   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[] = {20, 50, 130};

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

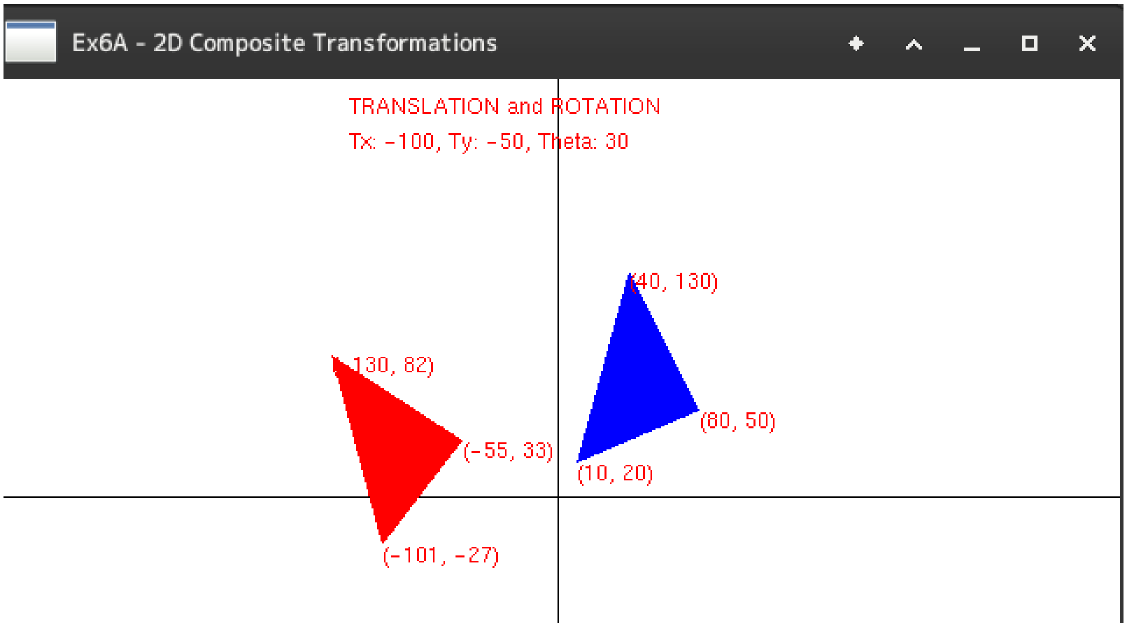


```
   // printf("\nVertex 2: ");
   // scanf("%d %d", &xs[1], &ys[1]);
   // printf("\nVertex 3: ");
   // scanf("%d %d", &xs[2], &ys[2]);

   /* 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 + ROTATION */
   markString("TRANSLATION and ROTATION", 200, 220, -320, 0);
   int tx = -100;
   int ty = -50;
   int theta = 30;
   // plot triangles
   glBegin(GL_TRIANGLES);
   float **tr_matrix = makeTranslationMatrix(tx, ty);
   float **rot_matrix = makeRotationMatrix(theta);
   float **triangle_matrix = makeTriangleMatrix(xs, ys);
   float **transformed_triangle = multiplyMatrices(
   tr_matrix,
   multiplyMatrices(
                   rot_matrix,
                   triangle_matrix, 3, 3, 3),
   3, 3, 3
   );
   for(int i=0; i<3; i++)  {
       glVertex2d((int)transformed_triangle[0][i],
(int)transformed_triangle[1][i]);
   }
   glEnd();
   // label translation + rotation
   char *string = (char*)malloc(sizeof(char)*100);
   sprintf(string, "Tx: %d, Ty: %d, Theta: %d", tx, ty, theta);
   markString(string, 200, 200, -320, 0);
   markString(string, 200, 200, -320, 0);
   for(int i=0; i<3; i++)  {
       sprintf(string, "(%d, %d)", (int)triangle_matrix[0][i],
(int)triangle_matrix[1][i]);
       markString(string, (int)triangle_matrix[0][i],
(int)triangle_matrix[1][i], 0, -10);
       sprintf(string, "(%d, %d)", (int)transformed_triangle[0][i],
(int)transformed_triangle[1][i]);
       markString(string, (int)transformed_triangle[0][i],
(int)transformed_triangle[1][i], 0, -10);
   }


   /* SCALING + REFLECTION */
   // markString("SCALING and REFLECTION", 200, 220, -320, 0);
   // int x0 = -100;
   // int y0 = -100;
   // float sx = 0.75;
   // float sy = 1.25;
   // // plot triangles
   // glBegin(GL_TRIANGLES);
   // float **scale_matrix = makeScalingMatrix(sx, sy);
   // float **refxy_matrix = makeReflectionMatrix(1, 1, 0);
   // float **triangle_matrix = makeTriangleMatrix(xs, ys);
   // float **transformed_triangle = multiplyMatrices(
   //     scale_matrix,
   //     multiplyMatrices(
   //                         refxy_matrix,
   //                         triangle_matrix, 3, 3, 3),
   //     3, 3, 3
   // );
   // for(int i=0; i<3; i++)  {
   //     glVertex2d((int)transformed_triangle[0][i],
(int)transformed_triangle[1][i]);
   // }
   // glEnd();
   // // label translation + rotation
   // char *string = (char*)malloc(sizeof(char)*100);
   // sprintf(string, "X0: %d, Y0: %d, Sx: %.2f Sy: %.2f", x0, y0, sx, sy);
   // markString(string, 200, 200, -320, 0);
   // markString(string, 200, 200, -320, 0);
   // for(int i=0; i<3; i++)  {
   //     sprintf(string, "(%d, %d)", (int)triangle_matrix[0][i], (int)triangle_matrix[1][i]);
   //     markString(string, (int)triangle_matrix[0][i], (int)triangle_matrix[1][i], 0, -10);
   //     sprintf(string, "(%d, %d)", (int)transformed_triangle[0][i], (int)transformed_triangle[1][i]);
   //     markString(string, (int)transformed_triangle[0][i], (int)transformed_triangle[1][i], -60, -10);
   // }


   /* SHEARING and TRANSLATION*/
   // markString("SHEARING and TRANSLATION", 200, 220, -320, 0);
   // int yref = -1;
   // int xref = -2;
   // float xshear = 0.2;
   // float yshear = 0.6;
   // int tx = -200;
   // int ty = 0;
   // glBegin(GL_TRIANGLES);
   // float **shear_matrix = makeShearingMatrix(xshear, yshear, xref, yref);
   // float **tr_matrix = makeTranslationMatrix(tx, ty);
   // float **triangle_matrix = makeTriangleMatrix(xs, ys);
   // float **transformed_triangle = multiplyMatrices(
   //     shear_matrix,
   //     multiplyMatrices(
   //         tr_matrix,
   //         triangle_matrix,
   //         3, 3, 3),
   //     3, 3, 3
   // );
   // displayMatrix(transformed_triangle, 3, 3);
   // for(int i=0; i<3; i++)  {
   //     glVertex2d((int)transformed_triangle[0][i],
(int)transformed_triangle[1][i]);
   // }
   // glEnd();
   // // label translation + rotation
   // char *string = (char*)malloc(sizeof(char)*100);
   // sprintf(string, "Tx: %d, Ty: %d, Yref: %d, Xref: %d, Xshear: %.2f
Yshear: %.2f", tx, ty, yref, xref, xshear, yshear);
   // markString(string, 200, 200, -320, 0);
   // for(int i=0; i<3; i++)  {
   //     sprintf(string, "(%d, %d)", (int)triangle_matrix[0][i],
(int)triangle_matrix[1][i]);
   //     markString(string, (int)triangle_matrix[0][i],
(int)triangle_matrix[1][i], 0, -10);
   //     sprintf(string, "(%d, %d)", (int)transformed_triangle[0][i],
(int)transformed_triangle[1][i]);
   //     markString(string, (int)transformed_triangle[0][i],
(int)transformed_triangle[1][i], 0, -10);
   // }

   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("Ex6A - 2D Composite Transformations");
    // glutDisplayFunc(display_transforms);
    glutCreateWindow("Ex6B - World to Viewport");
    glutDisplayFunc(display_viewport_transform);

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

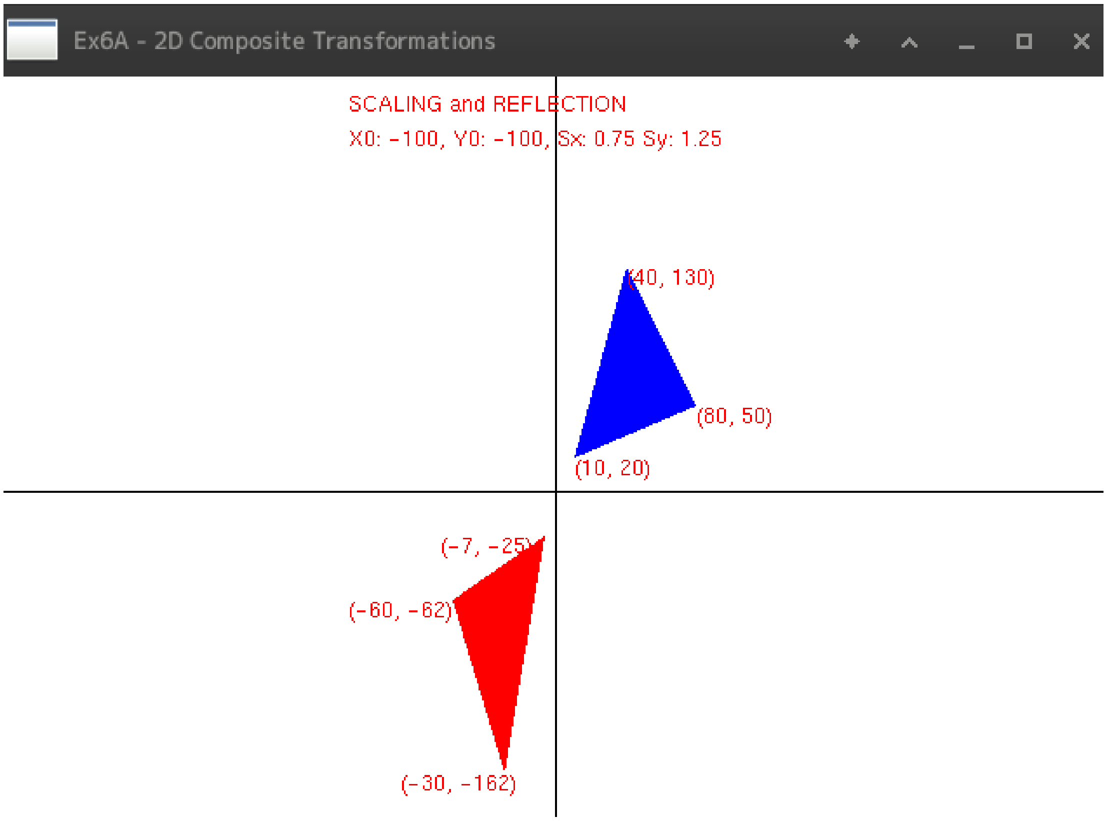
## Sample Output

- **2D Translation and Rotation**



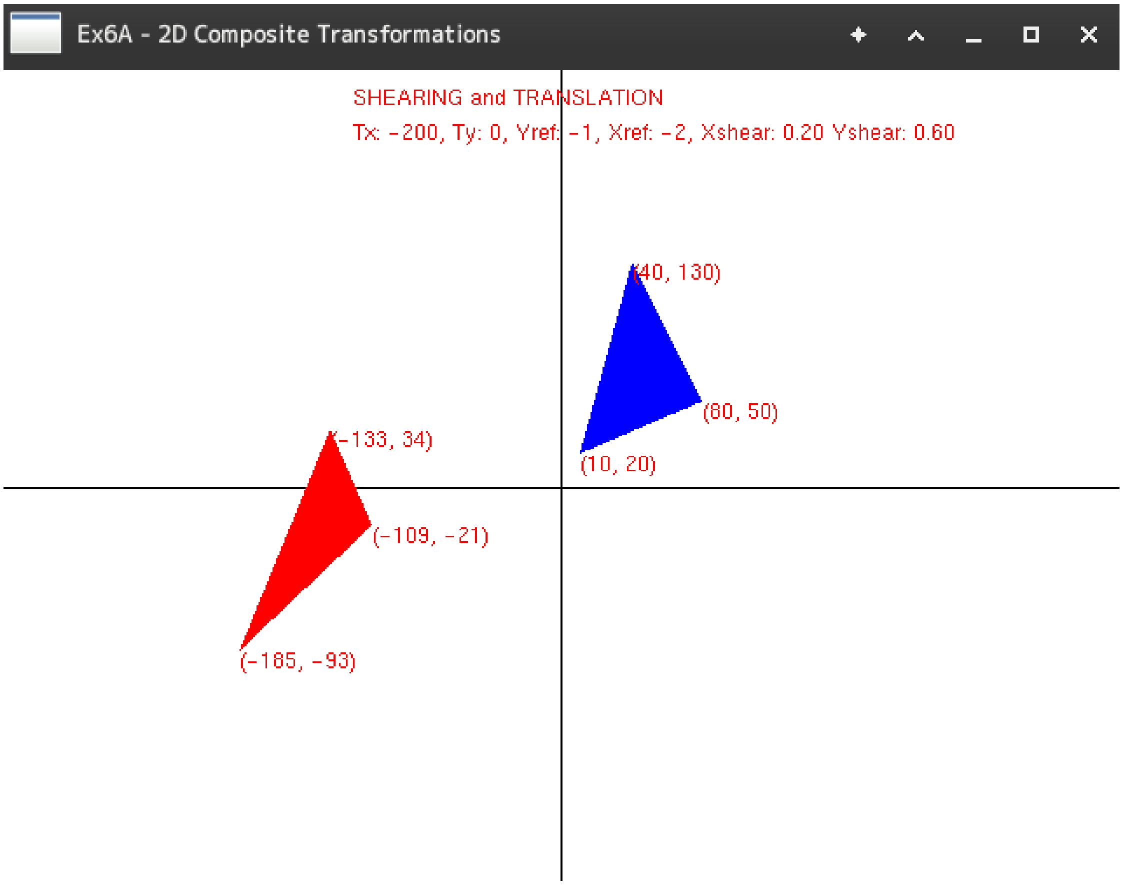
```
TRANSLATION and ROTATION
Tx: -100, Ty: -50, Theta: 30
```

- **2D Scaling and Reflection**



```
SCALING and REFLECTION
X0: -100, Y0: -100, Sx: 0.75 Sy: 1.25
```

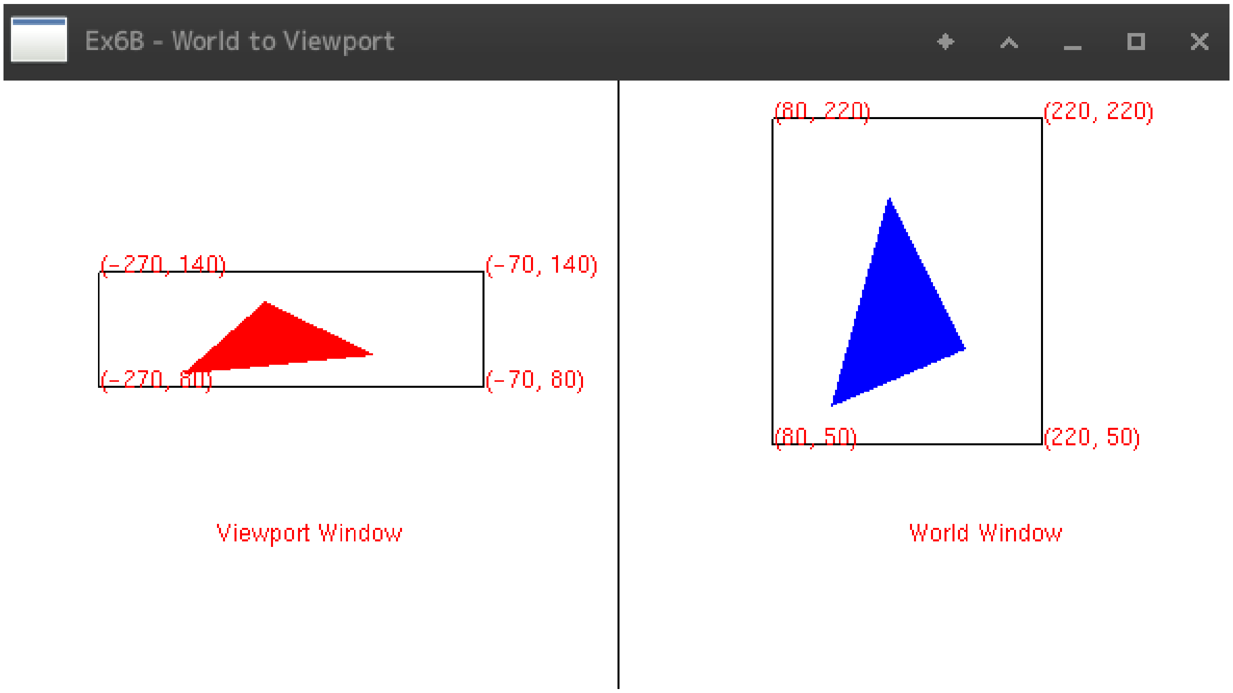
- **2D Shearing and Translation**



SHEARING and TRANSLATION

Tx: -200, Ty: 0, Yref: -1, Xref: -2, Xshear: 0.20 Yshear: 0.60

- **Window to Viewport Transformation**



## 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 and their combinations — reflection, rotation, scaling, shearing and translation.
2. The use and application of homogeneous coordinates when rendering transformations using matrices.
3. The working of the world to viewport transformation for windowing.
4. General understanding of the OpenGL framework and its APIs.