#include <GL/glut.h>

#include <stdlib.h>

#include <math.h>

/\* Set initial display-window size. \*/

GLsizei winWidth = 600, winHeight = 600;

/\* Set range for world coordinates. \*/

GLfloat xwcMin = 0.0, xwcMax = 225.0;

GLfloat ywcMin = 0.0, ywcMax = 225.0;

class wcPt2D {

public:

GLfloat x, y;

};

typedef GLfloat Matrix3x3 [3][3];

Matrix3x3 matComposite;

const GLdouble pi = 3.14159;

void init (void)

{

/\* Set color of display window to white. \*/

glClearColor (1.0, 1.0, 1.0, 0.0);

}

/\* Construct the 3 x 3 identity matrix. \*/

void matrix3x3SetIdentity (Matrix3x3 matIdent3x3)

{

GLint row, col;

for (row = 0; row < 3; row++)

for (col = 0; col < 3; col++)

matIdent3x3 [row][col] = (row == col);

}

void matrix3x3PreMultiply (Matrix3x3 m1, Matrix3x3 m2)

{

GLint row, col;

Matrix3x3 matTemp;

for (row = 0; row < 3; row++)

for (col = 0; col < 3 ; col++)

matTemp [row][col] = m1 [row][0] \* m2 [0][col] + m1 [row][1] \*

m2 [1][col] + m1 [row][2] \* m2 [2][col];

for (row = 0; row < 3; row++)

for (col = 0; col < 3; col++)

m2 [row][col] = matTemp [row][col];

}

void translate2D (GLfloat tx, GLfloat ty)

{

Matrix3x3 matTransl;

/\* Initialize translation matrix to identity. \*/

matrix3x3SetIdentity (matTransl);

matTransl [0][2] = tx;

matTransl [1][2] = ty;

/\* Concatenate matTransl with the composite matrix. \*/

matrix3x3PreMultiply (matTransl, matComposite);

}

void rotate2D (wcPt2D pivotPt, GLfloat theta)

{

Matrix3x3 matRot;

/\* Initialize rotation matrix to identity. \*/

matrix3x3SetIdentity (matRot);

matRot [0][0] = cos (theta);

matRot [0][1] = -sin (theta);

matRot [0][2] = pivotPt.x \* (1 - cos (theta)) +

pivotPt.y \* sin (theta);

matRot [1][0] = sin (theta);

matRot [1][1] = cos (theta);

matRot [1][2] = pivotPt.y \* (1 - cos (theta)) -

pivotPt.x \* sin (theta);

/\* Concatenate matRot with the composite matrix. \*/

matrix3x3PreMultiply (matRot, matComposite);

}

void scale2D (GLfloat sx, GLfloat sy, wcPt2D fixedPt)

{

Matrix3x3 matScale;

/\* Initialize scaling matrix to identity. \*/

matrix3x3SetIdentity (matScale);

matScale [0][0] = sx;

matScale [0][2] = (1 - sx) \* fixedPt.x;

matScale [1][1] = sy;

matScale [1][2] = (1 - sy) \* fixedPt.y;

/\* Concatenate matScale with the composite matrix. \*/

matrix3x3PreMultiply (matScale, matComposite);

}

/\* Using the composite matrix, calculate transformed coordinates. \*/

void transformVerts2D (GLint nVerts, wcPt2D \* verts)

{

GLint k;

GLfloat temp;

for (k = 0; k < nVerts; k++) {

temp = matComposite [0][0] \* verts [k].x + matComposite [0][1] \*

verts [k].y + matComposite [0][2];

verts [k].y = matComposite [1][0] \* verts [k].x + matComposite [1][1] \*

verts [k].y + matComposite [1][2];

verts [k].x = temp;

}

}

void triangle (wcPt2D \*verts)

{

GLint k;

glBegin (GL\_TRIANGLES);

for (k = 0; k < 3; k++)

glVertex2f (verts [k].x, verts [k].y);

glEnd ( );

}

void displayFcn (void)

{

/\* Define initial position for triangle. \*/

GLint nVerts = 3;

wcPt2D verts [3] = { {50.0, 25.0}, {150.0, 25.0}, {100.0, 100.0} };

/\* Calculate position of triangle centroid. \*/

wcPt2D centroidPt;

GLint k, xSum = 0, ySum = 0;

for (k = 0; k < nVerts; k++) {

xSum += verts [k].x;

ySum += verts [k].y;

}

centroidPt.x = GLfloat (xSum) / GLfloat (nVerts);

centroidPt.y = GLfloat (ySum) / GLfloat (nVerts);

/\* Set geometric transformation parameters. \*/

wcPt2D pivPt,fixedPt;

pivPt = centroidPt;

fixedPt = centroidPt;

GLfloat tx = 0.0, ty = 100.0;

GLfloat sx = 0.5, sy = 0.5;

GLdouble theta = pi/2.0;

glClear (GL\_COLOR\_BUFFER\_BIT); // Clear display window.

glColor3f (0.0, 0.0, 1.0); // Set initial fill color to blue.

triangle (verts); // Display blue triangle.

/\* Initialize composite matrix to identity. \*/

matrix3x3SetIdentity (matComposite);

/\* Construct composite matrix for transformation sequence. \*/

scale2D (sx, sy, fixedPt); // First transformation: Scale.

rotate2D (pivPt, theta); // Second transformation: Rotate

translate2D (tx, ty); // Final transformation: Translate.

/\* Apply composite matrix to triangle vertices. \*/

transformVerts2D (nVerts, verts);

glColor3f (1.0, 0.0, 0.0); // Set color for transformed triangle.

triangle (verts);

glFlush ( );

}

void winReshapeFcn (GLint newWidth, GLint newHeight)

{

glMatrixMode (GL\_PROJECTION);

glLoadIdentity ( );

gluOrtho2D (xwcMin, xwcMax, ywcMin, ywcMax);

glClear (GL\_COLOR\_BUFFER\_BIT);

}

int main (int argc, char \*\* argv)

{

glutInit (&argc, argv);

glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition (50, 50);

glutInitWindowSize (winWidth, winHeight);

glutCreateWindow ("Geometric Transformation Sequence");

init ( );

glutDisplayFunc (displayFcn);

glutReshapeFunc (winReshapeFcn);

glutMainLoop ( );

return 0;

}