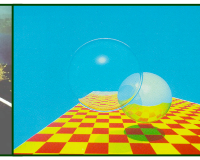


# Chap. 3: Geometric Transformations

# Summary

- Motivation.
- Euclidean transformations: translation and rotation.
- Euclidean geometry.
- Homogeneous coordinates.
- Affine transformations: translation, rotation, scaling, and shearing.
- Matrix representation of affine transformations.
- Composition of geometric transformations in 2D and 3D.
- Affine transformations in OpenGL.
- OpenGL matrix operations and arbitrary geometric transformations.
- Examples in OpenGL.

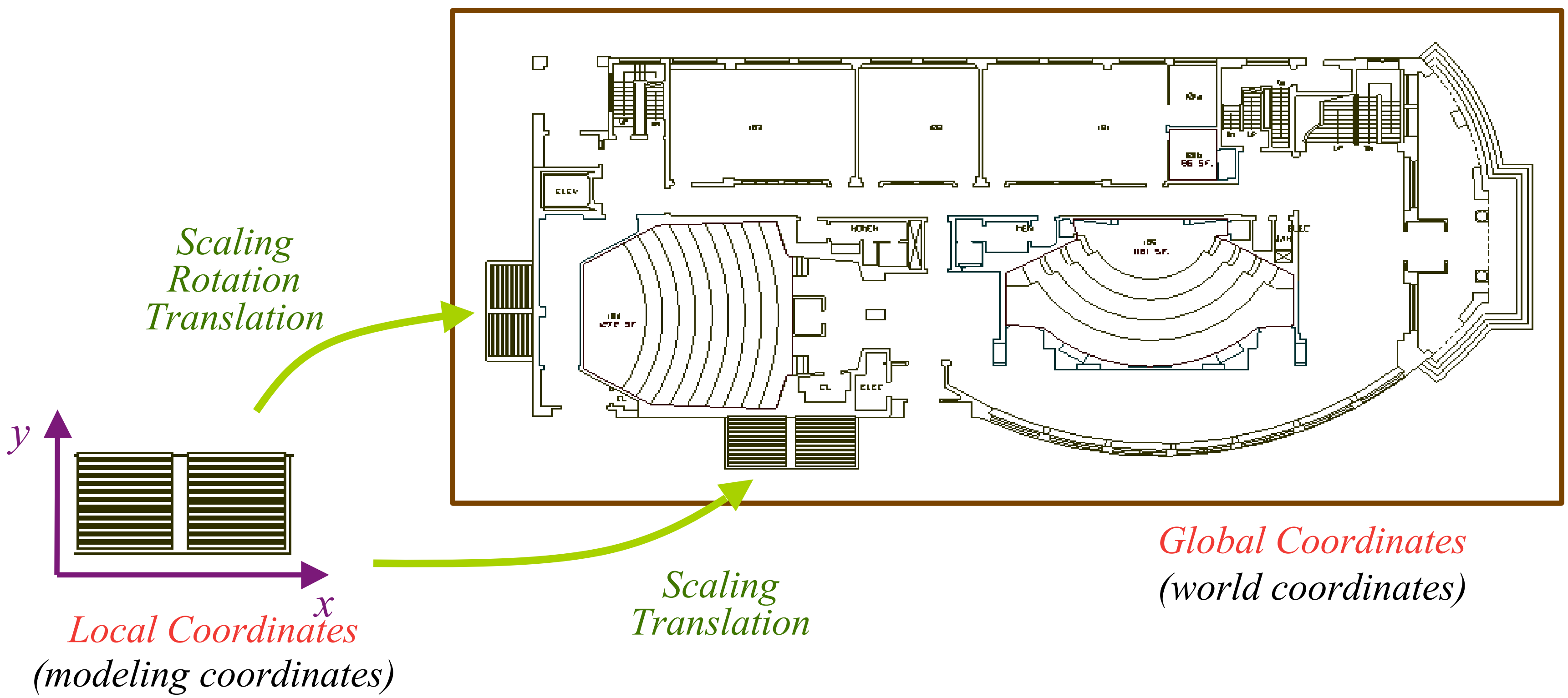
# Motivation

- Geometric transformations
  - Translation, rotation, reflection
  - Scaling, shearing
  - Orthogonal projection, perspective projection
- Why are geometric transformations necessary?
  - for positioning geometric objects in 2D and 3D.
  - for modelling geometric objects in 2D and 3D
  - For viewing geometric objects in 2D and 3D.

# Motivation (cont.): modelling objects in 2D

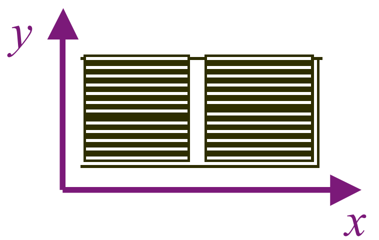
- Geometric transformations can specify object modelling operations
  - They allow us to define an object through its local coordinate system (modeling coordinates)
  - They allow us to define an object several times in a scene with a global coordinate system (world coordinates)

# Motivation (cont.): modelling objects in 2D

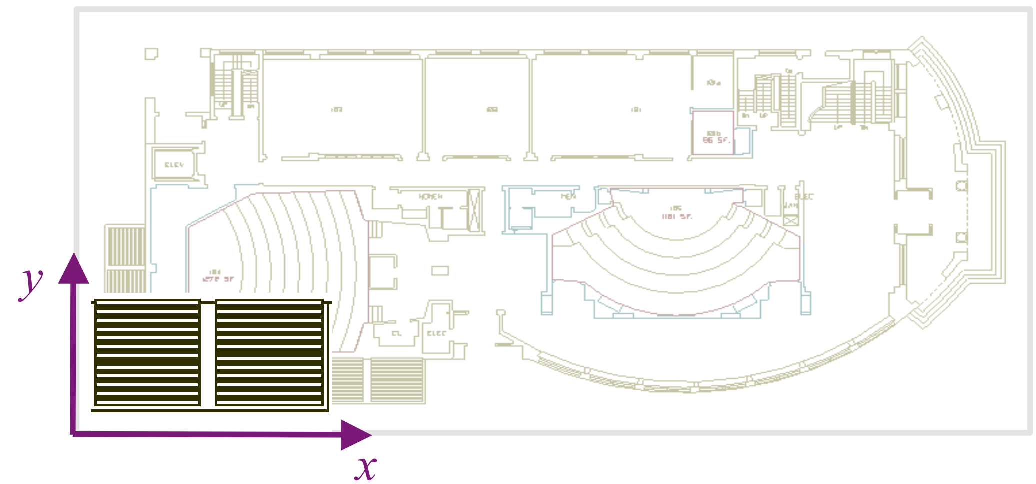


# Motivation (cont.): modelling objects in 2D

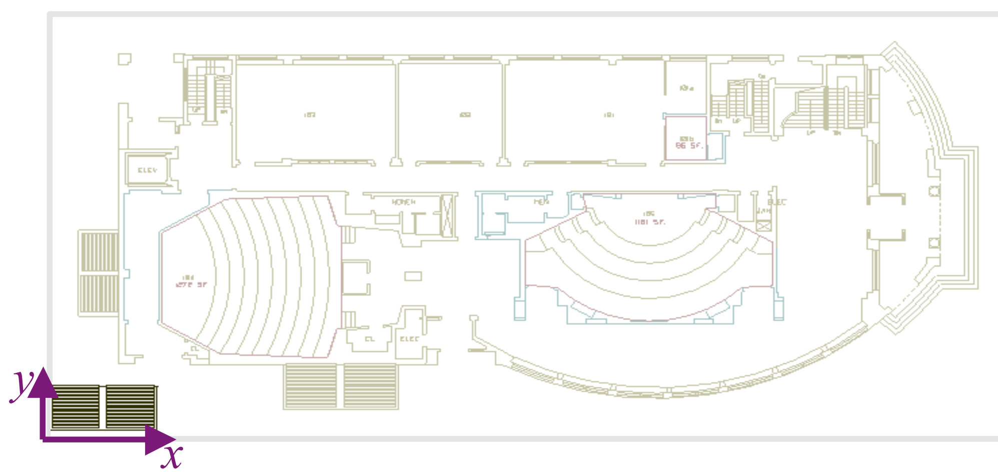
*Local Coordinates*




*Global Coordinates*



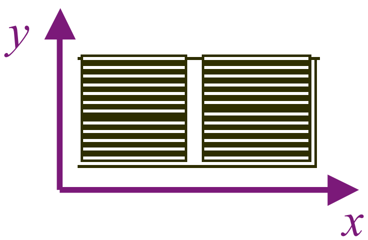

*Positioning*



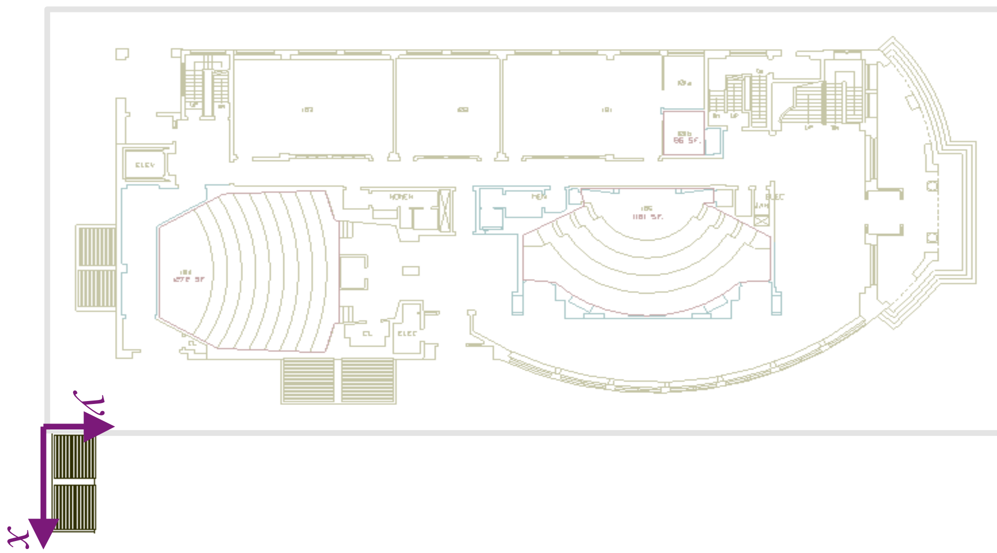

*Scaling*

# Motivação (cont.): modelação de objectos em 2D

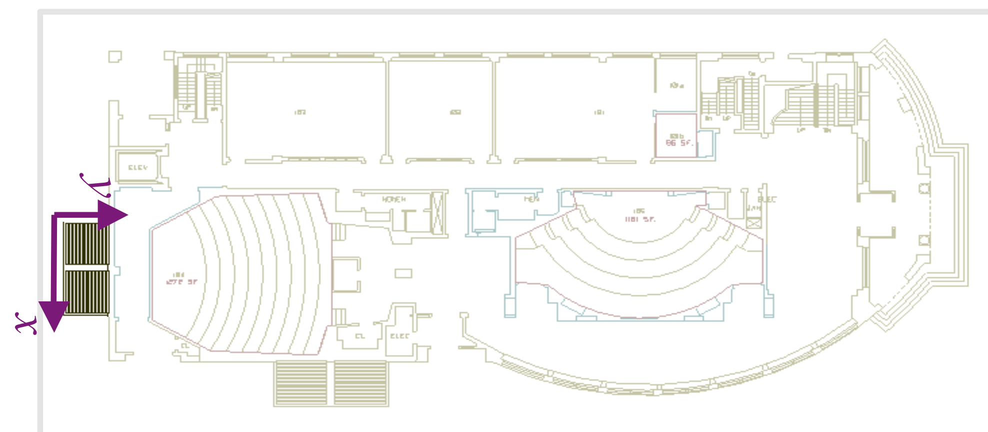
*Local Coordinates*




*Global Coordinates*




*Rotation*

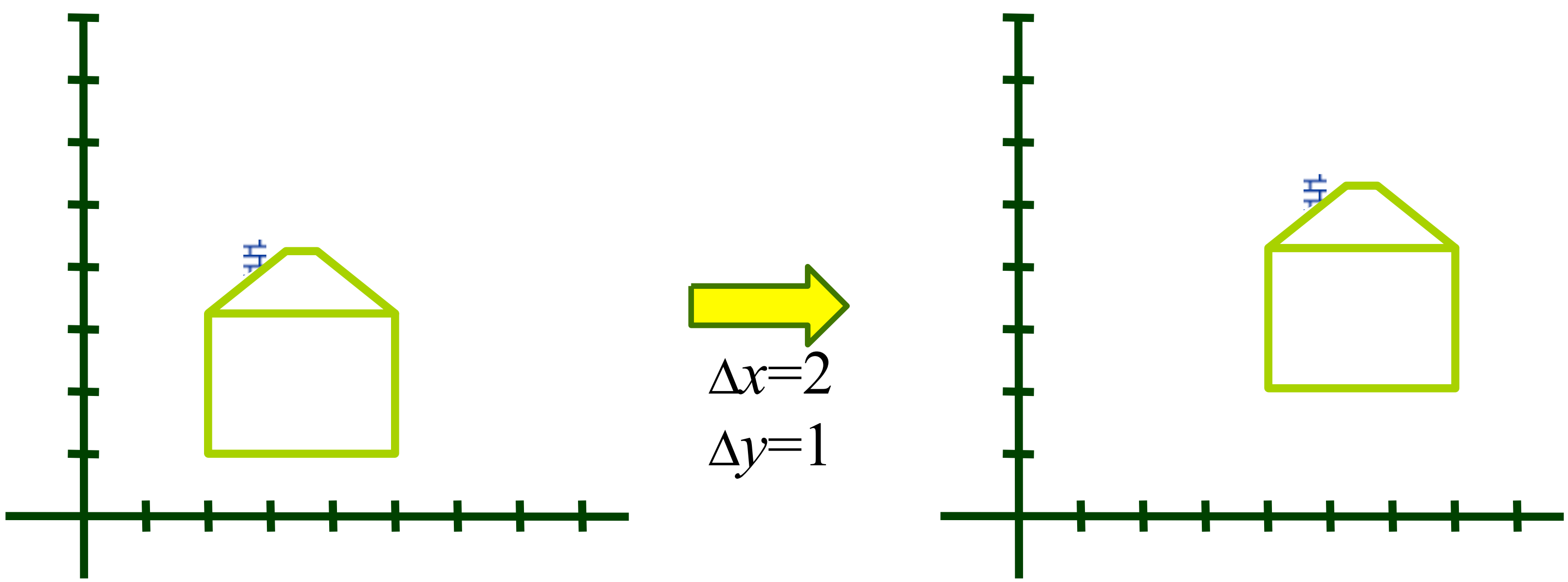



*Translation*

# Translation 2D

$$\begin{cases} x' = x + \Delta x \\ y' = y + \Delta y \end{cases}$$

*Translating* a point $(x, y)$ means to move it by $(\Delta x, \Delta y)$.



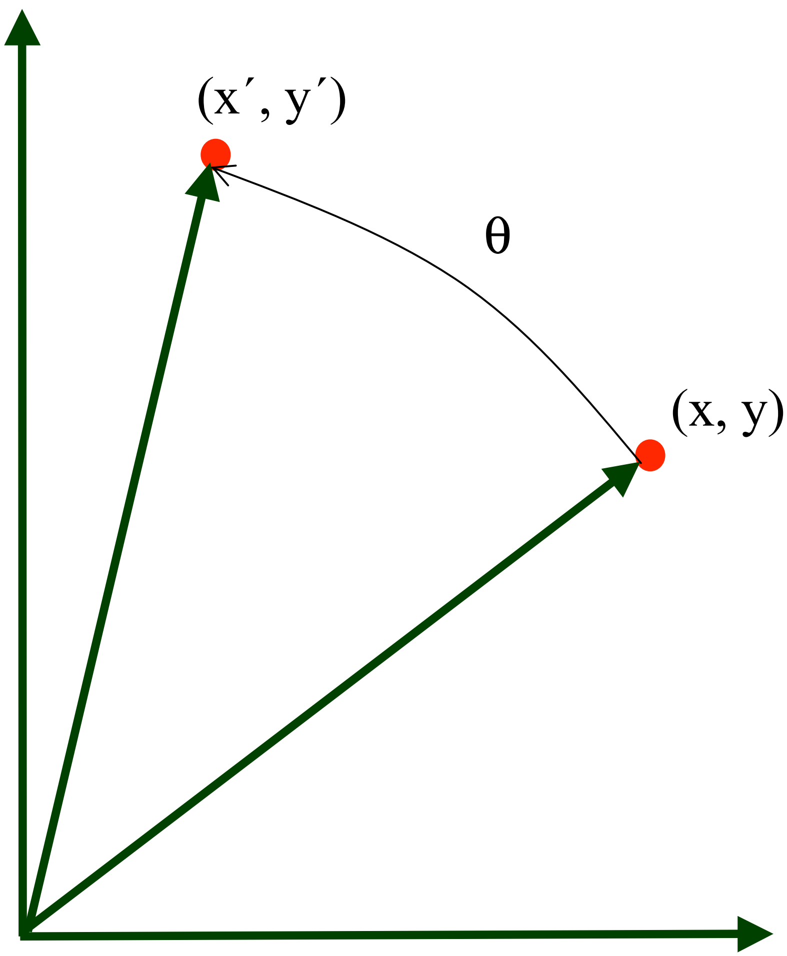
http://encyclopedia.laborlawtalk.com/Linear_combination

# Translation 2D: matrix representation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

- x' is not a linear combination of x and y
- y' is not a linear combination of x and y

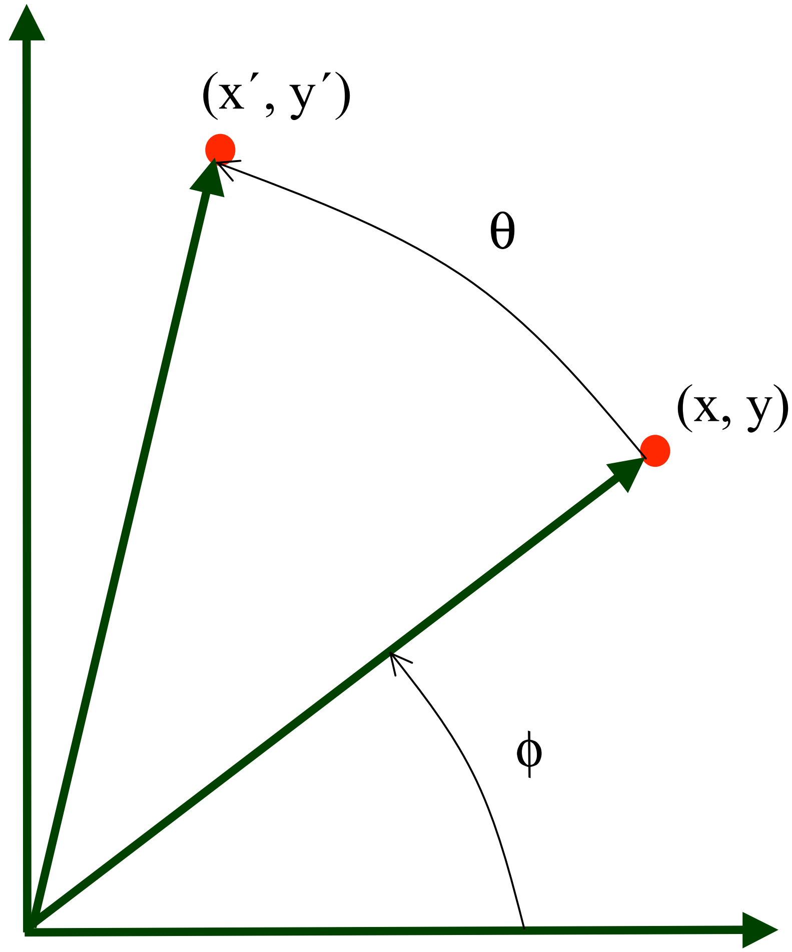
# Rotation 2D




$$\begin{cases} x' = x\cos\theta - y\sin\theta \\ y' = x\sin\theta + y\cos\theta \end{cases}$$

*Rotating* a point P=(x,y) through an angle θ about the origin O(0,0) counterclockwise means to determine another point Q=(x′,y′) on the circle centred at O such that θ=∠POQ.

# Rotation 2D: equations




$$\begin{cases} x = r\cos\phi \\ y = r\sin\phi \end{cases}$$

$$\begin{cases} x' = r\cos(\phi+\theta) \\ y' = r\sin(\phi+\theta) \end{cases}$$

Expanding the expressions of x´ and y´, we have:

$$\begin{cases} x' = r\cos\phi\cos\theta - r\sin\phi\sin\theta \\ y' = r\cos\phi\sin\theta + r\sin\phi\cos\theta \end{cases}$$

Replacing r cos(f) and r sin(f) by x and y in the previous equations, we get:

$$\begin{cases} x' = x\cos\theta - y\sin\theta \\ y' = x\sin\theta + y\cos\theta \end{cases}$$

# Rotation 2D: matrix representation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

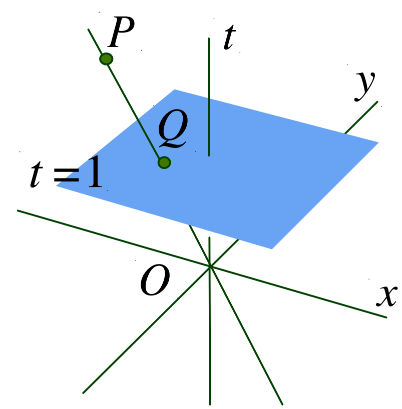
- Although sin(θ) and cos(θ) are not linear functions of θ,
  - x' is a linear combination of x and y
  - y' is a linear combination of x and y

# Fundamental problem of geometric transformations

- **Translation is not a linear transformation of x and y**.
- Consequence: we are not allowed to effect a sequence of transformations (tranlations and rotations) through a product of matrices 2x2.
- But, we can always produce *k* rotations by computing the product of *k* rotation matrices.
- SOLUTION: **homogeneous coordinates**!

# Homogeneous coordinates

- A triple of real numbers (x,y,t), with t≠0, is a set of homogeneous coordinates for the point P with cartesian coordinates (x/t,y/t).
- Thus, the same point has many sets of homogeneous coordinates. So, (x,y,t) e (x',y',t') represent the same point if and only if there is some real scalar α such that x'= αx, y'= αy and t'= αt.
- So, if P has cartesian coordinates (x,y), one set of homogeneous coordinates for P is (x,y,1), being this set the most used in computer graphics.



# Translation and Rotation in 2D: homogeneous coordinates

*Translation*

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

*Rotation*

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

# Euclidean metric geometry

- Set of geometric transformations: **translations** and **rotations** (also called isometries).
- By using homogeneous coordinates, these transformations can be represented through matrices 3x3. This enables the use of product operator for matrices to evaluate a sequence of translations and rotations
- The set of isometries **I**(n) in $\mathbf{R}^n$ and the concatenation operator • form a group **GI**(n)=(**I**(n),•).
- Fundamental metric invariant:
  - distance between points.
- Other metric invariants:
  - angles
  - lengths
  - areas
  - volumes
- 2-dimensional Euclidean geometry : ($\mathbf{R}^2$,**GI**(2))

# Definition of group: remind

A set C and an operation $\circ$ form a group $(C, \circ)$ if:

- Closure Axiom. For all $c_1, c_2 \in C$, $c_1 \circ c_2 \in C$.
- Identity Axiom. There exists an identity element $i \in C$ such that $c \circ i = c = i \circ c$, for any $c \in C$.
- Inverse Element Axiom. For any $c \in C$, there exists an inverse elemen $c^{-1} \in C$ such that

$$c \circ c^{-1} = i = c^{-1} \circ c$$

- Associativity Axiom. For all $c_1, c_2, c_3 \in C$,

$$c_1 \circ (c_2 \circ c_3) = (c_1 \circ c_2) \circ c_3$$

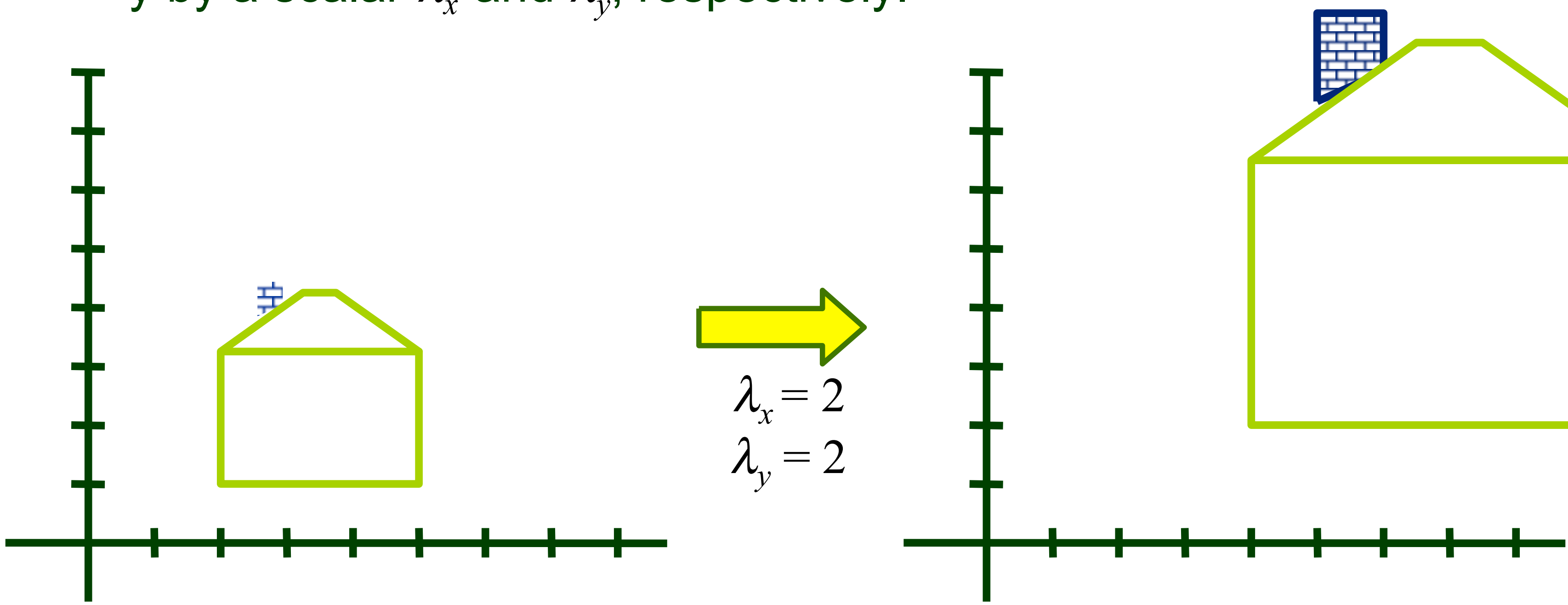
# Geometria afim

- It generalizes the Euclidean geometry.
- Set of affine transformations (or affinities): translation, rotation, **scaling** and **shearing**.
- The set **A**(n) of affinities in $\mathbf{R}^n$ and the concatenation operator • form a group **GA**(n)=(**A**(n),•).
- Fundamental invariant:
  - parallelism.
- Other invariants:
  - distance ratios for any three point along a straight line
  - co-linearity
- Examples:
  - a square can be transformed into a rectangle
  - a circle can be transformed into an ellipsis
- 2-dimensional affine geometry: ($\mathbf{R}^2$,**GA**(2))

# Scaling 2D

$$\begin{cases} x' = \lambda_x x \\ y' = \lambda_y y \end{cases}$$

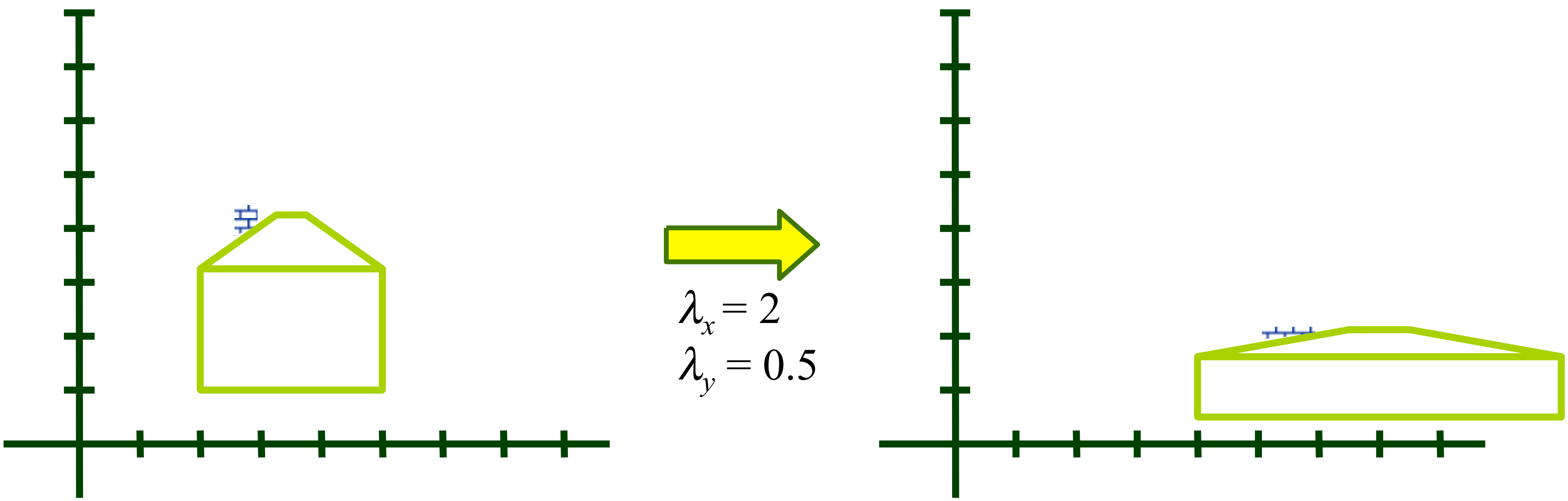
*Scaling* an object consists of multiplying each of its point component x and y by a scalar $\lambda_x$ and $\lambda_y$, respectively.



# Non-uniform scaling

$$\begin{cases} x' = \lambda_x . x \\ y' = \lambda_y . y \end{cases} \quad \text{com} \quad \lambda_x \neq \lambda_y$$

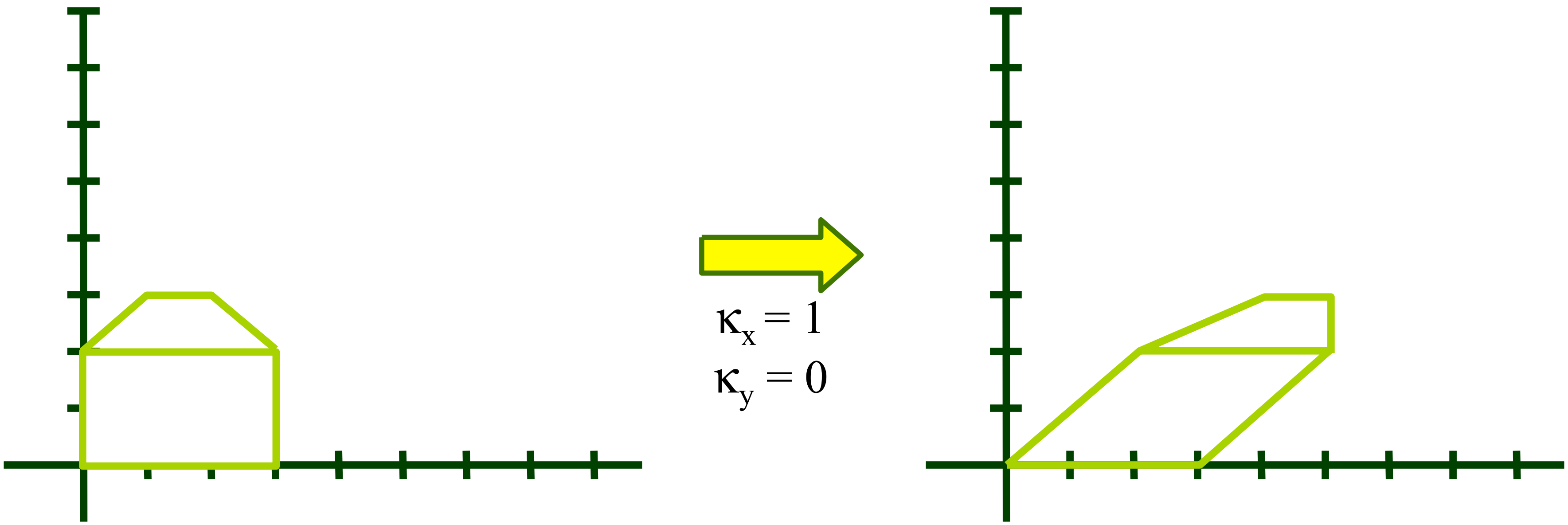
*Non-uniform scaling* an object consists of multiplying each of its point component x and y by a scalar $\lambda_x$ and $\lambda_y$, respectively, with $\lambda_x \neq \lambda_y$.



# Shearing

$$\begin{cases} x' = x + \kappa_x y \\ y' = y + \kappa_y x \end{cases}$$

*Shearing* an object consists of linearly deforming it along either x-axis or y-axis or both.



*Only linear transformations can be represented by matrices 2x2*

# Matrix representation 3x3 for 2D affine transformations

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

*Translation*

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \lambda_x & 0 & 0 \\ 0 & \lambda_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

*Scaling*

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

*Rotation*

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \kappa_x & 0 \\ \kappa_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

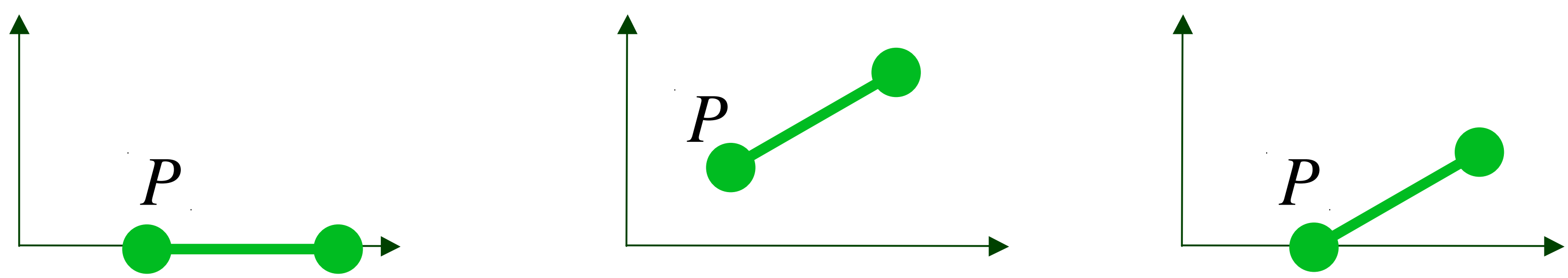
*Shearing*

# Composition of 2D affine transformations

- The composition operator is the product of matrices.
- It is a consequence of the Associativity Axiom of the affine geometry and the dimension 3x3 of the matrices associated to 2D affine transformations.
- REMARK:
  - The order of the composition matters.
  - The matrix product is not commutative.
  - The affine geometry does not satisfy the Commutativity Axiom.
- Example:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \left( \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \lambda_x & 0 & 0 \\ 0 & \lambda_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

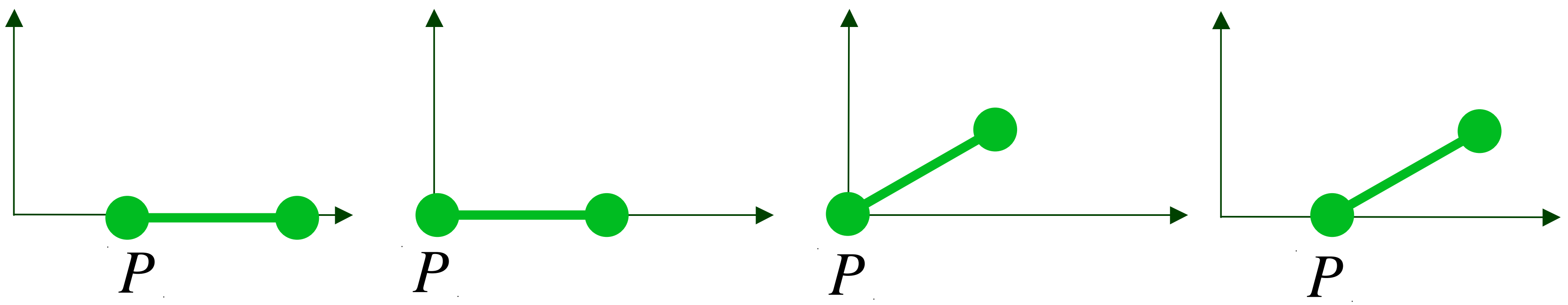
## Example: rotation $\theta = 30°$ of an segment $\overline{PQ}$ about $P(2,0)$




Wrong
Rot(30)

Right
Tr(-2) Rot(30) Tr(2)

# **Exemplo:** rotation $\theta = 30^\circ$ of an segment $\overline{PQ}$ about $P(2,0)$




$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \left( \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

# 3D affine transformations

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

*Identity*

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \lambda_x & 0 & 0 & 0 \\ 0 & \lambda_y & 0 & 0 \\ 0 & 0 & \lambda_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

*Scaling*

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

*Translation*

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

*Reflection across YZ*

# Other 3D affine transformations

*Rotation about z-axis*

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

*Rotation about y-axis*

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

*Rotation about x-axis*

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

# Affine transformation in OpenGL

- There are two ways to specify a geometric transformation:
  - Pre-defined transformations: `glTranslate`, `glRotate` and `glScale`.
  - Arbitary transformations by direct specification of matrices: `glLoadMatrix`, `glMultMatrix`
- These transformations are effected by the ***modelview*** matrix.
- For that, we have to say that it is the current matrix. This is done through the statement `glMatrixMode(GL_MODELVIEW)`.
- Let us say that the OpenGL has even a stack for each sort of matrix. It has 4 matrix sorts: *modelview*, *projection*, *texture*, and *colour* matrices.
- The *modelview* stack is initialized with the identity matrix.

# Pre-defined affine transformations in OpenGL

- `glTranslate{f,d}(dx, dy, dz)`
  - Causes subsequently defined coordinate positions to be translated by the vector `(dx,dy,dz)`, where `dx`, `dy`, `dz` are either **f**loating-point or **d**ouble precision numbers.
- `glRotate{f,d}(angle, vx, vy, vz)`
  - Causes subsequently defined coordinate positions to be rotated by angle degrees about the axis `(vx,vy,vz)` through the origin.
- `glScale{f,d}(sx, sy, sz)`
  - Causes subsequently defined coordinate positions to be scaled by factors `sx`,`sy`,`sz` along x, y, and z, respectively.
  - NOTE: setting any of these factors to zero can cause a processing error.

# Matrix operations

## em OpenGL

- `glLoadIdentity()`
  - Sets the current matrix to the identity.
- `glLoadMatrix{f,d}(<array>)`
  - Sets the current matrix to be given by the 16 elements of argument 1-dimensional array.
  - The entries must be given in **column major order**.
- `glMultMatrix{f,d}(<array>)`
  - Post-multiplies the current matrix *M* with the matrix *N* that is specified by the elements in the argument 1-dimensional array: $M=M \cdot N$

# **Example:** producing the *modelview* matrix $M = M_2 \cdot M_1$

- The sequence of statements is as follows:

```
glLoadIdentity();
glMultMatrixf(<array of M2>);
glMultMatrixf(<array of M1>);
```

- Note that the first transformation applied is the last specified.

# Examples in OpenGL

- Cumulative affine transformations
- Non-cumulative affine transformations
- Stack-controlled cumulative affine transformations

# Cumulative affine transformations in 2D

```
/* * cum-2D-trf.cc - Cumulative 2D transformations * Abel Gomes  */
#include <OpenGL/gl.h>          // Header File For The OpenGL Library
#include <OpenGL/glu.h>         // Header File For The GLu Library
#include <GLUT/glut.h>          // Header File For The GLut Library
#include <stdlib.h>

void draw(){
          // Make background colour yellow
     glClearColor( 100, 100, 0, 0 );
     glClear ( GL_COLOR_BUFFER_BIT );
          // modelview matrix for modeling transformations
     glMatrixMode(GL_MODELVIEW);
          // x-axis
     glColor3f(0,0,0);
     glBegin(GL_LINES);
          glVertex2f(0.0,0.0);
          glVertex2f(0.5,0.0);
     glEnd();
     // y-axis
     glColor3f(0,0,0);
     glBegin(GL_LINES);
          glVertex2f(0.0,0.0);
          glVertex2f(0.0,0.5);
     glEnd();
```

# Cumulative affine transformations in 2D (cont.)

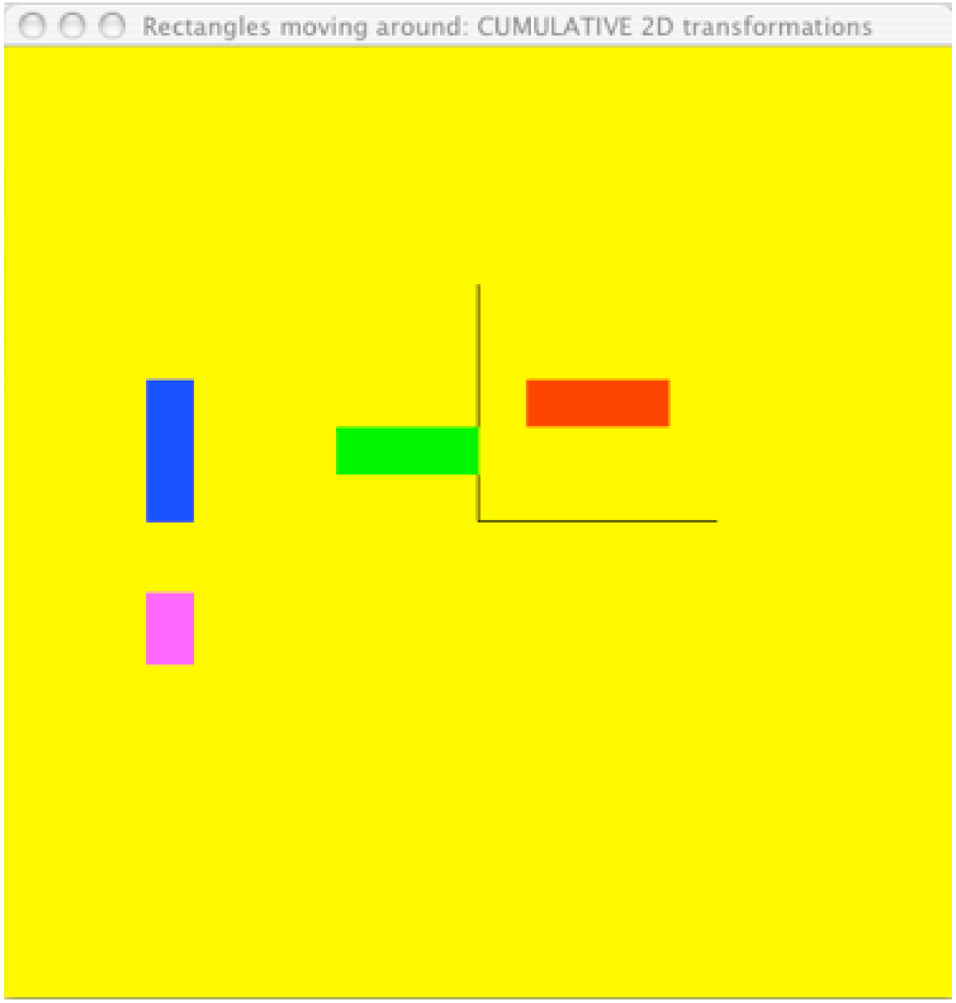
```
        // RED rectangle
    glColor3f( 1, 0, 0 );
    glRectf(0.1,0.2,0.4,0.3);
        // Translate GREEN rectangle
    glColor3f( 0, 1, 0 );
    glTranslatef(-0.4, -0.1, 0.0);
    glRectf(0.1,0.2,0.4,0.3);
        // Rotate and translate BLUE rectangle
    glColor3f( 0, 0, 1 );
        //glLoadIdentity();// reset the modelview matrix
    glRotatef(90, 0.0, 0.0,1.0);
    glRectf(0.1,0.2,0.4,0.3);
        // Scale, rotate and translate MAGENTA rectangle
    glColor3f( 1, 0, 1 );
        //glLoadIdentity();// reset the modelview matrix
    glScalef(-0.5, 1.0, 1.0);
    glRectf(0.1,0.2,0.4,0.3);
        // display rectangles
    glutSwapBuffers();
}                                       // end of draw()
```

# Cumulative affine transformations in 2D (cont.)

```
// Keyboard method to allow ESC key to quit
void keyboard(unsigned char key,int x,int y)
{
     if(key==27) exit(0);
}

int main(int argc,  char ** argv)
{
     glutInit(&argc, argv);
          // Double Buffered RGB display
     glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE);
          // Set window size
     glutInitWindowSize( 500,500 );
     glutCreateWindow("Rectangles moving around: CUMULATIVE 2D transformations");
          // Declare the display and keyboard functions
     glutDisplayFunc(draw);
     glutKeyboardFunc(keyboard);
          // Start the Main Loop
     glutMainLoop();
     return 0;
}
```

# Cumulative affine transformations in 2D (cont.): output



# Non-cumulative affine transformations in 2D

```
/* * noncum-2D-trf.cc - Non-cumulative 2D transformations * Abel Gomes  */
#include <OpenGL/gl.h>        // Header File For The OpenGL Library
#include <OpenGL/glu.h>       // Header File For The GLu Library
#include <GLUT/glut.h>        // Header File For The GLut Library
#include <stdlib.h>

void draw(){
          // Make background colour yellow
     glClearColor( 100, 100, 0, 0 );
     glClear ( GL_COLOR_BUFFER_BIT );
          // modelview matrix for modeling transformations
     glMatrixMode(GL_MODELVIEW);
          // x-axis
     glColor3f(0,0,0);
     glBegin(GL_LINES);
          glVertex2f(0.0,0.0);
          glVertex2f(0.5,0.0);
     glEnd();
     // y-axis
     glColor3f(0,0,0);
     glBegin(GL_LINES);
          glVertex2f(0.0,0.0);
          glVertex2f(0.0,0.5);
     glEnd();
```

# Non-cumulative affine transformations in 2D (cont.)

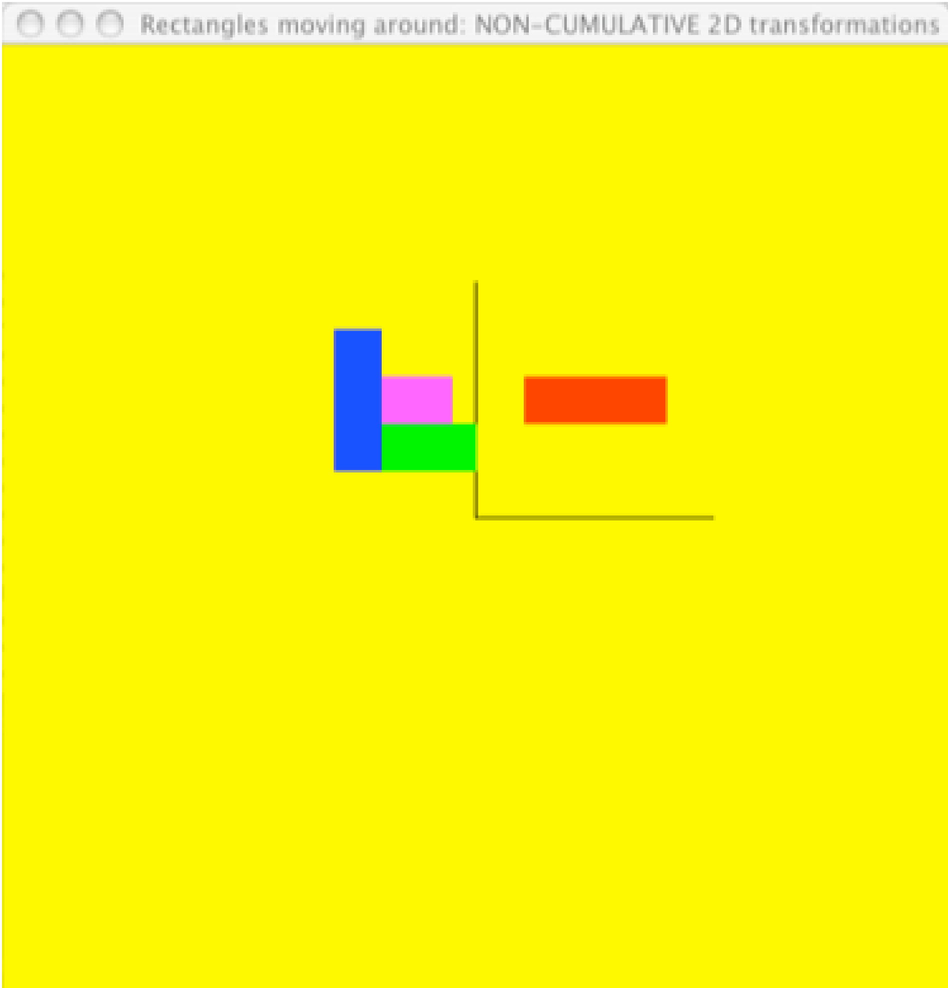
```
        // RED rectangle
    glColor3f( 1, 0, 0 );
    glRectf(0.1,0.2,0.4,0.3);
        // Translate GREEN rectangle
    glColor3f( 0, 1, 0 );
    glTranslatef(-0.4, -0.1, 0.0);
    glRectf(0.1,0.2,0.4,0.3);
        // Rotate BLUE rectangle
    glColor3f( 0, 0, 1 );
    glLoadIdentity();       // reset the modelview matrix
    glRotatef(90, 0.0, 0.0,1.0);
    glRectf(0.1,0.2,0.4,0.3);
        // Scale MAGENTA rectangle
    glColor3f( 1, 0, 1 );
    glLoadIdentity();       // reset the modelview matrix
    glScalef(-0.5, 1.0, 1.0);
    glRectf(0.1,0.2,0.4,0.3);
        // display rectangles
    glutSwapBuffers();
}                                                 // end of draw()
```

# Transformações 2D s/ acumulação (cont.)

```
// Keyboard method to allow ESC key to quit
void keyboard(unsigned char key,int x,int y)
{
	if(key==27) exit(0);
}

int main(int argc,  char ** argv)
{
	glutInit(&argc, argv);
		// Double Buffered RGB display
	glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE);
		// Set window size
	glutInitWindowSize( 500,500 );
	glutCreateWindow("Rectangles moving around: NON-CUMULATIVE 2D transformations");
		// Declare the display and keyboard functions
	glutDisplayFunc(draw);
	glutKeyboardFunc(keyboard);
		// Start the Main Loop
	glutMainLoop();
	return 0;
}
```

# Non-cumulative affine transformations in 2D (cont.): output



# Stack-controlled cumulative affine transformations in 2D

```
/* * stack-cum-2D-trf.cc - Stack-cumulative 2D transformations * Abel Gomes  */
#include <OpenGL/gl.h>         // Header File For The OpenGL Library
#include <OpenGL/glu.h>        // Header File For The GLu Library
#include <GLUT/glut.h>         // Header File For The GLut Library
#include <stdlib.h>

void draw(){
          // Make background colour yellow
     glClearColor( 100, 100, 0, 0 );
     glClear ( GL_COLOR_BUFFER_BIT );
          // modelview matrix for modeling transformations
     glMatrixMode(GL_MODELVIEW);
          // x-axis
     glColor3f(0,0,0);
     glBegin(GL_LINES);
          glVertex2f(0.0,0.0);
          glVertex2f(0.5,0.0);
     glEnd();
     // y-axis
     glColor3f(0,0,0);
     glBegin(GL_LINES);
          glVertex2f(0.0,0.0);
          glVertex2f(0.0,0.5);
     glEnd();
```

# Stack-controlled cumulative affine transformations in 2D (cont.)

```
        // RED rectangle
    glColor3f( 1, 0, 0 );
    glRectf(0.1,0.2,0.4,0.3);
        // Translate GREEN rectangle
    glColor3f( 0, 1, 0 );
    glTranslatef(-0.4, -0.1, 0.0);
    glRectf(0.1,0.2,0.4,0.3);
        // save modelview matrix on the stack
    glPushMatrix();
        // Rotate and translate BLUE rectangle
    glColor3f( 0, 0, 1 );
    glRotatef(90, 0.0, 0.0,1.0);
    glRectf(0.1,0.2,0.4,0.3);
        // restore modelview matrix from the stack
    glPopMatrix();

        // Scale and translate MAGENTA rectangle
    glColor3f( 1, 0, 1 );
    glScalef(-0.5, 1.0, 1.0);
    glRectf(0.1,0.2,0.4,0.3);
        // display rectangles
    glutSwapBuffers();
}                                             // end of draw()
```

# Stack-controlled cumulative affine transformations in 2D (cont.):

```
// Keyboard method to allow ESC key to quit
void keyboard(unsigned char key,int x,int y)
{
	if(key==27) exit(0);
}

int main(int argc,  char ** argv)
{
	glutInit(&argc, argv);
		// Double Buffered RGB display
	glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE);
		// Set window size
	glutInitWindowSize( 500,500 );
	glutCreateWindow("Rectangles moving around: STACK-CUMULATIVE 2D transformations");
		// Declare the display and keyboard functions
	glutDisplayFunc(draw);
	glutKeyboardFunc(keyboard);
		// Start the Main Loop
	glutMainLoop();
	return 0;
}
```

# Stack-controlled cumulative affine transformations in 2D (cont.): output