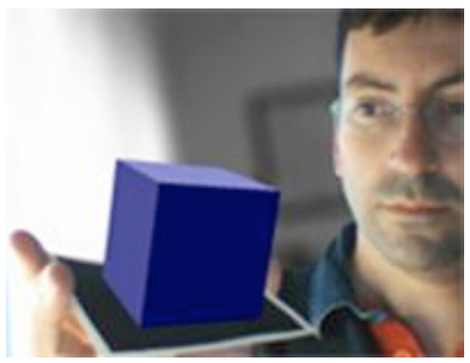
Computer Graphics

2D transformations



Jordi Linares i Pellicer
Escola Politècnica Superior d'Alcoi
Dep. de Sistemes Informàtics i Computació
jlinares@dsic.upv.es
http://www.dsic.upv.es/~jlinares

2D transformations

- 2D geometric transformations are essential in transforming and visualising our model
- The affine basic transformation are: translation, rotation and scale
- The transformation matrices are:

$$T = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \qquad E = \begin{bmatrix} E_x & 0 & 0 \\ 0 & E_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad R = \begin{bmatrix} \cos(\alpha) & -\text{sen}(\alpha) & 0 \\ \text{sen}(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2D transformations

- We can combine several transformations in order to define more complex ones
- For example, rotating around an arbitrary point (xc, yc):

$$\begin{bmatrix} x_3 \\ y_3 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_c \\ 0 & 1 & -y_c \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & (x_c - \cos\alpha \cdot x_c + \sin\alpha \cdot y_c) \\ \sin\alpha & \cos\alpha & (y_c - \sin\alpha \cdot x_c - \cos\alpha \cdot y_c) \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D transformations

- *processing* allows that every drawing primitive could be affected by an internal transformation matrix, generally known as model/view matrix
- Specifying this matrix we will be able to transform the coordinate system of our model to the coordinate system of the window, or any arbitrary complex transformation
- At the beginning, the model/view matrix is the identity matrix, not affecting any drawing primitive
- We can define this matrix by using the functions `rotate(angle)`, `translate(tx, ty)` and `scale(sx, sy)`
- The effect of these functions is to assign to the model/view matrix the product of the previous value of this matrix and the matrix defined by the function used (rotate, translate or scale). For example:
- `rotate(PI)` => `M = M • rotate(PI)`
- In practice, that means that the order in which we have to codify these functions is the reverse order in which they will be applied, i.e., the order will be from bottom to top

2D transformations

- Example:

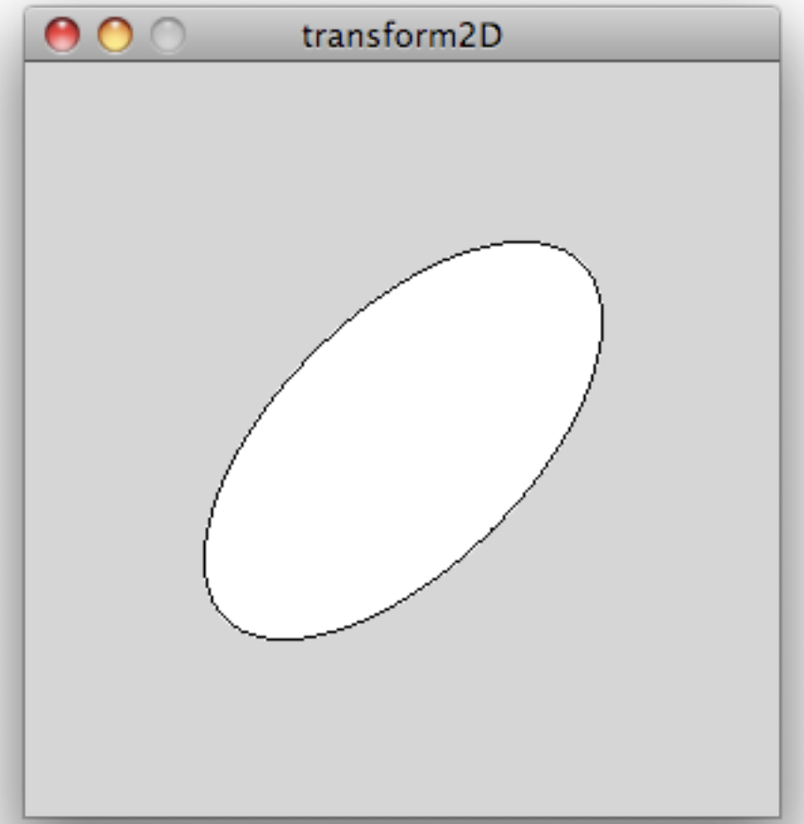
```
// Rotating around an arbitrary point
int xc, yc;

size(300, 300);

// Let's rotate around the middle of the window
xc = width / 2;
yc = height / 2;

// We have to specify the matrices in
// the reverse order in which they will
// affect our model
translate(xc, yc);
rotate(PI / 4.0);
translate(-xc, -yc);

// Now we draw the ellipse with center
// at (xc, yc), with horizontal diameter of
// 100 and 200 of vertical diameter.
// The effect will be the ellipse rotated
// 45 degrees around its center
ellipse(xc, yc, 100, 200);
```



Practice 4-1

- Modify the practice 2-1 (visualising trigonometric functions) changing the coordinate system from `[0, 2*PI]` to `[0, width]` and from `[-1, +1]` to `[height, 0]`, through the definition of the appropriate model/view matrix

- Modify the practice 3-1 in order to visualise an image to its maximum proportional size over the windows, through the definition of the appropriate model/view matrix

2D transformations

Other functions related to the model/view matrix:

`resetMatrix()`

- Initiates the model/view matrix making it equal to the identity matrix

`pushMatrix()`
`popMatrix()`

- *processing* offers what it is known as a stack of matrices
- When `pushMatrix()` is executed, the current model/view matrix is pushed into this stack (without losing its current value)
- When `popMatrix()` is executed, the current model/view matrix is assigned to the top of the stack. This top is popped from the stack
- In a chain of transformations (a series of calls to either `translate()`, `scale()` or `rotate()`) `pushMatrix()` will be called at any intermediate step to which we will want to return by calling to `popMatrix()`
- There have to be as many calls to `pushMatrix()` as to `popMatrix()`

2D transformations

```
// An example of the use of
// pushMatrix() and popMatrix()
size(300, 300);

noFill();
background(0);
stroke(255);

// A first change to a new
// coordinate system:
// We move (0,0) to the middle
// of the window
translate(width / 2, height / 2);

// We draw in this system an
// ellipse in the middle of
// the window
ellipse(0, 0, 100, 200);

// We save this first coordinate
// system
pushMatrix();

// A second coordinate system:
// Additionally to the previous
// change, we apply a rotation
rotate(PI / 4.0);

// We draw the same ellipse,
// but now it will be affected
// by the two previous
// transformations
ellipse(0, 0, 100, 200);

// We recover the first coordinate
// system
popMatrix();

// We draw again.
// Now only affected by
// the first translate
rectMode(CENTER);
rect(0, 0, 100, 200);
```