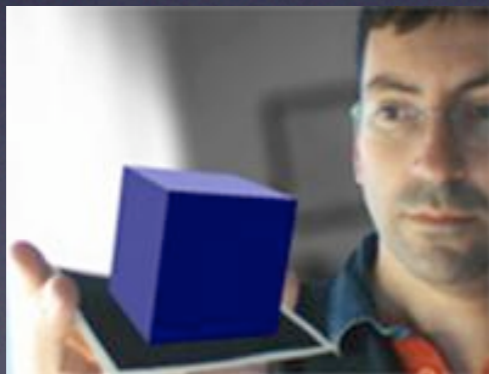


Gráficos por Computador

Transformaciones 2D



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>

Transformaciones 2D

- Las transformaciones geométricas 2D son imprescindibles en la transformación y visualización de nuestro modelo
- Las transformaciones básicas son las afines: traslación, rotación y escalado
- Las matrices de transformación son las siguientes:

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

Transformaciones 2D

- Podemos combinar varias transformaciones para definir transformaciones más complejas
- Por ejemplo, rotación respecto a un punto cualquier (x_c, y_c) :

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

Transformaciones 2D

- *processing* permite que toda primitiva de dibujo quede afectada por una matriz de transformación interna, conocida generalmente como la matriz modelo/vista
- La caracterización de esta matriz nos permite transformar el sistema de coordenadas original de nuestro modelo en el sistema de coordenadas de nuestro dispositivo, o cualquier transformación arbitrariamente compleja
- Inicialmente, la matriz modelo/vista es la matriz identidad, no afectando a ninguna primitiva de dibujo
- La definición de esta matriz se lleva a cabo mediante las funciones `rotate(angulo)`, `translate(tx, ty)` y `scale(sx, sy)`
- El efecto de estas funciones es el de asignar a la matriz modelo/vista el producto de ella misma por la matriz de transformación que indica la función. Por ejemplo:
 - `rotate(PI) => M = M • rotate(PI)`
 - Esto a efectos prácticos implica que el orden en el que se deben codificar estas funciones corresponde al orden inverso en el que se aplicarán, es decir, el orden de aplicación de las mismas será de abajo a arriba

Transformaciones 2D

- Ejemplo:

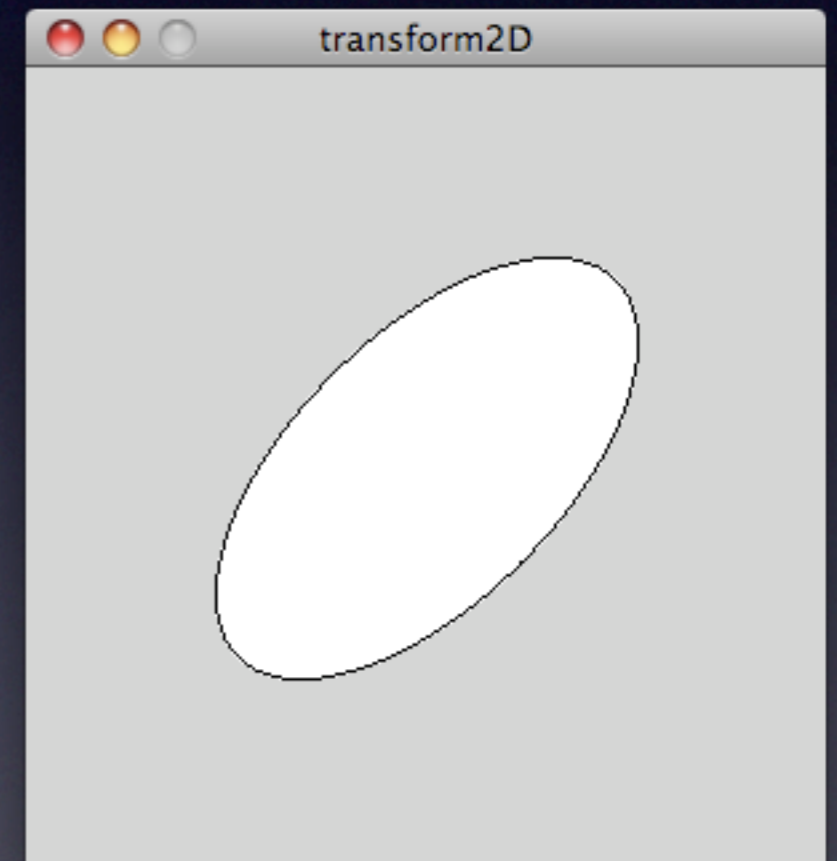
```
// Rotación respecte a un punto arbitrario
int xc, yc;

size(300, 300);

// Rotamos respecto al centro de la ventana
xc = width / 2;
yc = height / 2;

// Debemos especificar las matrices en
// orden inverso al que afectarían realmente
// a nuestro modelo
translate(xc, yc);
rotate(PI / 4.0);
translate(-xc, -yc);

// Ahora dibujamos una elipse haciendo
// que su centro esté en (xc, yc), con
// diámetro horizontal de 100 y 200 de
// diámetro vertical.
// El efecto será su rotación en 45 grados
// respecto a su centro
ellipse(xc, yc, 100, 200);
```



Práctica 4-1

- Modificar la práctica 2-1 de visualización de funciones trigonométricas de forma que se realice un cambio de sistemas de coordenadas de $[0, 2 * \text{PI}]$ a $[0, \text{width}]$ y de $[-1, +1]$ a $[\text{height}, 0]$, mediante la definición de la correspondiente matriz modelo/vista
- Modificar la práctica 3-1 para conseguir visualizar una imagen a su máximo tamaño proporcional sobre la ventana mediante la definición de la correspondiente matriz modelo/vista

Transformaciones 2D

Otras operaciones relacionadas con la matriz modelo/vista:

`resetMatrix()`

- Inicializa la matriz modelo/vista igualándola con la matrix identidad

`pushMatrix()`

`popMatrix()`

- *processing* dispone de lo que se denomina una pila de matrices
- Cuando se ejecuta `pushMatrix()`, la matriz de transformación modelo/vista actual es apilada en esta pila (sin perder su actual valor)
- Cuando se ejecuta `popMatrix()`, la matriz de transformación modelo/vista actual pierde su valor por el del tope de la pila de matrices. Este tope es efectivamente desapilado de la pila de matrices
- En una cadena de transformaciones (llamadas sucesivas a `translate()`, `scale()` o `rotate()`) se llamará a `pushMatrix()` cada paso intermedio (estado de transformación intermedio) al cual se quiera volver posteriormente llamando a `popMatrix()`
- Deben haber tantas llamadas a `pushMatrix()` como a `popMatrix()`

Transformaciones 2D

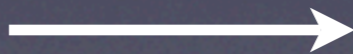
```
// Un ejemplo de uso de
// pushMatrix() y popMatrix()
size(300, 300);

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

// Un primer cambio de sistema de
// coordenadas:
// Movemos el (0,0) a la mitad
// de la ventana
translate(width / 2, height / 2);

// Dibujamos en este sistema
// una elipse en la mitad de
// la pantalla
ellipse(0, 0, 100, 200);

// Almacenamos este primer
// sistema de coordenadas
pushMatrix();
```



```
// Un segundo sistema de
// coordenadas:
// Adicionalmente al cambio
// anterior,
// aplicamos una rotación
rotate(PI / 4.0);

// Dibujamos la misma elipse,
// pero en este caso quedará
// afectada por las dos
// transformaciones
// anteriores
ellipse(0, 0, 100, 200);

// Recuperamos el primer sistema
// de coordenadas
popMatrix();

// Dibujamos de nuevo.
// Ahora estamos afectados
// únicamente
// por el primer translate
rectMode(CENTER);
rect(0, 0, 100, 200);
```

