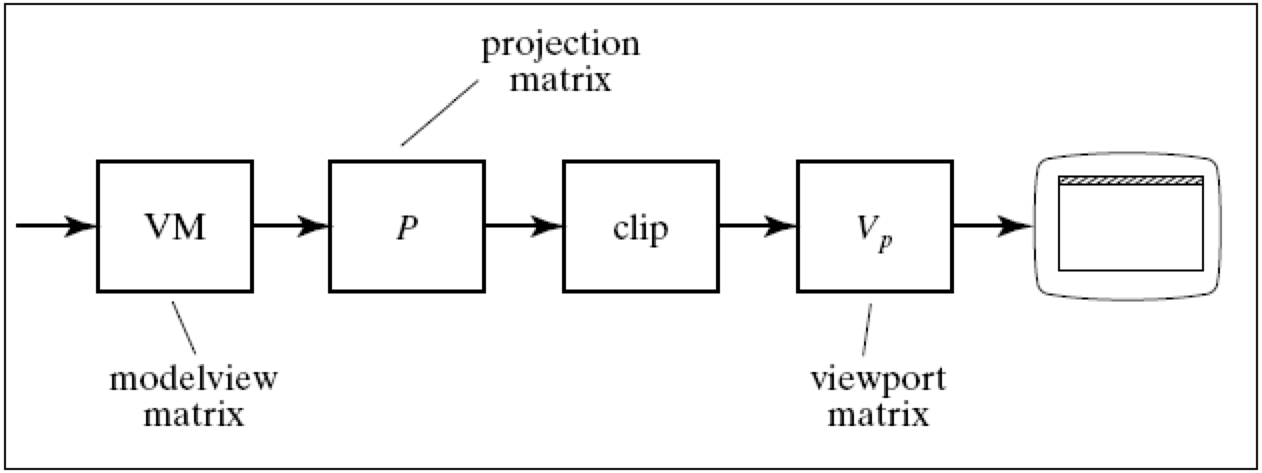
Transformaciones afines

P. J. Martín, A. Gavilanes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

- Imágenes virtuales vs modelado físico:
  - La matriz de modelado se aplica a los vértices.
- Aplicación de las transformaciones:
  - Replicar imágenes virtuales de un mismo modelo físico: escenas complejas.
  - Explorar la escena: mover objetos vs mover la cámara.
  - Animar objetos: se mueve la imagen virtual.
- En OpenGL: GL_MODELVIEW, GL_PROJECTION, matriz del puerto de vista.



Transformaciones afines

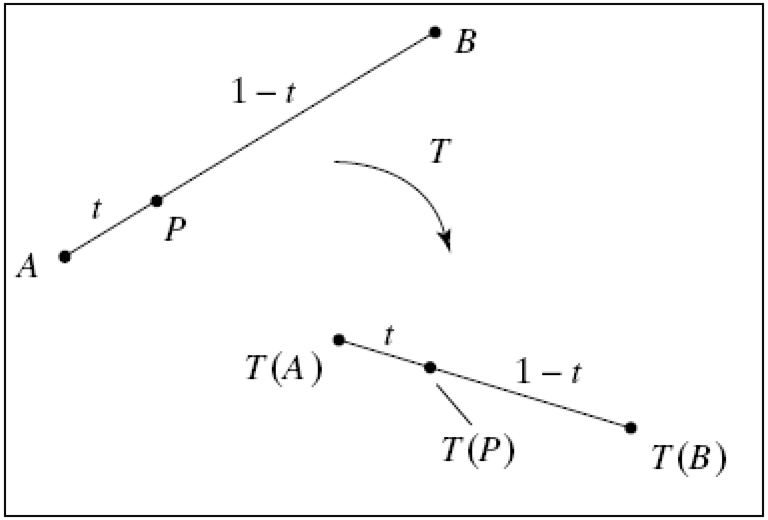
- Están definidas por matrices de la forma:

$$\begin{pmatrix} x' \\ y' \\ z' \\ \blacksquare \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ \blacksquare \end{pmatrix}$$

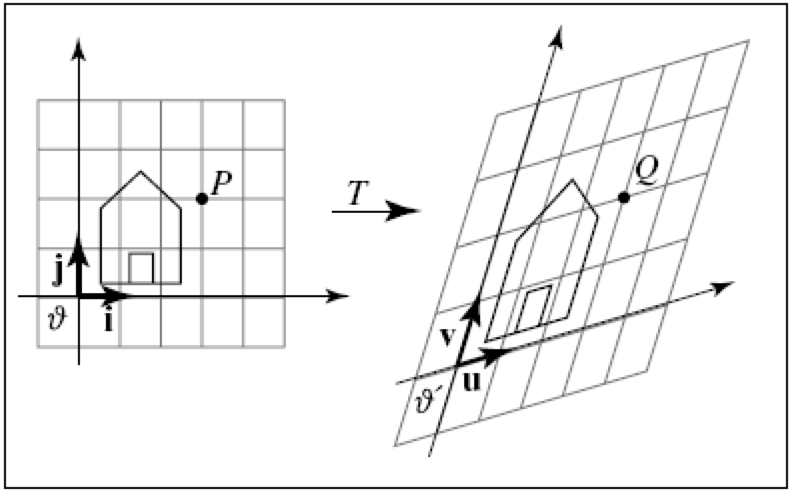
Coordenada homogénea: $\blacksquare \in \{0,1\}$
Puntos (1) y vectores (0) se transforman de forma análoga!

- Propiedades de las transformaciones afines:
  - Las combinaciones afines de puntos se transforman en combinaciones afines de los puntos transformados
  - Las líneas y planos se transforman en líneas y planos, respectivamente
  - Líneas y planos paralelos se transforman en líneas y planos paralelos

- Las transformaciones afines preservan las proporciones, pero no los ángulos.



- Aplicación de una transformación afín T a una rejilla.



- Traslaciones

$$\begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x+T_x \\ y+T_y \\ z+T_z \\ 1 \end{pmatrix}$$

- El vector de traslación es **t** = (Tx, Ty, Tz, 0).
- El comando de OpenGL:

```
glTranslated(Tx, Ty, Tz);
```

  post-multiplica la matriz de modelado-vista por la matriz de traslación asociada al vector determinado por los tres parámetros.
- Cuando Tz=0.0 se obtienen las traslaciones 2D.

- Escalaciones

$$\begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \cdot S_x \\ y \cdot S_y \\ z \cdot S_z \\ 1 \end{pmatrix}$$

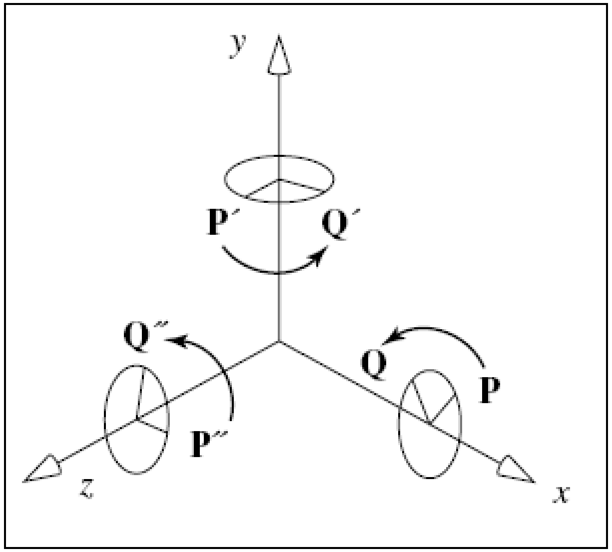
- Los factores de escalación son Sx, Sy, Sz en los ejes X, Y, Z, respectivamente.
- El comando de OpenGL:

```
glScaled(Sx, Sy, Sz);
```

post-multiplica la matriz de modelado-vista por la matriz de escalación asociada a los factores determinados por los parámetros.

- Cuando Sz=1.0 se obtienen las escalaciones 2D.

- Rotaciones elementales alrededor de los ejes



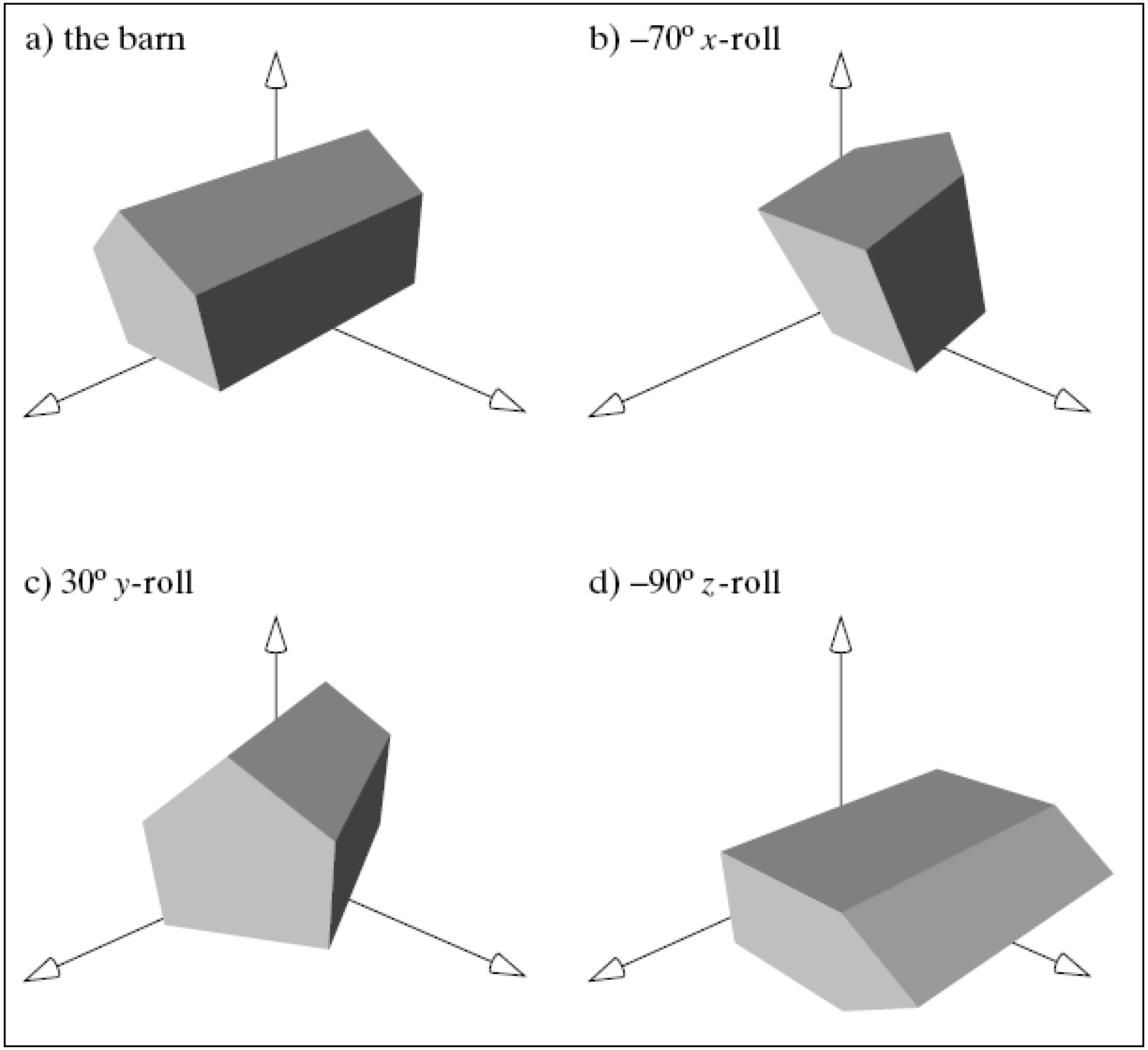
- Las rotaciones se miden en sentido anti-horario, cuando se mira el origen desde la parte positiva del eje respectivo.
- Rotaciones elementales Rx(θ), Ry(θ), Rz(θ).

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

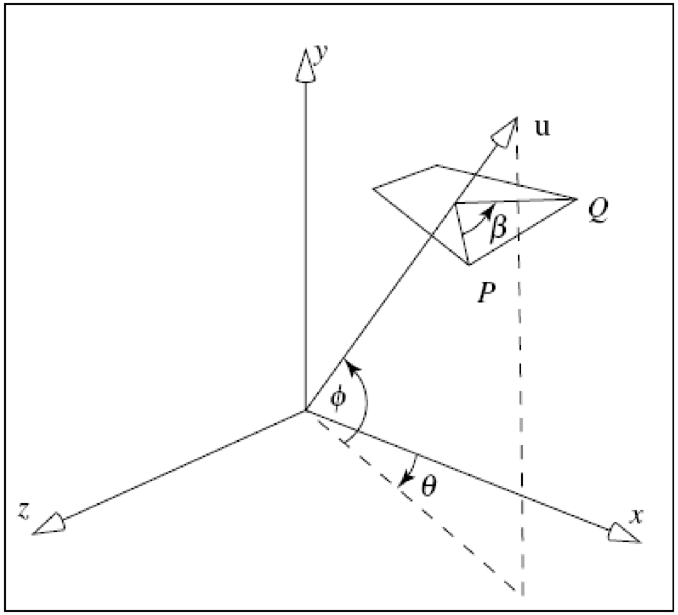
$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformaciones elementales 3D en OpenGL



- Rotación alrededor de un eje que pasa por el origen
- Una rotación alrededor de una recta que pasa por el origen y tiene vector director **u**, de β grados medidos en sentido anti-horario, tomado éste cuando se mira desde el punto señalado por **u** al origen, se puede expresar como una composición de 5 rotaciones elementales alrededor de los ejes coordenados

$$R\mathbf{u}(\beta) = Ry(-\theta)\cdot Rz(\phi)\cdot Rx(\beta)\cdot Rz(-\phi)\cdot Ry(\theta)$$



- El comando de OpenGL:

```
glRotated(β, x, y, z);
```

post-multiplica la matriz de modelado-vista por la matriz correspondiente a una rotación con respecto a una recta que pasa por el origen y tiene vector director **u**=(x, y, z, 0), de β grados en sentido anti-horario, tomado éste cuando se mira desde el punto (x, y, z, 1), al origen.

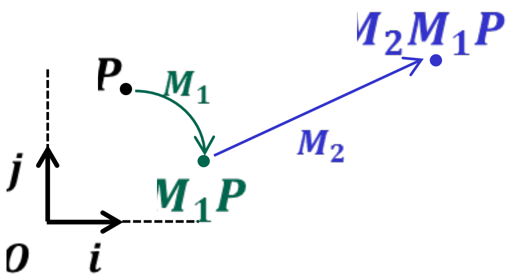
- La matriz correspondiente a esta rotación es:

$$\begin{pmatrix} x^2(1-c)+c & xy(1-c)-zs & xz(1-c)+ys & 0 \\ xy(1-c)+zs & y^2(1-c)+c & yz(1-c)-xs & 0 \\ xz(1-c)-ys & yz(1-c)+xs & z^2(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

donde c=cos(β), s=sen(β), y **u**=(x, y, z).

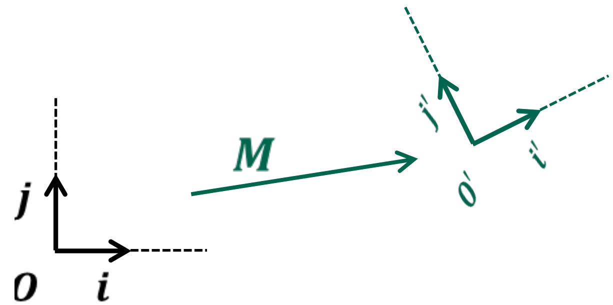
- Cuando la rotación es con respecto al eje Z de coordenadas se obtienen las rotaciones 2D.

- Las transformaciones afines vistas como transformaciones de objetos transforman puntos y vectores en un marco de coordenadas fijo.
- $M\begin{pmatrix} x \\ y \\ z \\ \blacksquare \end{pmatrix}$ son las coordenadas del punto/vector ($\blacksquare \in \{0, 1\}$) transformado.
- Composición de transformaciones: primero M_1 y luego M_2



  - La transformación afín resultante es M_2M_1
  - Pre-multiplicación de matrices.
  - El producto de matrices no es conmutativo $\Longrightarrow$ el orden en una sucesión de transformaciones importa!

- Las transformaciones afines vistas como transformaciones de marcos transforman marcos de coordenadas.



Simplificación en 2D!!

- El marco $\langle \boldsymbol{i}, \boldsymbol{j}, \boldsymbol{0} \rangle$ se transforma en el marco $\langle \boldsymbol{i}', \boldsymbol{j}', \boldsymbol{O}' \rangle$

  - $\boldsymbol{i}' = \boldsymbol{M}\begin{pmatrix}1\\0\\0\end{pmatrix} = \begin{pmatrix}m_{11}\\m_{21}\\0\end{pmatrix}$ es el eje x transformado (primera columna de $\boldsymbol{M}$).
  - $\boldsymbol{O}' = \boldsymbol{M}\begin{pmatrix}0\\0\\1\end{pmatrix} = \begin{pmatrix}m_{13}\\m_{23}\\1\end{pmatrix}$ es el origen transformado (última columna de $\boldsymbol{M}$).

Coordenadas de → en el marco ↓	$\boldsymbol{i}'$	$\boldsymbol{j}'$	$\boldsymbol{O}'$
$\langle \boldsymbol{i}, \boldsymbol{j}, \boldsymbol{O} \rangle$	$\begin{pmatrix}m_{11}\\m_{21}\\0\end{pmatrix}$	$\begin{pmatrix}m_{12}\\m_{22}\\0\end{pmatrix}$	$\begin{pmatrix}m_{13}\\m_{23}\\1\end{pmatrix}$
$\langle \boldsymbol{i}', \boldsymbol{j}', \boldsymbol{O}' \rangle$	$\begin{pmatrix}1\\0\\0\end{pmatrix}$	$\begin{pmatrix}0\\1\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\1\end{pmatrix}$

- Dado un punto/vector de coordenadas $\begin{pmatrix} x \\ y \\ \blacksquare \end{pmatrix}$ expresadas en el marco transformado $\langle \boldsymbol{i}', \boldsymbol{j}', \boldsymbol{O}' \rangle$, $\boldsymbol{M} \begin{pmatrix} x \\ y \\ \blacksquare \end{pmatrix}$ calcula las coordenadas de ese punto/vector en el marco original $\langle \boldsymbol{i}, \boldsymbol{j}, \boldsymbol{O} \rangle$:

$$\begin{pmatrix} x \\ y \\ \blacksquare \end{pmatrix}_{\langle i',j',O' \rangle} = x\,\boldsymbol{i}' + y\,\boldsymbol{j}' + \blacksquare\,\boldsymbol{O}' = x(\boldsymbol{M}\,\boldsymbol{i}) + y\,(\boldsymbol{M}\,\boldsymbol{j}) + \blacksquare\,(\boldsymbol{M}\,\boldsymbol{O}) =$$

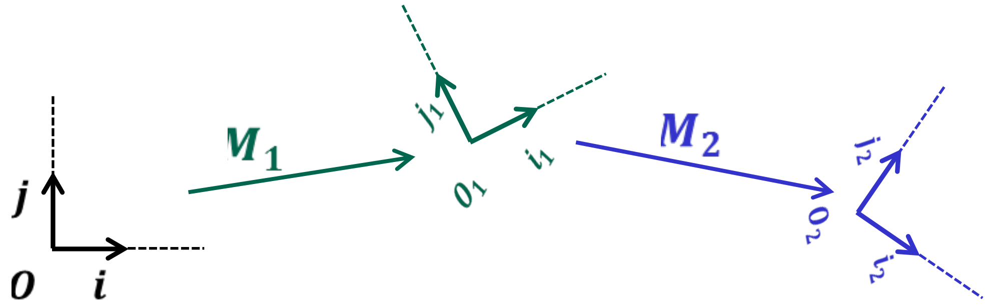
$$= \boldsymbol{M}\,(x\,\boldsymbol{i}) + \boldsymbol{M}\,(y\,\boldsymbol{j}) + \blacksquare\,(\boldsymbol{M}\,\boldsymbol{O}) = \boldsymbol{M}\,(x\,\boldsymbol{i} + y\,\boldsymbol{j} + \blacksquare\,\boldsymbol{O}) = \left[\boldsymbol{M}\begin{pmatrix} x \\ y \\ \blacksquare \end{pmatrix}\right]_{\langle i,j,O \rangle}$$

M es la matriz del cambio de coordenadas desde el marco transformado al marco original: $\boldsymbol{M}: \langle \boldsymbol{i}', \boldsymbol{j}', \boldsymbol{O}' \rangle \longrightarrow \langle \boldsymbol{i}, \boldsymbol{j}, \boldsymbol{O} \rangle$

- Consecuencia en el modelado:
  1. Modelamos localmente.
  2. Colocamos el marco local en el marco global.
  3. $\boldsymbol{M}$ resuelve el cambio de coordenadas de locales a globales!

OpenGL trabaja así!!

- Composición de transformaciones: primero M_1 y luego M_2



- $P_{\langle i_2, j_2, o_2 \rangle} =$
  $= (M_2 P)_{\langle i_1, j_1, o_1 \rangle}$
  $= (M_1 M_2 P)_{\langle i, j, o \rangle}$

- La transformación afín resultante es $M_1 M_2$
- Post-multiplicación de matrices.
- El producto de matrices no es conmutativo $\Longrightarrow$ el orden en una sucesión de transformaciones importa!

- OpenGL interpreta las transformaciones afines como transformación de marcos de coordenadas.
- Cada comando OpenGL de traslación, escalación o rotación, con matriz asociada M, se traduce en la post-multiplicación de la transformación actual por M. Es decir, si CT es la transformación actual se ejecuta:

```
CT= CT*M;
```

- Esta operación puede aplicarse sobre cualquiera de las matrices de OpenGL: `GL_MODELVIEW`, `GL_PROJECTION`... Por eso debemos activar la matriz de modelado-vista antes con `glMatrixMode(GL_MODELVIEW);`
- OpenGl usa esta matriz para pasar de coordenadas locales a coordenadas globales.
- En realidad, OpenGL no mantiene una matriz de cada tipo, sino una pila de matrices para cada categoría: la de modelado-vista, la de proyección, la de color y la de textura.
- Podemos apilar/desapilar en la pila de matrices de modelado-vista para deshacer transformaciones!

Manejo de la pila de matrices de modelado en OpenGL

- Manejo de una pila de matrices en OpenGL:
  - 1.- Se activa la pila de matrices

```
glMatrixMode(GL_MODELVIEW);
```

  - 2.- Se duplica la cima de la pila

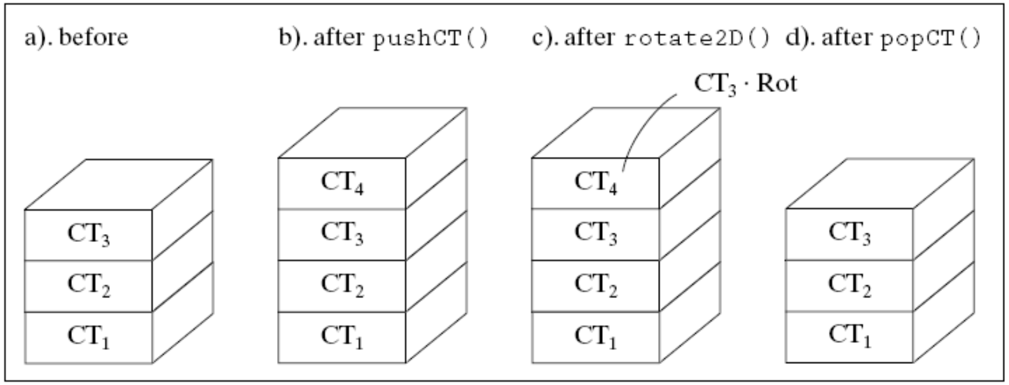
```
glPushMatrix();
```

  - 3.- Se realiza la transformación afín o sucesión de transformaciones afines que se quiera

```
glRotatef(…);
```

  - 4.- Se elimina la cima duplicada

```
glPopMatrix();
```



Manejo de la pila de matrices de modelado en OpenGL

- Antes de usar una pila de matrices, hay que recordar siempre activarla.
- La pila de matrices se inicializa con la matriz identidad:

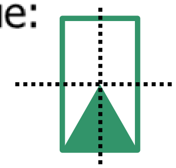
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

- Los bloques

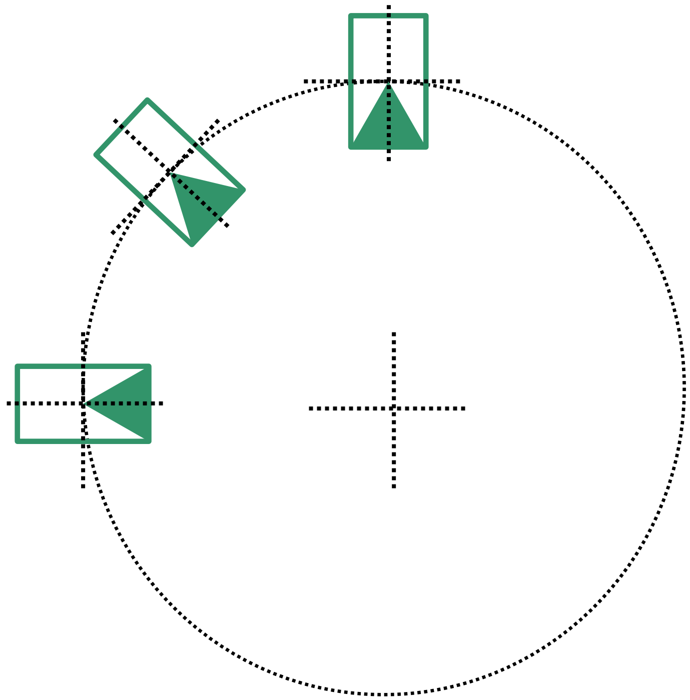
```
glPushMatrix()
    …
glPopMatrix();
```

  se pueden anidar.

- Supongamos que el método `rectangle()` dibuja, en el marco local que reciba, un rectángulo como el que sigue:



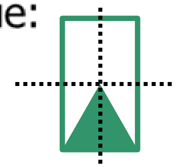
- Objetivo: visualizar 8 imágenes del rectángulo dispuestas uniformemente como sigue sobre una circunferencia de radio R centrada en el origen:



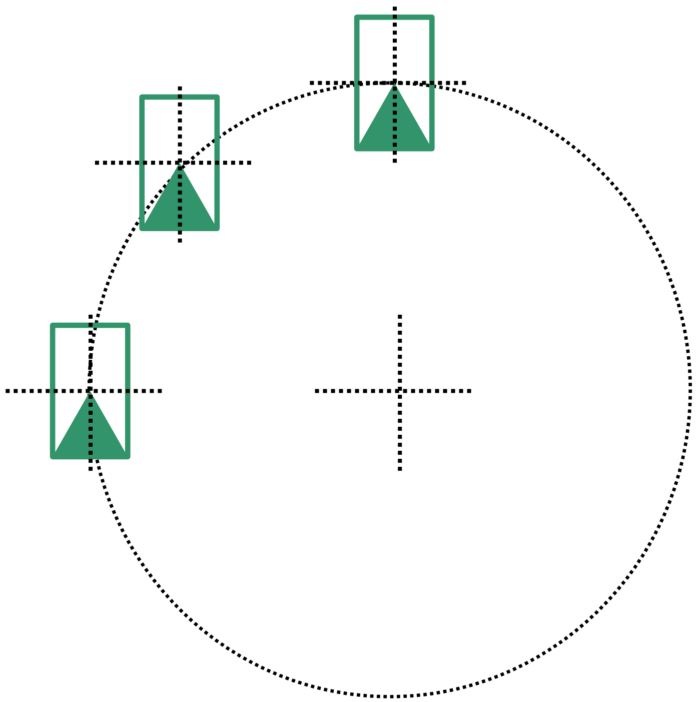
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
for(int i=0; i<8, i++){
  glPushMatrix();
  glRotated(45*i, 0, 0, 1); //2D
  glTranslated(0, R, 0);    //2D
  rectangle();
  glPopMatrix();
}
```

Ejemplo en 2D

- Supongamos que el método `rectangle()` dibuja, en el marco local que reciba, un rectángulo como el que sigue:

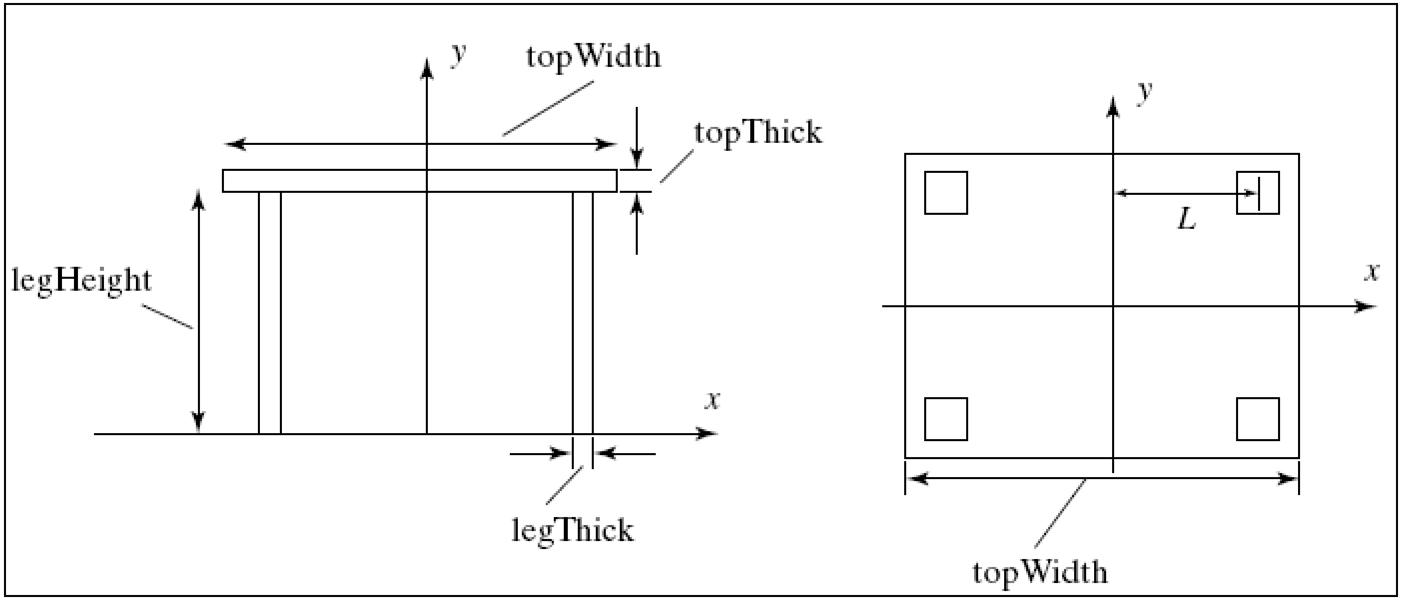


- Objetivo: visualizar 8 imágenes del rectángulo dispuestas uniformemente como sigue sobre una circunferencia de radio R centrada en el origen:



```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
for(int i=0; i<8, i++){
  glPushMatrix();
  glRotated(45*i, 0, 0, 1); //2D
  glTranslated(0, R, 0);    //2D
  glRotated(-45*i, 0, 0, 1);//2D
  rectangle();
  glPopMatrix();
}
```

- Dibujo de una mesa.
- Se usa el método `glutSolidCube(1);` que dibuja un cubo de arista 1, centrado en el origen.



- Dibujo de la pata de la mesa.

```
,
//<<<<<<<<<<<<<<<<<<< tableLeg >>>>>>>>>>>>>>>>>>>>
void tableLeg(double thick, double len)
{
	glPushMatrix();
	glTranslated(0, len/2, 0);
	glScaled(thick, len, thick);
	glutSolidCube(1.0);
	glPopMatrix();
}
```

- Dibujo del tablero de la mesa.

```
//<<<<<<<<<<<<<<<<<<<<<<< table >>>>>>>>>>>>>>>>>>>>
void table(double topWid, double topThick, double legThick, double legLen)
{ // draw the table - a top and four legs
      glPushMatrix(); // draw the table top
      glTranslated(0, legLen, 0);
      glScaled(topWid, topThick, topWid);
      glutSolidCube(1.0);
      glPopMatrix();
      double dist = 0.95 * topWid/2.0 - legThick / 2.0;
      glPushMatrix();
      glTranslated(dist, 0, dist);
      tableLeg(legThick, legLen);
      glTranslated(0, 0, -2 * dist);
      tableLeg(legThick, legLen);
      glTranslated(-2 * dist, 0, 2*dist);
      tableLeg(legThick, legLen);
      glTranslated(0, 0, -2*dist);
      tableLeg(legThick, legLen);
      glPopMatrix();
}
```