GRAFICACIÓN
Unidad II

Profr. Hilario Salazar Martínez

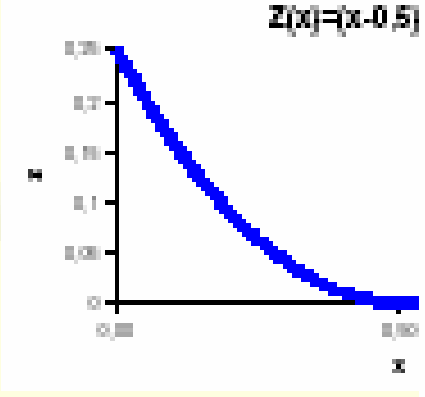
OBJETIVO ESPECIFICO:

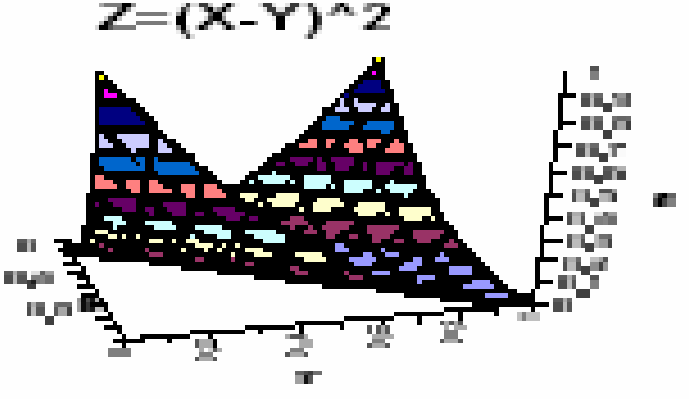
El estudiante conocerá los algoritmos y técnicas de graficado en dos dimensiones

Representación

Ecuaciones explícitas

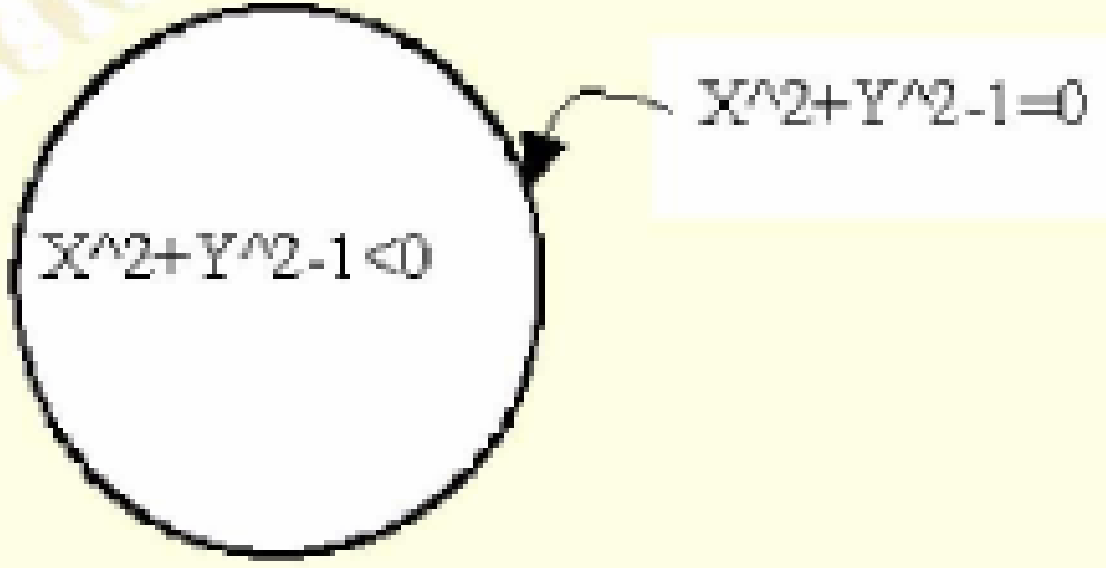
3D	$z = f(x,y)$	(superficie)
	ó $y = f_1(x), \quad z = f_2(x)$	(curva en el espacio)
2D	$y = f(x)$	(curva en el plano)





Ecuaciones implicitas

3D	f(x,y,z) = 0	(superficie)
2D	f(x,y) = 0	(curva en el plano)

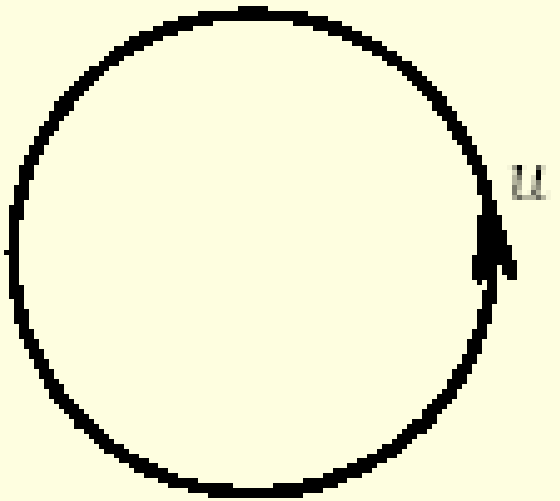


Ecuaciones implícitas

3D $x = f_1(u,v)$,
$y = f_2(u,v)$,
$z = f_3(u,v)$ $u \in [u_1,u_2]$, $v \in [v_1,v_2]$ (superficie)

2D $x = f_1(u)$
$y = f_2(u)$ $u \in [u_1,u_2]$ (curva en el plano)

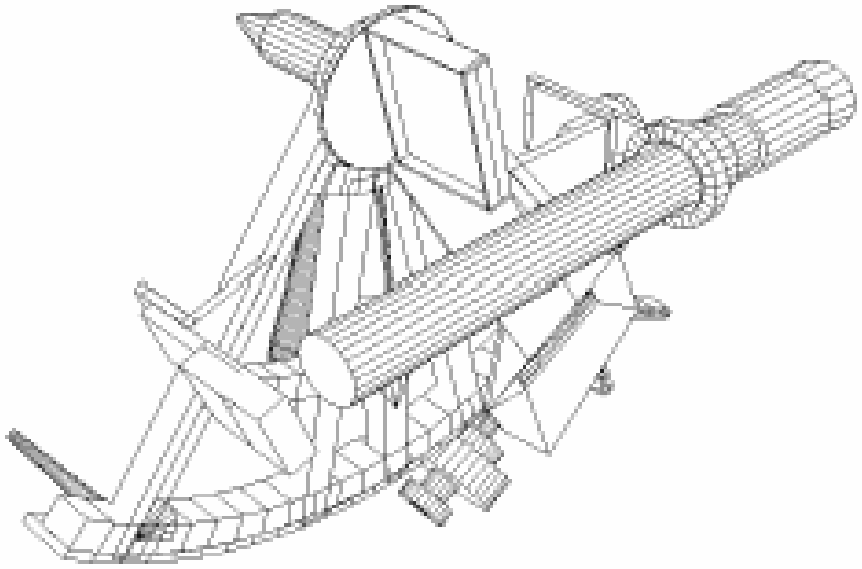
$x = \cos(2\pi u)$
$y = \operatorname{sen}(2\pi u)$ $u \in [0,1]$

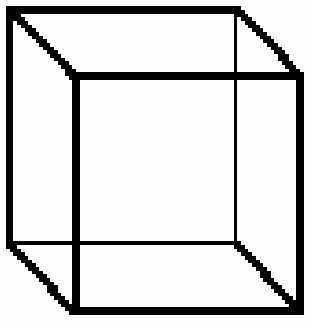


Modelado geométrico

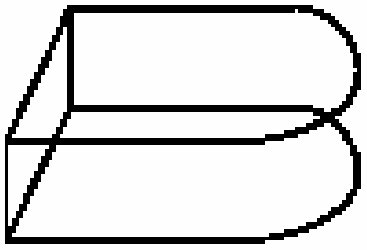
Modelos alámbricos

- Elementos
  - Puntos, líneas, arcos y círculos, cónicas, y curvas
- Ventajas
  - Fácil de construir, pocas necesidades de memoria y almacenamiento
  - Visualización muy rápida
- Desventajas
  - Representación ambigua (algoritmos de eliminación de líneas ocultas)
  - Falta de coherencia visual (algoritmos de inclusión de aristas)





Ambigüedad



Falta de coherencia

Modelos poligonales

- Ecuación del plano

$$Ax + By + Cz + D = 0$$

  - donde (x,y,z) es un punto cualquiera del plano
- Los coeficientes A, B y C definen la normal del plano y pueden obtenerse a partir de los vértices

$$N = (V_2 - V_1) \times (V_3 - V_1)$$

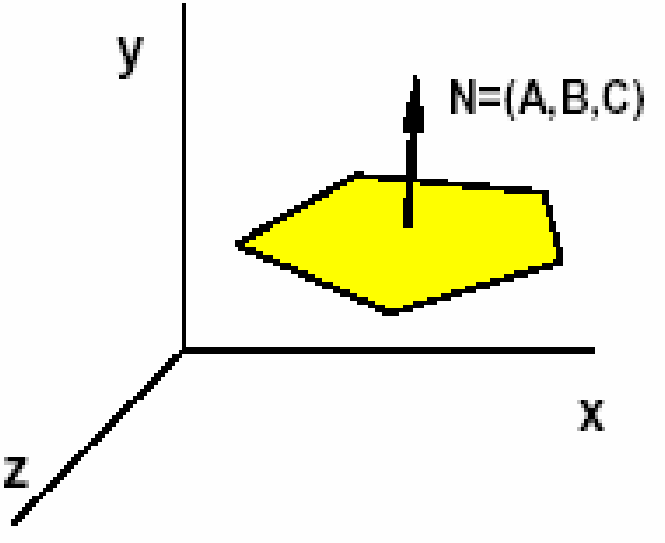
$$NV_1 = -D$$

- La ecuación del plano también se utiliza clasificación espacial
  - Punto interior

$$Ax + By + Cz + D < 0$$

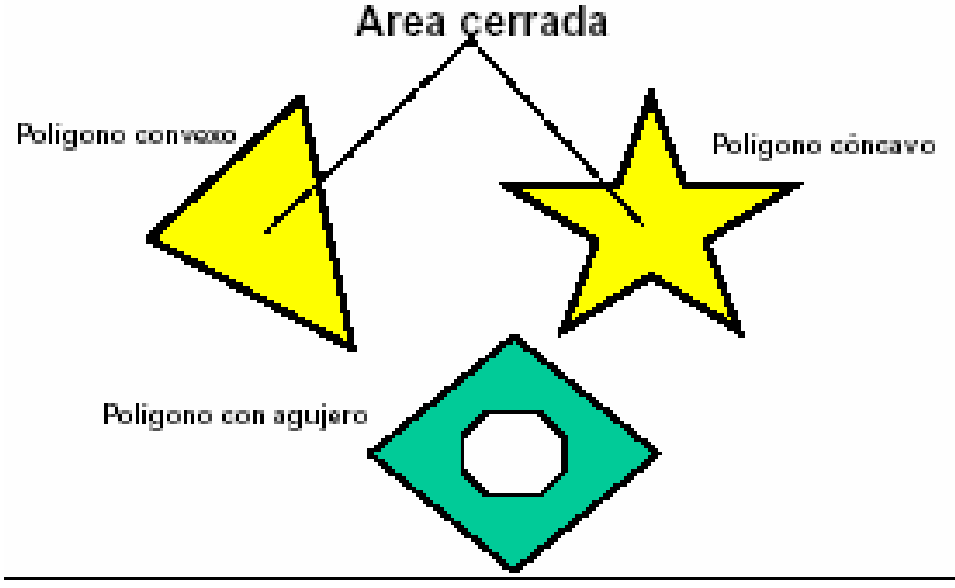
  - Punto exterior

$$Ax + By + Cz + D > 0$$

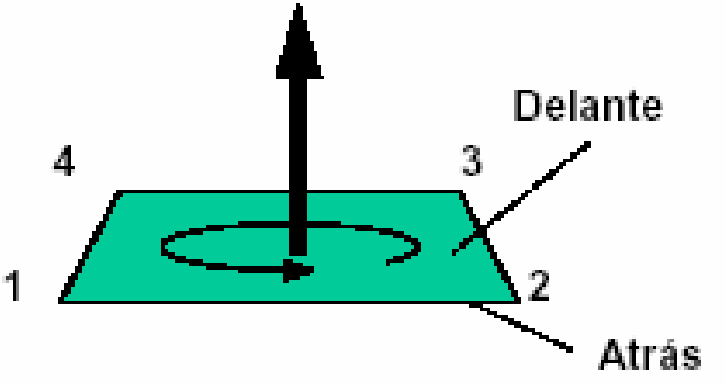


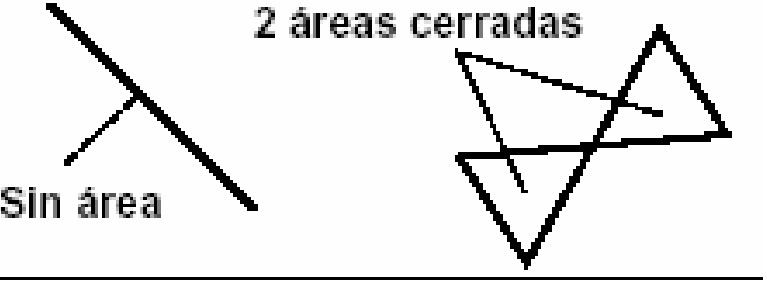
Modelos poligonales

- Polígono
  - Conjunto de líneas rectas que no se cruzan y que unen un conjunto coplanar de puntos (vértices) definiendo un área simple (habitualmente convexa y sin agujeros)



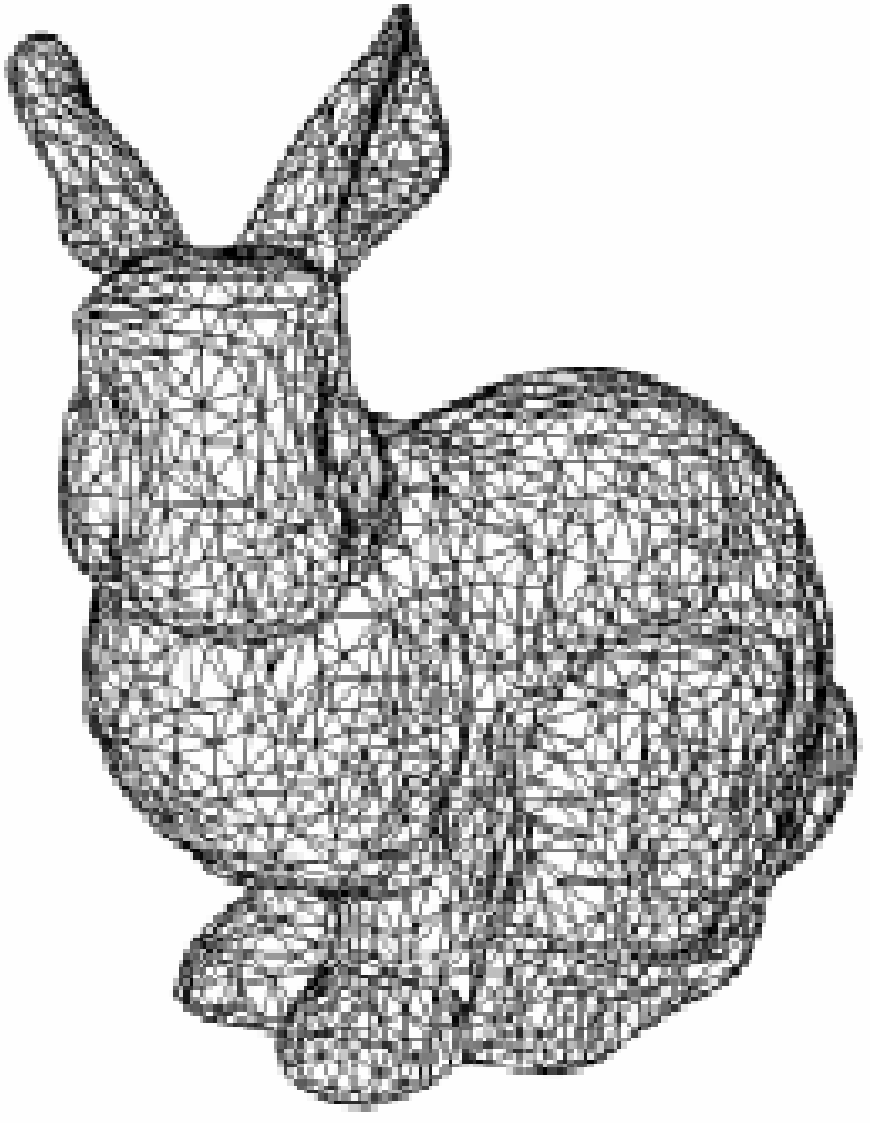
- Descripción
  - Lista ordenada de vértices (sentido horario o antihorario)
  - Dos caras (front, back)





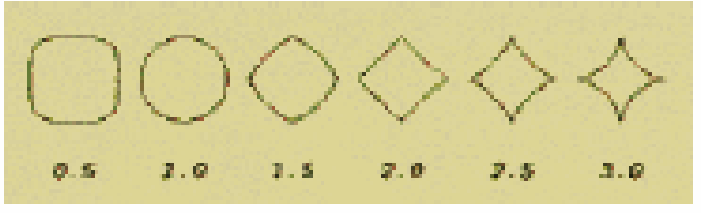
Modelos poligonales

- Malla de polígonos
  - colección de vértices, aristas y polígonos conectados
  - vértice: punto de coordenadas (x,y,z)
  - arista: segmento de línea que une dos vértices
  - polígono: secuencia cerrada de aristas
- Tipos de representaciones básicos
  - Explícita
  - Lista de vértices
  - Lista de aristas



Modelos de superficies curvas

- Supercuádricas
  - Generalización de las superficies cuádricas con parámetros adicionales



Super elipses

$$\left(\frac{x}{r_x}\right)^{2/s} + \left(\frac{y}{r_y}\right)^{2/s} = 1$$

$$x = r \cdot \cos^s \theta$$

$$y = r \cdot \mathrm{sen}^s \phi$$

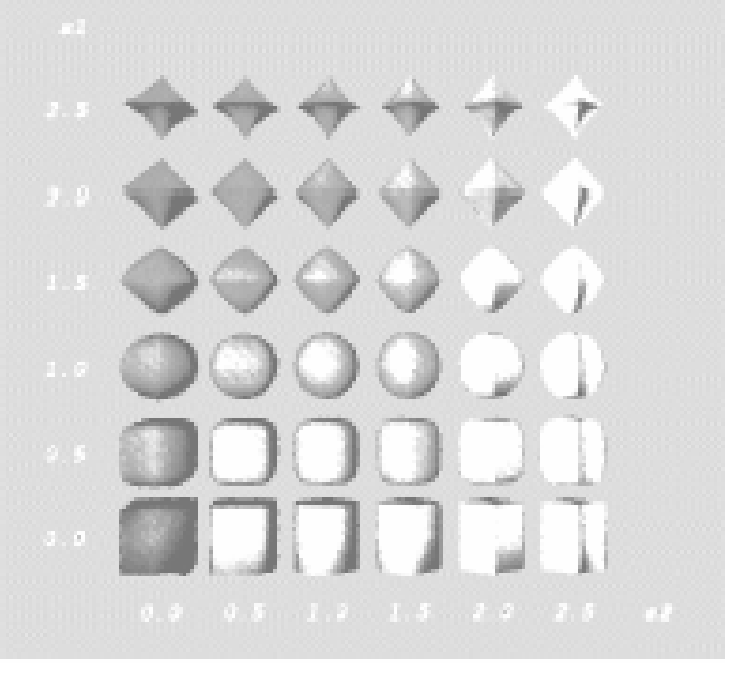
Super elipsoides

$$\left[\left(\frac{x}{r_x}\right)^{2/s_2} + \left(\frac{y}{r_y}\right)^{2/s_2}\right]^{s_2/s_1} + \left(\frac{z}{r_z}\right)^{2/s_1} = 1$$

$$x = r_x \cdot \cos^{s_1} \phi \cos^{s_2} \theta$$

$$y = r_y \cdot \mathrm{sen}^{s_1} \phi$$

$$z = r_z \cdot \cos^{s_1} \phi \, \mathrm{sen}^{s_2} \theta$$



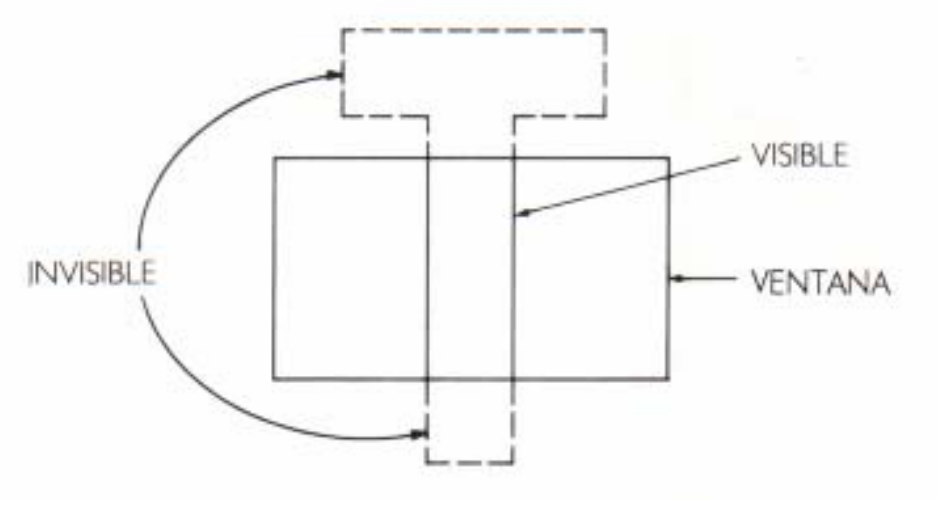
2.1 Recorte

El **recorte** es el proceso que determina la porción visible de la imagen quedando dentro de la ventana. Los puntos (y las rectas) que sobreviven al recorte son enviadas al marco visual por medio de la correspondencia ventana-marco visual.

METODO DIRECTO

Un ***punto (x,y)*** se halla dentro de la ventana rectangular si la coordenada ***x*** se encuentra entre los lados izquierdo y derecho del rectángulo y la coordenada ***y*** entre los lados superior e inferior del rectángulo:

$$xizquierda \leq x \leq xderecha \quad y \quad yabajo \leq y \leq yarriba.$$



ALGORITMO DE RECORTE DE
COHEN-SUTHERLAND (RECTAS)

Reglas

EL algoritmo empieza dividiendo el sistema de coordenadas del mundo en nueve regiones. Estas regiones se obtienen extendiendo los bordes de la ventana (fig. 2-1).

El extremo de un segmento de rectas coincide sólo con una de las nueve regiones. Cualquier punto que se halle en un borde extendido se considerará como perteneciente al lado de la ventana de ese borde.

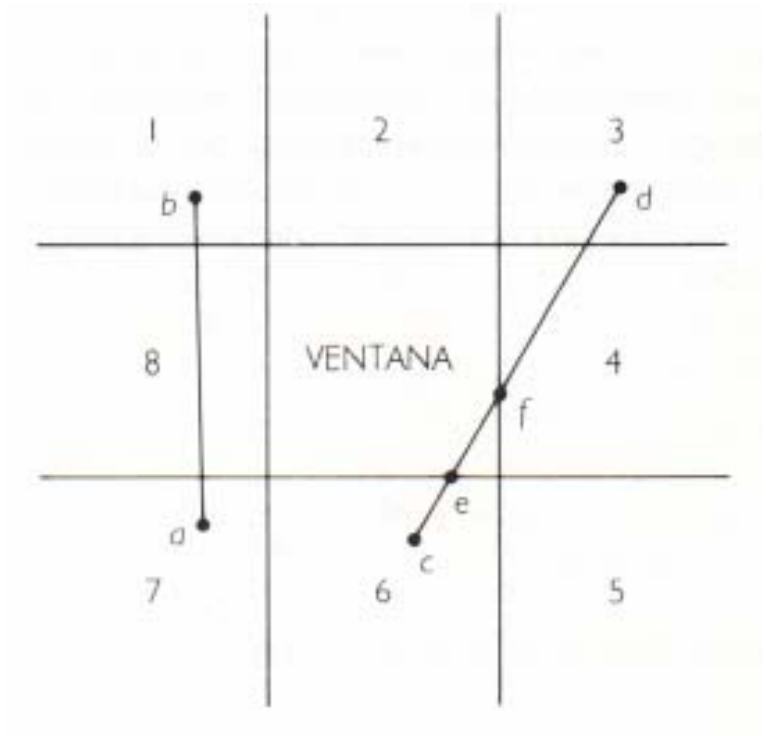


figura 2-1

Matemáticas preliminares.

Para calcular el punto de intersección del segmento de recta con el borde extendido de la ventana, se usa la formula de la pendiente ***m*** de la recta.

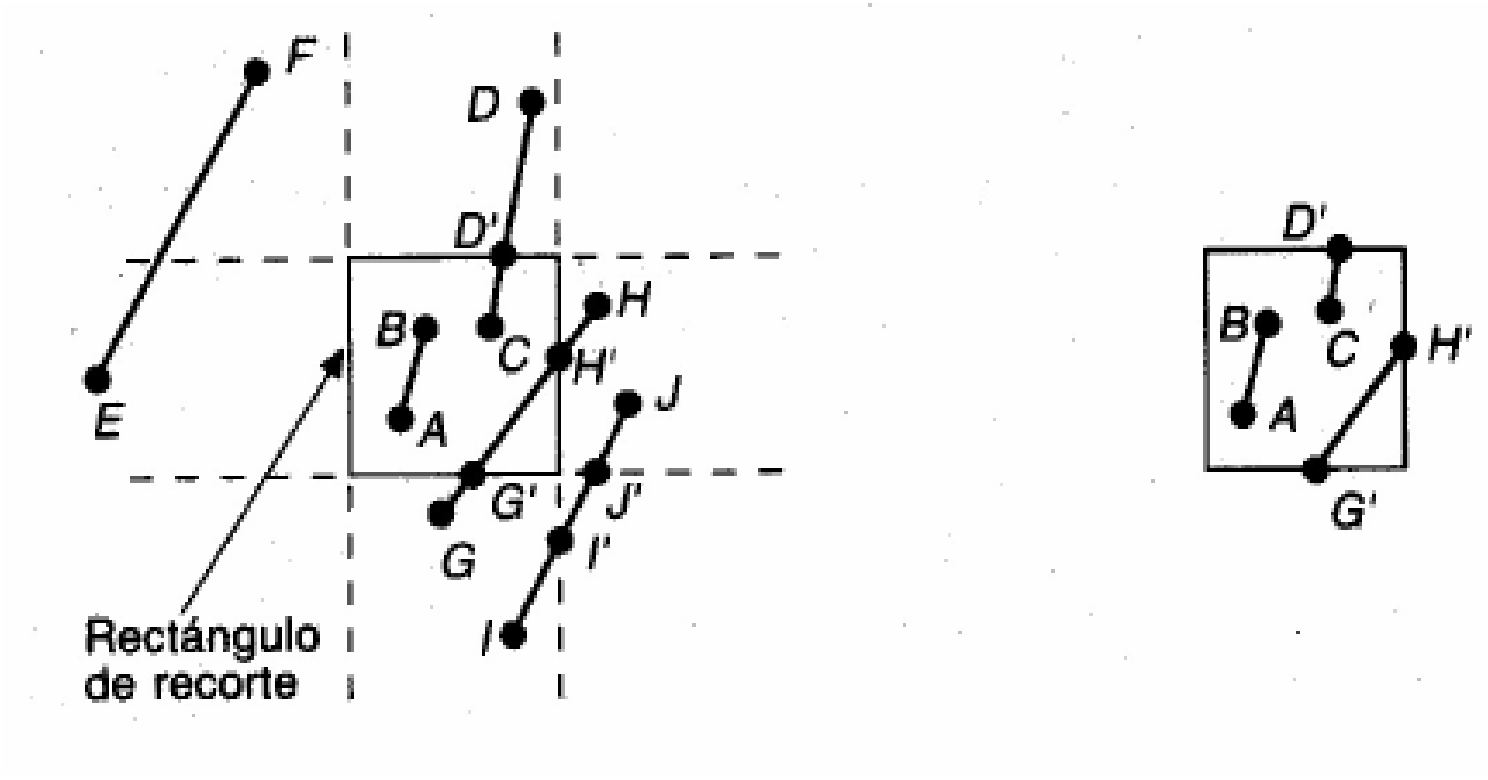
Si *m* es la pendiente del segmento de recta entre los puntos (x_1,y_1) y (x_2,y_2), entonces si $x_1 \neq x_2$ $m = (y_2 - y_1) / (x_2 - x_1)$ y para cualquier otro punto (x,y) sobre la recta $m = (y - y_1) / (x - x_1)$

Entonces obtenemos los valores de **x** e **y** para la intersección con los bordes de la ventana.

$$y = m * (x - x_1) + y_1 \qquad y$$
$$x = 1/m * (y - y_1) + x_1$$

Recorte de líneas.

- Las líneas que intersecan una región de recorte rectangular siempre se recortan a un solo segmento de línea.
- Para recortar una línea sólo hay que considerar sus dos puntos extremos.



Algoritmo de recorte de líneas de Cohen-Sutherland

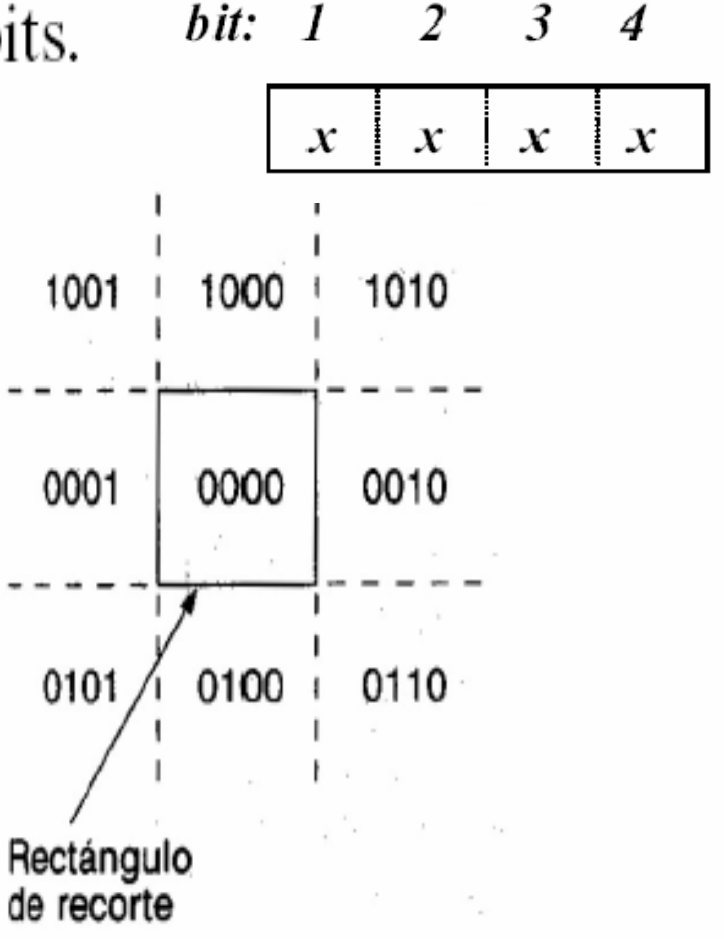
•Este algoritmo iterativo revisa en primer lugar el par de puntos extremos para determinar la posibilidad de su aceptación trivial.

• Si la línea no puede aceptarse trivialmente, se efectúan entonces revisiones de región, para comprobar si se puede rechazar trivialmente.

  – Por ejemplo, si los dos puntos tienen una coordenada *x* menor que *x*min, todo el segmento quedaría a la izquierda del rectángulo de recorte, y se rechazaría de forma trivial, no hay que recortarlo ni dibujarlo.

• Si el segmento de línea no puede aceptarse o rechazarse trivialmente, se divide en dos segmentos con respecto a una arista de recorte, de manera que uno de ellos pueda rechazarse trivialmente.

- Aceptación y rechazo trivial:
  - Se divide el plano en nueve regiones, y a cada una se le asigna un código de cuatro bits.
    - Primer bit $y > y_{max}$
    - Segundo bit $y < y_{min}$
    - Tercer bit $x > x_{max}$
    - Cuarto bit $x < x_{min}$



Ejemplo:

Si $y>y_{max}$ { V ⇒ 1 ; F ⇒ 0

- Aceptación y rechazo trivial (cont.):
  - Se determina entonces qué código le corresponde a cada extremo de la línea:
    - Si ambos son 0000, entonces la línea es aceptada de manera trivial.
    - Si no, se calcula la operación AND a nivel de bits entre los dos códigos y si el resultado es distinto de 0000 entonces la línea es rechazada de manera trivial.Si no, habrá que dividir.
      - Por ejemplo, un extremo podría tener código 1001 y el otro 0001, el resultado del AND sería entonces 0001 y la línea sería rechazada por quedar completamente a la izquierda.

- Dividir la línea en dos:
  - Si no podemos aceptar ni rechazar trivialmente, hay que dividir la línea en dos segmentos, de manera que uno pueda rechazarse de manera trivial y continuar el recorte con el otro.
  - Tomamos como código de región para la decisión el de un extremo que no sea 0000 (si los dos fueran 0000 la línea sería aceptada de manera trivial).
- Dividir la línea en dos (cont.):
  - Se evalúan entonces los bits del código de región elegido, siguiendo el orden que se desee.
    - Por ejemplo, podría seguirse el orden arriba, abajo, izquierda y derecha.
  - Un bit a 1 representa una arista respecto a la cual podemos recortar.
    - Por ejemplo, si recortamos contra la arista superior, entonces nos quedaríamos con el trozo de segmento que quede debajo, y realizamos una nueva iteración del algoritmo con ese trozo.

Algoritmo de Cohen-Sutherland

Ejemplos:

→ *AB* es trivialmente aceptada porque

A: 0000

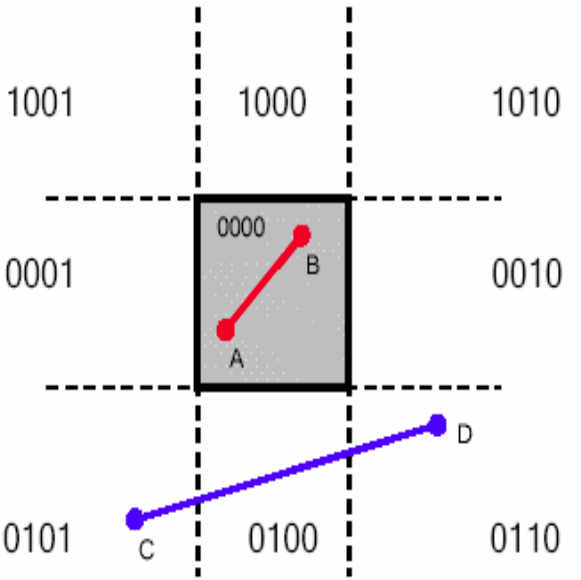
B: 0000

→ CD es trivialmente rechazada porque

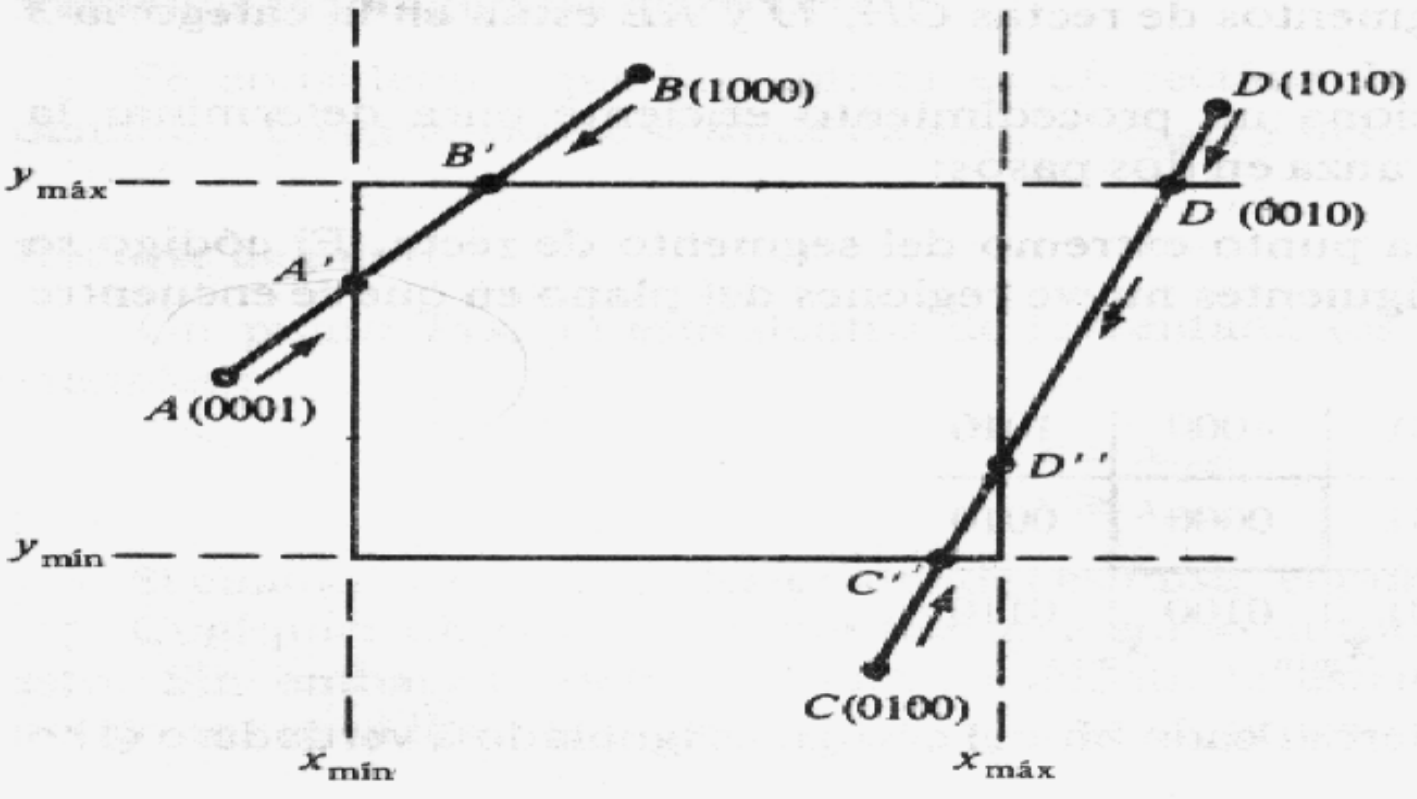
D: 0110

C: 0101

AND = 0100



Si el bit 1 es 1, se interseca con la línea $y = y_{máx}$.
Si el bit 2 es 1, se interseca con la línea $y = y_{mín}$.
Si el bit 3 es 1, se interseca con la línea $x = x_{máx}$.
Si el bit 4 es 1, se interseca con la línea $x = x_{mín}$.



Pseudo código para el Algoritmo de recorte de **Cohen-Sutherland**

$y = y_0 + m * (x - x_0)\ y$

$x = x_0 + (1/m) * (y - y_0)$

ComputeOutCode(x_0, y_0, outcode0)

ComputeOutCode(x_1, y_1, outcode1)

repeat

check for trivial reject or trivial accept

pick the point that is outside the clip rectangle

if TOP then $x = x_0 + (x_1 - x_0) * (y_{max} - y_0)/(y_1 - y_0)$; $y = y_{max}$;

else if BOTTOM then $x = x_0 + (x_1 - x_0) * (y_{min} - y_0)/(y_1 - y_0)$; $y = y_{min}$;

else if RIGHT then $y = y_0 + (y_1 - y_0) * (x_{max} - x_0)/(x_1 - x_0)$; $x = x_{max}$;

else if LEFT then $y = y_0 + (y_1 - y_0) * (x_{min} - x_0)/(x_1 - x_0)$; $x = x_{min}$;

end {calculate the line segment}

if (outcode1 = outcode0) **then** $x_0 = x$; $y_0 = y$;

ComputeOutCode(x_0, y_0, outcode0)

else $x_1 = x$; $y_1 = y$;

ComputeOutCode(x_1, y_1, outcode1)

end {Subdivide}

until done

- Ventajas:
  - Sencillo, muy eficiente con rectángulos de recorte muy grandes o muy pequeños, donde hay más probabilidad de aceptación o rechazo trivial.
- Desventaja:
  - Puede efectuar recortes innecesarios.

ALGORITMO DE RECORTE DE SUTHERLAND-HODGEMAN (POLÍGONOS)

las siguiente figuras 2-2a a 2-2b representan dos polígonos y los resultados del recorte.

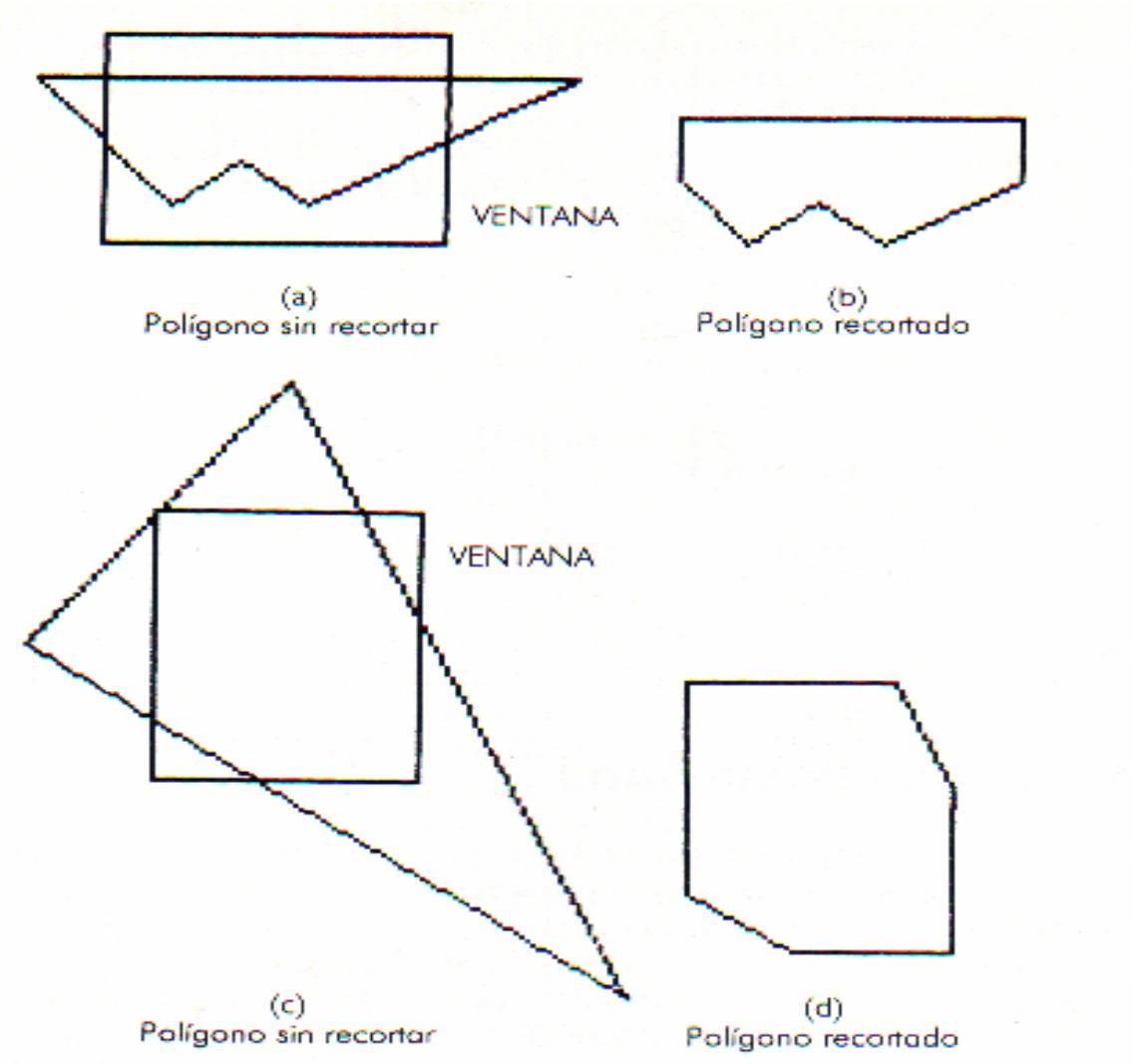


figura 2-2

El algoritmo consiste en cortar un polígono contra cada lado de la ventana, uno a la vez. Específicamente, para cada lado de la ventana introduce una lista de vértices y proporciona una lista nueva. La cual es sometida al algoritmo para recorta contra el siguiente lado de la ventana. **La lista de entrada** es una secuencia de vértices consecutivos del polígono, obtenida a partir del recorte en el lado anterior. El primer lado de la ventana tiene como entrada el conjunto original de vértices. Después del recorte contra el ultimo lado, **la lista de salida** se compone de los vértices que describen el polígono recortado.

Algoritmo.

1) Prueba un par de vértice consecutivos contra un lado de la ventana (figura 2-3) y tiene

4 casos posibles, El polígono ejemplificado tiene 4 vértices ***{v1, v2, v3, v4}***

caso I) ***v1*** **y** ***v2*** dentro de la ventana y ***v2*** es enviado a la lista de salida,

caso II) ***v2*** dentro y ***v3*** fuera de la ventana. El punto de intercepción, ***i1***, de lado del polígono que une a los vértices y el lado es agregado a la lista de salida.

caso III) **v3 y v4** fuera de la ventana y ningún punto es sacado

caso IV) ***v4*** esta fuera y ***v1*** dentro de la ventana. El punto de intercepción, ***i2***, y el segundo vértice ***v1*** son agregados a la lista de salida. El resultado de este recorte izquierdo es la transformación de la **lista de entrada *{v1, v2, v3, v4}*** a la **lista de salida *{v2,i1,i2,v1}*** figura 2-4 ilustra recortes sucesivos contra cuatro fronteras.

Se recurre al algoritmo 4 veces, uno para cada lado de la ventana.

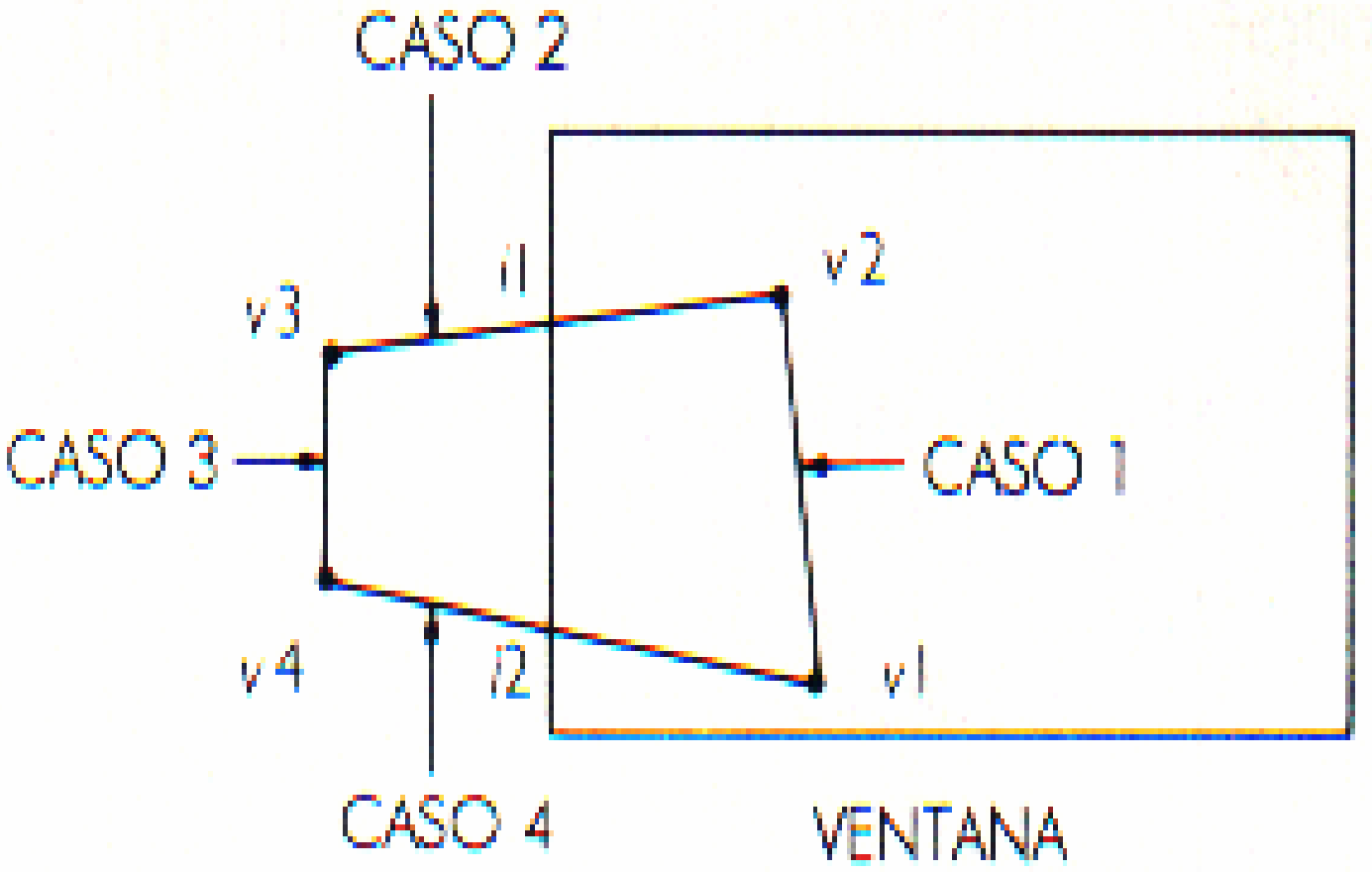


Figura 2-3

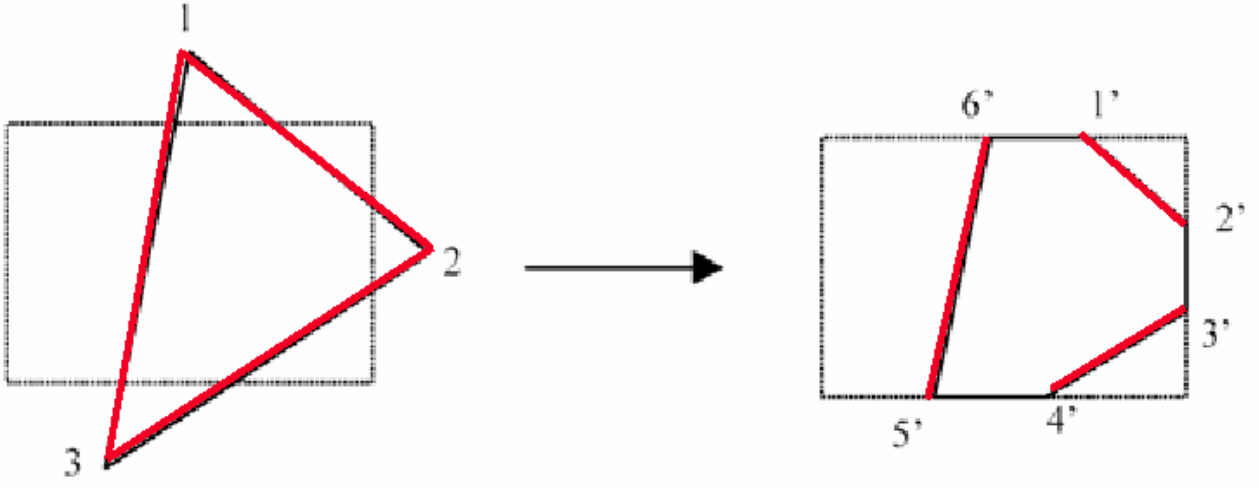
Algoritmo de Sutherland-Hodgeman

Dado el polígono que se ve en la figura:

Polígono de entrada: <1, 2, 3> donde 12, 23, 31 son lados

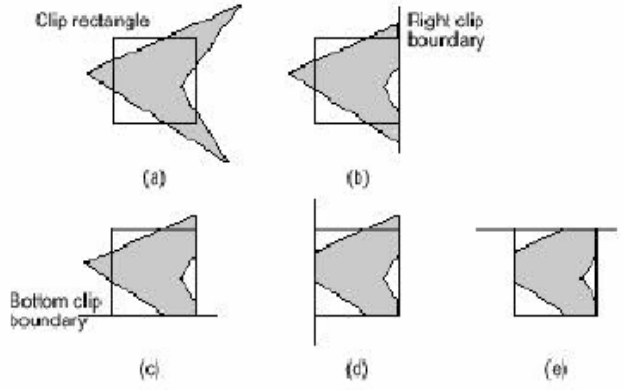
El clipping de polígonos debe hacerse de modo tal que se obtenga el polígono:

Polígono de salida: <1', 2', 3', 4', 5', 6'>

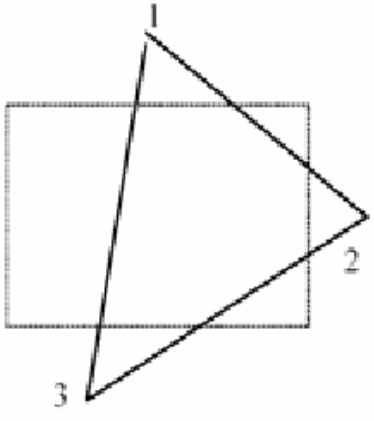


Algoritmo de Sutherland-Hodgeman

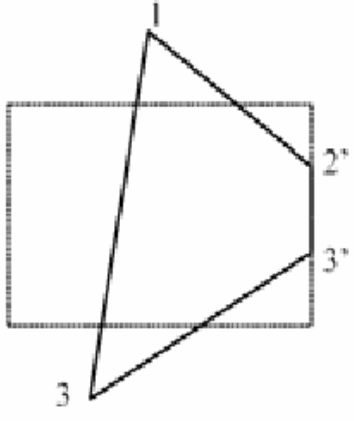
El algoritmo de Sutherland-Hodgeman procesa cada polígono, en orden, con cada uno de los bordes.



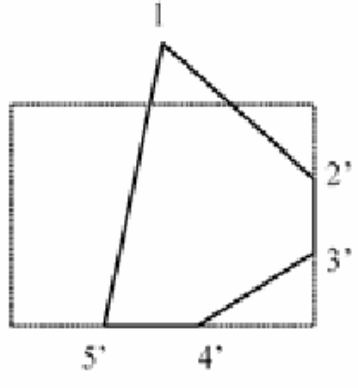
Izquierdo



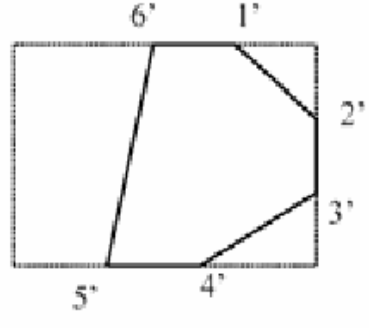
Derecho



Abajo

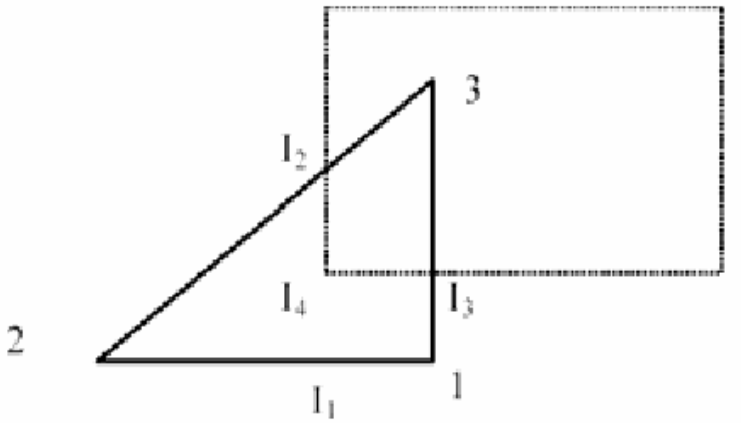


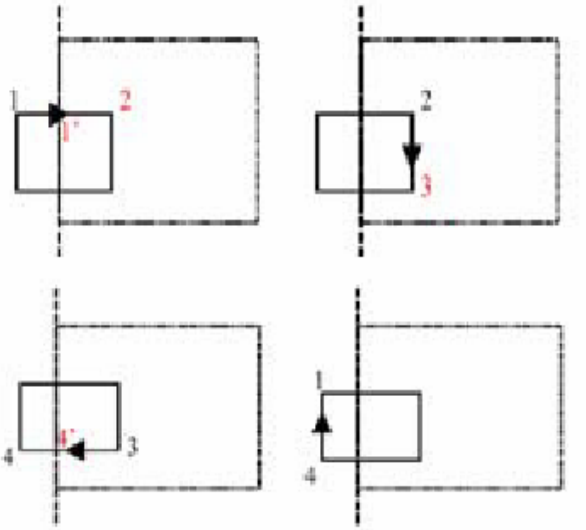
Arriba

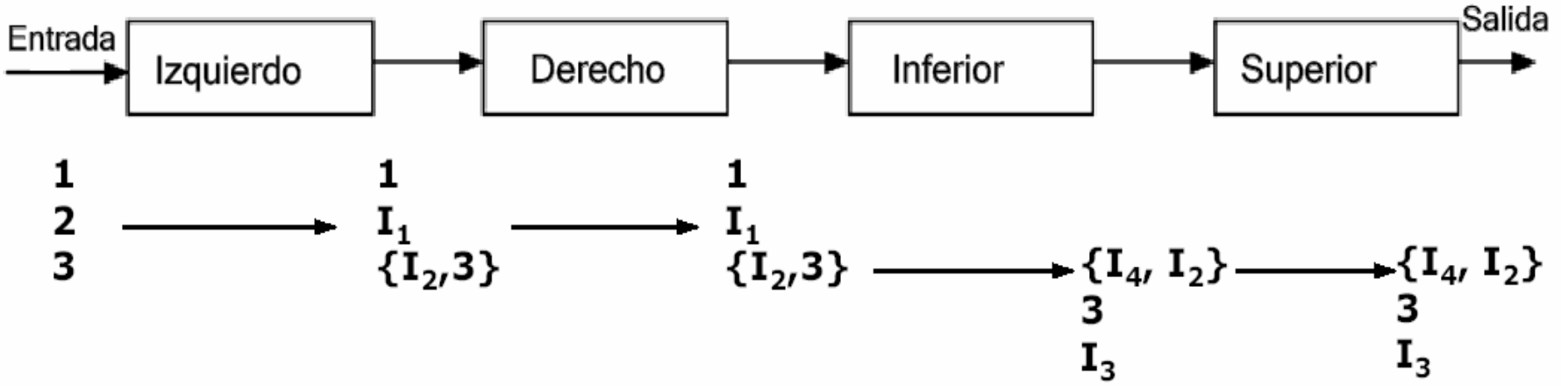


Algoritmo de Sutherland-Hodgeman

Ejemplo







2.3 TRANSFORMACIONES GEOMÉTRICAS EN 2D

Introducción a las transformaciones en dos dimensiones (2D)

- Un Sistema gráfico debería permitir la definición de objetos o imágenes que incluyan una serie de transformaciones.
- Estas transformaciones son el medio para construir o modificar imágenes u objetos.
- Una *rotación, traslación y escalamiento* entre otras, son tales transformaciones.
- Cada transformación utiliza un punto (x, y) para generar un nuevo punto (x', y').

Transformaciones en dos dimensiones

Los objetos se definen mediante un conjunto de puntos. Las transformaciones son procedimientos para calcular nuevas posiciones de estos puntos, cambiado el tamaño y orientación del objeto.

Las operaciones básicas de transformación son

- Traslación
- Escalamiento
- Rotación.

Composición de funciones

Si el proceso efectuado por una función H puede describirse por los pasos sucesivos de aplicar primero una función G y después una función F a los resultados de G, se dice que H es la *composición* de F y G. Se escribe $H = F \circ G$. Si la entrada a la función se denota como x, la salida $H(x)$ se evalúa por

$$H(x) = F[G(x)]$$

Esto es, G opera primero sobre x; después el resultado $G(x)$ se pasa a F como entrada.

En general, la composición de funciones no es conmutativa; esto es, $F \circ G \neq G \circ F$.

El concepto de composición no está restringido a sólo dos funciones, sino que se amplía a cualquier número de funciones. Para aquellas funciones que están representadas por matrices, la composición de funciones equivale a la multiplicación matricial; esto es, $\mathbf{A} \circ \mathbf{B} = \mathbf{AB}$.

EJEMPLO 13.

1. Si $f(x) = x^2 + 2$ y $g(x) = 2x + 1$, entonces $f[g(x)] = [g(x)]^2 + 2 = (2x + 1)^2 + 2 = 4x^2 + 4x + 3$.
2. Si

$$\mathbf{A} = \begin{pmatrix} 1 & 3 \\ 0 & 2 \end{pmatrix} \qquad \text{y} \qquad \mathbf{B} = \begin{pmatrix} -5 & 4 \\ 2 & 2 \end{pmatrix}$$

entonces

$$\mathbf{A} \circ \mathbf{B} = \mathbf{AB} = \begin{pmatrix} 1 & 3 \\ 0 & 2 \end{pmatrix}\begin{pmatrix} -5 & 4 \\ 2 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 10 \\ 4 & 4 \end{pmatrix}$$

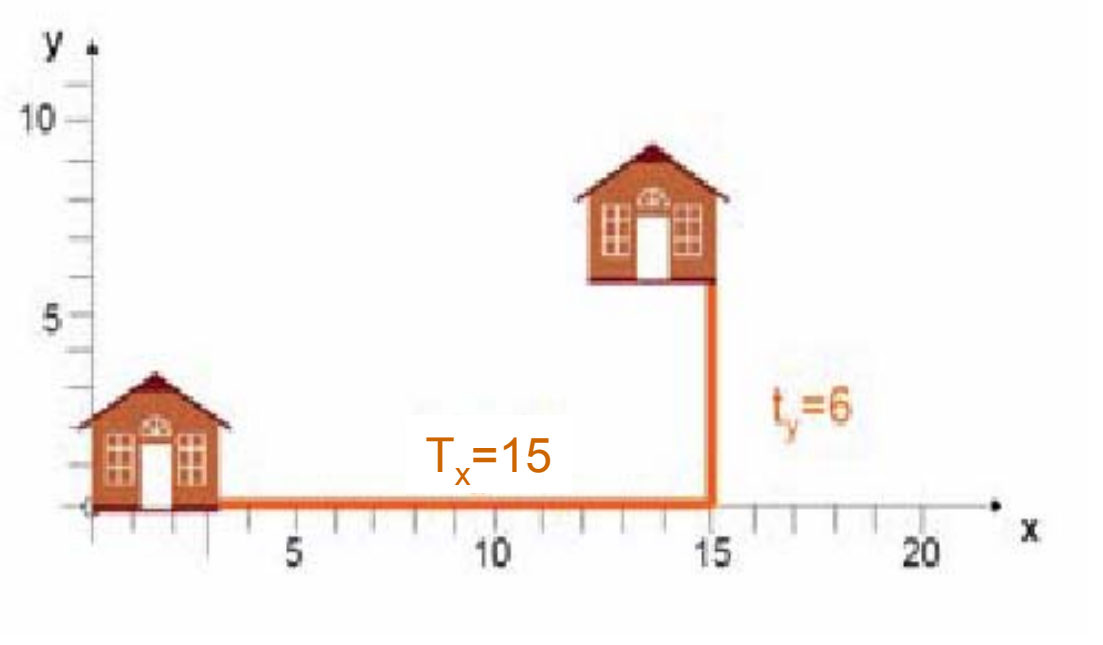
Traslación

Las coordenadas (x, y) de un objeto se transforman a (x', y') de acuerdo a las fórmulas:

$$x' = x + T_x$$

$$y' = y + T_y$$

El par (T_x, T_y) se conoce como *vector de traslación* o *vector de cambio*



Representación matricial de traslaciones

Haciendo uso de coordenadas homogéneas la traslación puede representarse como:

$$x' = x + t_x$$
$$y' = y + t_y$$

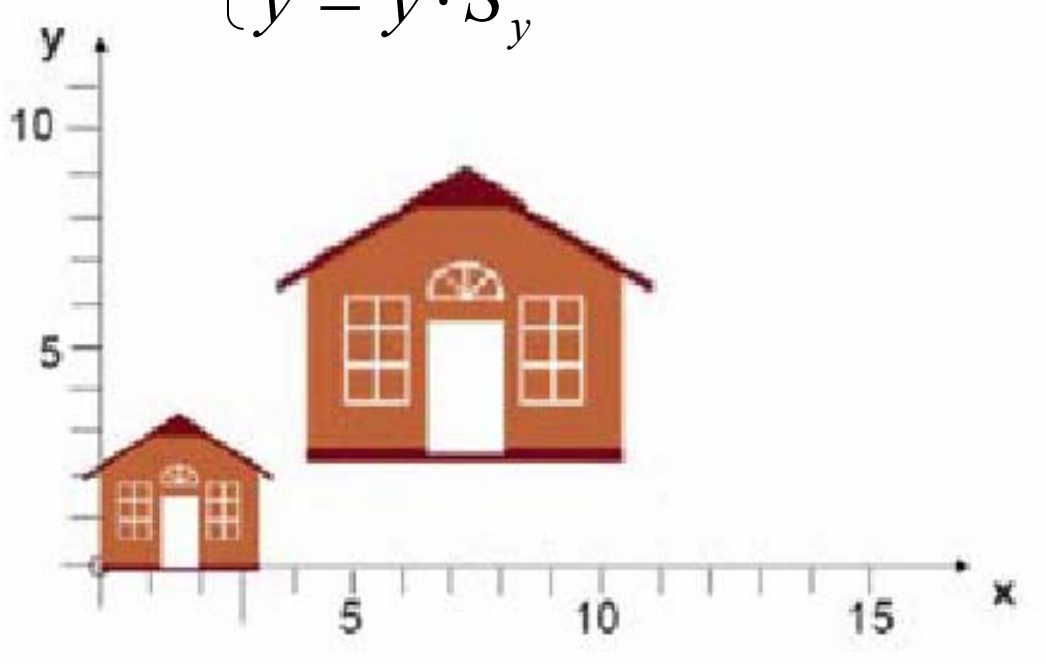
$$\Rightarrow \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{T}(t_x, t_y) \cdot \mathbf{P}$$

Escalamiento

El escalamiento modifica el tamaño de un polígono. Para obtener este efecto, se multiplica cada par coordenado (x, y) por un factor de escala en la dirección x y en la dirección y para obtener el par (x', y').

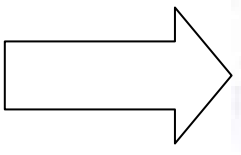
Las fórmulas son $\begin{cases} x' = x \cdot S_x \\ y' = y \cdot S_y \end{cases}$



Representación matricial de escalamientos

Haciendo uso de coordenadas homogéneas el escalamiento puede representarse como:

$X' = S_x X$
$Y' = S_y Y$



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{S}(s_x, s_y) \cdot \mathbf{P}$$

Rotación

La rotación gira los puntos de una figura alrededor de un punto fijo. De la figura se obtiene

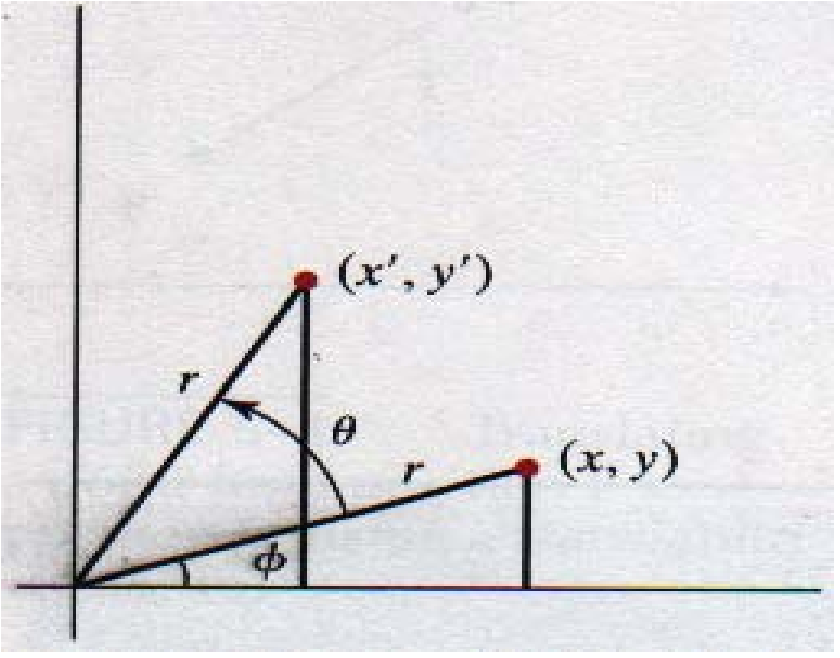
$$x' = r\cos(\phi + \theta) = r\cos\phi\cos\theta - r\,\text{sen}\,\phi\,\text{sen}\,\theta$$

$$y' = r\,\text{sen}(\phi + \theta) = r\,\text{sen}\,\phi\cos\theta + r\cos\phi\,\text{sen}\,\theta$$

Simplificando

$$x' = x\cos\theta - y\,\text{sen}\,\theta$$

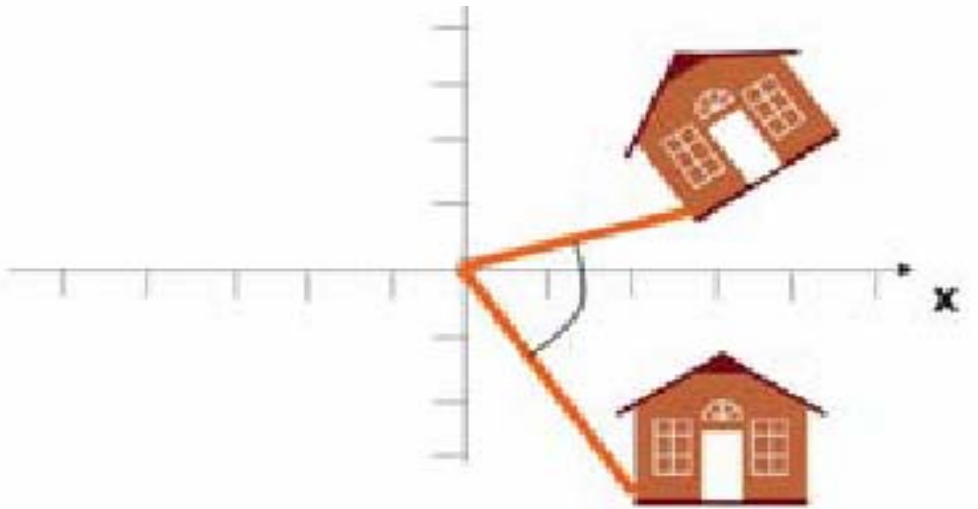
$$y' = y\cos\theta + x\,\text{sen}\,\theta$$



$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

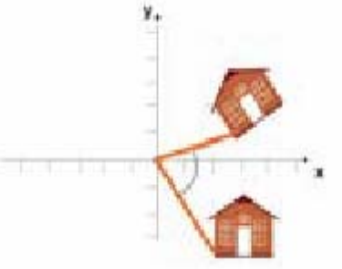
$$\mathbf{P}' = \mathbf{R} \cdot \mathbf{P}$$

$$\begin{aligned} x' &= x\cos\alpha - y\sin\alpha \\ y' &= x\sin\alpha + y\cos\alpha \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



En resumen

Si tenemos las transformaciones :



$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \qquad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} e_x & 0 \\ 0 & e_y \end{pmatrix} * \begin{pmatrix} x \\ y \end{pmatrix} \qquad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} * \begin{pmatrix} x \\ y \end{pmatrix}$$

… podemos escribir cada punto transformado como:

$$p' = p + T \qquad p' = E\,p \qquad p' = Rp$$

Escalamiento respecto a un punto fijo

El procedimiento de escalamiento respecto a un punto fijo es el siguiente:

1. Trasladando primero ese punto al origen
2. después escalando
3. luego regresando el objeto a la posición original.

Las ecuaciones son

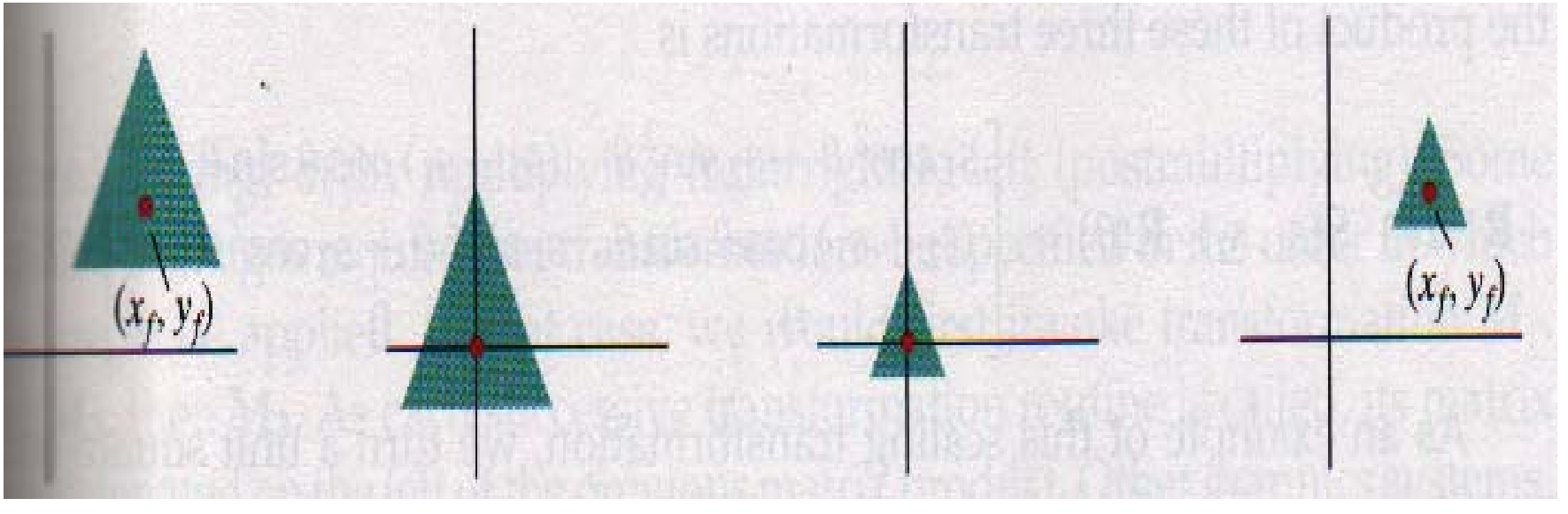
$$x' = x_F + (x - x_F)S_x, \quad y' = y_F + (y - y_F)S_y$$

Reacomodando

$$x' = x \cdot S_x + (1 - S_x) x_F$$
$$y' = y \cdot S_y + (1 - S_y) y_F$$

En forma matricial

$$\begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & x_f(1 - s_x) \\ 0 & s_y & y_f(1 - s_y) \\ 0 & 0 & 1 \end{bmatrix}$$

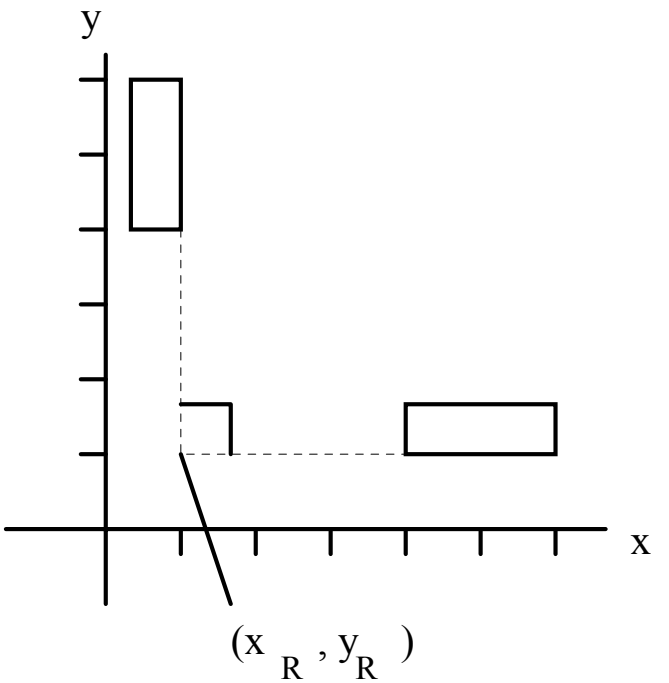


Rotación respecto a un punto arbitrario

La rotación respecto a un punto arbitrario es

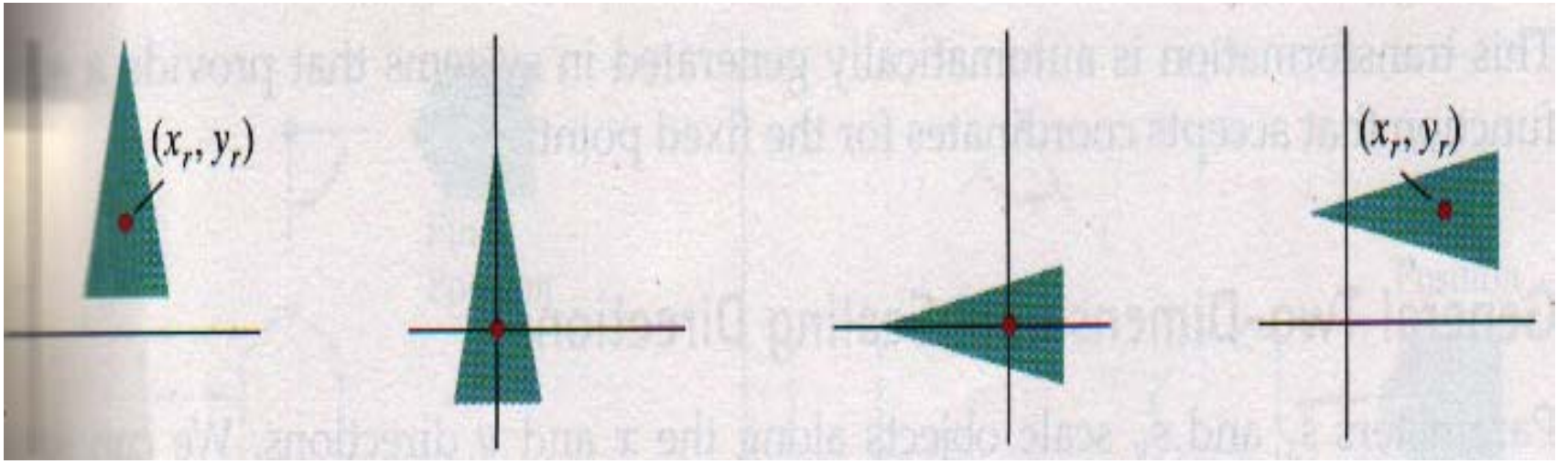
$$x' = x_R + (x - x_R)\cos\theta - (y - y_R)\,\text{sen}\,\theta$$

$$y' = y_R + (y - y_R)\cos\theta + (x - x_R)\,\text{sen}\,\theta$$



$$\begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & x_f(1-s_x) \\ 0 & s_y & y_f(1-s_y) \\ 0 & 0 & 1 \end{bmatrix}$$

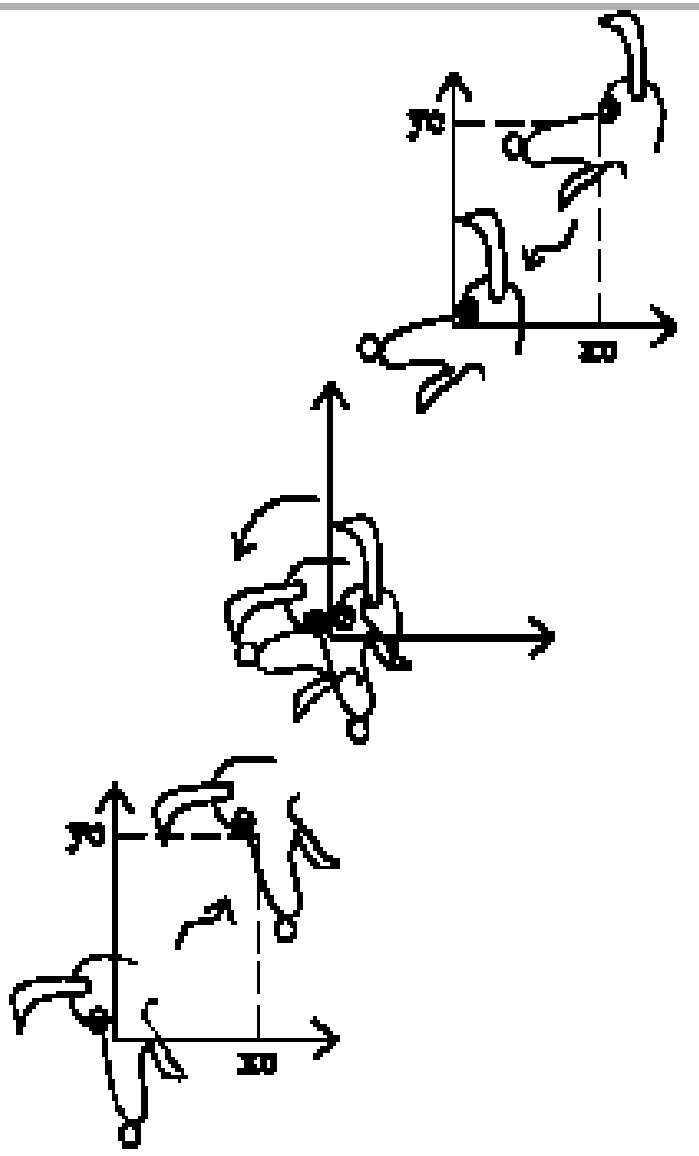
$$\mathbf{T}(x_f, y_f) \cdot \mathbf{S}(s_x, s_y) \cdot \mathbf{T}(-x_f, -y_f) = \mathbf{S}(x_f, y_f, s_x, s_y)$$



Concatenación de transformaciones

- Concatenación de transformaciones
  - Podemos combinar varias transformaciones para obtener operaciones más complejas
  - Por ejemplo -> Rotación respecto a un punto cualquiera (x_c , y_c)
    - *En tres pasos:*
      - Traslación $(-x_c , -y_c)$,
      - Rotación α grados
      - Traslación (x_c , y_c)
    - *Como las matrices son cuadradas se obtiene una única matriz*

$$P3 = T(x_c , y_c) \cdot R(\alpha) \cdot T(-x_c , -y_c) \cdot P$$



Composición de transformaciones

Para aplicar varias transformaciones a un conjunto de puntos basta con combinar las matrices de transformación en una sola, mediante multiplicación matricial. En caso de tener solo transformaciones del mismo tipo, la combinación sigue reglas muy simples.

Traslación:

$$\begin{cases} P' = T(T_{x2}, T_{y2}) \cdot \{T(T_{x1}, T_{y1}) \cdot P\} = \{T(T_{x2}, T_{y2}) \cdot T(T_{x1}, T_{y1})\} \cdot P \\ T(T_{x2}, T_{y2}) \cdot T(T_{x1}, T_{y1}) = T(T_{x1} + T_{x2}, T_{y1} + T_{y2}) \end{cases}$$

Escalamiento: $S(S_{x2}, S_{y2}) \cdot \{S(S_{x1}, S_{y1})P\} = S(S_{x1} \cdot S_{x2}, S_{y1} \cdot S_{x2})P$

Rotación:

$$\begin{cases} P' = R(\theta_2) \cdot \{R(\theta_1) \cdot P\} = \{R(\theta_2) \cdot R(\theta_1)\} \cdot P \\ R(\theta_2) \cdot R(\theta_1) = R(\theta_1 + \theta_2) \end{cases}$$

Otras transformaciones

Otras transformaciones que permiten llevar a cabo operaciones muy útiles, estas son:

- Reflexiones
- Corte.

Reflexiones en *x* y *y*

Las reflexiones respecto al eje *x* y *y* se obtienen con las matrices siguientes:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

X=0

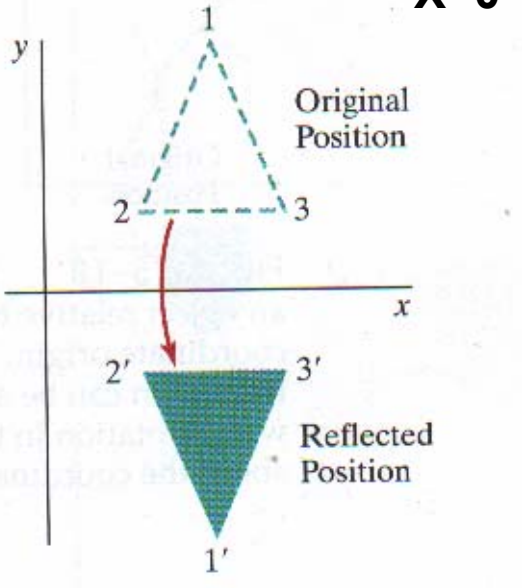


FIGURE 5-16 Reflection of an object about the *x* axis.

Y=0

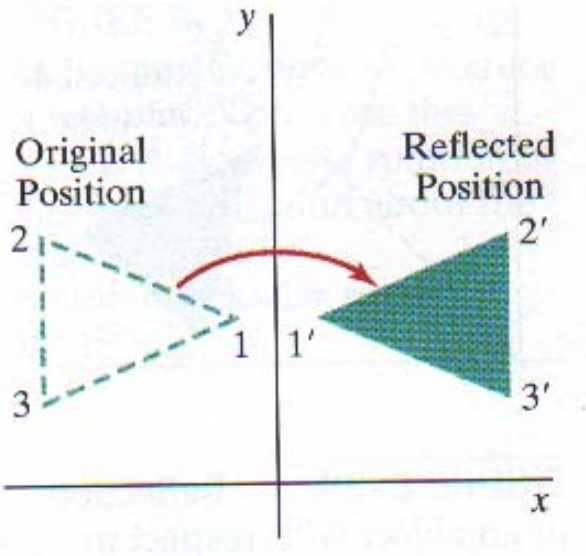
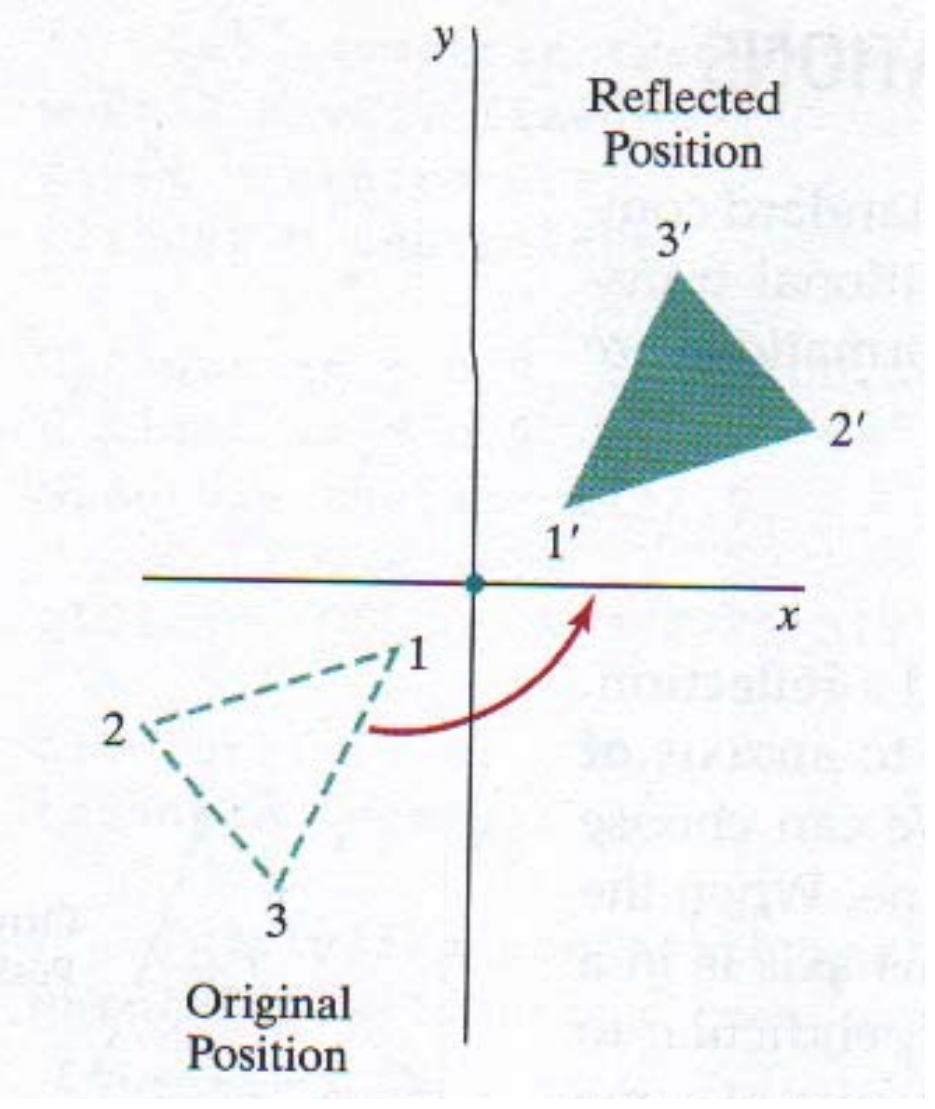


FIGURE 5-17 Reflection of an object about the *y* axis.

Reflexión respecto al origen

La reflexión respecto al origen se obtiene con :

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



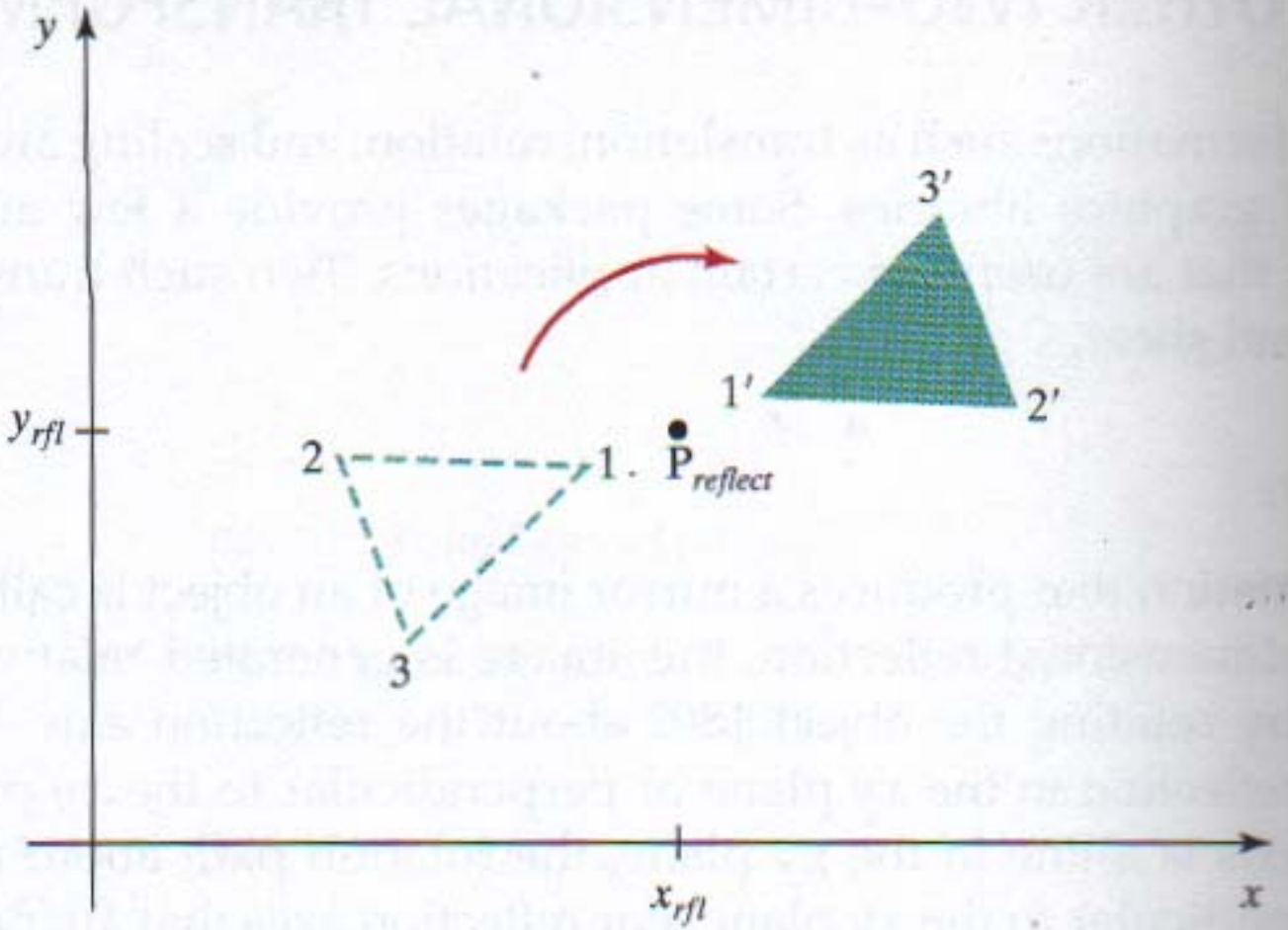
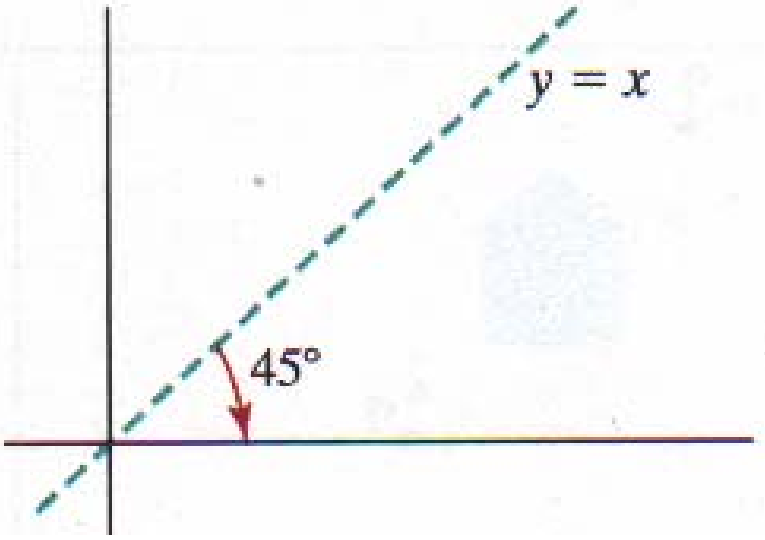


FIGURE 5-19 Reflection of an object relative to an axis perpendicular to the xy plane and passing through point $\mathbf{P}_{\text{reflect}}$.

Reflexión respecto a la recta $y = x$

Una reflexión respecto a la recta $y = x$, puede obtenerse en tres pasos: girar un ángulo de 45 en el sentido de las manecillas del reloj, una reflexión respecto al eje x, y una rotación de 45 grados en contra del sentido del reloj.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



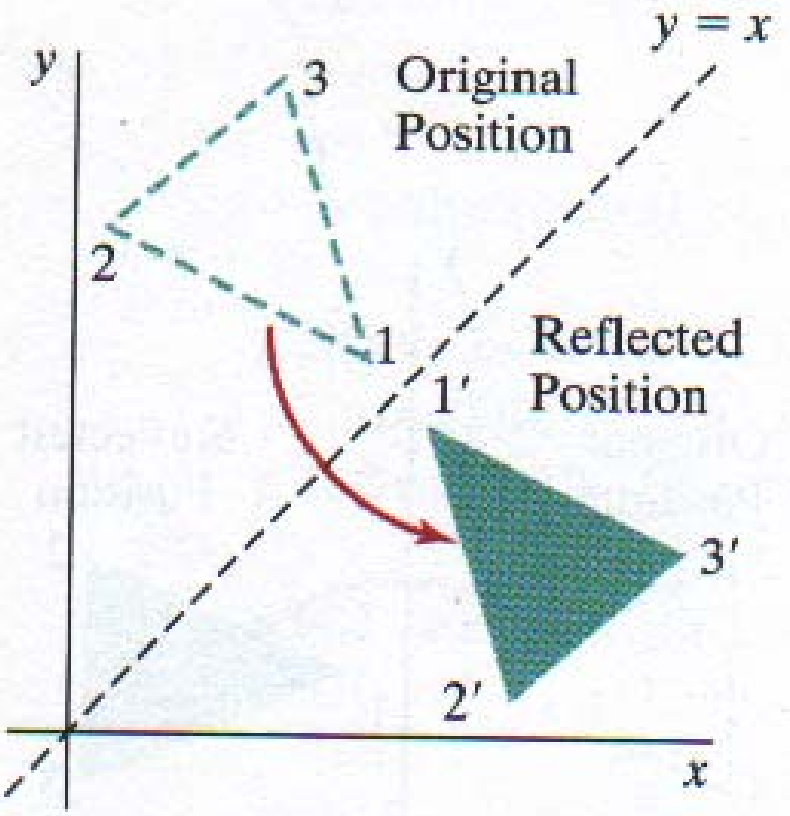
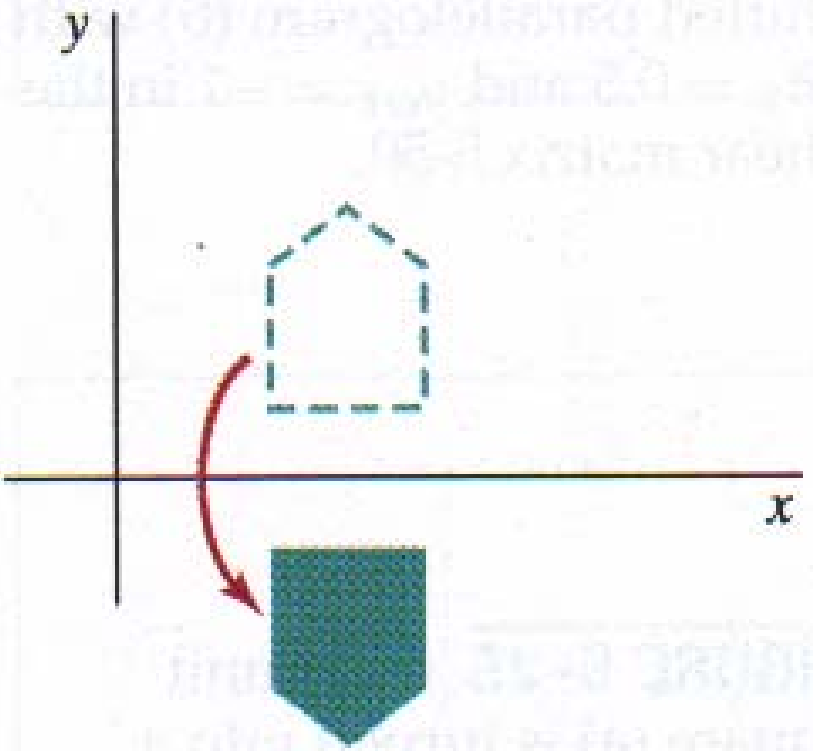
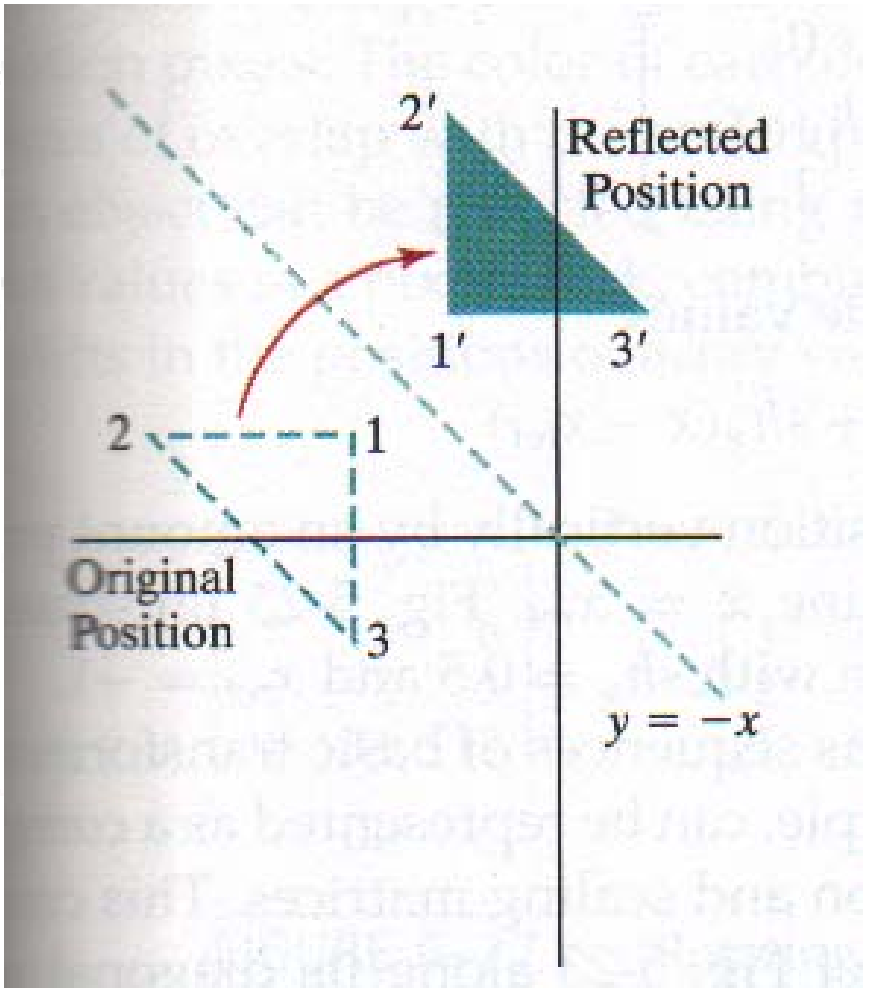


FIGURE 5-20 Reflection of an object with respect to the line $y = x$.

a)



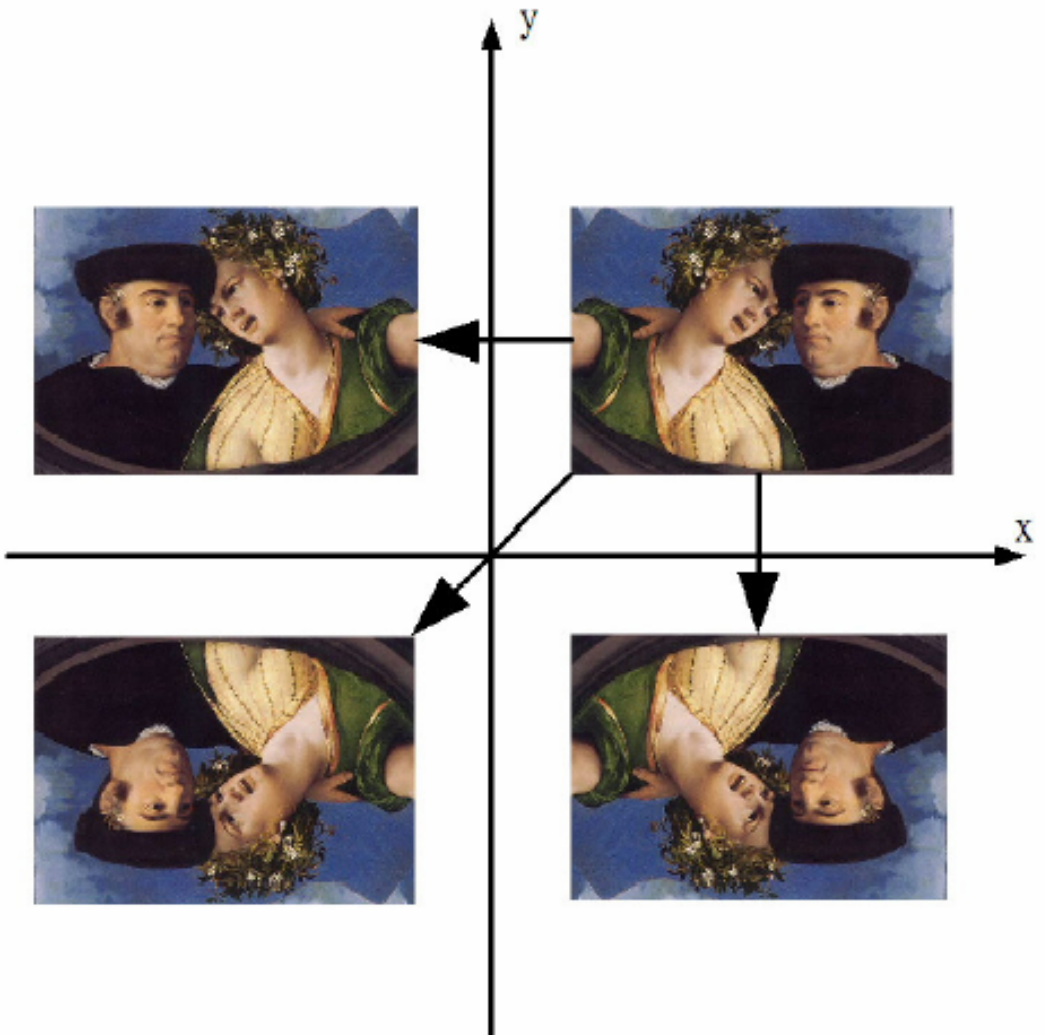
b)



c)

Reflexión en 2D

Corresponde a escalar usando coeficientes negativos:



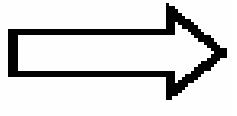
Sesgado en *x*

El sesgado produce una deformación similar al deslizamiento de una capa sobre otra. El corte en *x* se produce por la matriz:

$$x' = x + sh_x \cdot y, \qquad y' = y$$

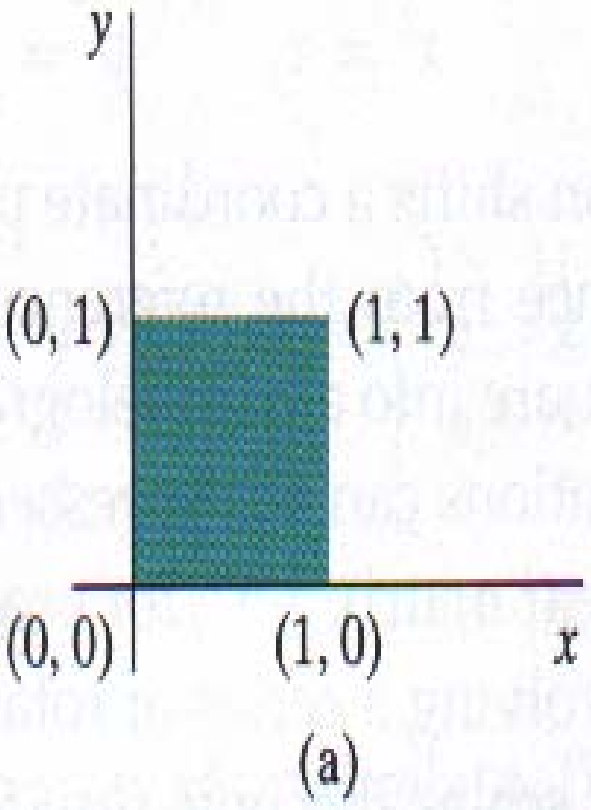
$$x' = x + \mathbf{Shx}$$

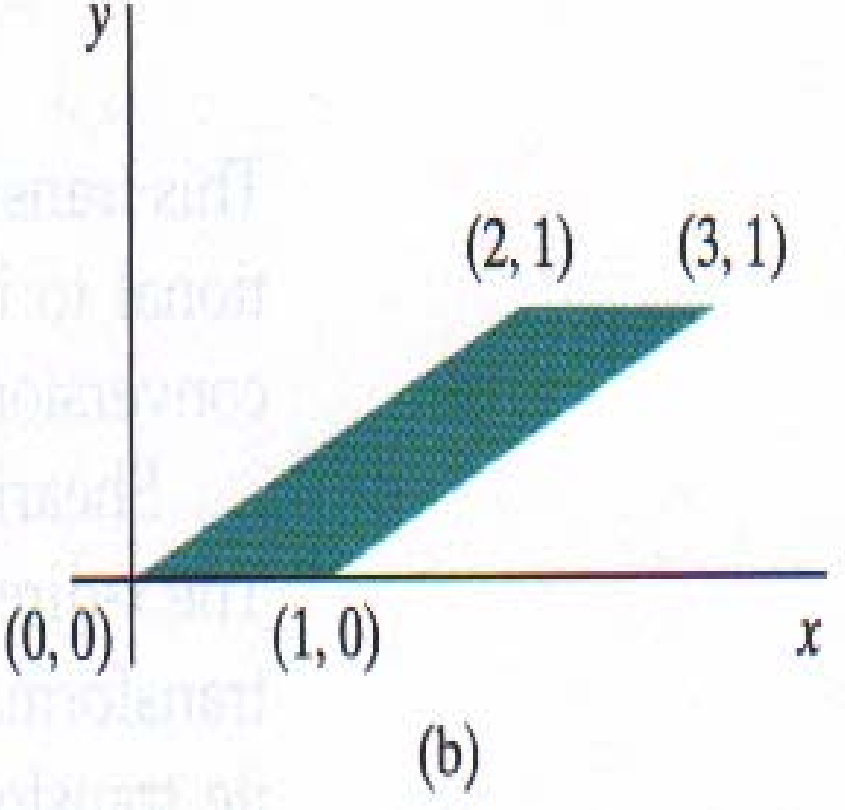
$$y' = y$$



$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$Sh_x=2$



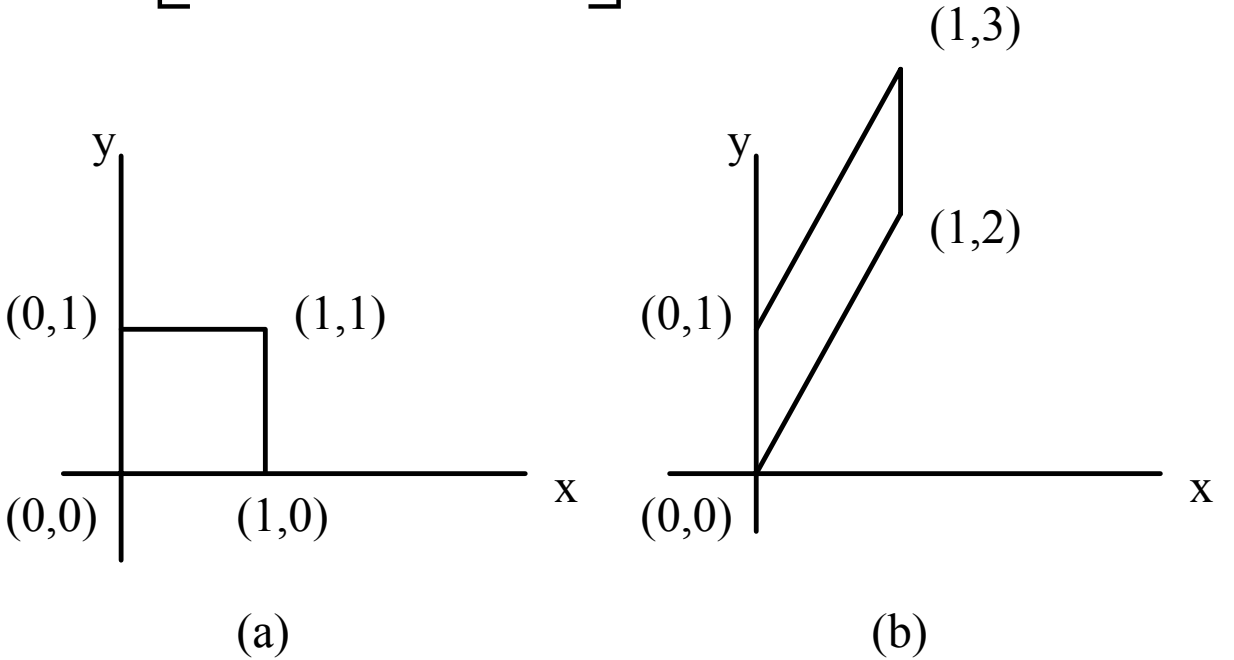


Sesgado en *y*

El sesgado en *y* se produce por la matriz

$$\begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

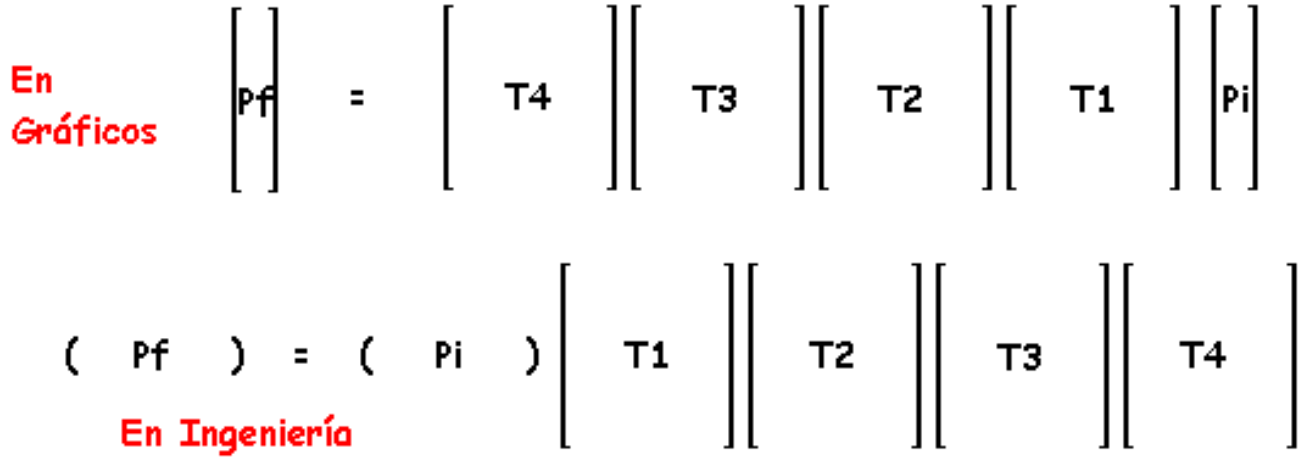
$Sh_y=2$



Premultiplicación y postmultiplicación

Existen dos convenciones en cuanto a uso de transformaciones geométricas: la de Robótica / Ingeniería y la de Gráficos. En ambos casos se realizan exactamente las mismas operaciones pues tanto puedo querer mover un brazo robot como un personaje sobre mi juego 3D. Pero en cada caso se sigue una metodología distinta.

En la convención de Gráficos, se postmultiplican las matrices, que los puntos se toman como vectores en columna que se multiplican a las matrices por la derecha. Y además el orden de las transformaciones, de primera a última a aplicar, es de derecha a izquierda.



Concepto de "pila" o "stack“

La función **glPushMatrix**() realiza una copia de la matriz superior y la pone encima de la pila, de tal forma que las dos matrices superiores son iguales. En la figura 1 se observa la pila en la situación inicial con una sola matriz, al llamar a la función **glPushMatrix**() se duplica la matriz superior. Las siguientes transformaciones que se realizan se aplican sólo a la matriz superior de la pila, quedando la anterior con los valores que tenía en el momento de llamar a la función **glPushMatrix**().

La función **glPopMatrix**() elimina la matriz superior, quedando en la parte superior de la pila la matriz que estaba en el momento de llamar a la función **glPushMatrix**().

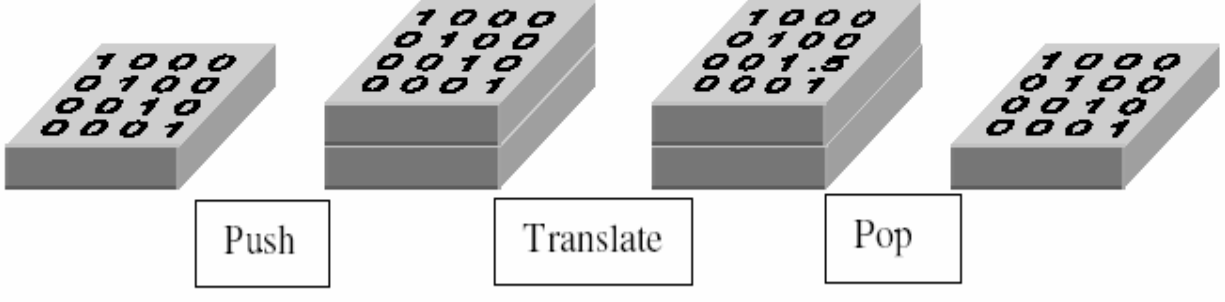


Figura PushMatrix y PopMatrix

Concepto de "pila" o "stack“

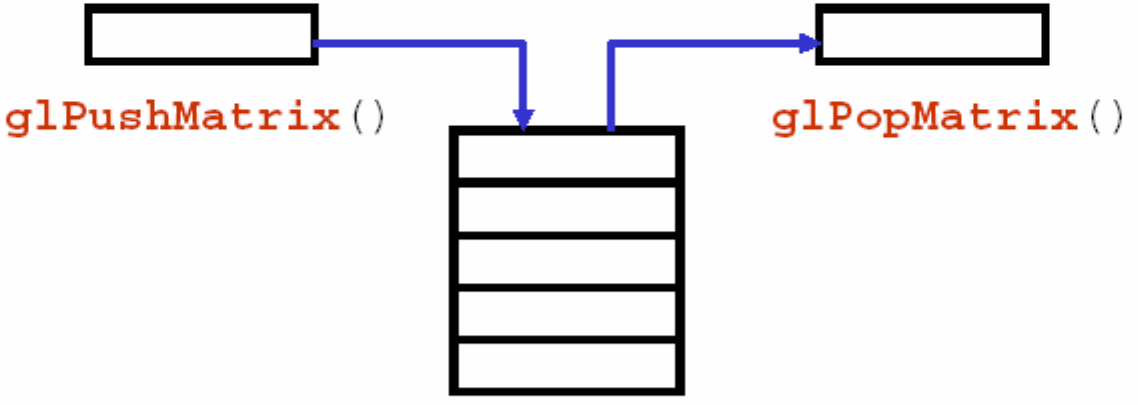
La matriz de transformación, la "model-view" debe entenderse como una pila.
Pues bien, cada transformación que añadimos entra a la pila como la última y por tanto al salir será la primera. Ahí tenéis el porque OpenGL funciona tal y como se ha comentaba.
Podemos salvar el estado de la pila en cualquier momento para recuperarlo después. Esto la haremos mediante las funciones:

```
glPushMatrix( ); /* Salvamos el estado actual de la matriz */
glPopMatrix( ); /* Recuperamos el estado de la matriz */
```

Esto nos servirá en el caso de que tengamos que aplicar algunas transformaciones a una pequeña parte de la geometría. El resto no debiera verse afectado por esos cambios. Lo que se hace es definir las transformaciones generales que afectan a todos. Entonces se salva la matriz y se añaden otras. Se dibuja la geometría "especial" y inmediatamente después se recupera la matriz. Ahora podemos dibujar todo el resto estando tranquilos pues no se verá afectado por las transformaciones que hayamos definido entre el glPush... y el glPop...

La pila de matrices

- No siempre es deseable reiniciar por completo la matriz de modelado
- A veces es preferible querer almacenar la matriz actual, y volverla a recuperar más adelante
- OpenGL mantiene una pila de matrices



- Para conocer la profundidad máxima de la pila:

```
glGet(GL_MAX_MODELVIEW_STACK_DEPTH)
```

mini-ejemplo:

```
.....
glRotatef... /* afectará a toda la geometría que dibuje a partir de ahora */
glTranslatef.... /* afectará a toda la geometría que dibuje a partir de ahora */
glPushMatrix( ); /* salvo el estado actual de la matriz, es decir, las 2 transformaciones anteriores */
glTranslatef.... /* afectará a sólo a la geometría que dibuje antes del glPop... */
glScalef..... /* afectará a sólo a la geometría que dibuje antes del glPop... */
dibujo_geometría_específica( ); /* Render de la geometría que pasará por 4 transformaciones */
glPopMatrix( ); /* recupero el estado de la matriz anterior */
dibujo_el_resto( ); /* Render de la geometría que pasará por 2 transformaciones */
.....
```

Crear matrices "a medida"

Por último comentar que también podemos crearnos matrices "a mano" para después pasarlas a la matriz de transformación de OpenGL. No disponemos tan sólo de las funciones de traslación, rotación... que os he comentado sinó que también podemos usar:

```
glLoadMatrixf(puntero_a_matriz);
glMultMatrixf(puntero_a_matriz);
```

En el primer caso substituimos a la matriz actual con la que le pasamos precalculada por nosotros mismos. En el segundo caso multiplicamos a lo que ya haya en la matriz por lo que nosotros pasamos.

El puntero a una matriz se asume como variable del tipo:

```
GLfloat M[16];
```

o

```
GLfloat M[4][4];
```

y lo importantísimo es que OpenGL asume que la matriz que se le pasará está definida por columnas, es decir:

|a0 a4 a8 a12|
|a1 a5 a9 a13|
|a2 a6 a10 a14|
|a3 a7 a11 a15|

Primero definimos a0, después a1, a2, a3, a4 ... y así sucesivamente.

Si utiliza estas funciones probar antes con ejemplos sencillos hasta entender perfectamente como pasar la matriz para que ocurra lo que espera.

Manejo de trasformaciones OpenGL

La matriz de modelado

- La matriz de modelado representa el sistema de coordenadas tranformado sobre el cual construiremos la escena
- Cada transformación que hagamos se multiplicará por la matriz, actualizando sus coeficientes

$$\begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \quad M$$

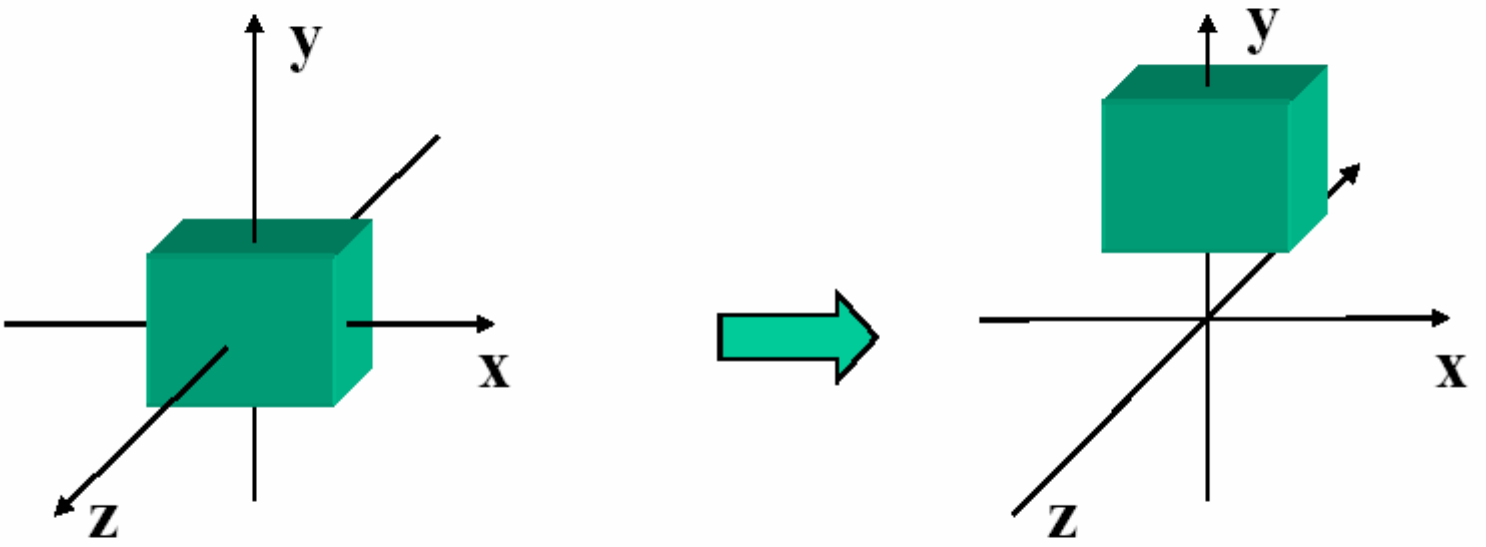
- Cada punto que pintemos, se multiplicará por la matriz

$$\begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x_m \\ y_m \\ z_m \\ w_m \end{pmatrix} \quad M$$

Traslación

- Para construir una matriz de traslación:

```
void glTranslatef (GLfloat x, GLfloat y, GLfloat z)
```

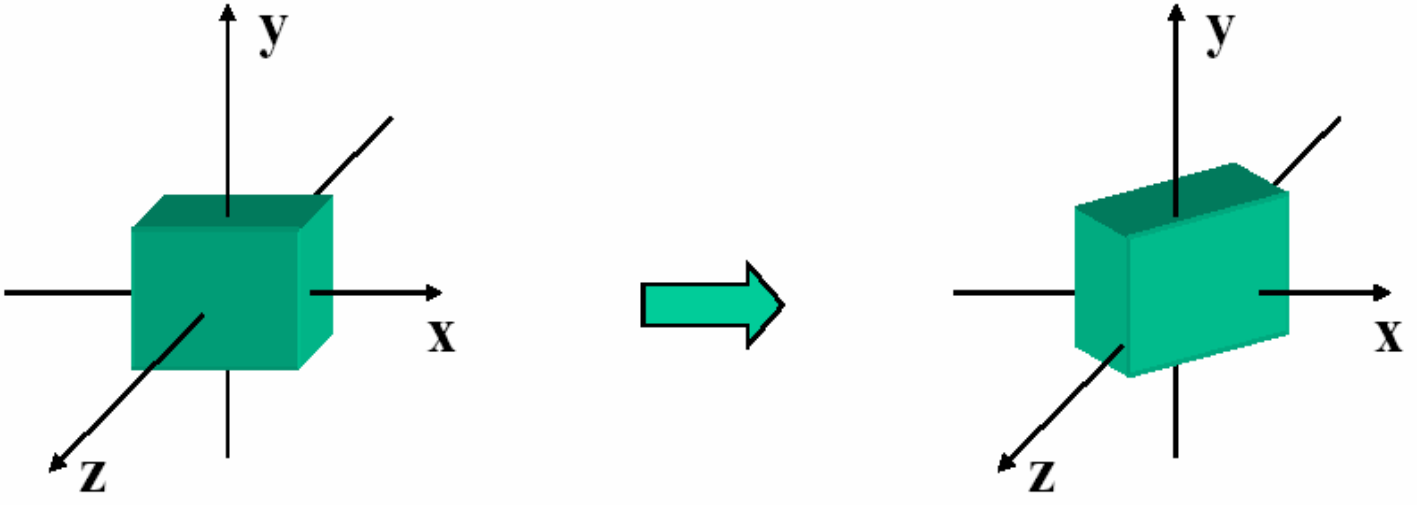



```
// Trasladar 10 unidades hacia arriba
glTranslatef (0.0f, 10.0f, 0.0f);

// Dibuja el cubo
glutSolidCube (10.0f);
```

Rotación

- Para construir una matriz de rotación:

```
void glRotatef (GLfloat ang, GLfloat x, GLfloat y, GLfloat z)
```

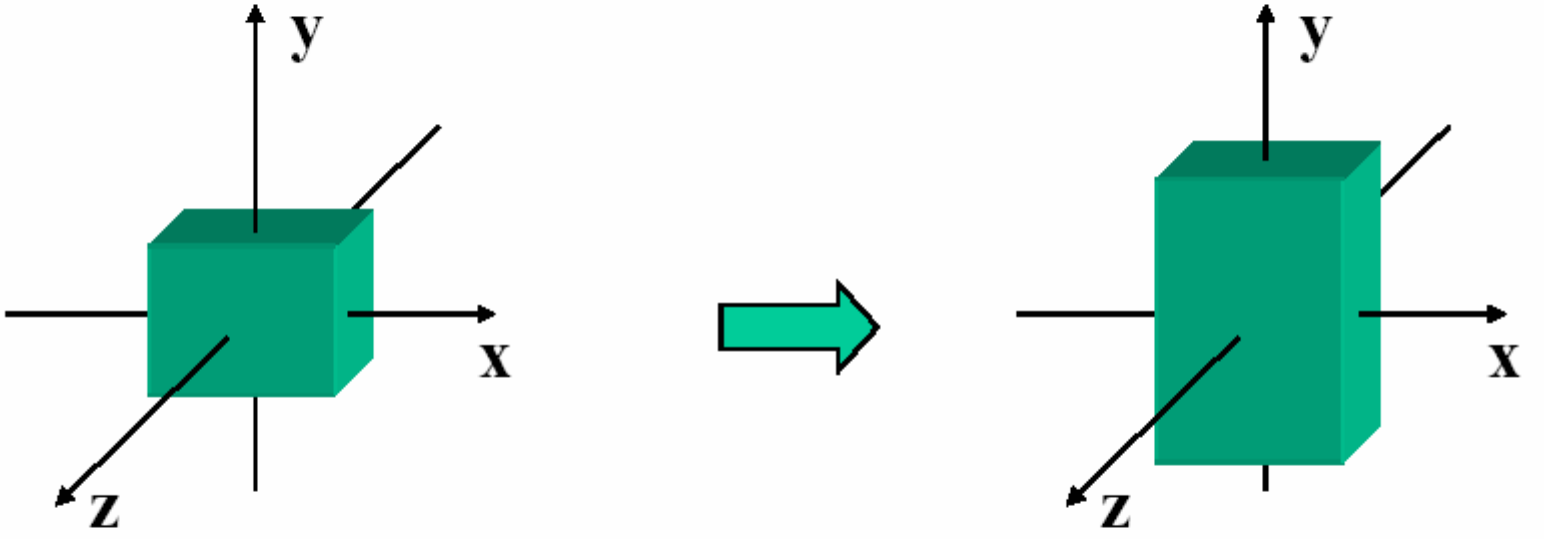



```
// Rotar 45 grados en el eje y
glRotatef (45.0f, 0.0f, 1.0f, 0.0f);

// Dibuja el cubo
glutSolidCube (10.0f);
```

Escalado

- Para construir una matriz de escalado:

```
void glScalef (GLfloat x, GLfloat y, GLfloat z)
```

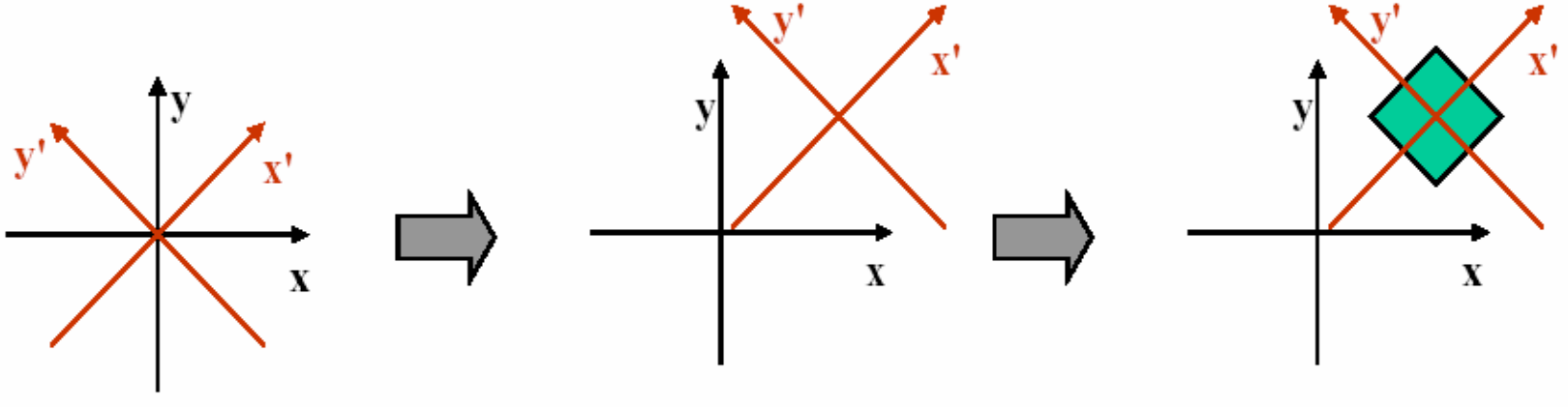


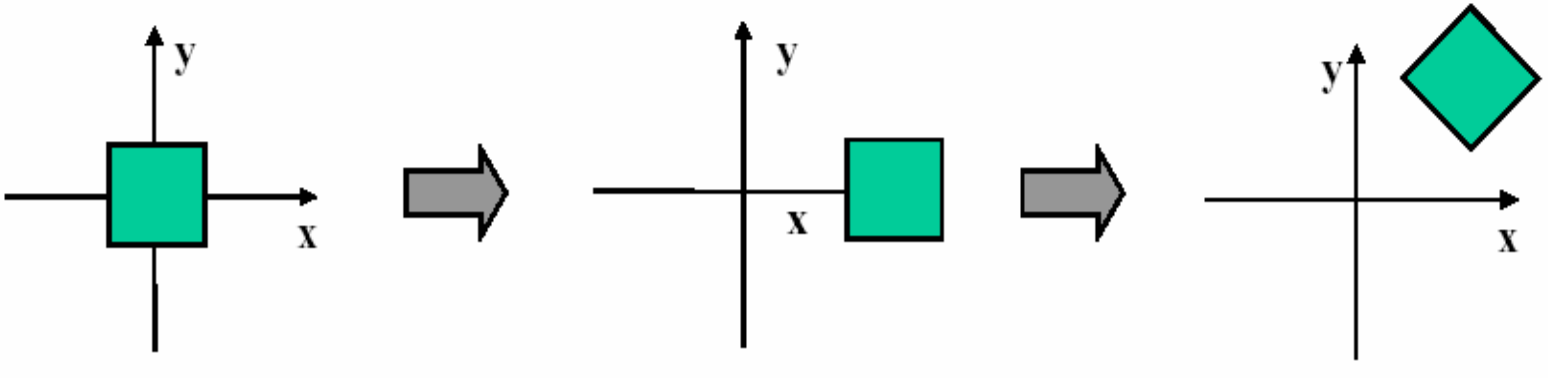

```
// Escalar el doble en vertical
glScalef (1.0f, 2.0f, 1.0f);

// Dibuja el cubo
glutSolidCube (10.0f);
```

La dualidad de la matriz de modelado

- Es igual aplicar la transformación al sistema de referencia de la escena, que aplicar la transformación inversa al objeto

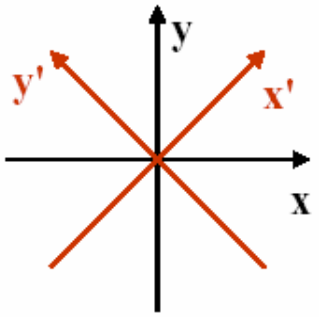
```
glRotatef(45,0,0,1);
glTranslatef(10,0,0);
glRectf(-5,-5,5,5);
```



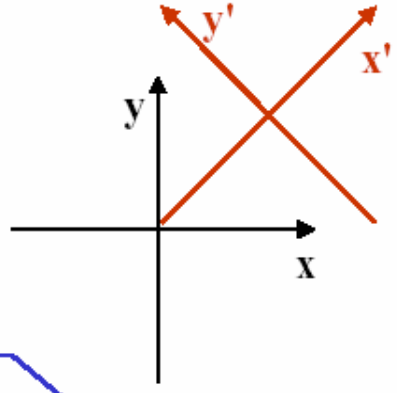


Orden de las transformaciones

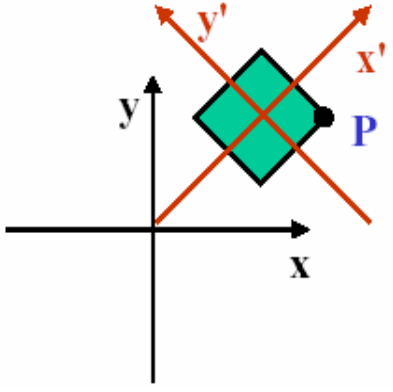
```
glRotatef(45,0,0,1);
glTranslatef(10,0,0);
glRectf(-5,-5,5,5);
```



$$M = \begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



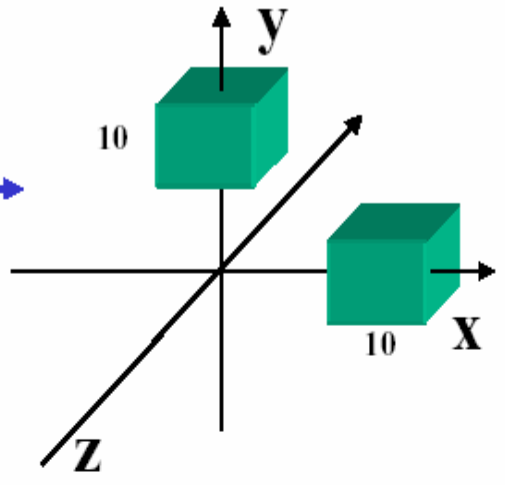
$$M = M \cdot \begin{pmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 & 10/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 & 10/\sqrt{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



$$P = M \cdot P = \begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 & 10/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 & 10/\sqrt{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 5 \\ -5 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 10\sqrt{2} \\ 5\sqrt{2} \\ 0 \\ 1 \end{pmatrix}$$

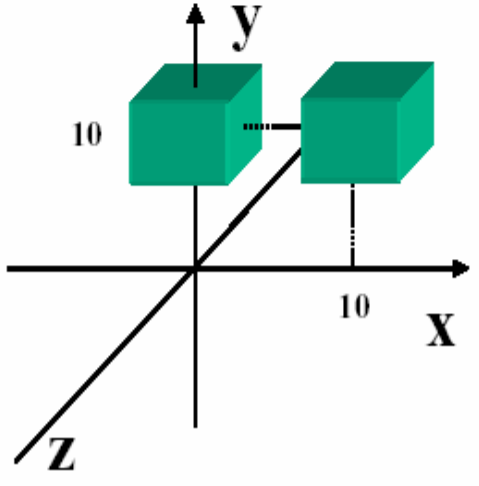
Resetear la matriz de modelado

- Si quisiéramos obtener esta escena,
  no podemos hacer esto




```
// Sube 10 unidades en y
glTranslatef (0.0f, 10.0f, 0.0f);

// Dibuja la primera esfera
glutSolidCube (1.0f);

// Mueve 10 unidades en x
glTranslatef (10.0f, 0.0f, 0.0f);

// Dibuja la segunda esfera
glutSolidCube (1.0f);
```

porque obtendríamos esto:

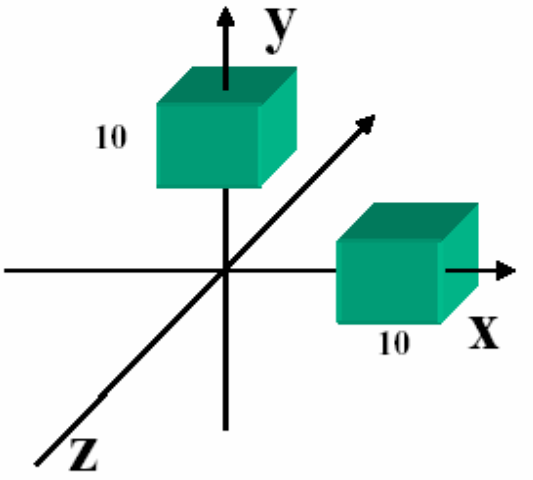


- Necesitamos una forma de resetear la matriz de modelado

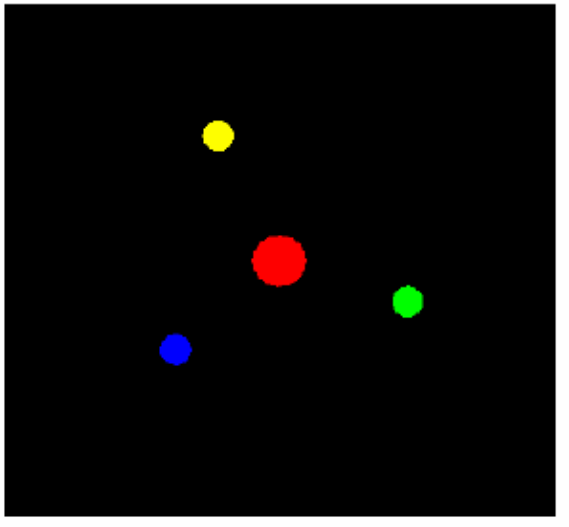
```
void glLoadIdentity()
```

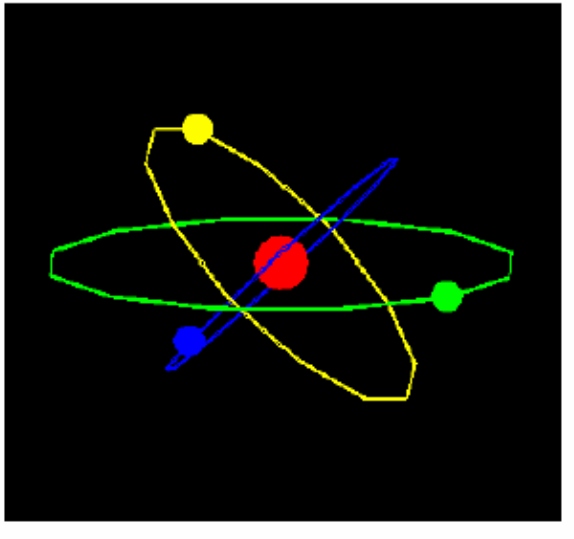
- El código correcto sería:

```
// Inicializa la matriz del modelador
glMatrixMode (GL_MODELVIEW);
glLoadIdentity();

// Sube 10 unidades en y
glTranslatef (0.0f, 10.0f, 0.0f);
// Dibuja la primera esfera
glutSolidCube (1.0f);

// Reinicia de nuevo la matriz
glLoadIdentity();

// Mueve 10 unidades en x
glTranslatef (10.0f, 0.0f, 0.0f);
// Dibuja la segunda esfera
glutSolidCube (1.0f);
```



```
// Inicialización
glClear (GL_COLOR_BUFFER_BIT);
glMatrixMode (GL_MODELVIEW);
glLoadIdentity();

// Dibuja el núcleo
glutSolidSphere(10,20,20);

// Dibuja el electrón amarillo
glColor3ub(255, 255, 0);
glPushMatrix();
   glRotatef(-45.0f, 0.0f, 0.0f, 1.0f);
   if (VerOrbitas) auxWireCylinder(70,1);
   glRotatef(angulo, 0.0f, 1.0f, 0.0f);
   glTranslatef(70.0f, 0.0f, 0.0f);
   glutSolidSphere(6,20,20);
glPopMatrix();

//dibuja los demás electrones

// incrementa el ángulo y dibuja
angulo+= 10;
glFlush();
```





Manipulación de matrices

- Podemos crear nuestra propia matriz y cargarla en la pila correspondiente

```
void glLoadMatrix (GLdouble *m);
```

- También podemos multiplicarla por la matriz actual

```
void glMultMatrix (GLdouble *m);
```

- Ejemplo:

```
GLfloat m[] = {1,0,5,0,
               0,1,5,0,
               0,0,1,0,
               0,0,0,1};

glMatrixMode (GL_MODELVIEW);
glLoadMatrix (m);
```

Transformaciones con OpenGL

- Indica la matriz que vas a utilizar: *glMatrixMode (GL_MODELVIEW);*
- Inicializar la matriz a la identidad: *void glLoadIdentity (void);*
- Traslación: *void glTranslate{fd}(x, y, z);*
- Rotación: *void glRotate{fd}(angle, x, y, z);*
- Escalado: *void glScale{fd}(x, y, z);*
- Ejemplo:

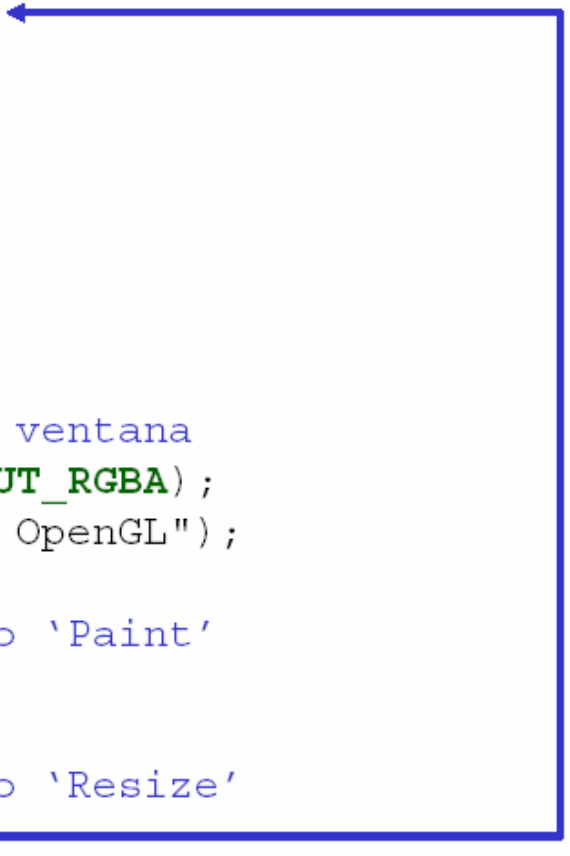
```
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ();
glTranslatef (10.0, 5.0, 1.0);
glutWireSphere (1.0);
glRotatef (15.0, 1.0, 1.0, 1.0);
glutWireTeapot (1.0);
```

- Pila de matrices:
  void glPushMatrix (void);
  void glPopMatrix (void);
- Ejemplo:

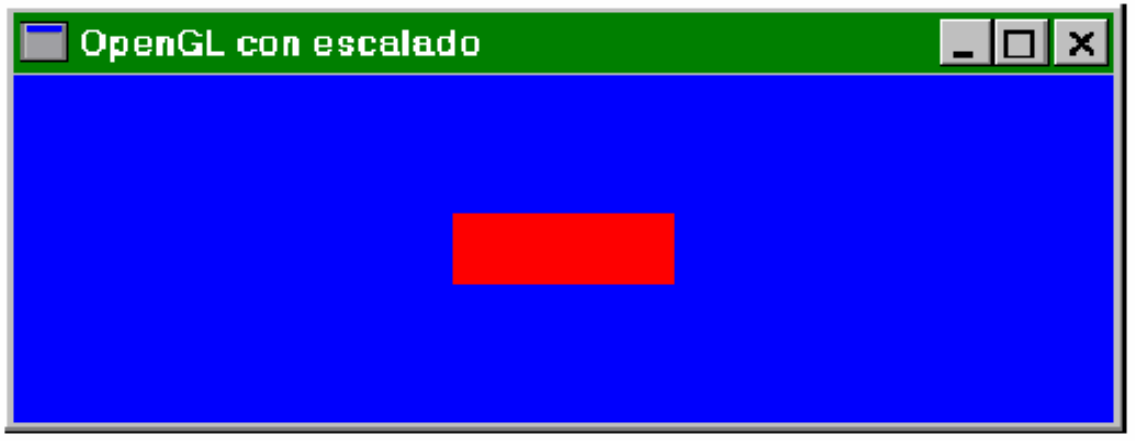
```
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ();
glPushMatrix();
  glTranslatef (10.0, 5.0, 1.0);
  glutWireSphere (1.0);
glPopMatrix();
glRotatef (15.0, 1.0, 1.0, 1.0);
glutWireTeapot (1.0);
```

Escalado de la ventana

```
void EscalaVentana(GLsizei w, GLsizei h)
   { .... }

void DibujaEscena()
   { .... }

void main()
{
   // Funciones GLUT para inicializar la ventana
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGBA);
   glutCreateWindow ("Mi primer programa OpenGL");

   // Indicamos la función para el evento 'Paint'
   glutDisplayFunc (DibujaEscena);

   // Indicamos la función para el evento 'Resize'
   glutReshapeFunc (EscalaVentana)

   // Lanzamos el bucle indefinido de eventos
   glutMainLoop();
}
```



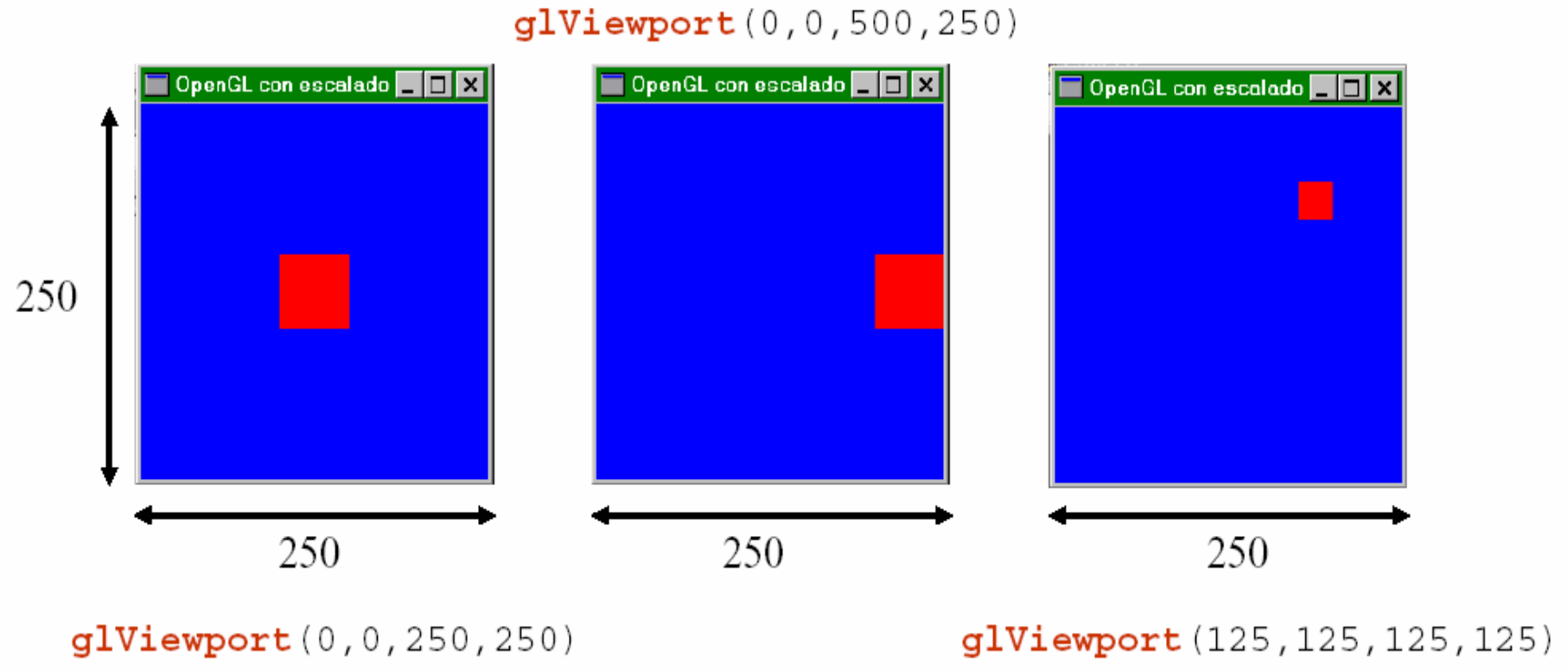
```
void EscalaVentana(GLsizei w, GLsizei h)
{
    // Evita una división por cero
    if (h == 0) h = 1;

    // Ajusta la vista a las dimensiones de la ventana
    glViewport(0, 0, w, h);

    // Reinicia el sistema de coordenadas
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity();

    // Establece el volumen de trabajo
    glOrtho(0.0f, 250.0f, 0.0f, 250.0f, -1.0, 1.0);

    glFlush();
}
```



Definición de la vista

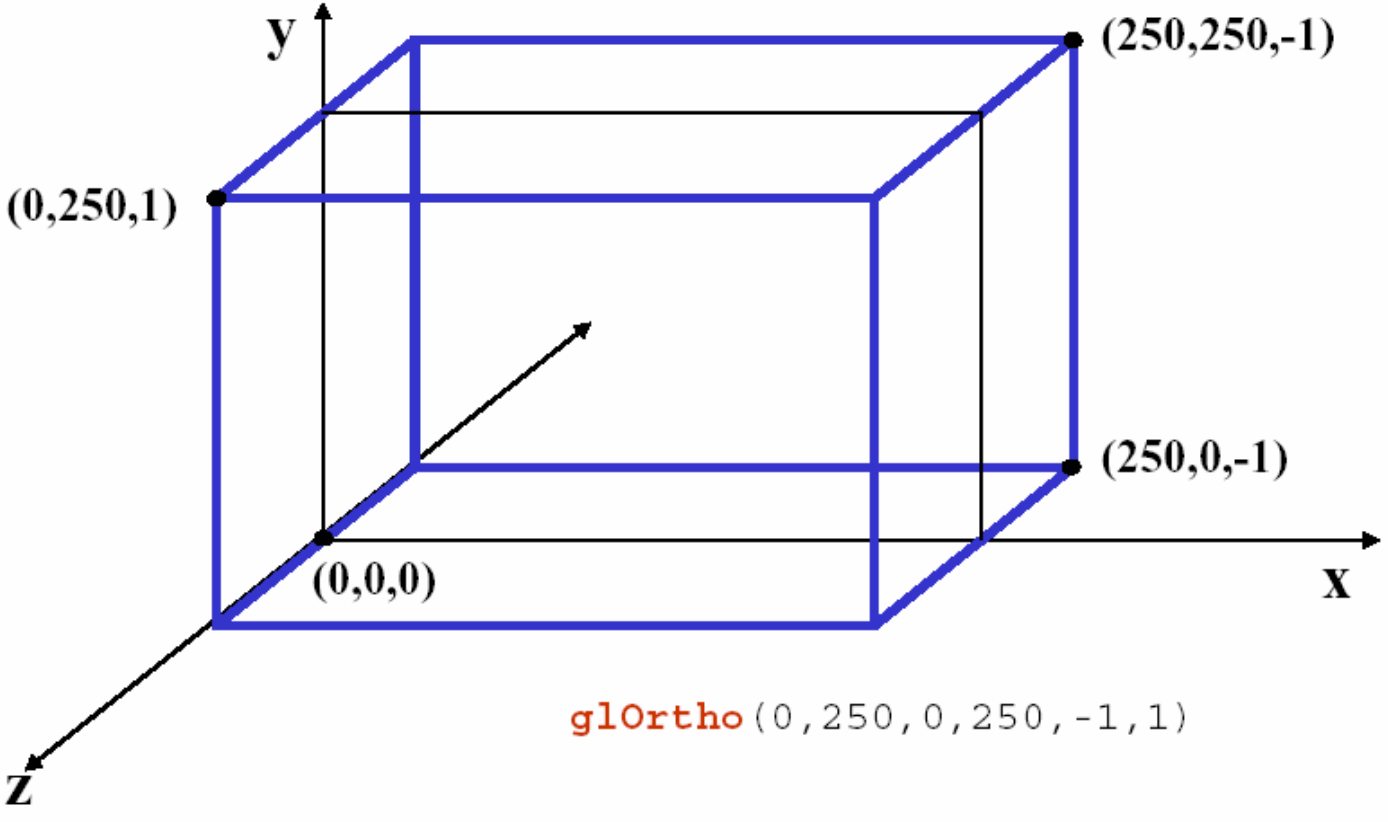
- Indica la posición y resolución de la foto final (pixels)

```
glViewport(GLint x, GLint y, GLsizei ancho, GLsize alto);
```



Definición del volumen de trabajo

```
glLoadIdentity();
glOrtho(GLdouble izquierda, GLdouble derecha, GLdouble abajo,
        GLdouble arriba, GLdouble detrás, GLdouble delante);
```




```
glOrtho(0,250,0,250,-1,1)
```

OpenGL: Gestión del ratón

- Asignación de las funciones de gestión de eventos del ratón:
  - `glutMouseFunc(funcion_gestion_boton);`
  - `glutMotionFunc(funcion_gestion_movimiento_pulsado);`
  - `glutPassiveMotionFunc(funcion_gestion_movimiento_no_pulsado);`

- Declaración de las funciones:
  - `funcion_gestion_boton(GLint boton,GLint estado,GLint x,GLint y)`
    - boton: `GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON, GLUT_MIDDLE_BUTTON`
    - estado: `GLUT_UP, GLUT_DOWN`
    - posición del ratón:
      - `(x,y)` en píxeles a partir de la esquina superior izquierda
      - **ATENCIÓN**: ventana visión OpenGL mide a partir de esquina **inferior** izquierda
        - correspondencia: OpenGL Y = altura_ventana - GLUT Y
      - **ATENCIÓN**: dimensiones de la ventana visión OpenGL no tienen que corresponder con dimensiones ventana GLUT
        - ejemplo: ventana GLUT: 400x300, ventana visión OpenGL 1x1
        - hay que **escalar** las posiciones GLUT para hacer corresponder con la ventana de visión

- `funcion_gestion_movimiento_pulsado(GLint x, GLint y)`
  - `(x,y)` posición a la que se ha movido el ratón con algún botón pulsado

- `funcion_gestion_movimiento_no_pulsado(GLint x, GLint y)`
  - `(x,y)` posición a la que se ha movido el ratón con botones sin pulsar

OpenGL: Gestión del teclado

- Asignación de las funciones de gestión de eventos del teclado:
  - `glutKeyboardFunc(funcion_gestion_teclado);`
  - `glutSpecialFunc(funcion_gestion_teclado_especial);`

- Declaración de las funciones:
  - `funcion_gestion_teclado(GLubyte tecla, GLint x, GLint y)`
    - tecla: código ASCII de la tecla que se ha pulsado
    - `(x,y)` posición del cursor cuando se pulsó la tecla

  - `funcion_gestion_teclado_especial(GLubyte tecla, GLint x, GLint y)`
    - tecla: función: `GLUT_KEY_F1` a `GLUT_KEY_F12`, cursor: `GLUT_KEY_UP`, `GLUT_KEY_DOWN`, `GLUT_KEY_RIGHT`, `GLUT_KEY_LEFT`, página: `GLUT_PAGE_UP`, `GLUT_PAGE_DOWN`
    - `(x,y)` posición del cursor cuando se pulsó la tecla

- `int glutGetModifiers();`
  - devuelve: `GLUT_ACTIVE_SHIFT`, `GLUT_ACTIVE_CTRL`, `GLUT_ACTIVE_ALT`
- GLUT **no** reconoce cuando la tecla se suelta

- Objetivo de la selección interactiva:
  - situar el cursor sobre un objeto de la ventana, pulsar y seleccionarlo

- Problema:
  - varios objetos 3D se proyectan sobre el mismo punto 2D de la ventana

- Posibles soluciones:

  - intersección de rayos:
  - cajas englobantes
  - buffer de color
  - lista de impactos

OpenGL: Menús desplegables

- Creación del menú:

  `GLint glutCreateMenu(funcion_gestion_menu);`

  - `void funcion_gestion_menu(GLint entrada_seleccionada)`:
    - es la función que se llama cuando se selecciona alguna entrada del menú
    - debe tener un parámetro de tipo `GLint` que corresponde a la entrada seleccionada
    - la última línea de esta función debe revisualizar: `glutPostRedisplay();`
  - devuelve el identificador del menú

- Definición de las entradas del menú:

  `glutAddMenuEntry(cadena,identificador_entrada);`

  - `cadena`: texto que aparecerá como entrada del menú
  - `identificador_entrada`: `GLint` con el identificador asociado a la entrada

- Asociación del menú a un botón del ratón:

  `glutAttachMenu(boton);`

  - `boton:GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON, GLUT_MIDDLE_BUTTON`
  - asocia el último menú creado

- Definición de submenús:
  1. Creación del submenú (como un menú normal) guardando su identificador

```
identificador_submenu=glutCreateMenu(funcion_gestion_submenu);
  glutAddMenuEntry("entrada_1_submenu",1);
  glutAddMenuEntry("entrada_2_submenu",2);
  ...
```

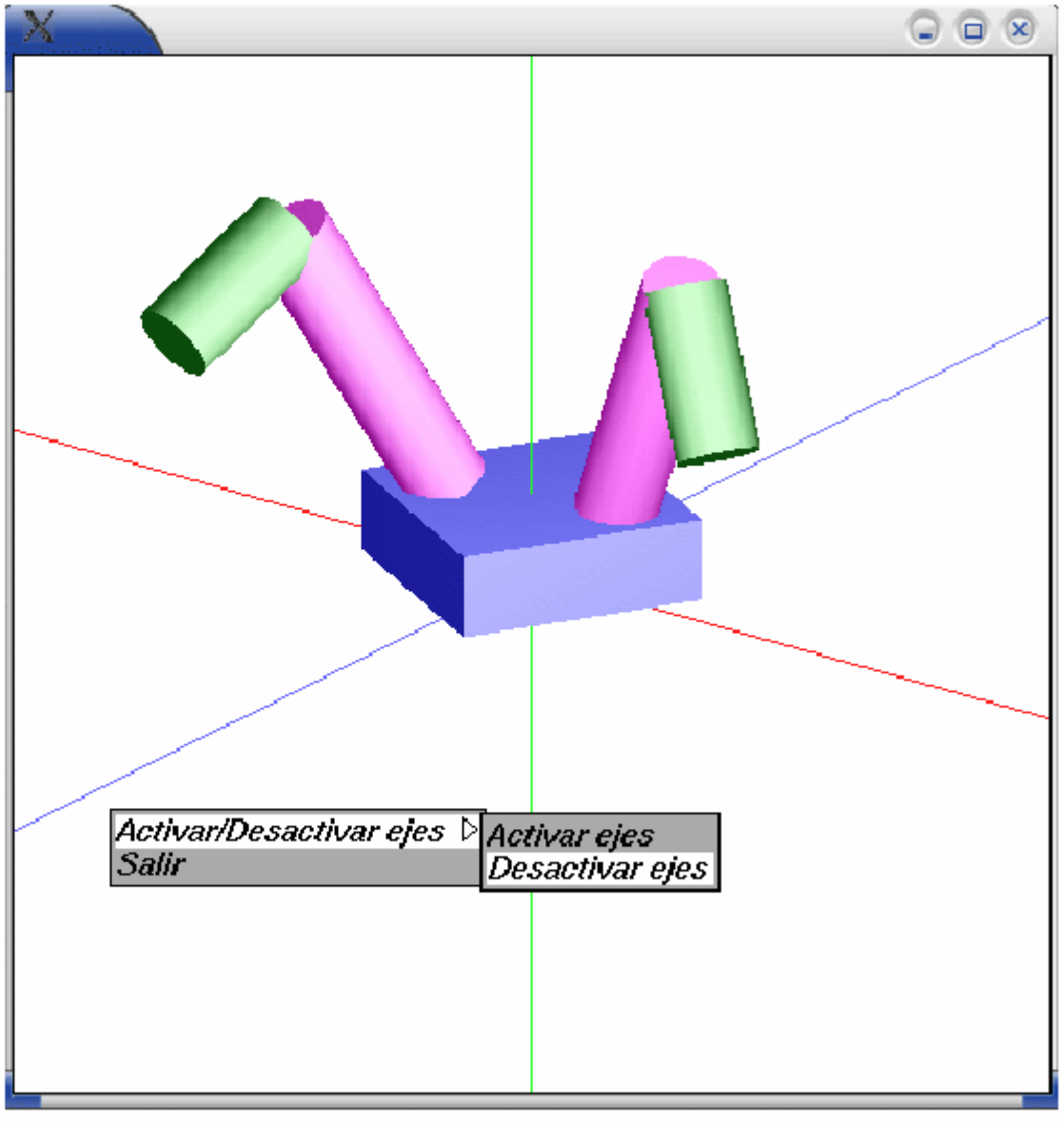
  2. Asociar el submenú como una entrada más del menú principal

```
glutCreateMenu(funcion_gestion_menu_principal);
  glutAddMenuEntry("entrada_1_menu_principal",1);
  glutAddMenuEntry("entrada_2_menu_principal",2);
  ...
  glutAddSubMenu("entrada_submenu",identificador_submenu);
  ...
```

  3. Asociar el menú principal al botón del ratón deseado

```
glutAttachMenu(boton);
```

- Ejemplo:


Activar/Desactivar ejes
Salir
Activar ejes
Desactivar ejes

Interacción con el teclado

La función que nos permite registrar este evento es *glutSpecialFunc*, que se encarga de manejar la pulsación de teclas "especiales", como los cursores.

La declaración de esta función es:

```
glutSpecialFunc(void (*func)(int key, int x, int y));
```

Es decir, recibe un puntero a función, que no devuelve nada, y recibe tres enteros: la tecla pulsada, y la posción X e Y del ratón en la pantalla.

La rotación alrededor de los ejes X e Y la vamos a controlar mediante dos variables globales (*rot_angle_x*, *rot_angle_y*).

Acceso directo a keys

El código de la función a la que hay que llamar al pulsar alguna tecla “especial” será el siguiente:

```
void specialKeys(int key, int x, int y)
{
  switch (key)
  {
    case GLUT_KEY_UP: rot_angle_x--;
         break;
    case GLUT_KEY_DOWN: rot_angle_x++;
         break;
    case GLUT_KEY_RIGHT: rot_angle_y++;
         break;
    case GLUT_KEY_LEFT: rot_angle_y--;
         break;
  }
  glutPostRedisplay();
}
```

Con esto, modificaremos la rotación según la tecla pulsada.

Además, para facilitar la salida del programa, vamos a añadir que, al pulsar la tecla "Esc.", el programa termine. Para ello utilizaremos la función *glutKeyboardFunc()*, que recibirá como parámetro un puntero a la siguiente función:

```
static void keys(unsigned char key, int x, int y)
{
  switch (key)
  {
    case 27: exit(0);
         break;
  }
  glutPostRedisplay();
}
```

El resto del programa es prácticamente idéndico al comentado en el capítulo de iluminación (la función *idle* se hace innecesaria, y hay que dar de alta los dos nuevos manejadores de eventos añadiendo al programa principal:

```
glutSpecialFunc(specialKeys);
glutKeyboardFunc(keys);
```