

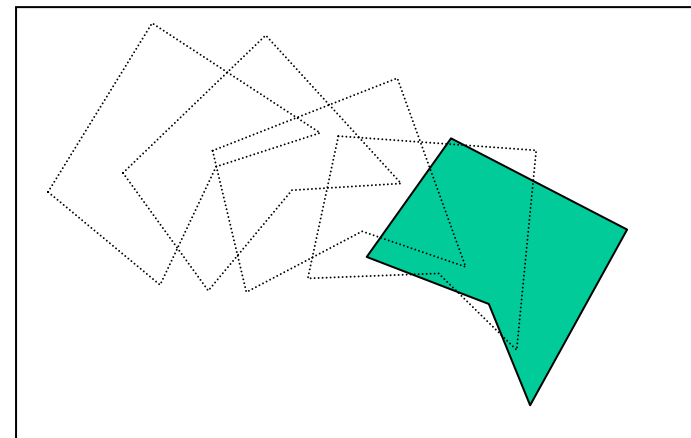
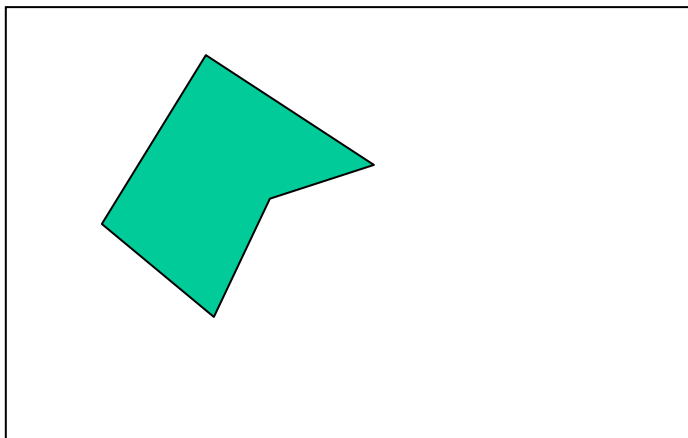
# TEMA 3: Transformaciones 2D

# Índice

1. Transformaciones Básicas
  1. Traslación
  2. Rotación
  3. Escalado
2. Representación Matricial y Coordenadas Homogéneas
3. Otras Transformaciones
  1. Reflexión
  2. Afilamiento
4. Transformación de Ventana a Pantalla
5. Algoritmos de Recorte
  1. Recorte de Puntos
  2. Recorte de Líneas
  3. Recorte de Polígonos

# Transformaciones geométricas

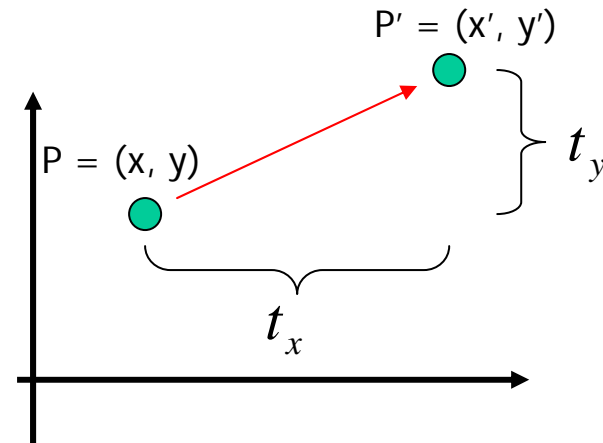
- Con los algoritmos de primitivas ya podemos dibujar en pantalla
- El siguiente paso consiste en permitir modificar o manipular dichas primitivas → Transformaciones Geométricas
  - Para poder implementar aplicaciones de diseño
  - Para poder realizar animaciones
  - Para interactuar con la escena
- Las transformaciones básicas que se necesitan son:
  - Traslación: cambios en la posición
  - Rotación: cambios en la orientación
  - Escalado: cambios en el tamaño



# Traslación

- Reposiciona un objeto desplazándolo a las nuevas coordenadas

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

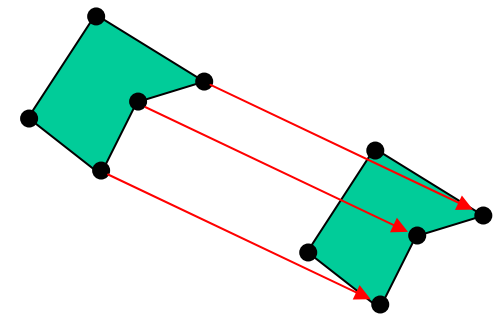


- En forma matricial:

$$P = (x, y) \quad P' = (x', y') \quad T = (t_x, t_y)$$

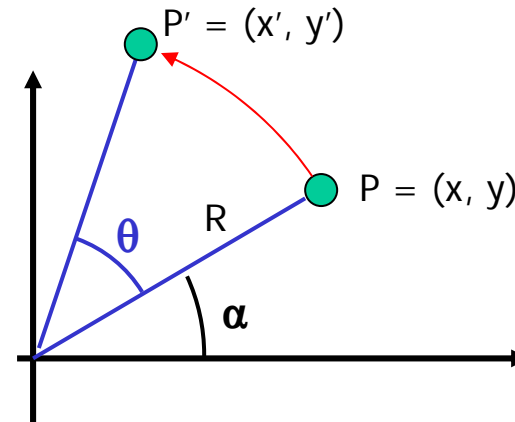
$$P' = P + T$$

- Es una transformación rígida  $\rightarrow$  el objeto no se deforma
- Para trasladar líneas rectas trasladamos sólo sus extremos
- Para trasladar polígonos, trasladamos sólo sus vértices y redibujamos



# Rotación con respecto al origen

- La posición de un punto es rotada alrededor del origen de coordenadas
- ¿Cómo sacamos la fórmula para obtener  $P'$  a partir de  $P$  y del ángulo?
- Solución: expresándolo en polares



$$\begin{cases} x = R \cos \alpha \\ y = R \sin \alpha \end{cases} \quad \begin{cases} x' = R \cos(\alpha + \theta) = \dots = x \cos \theta - y \sin \theta \\ y' = R \sin(\alpha + \theta) = \dots = x \sin \theta + y \cos \theta \end{cases}$$

- En forma matricial:

$$P = (x, y) \quad P' = (x', y') \quad R = \begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix}$$

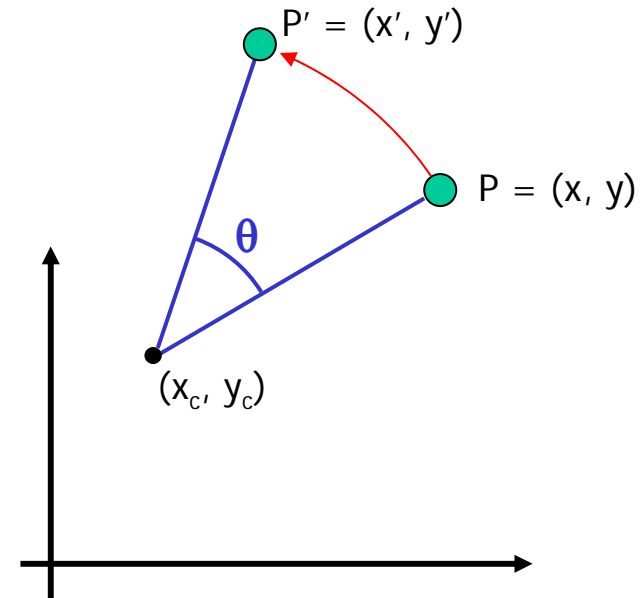
$$P' = P \cdot R$$

# Rotación general

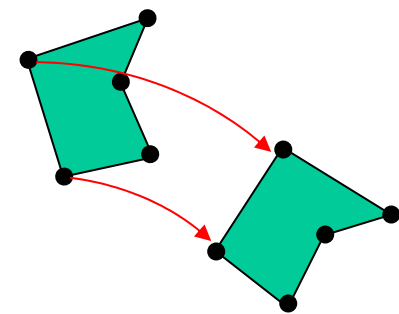
- ¿Cómo será la fórmula general cuando el punto sobre el que se rota no es el origen, sino un punto cualquiera  $(x_c, y_c)$ ?

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

- Encontrar la forma matricial para este caso es un poco complicado
- Más tarde lo haremos de otra forma mucho más fácil



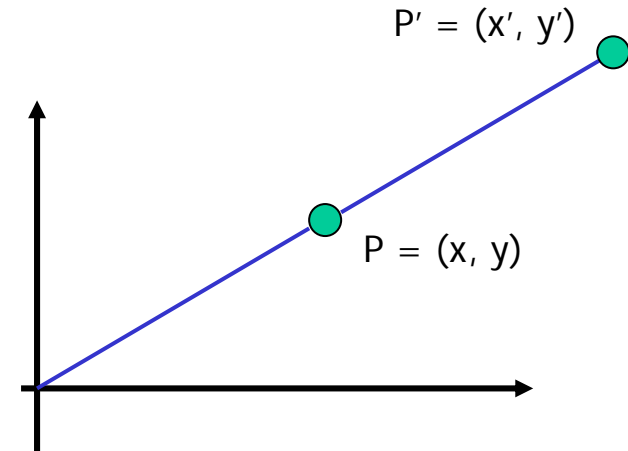
- Es una transformación rígida  $\rightarrow$  el objeto no se deforma
- Para rotar líneas rectas rotamos sólo sus extremos
- Para rotar polígonos, rotamos sólo sus vértices y redibujamos



# Escalado con respecto al origen

- La posición del punto se multiplica por una constante
- Hay que especificar dos factores de escala,  $s_x$  y  $s_y$

$$\begin{cases} x' = s_x \cdot x \\ y' = s_y \cdot y \end{cases}$$



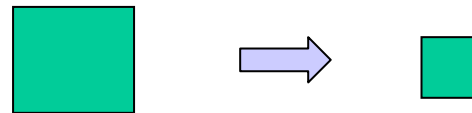
- En forma matricial:

$$P = (x, y) \quad P' = (x', y') \quad S = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \quad \boxed{P' = P \cdot S}$$

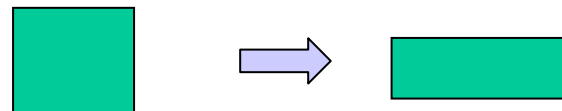
- Según el valor del factor de escala s:

- Si  $s > 1$  → aumento de tamaño
- Si  $s = 1$  → no cambia de tamaño
- Si  $s < 1$  → disminución de tamaño

- Si  $s_x = s_y$  → escalado uniforme



- Si  $s_x \neq s_y$  → escalado diferencial

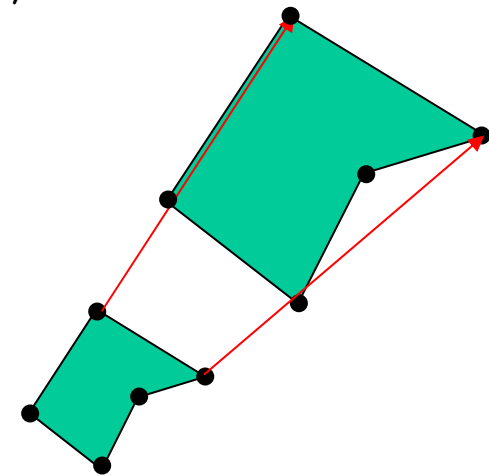
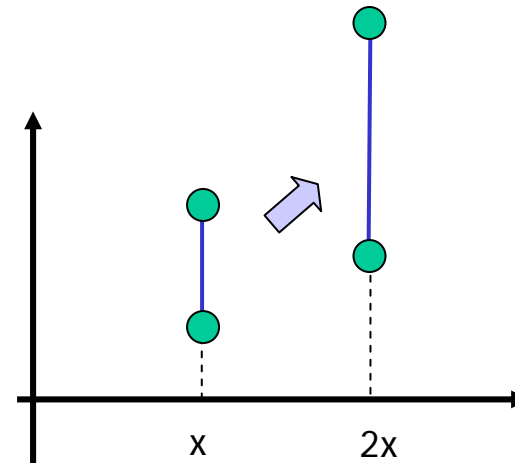


# Escalado general

- **NOTA:** si el origen de coordenadas no se encuentra en el interior del objeto, se produce un desplazamiento!
- Para evitarlo, se usa un punto fijo, y se escala a partir de él

$$\begin{cases} x' = x_c + s_x(x - x_c) \\ y' = y_c + s_y(y - y_c) \end{cases}$$

- El punto fijo podría ser el centro del objeto, o uno de sus vértices, o también un punto arbitrario
- Es una transformación rígida  $\rightarrow$  el objeto no se deforma
- Para escalar líneas rectas escalamos sólo sus extremos
- Para escalar polígonos, escalamos sólo sus vértices y redibujamos





# Representación matricial

- Muchas aplicaciones incluyen secuencias de transformaciones geométricas:
  - Una animación requiere que los objetos se trasladen y roten en cada fotograma
  - Un diseño CAD requiere muchas transformaciones hasta obtener el resultado final
- Debemos formular de forma muy eficiente toda la secuencia de transformaciones
- Cada transformación puede representarse como  $P' = P M_1 + M_2$
- La matriz  $M_1$  contiene la información de ángulos y factores de escala
- La matriz  $M_2$  contiene los términos de traslación asociados al punto fijo y al centro de rotación
- Para producir una secuencia de transformaciones hay que calcular las nuevas coordenadas en cada transformación!

$$P'' = P' M_3 + M_4 = \dots = P M_1 M_3 + M_2 M_3 + M_4$$

- Buscamos una solución más eficiente que permita combinar las transformaciones para obtener directamente las coordenadas finales a partir de las iniciales

# Coordenadas homogéneas

- ¿Cómo podríamos eliminar la matriz  $M_2$ , y usar una única matriz para transformación?
- Solución: pasando a matrices 3x3 en lugar de 2x2
- Para ello debemos representar un punto cartesiano  $(x, y)$  en coordenadas homogéneas
- Un punto  $(x, y)$  se representa en coordenadas homogéneas de la forma  $(hx, hy, h)$ , para cualquier  $h$  distinto de 0
- Esto significa que un mismo punto tiene infinitas representaciones en coordenadas homogéneas
- Ejemplo: el punto  $(4, 6)$  puede expresarse como
  - $(4, 6, 1)$   $(8, 12, 2)$   $(2, 3, 1/2)$   $(8/3, 4, 2/3)$   $(-12, -18, -3)$
- Lo habitual es tomar  $h=1$ , con lo que el punto  $(x, y)$  pasa a ser  $(x, y, 1)$
- Conclusión: el punto  $(a,b,c)$  en coordenadas homogéneas representa al punto  $(a/c, b/c)$

# Coordenadas homogéneas

- El uso de coordenadas homogéneas permite tratar todas las transformaciones geométricas como una multiplicación de matrices

- Traslación:  $(x', y', 1) = (x, y, 1) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{pmatrix}$   $P' = P \cdot T(t_x, t_y)$

- Rotación respecto al origen  $(x', y', 1) = (x, y, 1) \begin{pmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$   $P' = P \cdot R(\theta)$

- Escalado respecto al origen  $(x', y', 1) = (x, y, 1) \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$   $P' = P \cdot S(s_x, s_y)$

# Composición de transformaciones: traslaciones

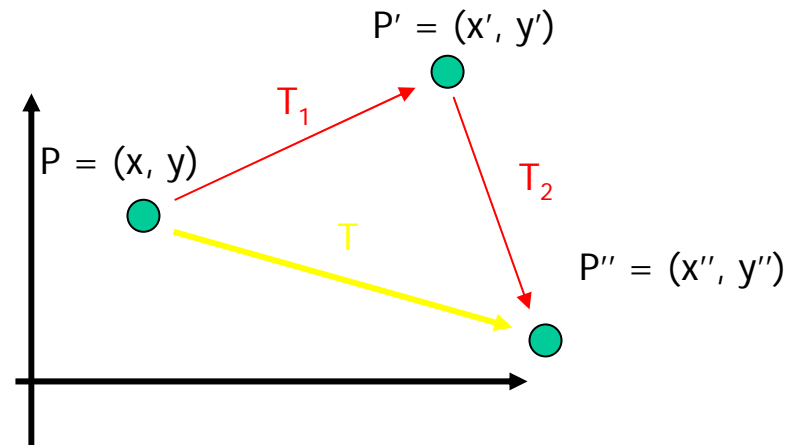
- Para cualquier secuencia de transformaciones, podemos calcular la matriz de transformación compuesta, calculando el producto de las transformaciones individuales!

- Traslaciones sucesivas:

$$P' = P \cdot T_1(t_{x1}, t_{y1})$$

$$P'' = P' \cdot T_2(t_{x2}, t_{y2})$$

$$P'' = P' \cdot T_2 = P \cdot T_1 \cdot T_2 = P \cdot T$$



$$T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_{x1} & t_{y1} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_{x2} & t_{y2} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_{x1} + t_{x2} & t_{y1} + t_{y2} & 1 \end{pmatrix}$$

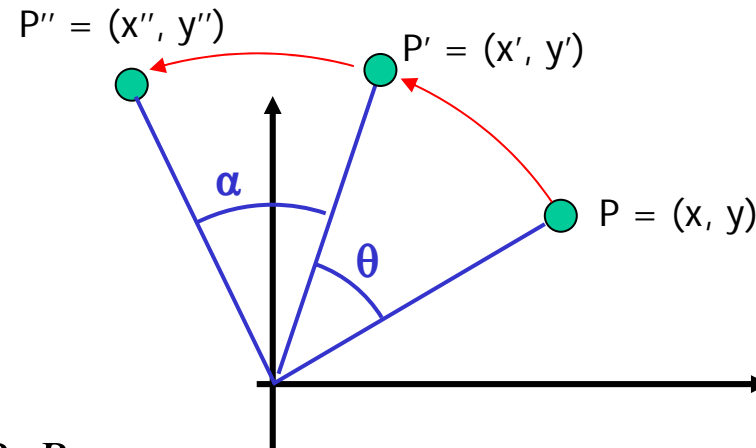
# Composición de transformaciones: rotaciones

- Rotaciones sucesivas:

$$P' = P \cdot R(\theta)$$

$$P'' = P' \cdot R(\alpha)$$

$$P'' = P' \cdot R(\alpha) = P \cdot R(\theta)R(\alpha) = P \cdot R$$



$$R = \begin{pmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta + \alpha) & \sin(\theta + \alpha) & 0 \\ -\sin(\theta + \alpha) & \cos(\theta + \alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

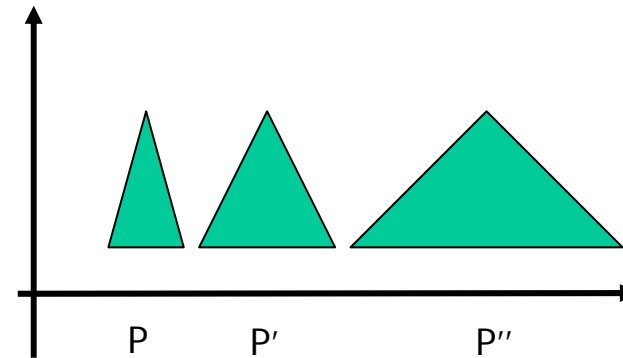
# Composición de transformaciones: escalados

- Escalados sucesivas:

$$P' = P \cdot S_1(s_{x1}, s_{y1})$$

$$P'' = P' \cdot S_2(s_{x2}, s_{y2})$$

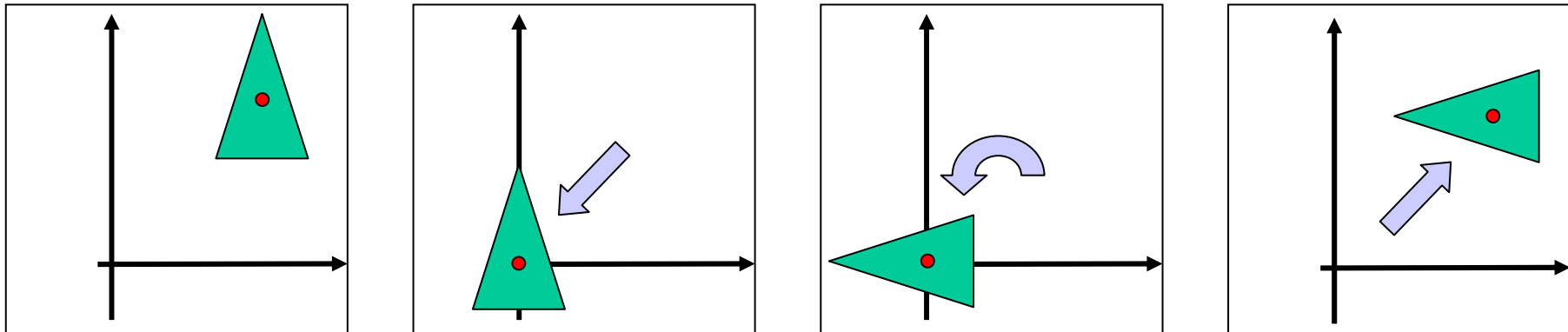
$$P'' = P' \cdot S_2 = P \cdot S_1 \cdot S_2 = P \cdot S$$



$$S = \begin{pmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} s_{x1}s_{x2} & 0 & 0 \\ 0 & s_{y1}s_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

# Rotación alrededor de un punto

- Para hacer una rotación general, podemos hacerlo mediante una composición de transformaciones básicas:

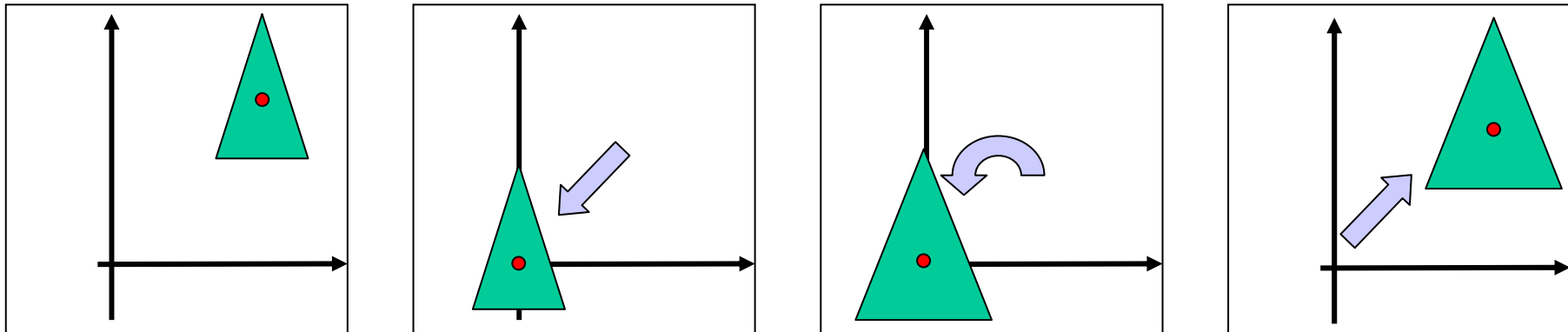


$$P' = P \cdot \underbrace{T(-x_c, -y_c) \cdot R(\theta) \cdot T(x_c, y_c)}_{R(x_c, y_c, \theta)}$$

$$R(x_c, y_c, \theta)$$

# Escalado alrededor de un punto

- Para hacer un escalado general, procedemos de igual manera:



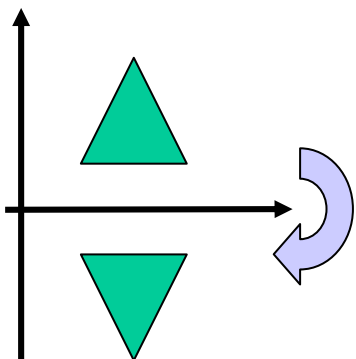
$$P' = P \cdot T(-x_c, -y_c) \cdot S(s_x, s_y) \cdot T(x_c, y_c)$$

$$S(x_c, y_c, s_x, s_y)$$



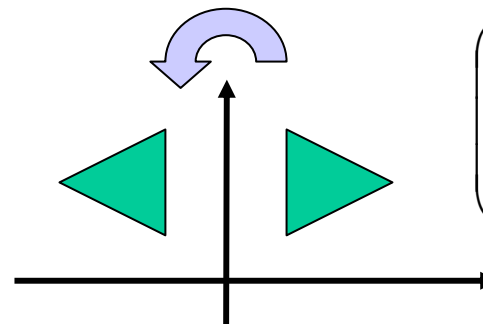
# Reflexión

Sobre el eje x



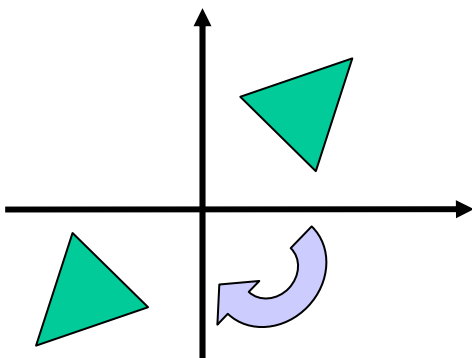
$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Sobre el eje y



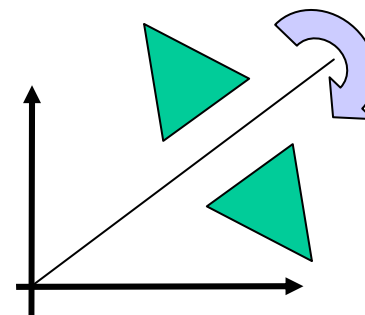
$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Sobre el origen de coordenadas



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

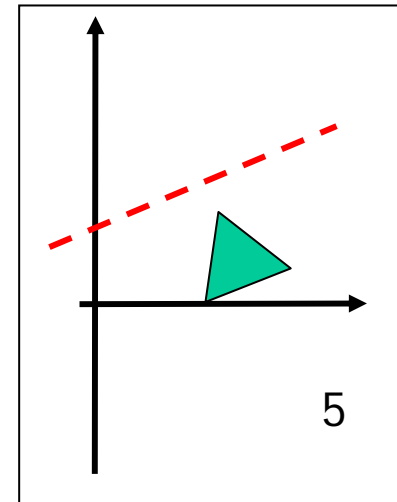
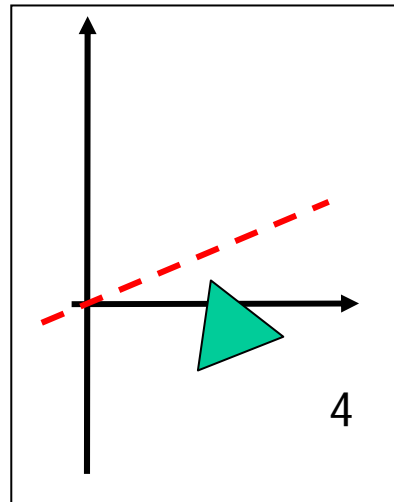
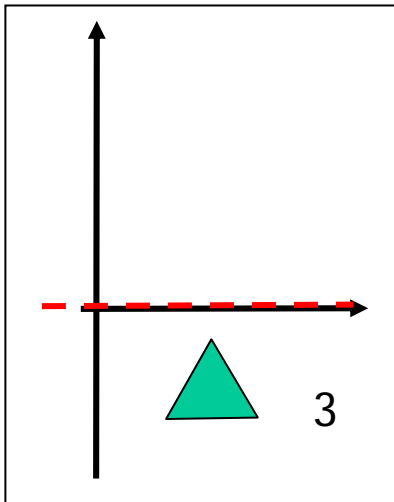
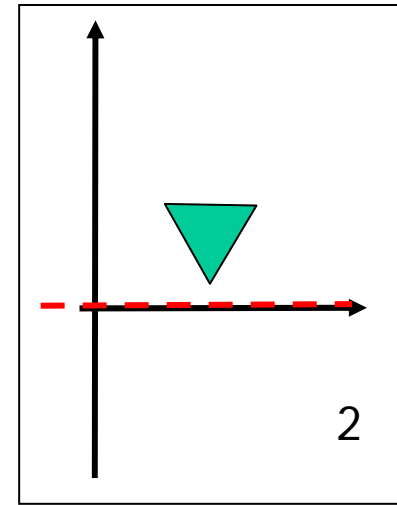
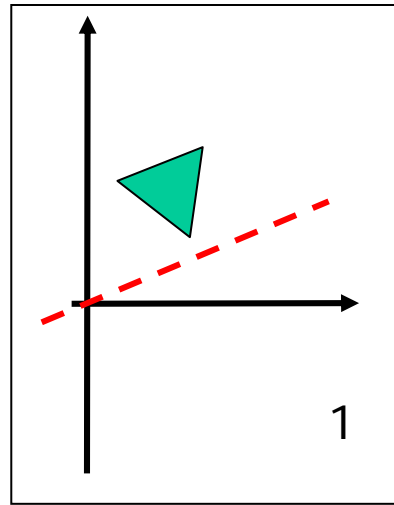
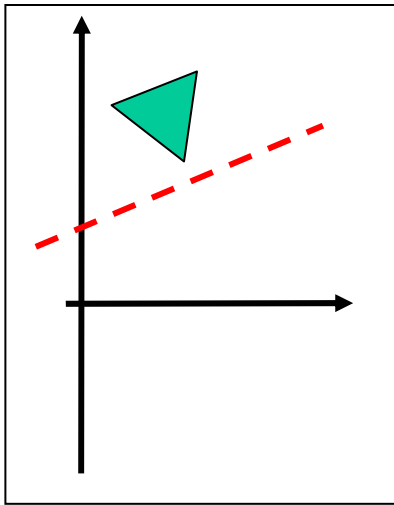
Sobre la recta  $y = x$



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

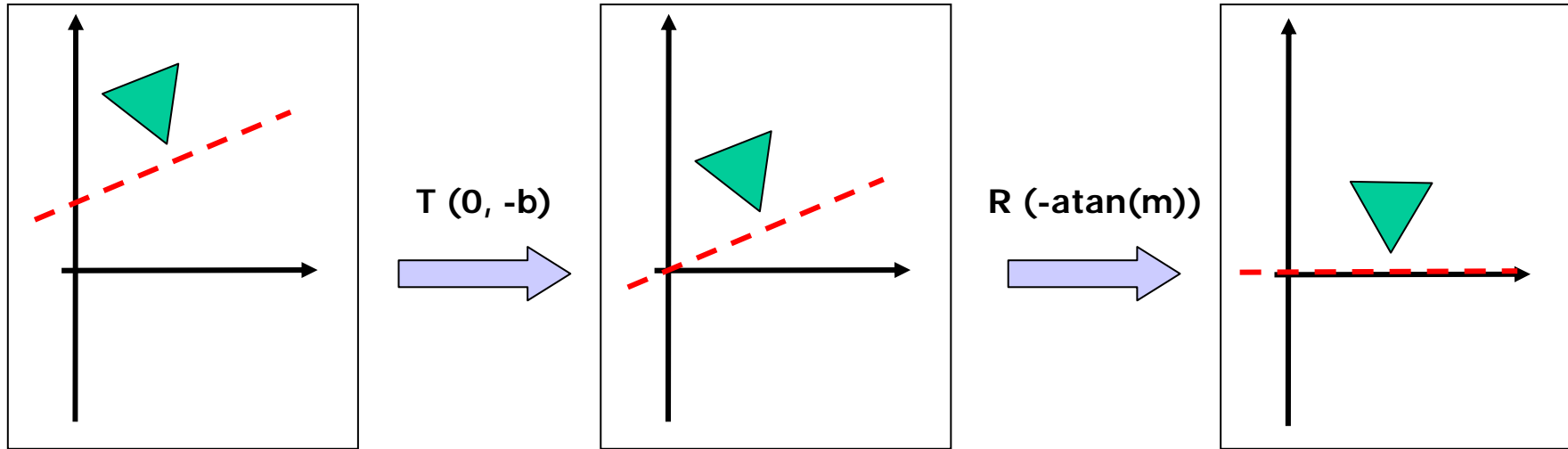
# Reflexión general

- Sobre una recta arbitraria  $y = mx + b$



# Reflexión general

- ¿Cuánto habrá que trasladar y rotar?



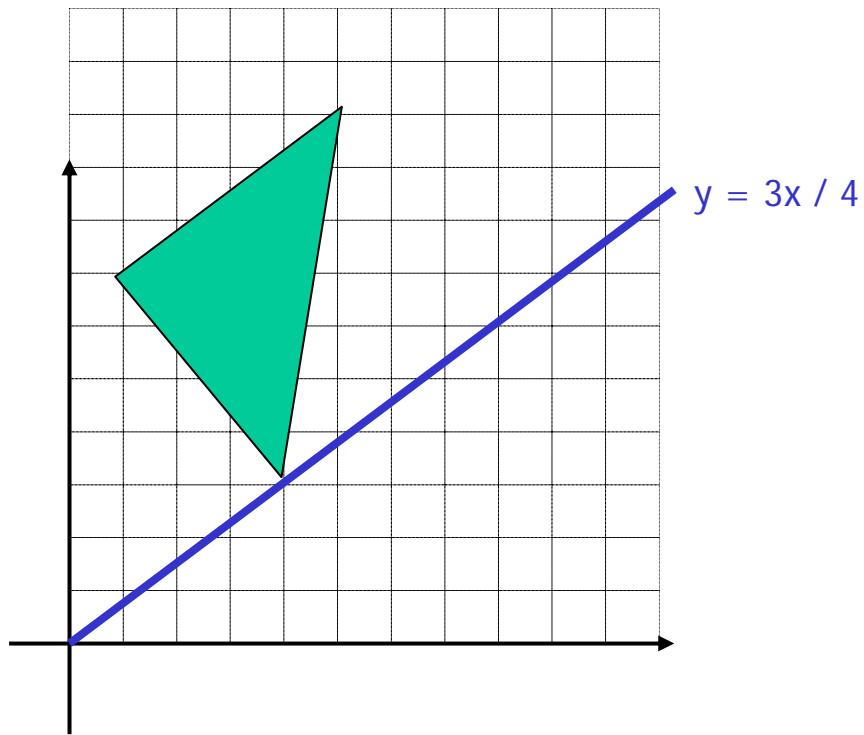
$$P' = P \cdot T(0, -b) \cdot R(-\arctan m) \cdot F \cdot R(\arctan m) \cdot T(0, b)$$

$F(m, b)$

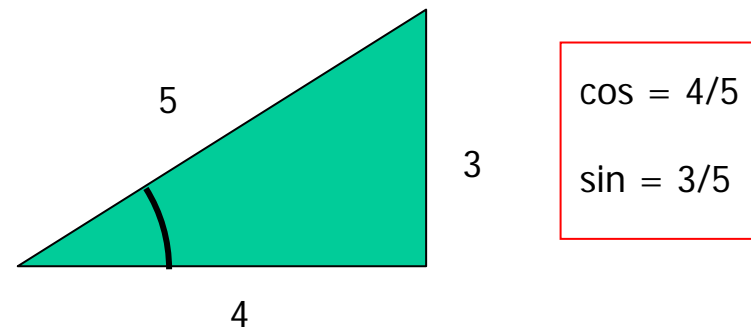
# Ejemplo de reflexión

## 2. Reflexión 2D.

- (a) Obtener la matriz de reflexión sobre la recta  $y = 3x/4$ . (1.5 puntos)
- (b) Aplicar dicha matriz para reflejar el triángulo formado por los puntos (4,3), (5,10) y (1,7). (0.5 puntos)



- ¿Cómo lo resolvemos?
- Por lo pronto no hay que trasladar
- ¿Cuánto habrá que rotar?
- Lo importante no es el ángulo en sí, sino los valores del seno y el coseno



## ...continuación...

- Comenzamos con la matriz de rotación

$$R(\theta) = \begin{pmatrix} 4/5 & -3/5 & 0 \\ 3/5 & 4/5 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Continuamos con la reflexión

$$F = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Deshacemos la rotación

$$R(-\theta) = \begin{pmatrix} 4/5 & 3/5 & 0 \\ -3/5 & 4/5 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$M = R(\theta) \cdot F \cdot R(-\theta) =$$

$$M = \begin{pmatrix} 7/25 & 24/25 & 0 \\ 24/25 & -7/25 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

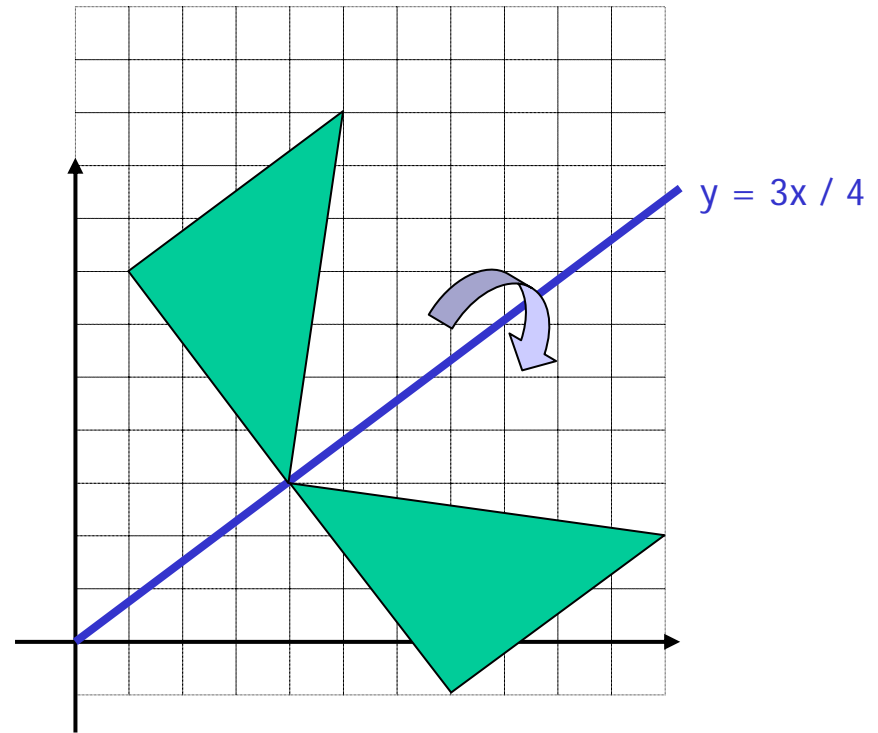
## ...continuación

- Para calcular los puntos transformados, simplemente los multiplicamos por la matriz final

$$(4,3,1) \cdot M = (4,3,1)$$

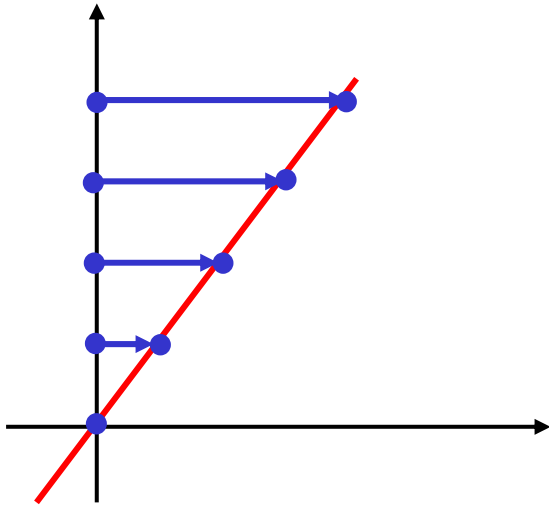
$$(5,10,1) \cdot M = (11,2,1)$$

$$(1,7,1) \cdot M = (7,-1,1)$$



# Afilamiento

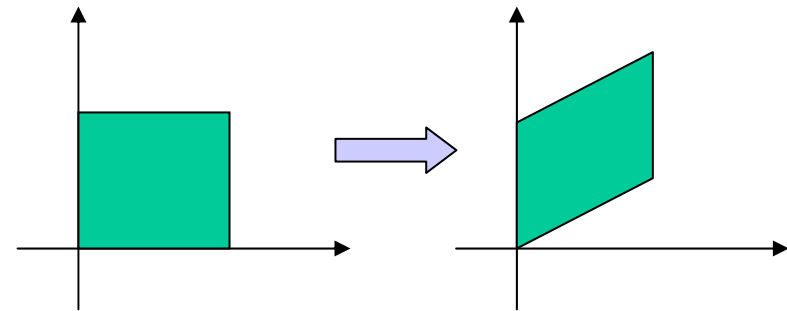
- Desplaza los puntos en función de los valores de sus coordenadas



$$\begin{cases} x' = x + ay \\ y' = y \end{cases} \longrightarrow A = \begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

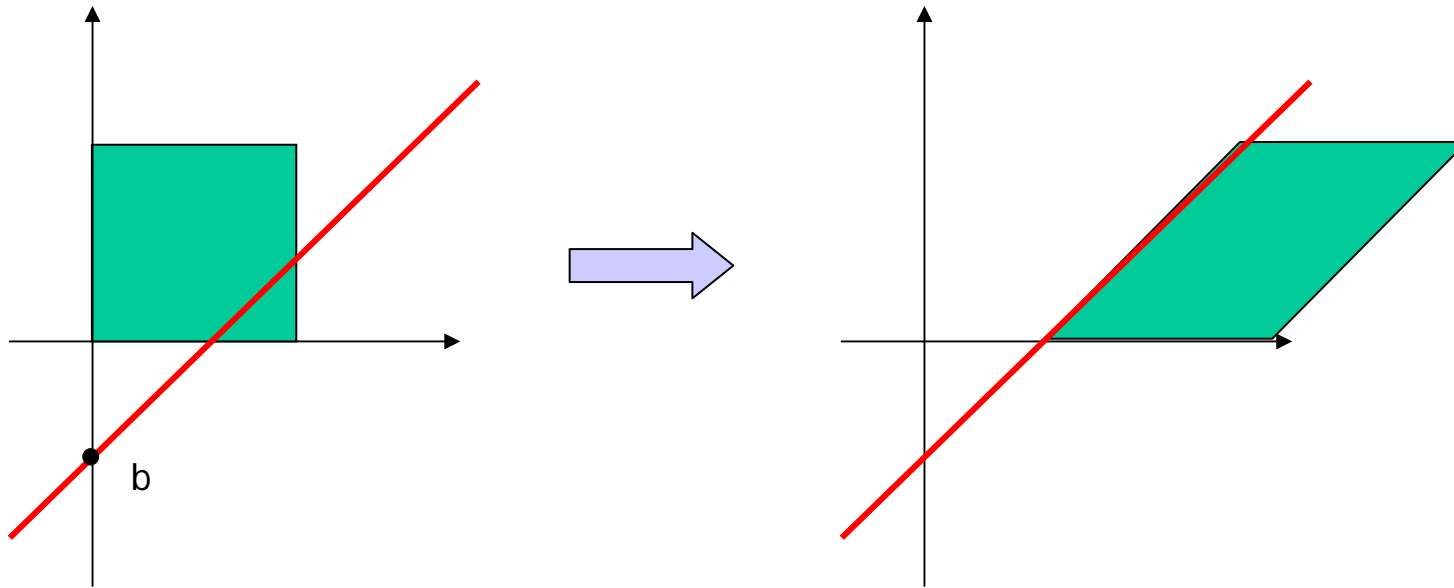
- En la otra dirección:

$$\begin{cases} x' = x \\ y' = y + bx \end{cases} \longrightarrow A = \begin{pmatrix} 1 & b & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



# Afilamiento general

- Afilarse con respecto a la recta  $y = mx + b$



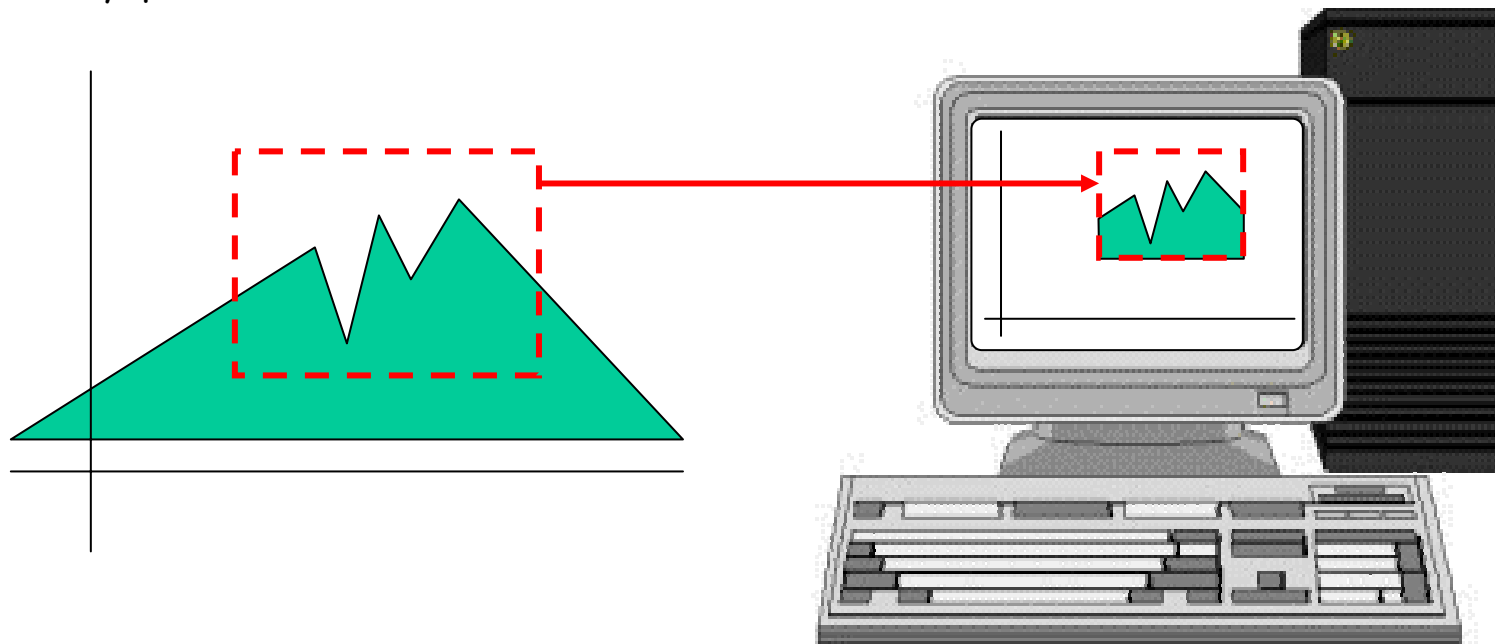
- La expresión de la recta despejando  $x$  es  $x = (y-b) / m$

$$\begin{cases} x' = x - (y-b) / m \\ y' = y \end{cases} \quad \longrightarrow \quad A = \begin{pmatrix} 1 & 0 & 0 \\ -1/m & 1 & 0 \\ b/m & 0 & 1 \end{pmatrix}$$

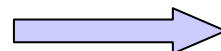


# Transformación a la ventana de visión

- La escena se almacenan según un sistema de coordenadas reales (metros, cm, pulgadas)
- El usuario verá en cada momento una subárea de la escena, o varias simultáneamente
- Cada subárea se mapeará en zonas distintas de la pantalla
- La pantalla viene definida por un sistema de coordenadas enteras (pixels)
- Hay que transformar de un sistema a otro



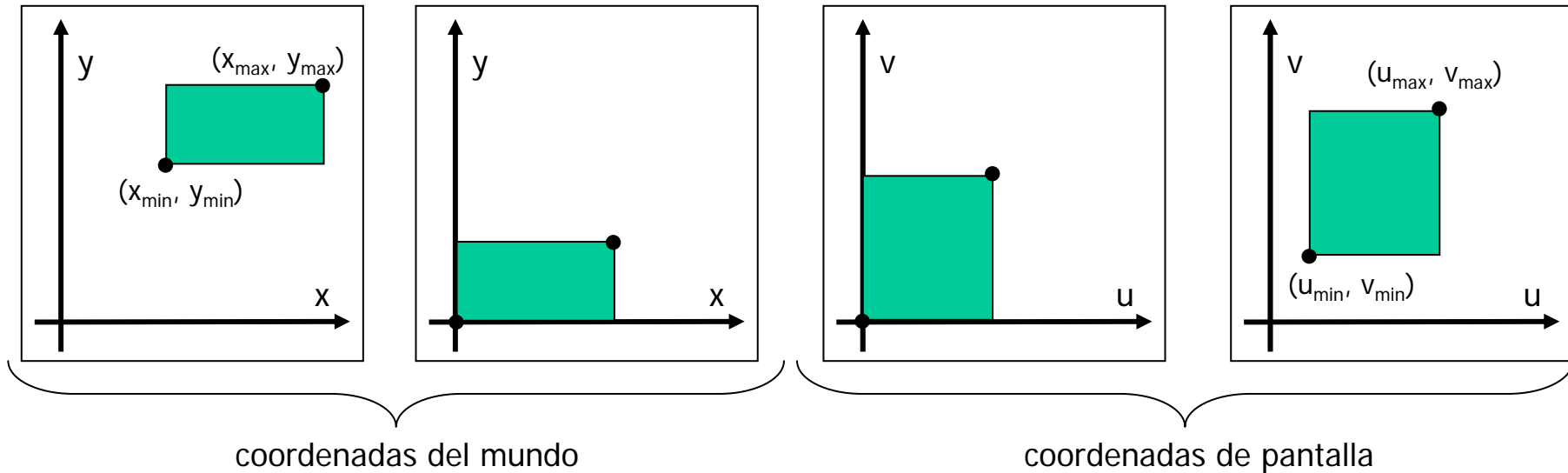
Escena (coordenadas reales)



Pantalla (coordenadas enteras)

# Matriz de transformación

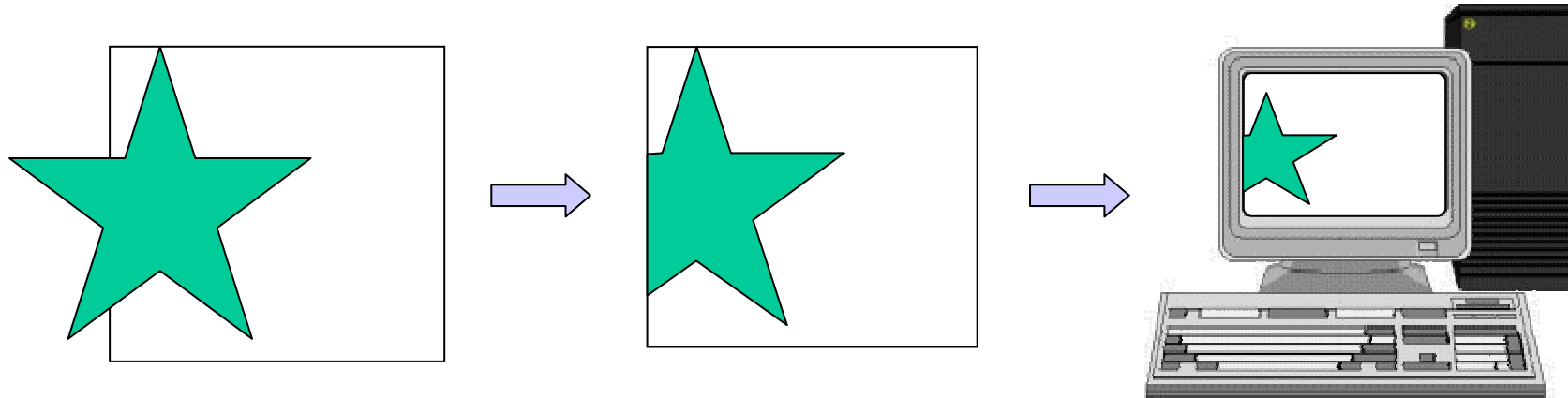
- Para pasar de un sistema a otro lo haremos también por medio de una matriz
- Dicha matriz la obtendremos a partir de una secuencia de transformaciones básicas



$$M = T(-x_{\min}, -y_{\min}) \cdot E \left( \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}}, \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} \right) \cdot T(u_{\min}, v_{\min})$$

# Algoritmos de recorte

- Después de la transformación, hay que recortar las partes de la escena que queden fuera de la ventana

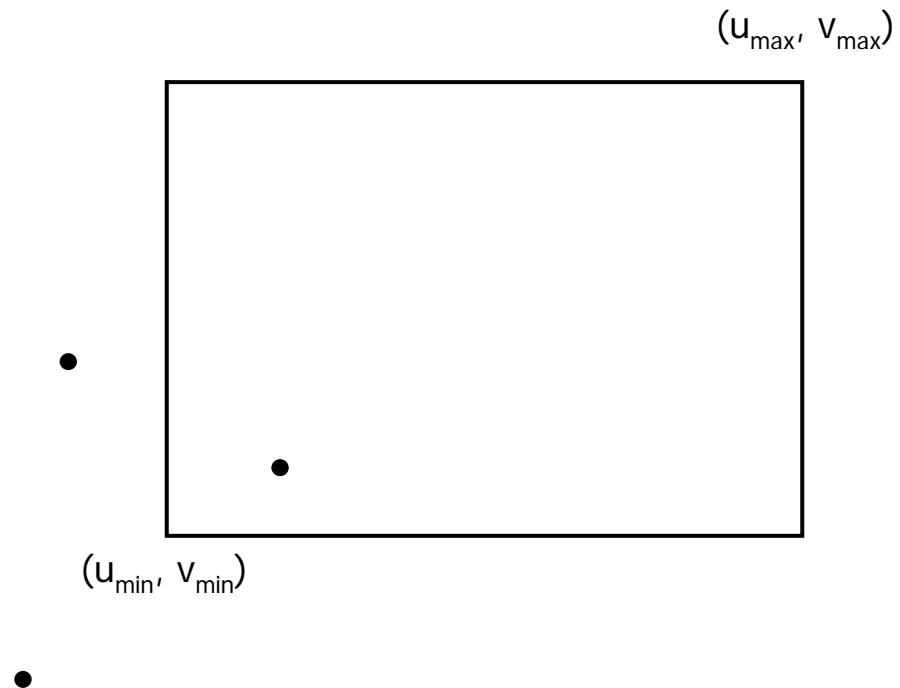


- Tenemos dos alternativas:
  - Podemos recortar en el mundo, y solo mapear lo que caiga dentro
  - Podemos transformar toda la escena, y recortar en la pantalla que es más rápido
- Estos algoritmos también se usan para permitir método de copiar y pegar en aplicaciones gráficas, pero a la inversa (recortando el interior)

# Recorte de puntos

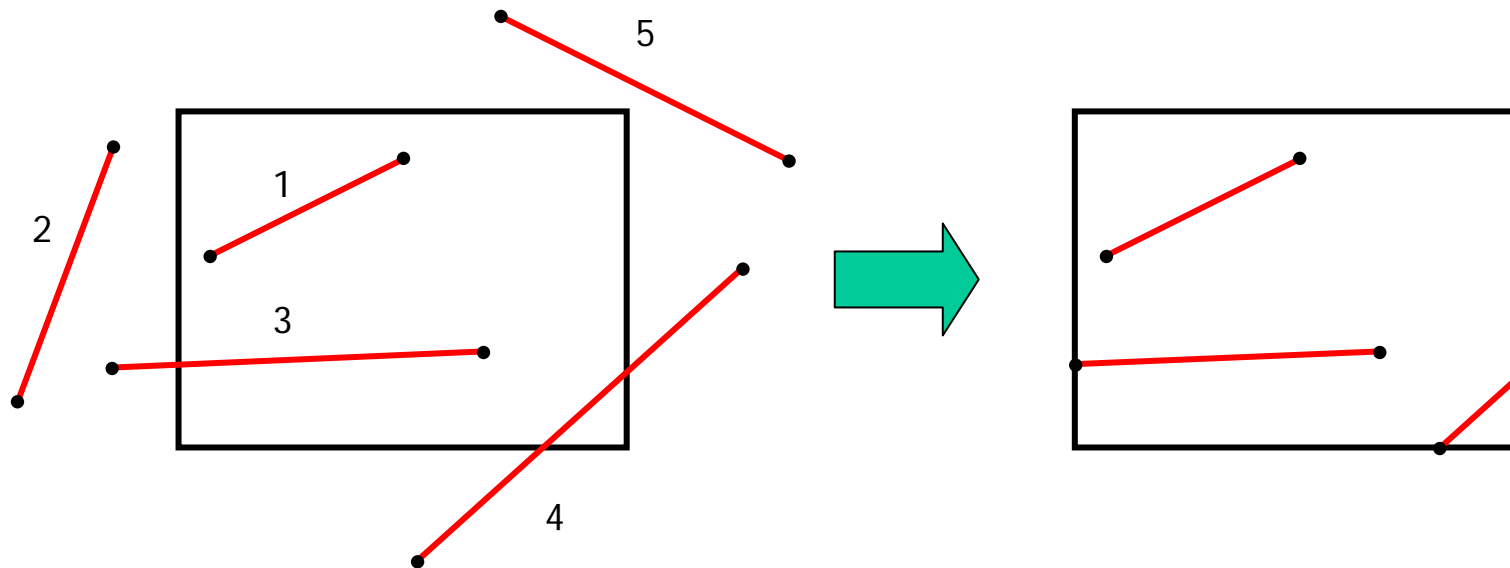
- Es el algoritmo más sencillo
- Asumiendo una ventana rectangular, sólo pintaremos los puntos que cumplan estas dos condiciones simultáneamente:

$$\begin{cases} u_{\min} \leq x \leq u_{\max} \\ v_{\min} \leq y \leq v_{\max} \end{cases}$$



# Recorte de líneas

- Dado un segmento recto, hay que testear qué parte cae dentro de la ventana para dibujar sólo esa parte
- ¿Cómo hacemos el test?

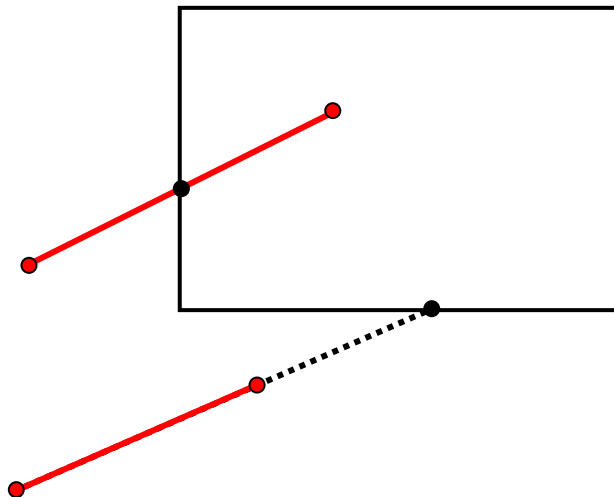


- Existe varios casos:
  - Si el segmento es completamente visible, lo dibujamos por completo
  - Si el segmento es completamente invisible, no lo dibujamos
  - Si sólo se ve parcialmente, habrá que calcular la intersección con los bordes de la ventana

# Cálculo de intersecciones

- ¿Cómo se calculan las intersecciones?
- En coordenadas explícitas es complicado, porque aparte de calcular el punto, debemos averiguar si pertenece al interior del segmento o no
- Es mucho más simple si usamos la ecuación paramétrica de la recta

$$\left\{ \begin{array}{l} x = x_1 + t(x_2 - x_1) \\ y = y_1 + t(y_2 - y_1) \end{array} \right\} \forall t \in [0,1]$$



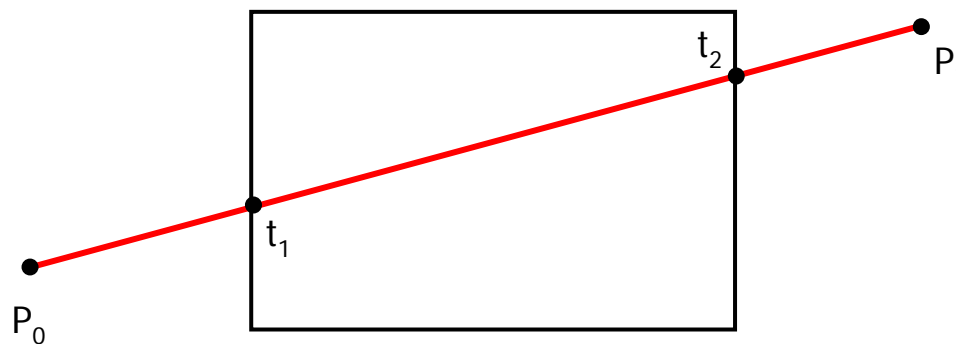
- Si la intersección se produce para un  $t$  fuera del intervalo  $(0,1)$ , el punto no pertenece al segmento
- Habría que aplicar este método para cada borde de la ventana
- Este test es muy costoso  $\rightarrow$  hay que resolver 2 ecuaciones para cada uno de los 4 bordes
- También hay que tener en cuenta el caso de las líneas que sean paralelas a un borde

# Algoritmo de recorte de Cyrus-Beck

- Funciona para cualquier tipo de ventana poligonal
- Se utiliza la ecuación paramétrica de la línea

$$P(t) = P_0 + t(P_1 - P_0)$$

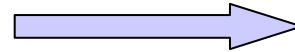
- Lo que se hace es calcular el valor de  $t$  de intersección con cada borde
- Una vez obtenidos los 4 valores, se buscan las intersecciones  $(x,y)$  correctas
- ¿Cómo se calcula el valor de  $t$  intersección?



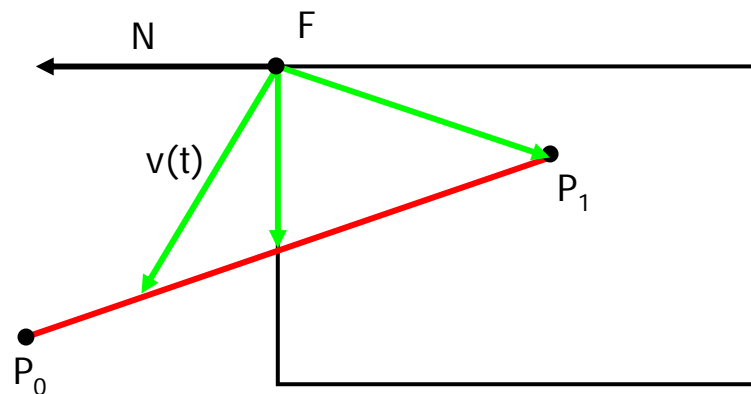
# Algoritmo de recorte de Cyrus-Beck

- Se elige un punto  $F$  arbitrario para cada borde
- Sea  $P(t)$  un punto sobre el segmento,  $t$  en  $[0,1]$   $P(t) = P_0 + t(P_1 - P_0)$
- Sea  $v(t)$  el vector desde  $F$  hasta un punto cualquiera del segmento  $v(t) = P(t) - F$
- Sea el producto escalar  $N_i \cdot v(t)$
- La intersección se produce cuando  $N_i \cdot v(t) = 0$
- Desarrollando y despejando  $t$  obtenemos:

$$N_i \cdot v(t) = N_i (P(t) - F) = \dots = 0$$



$$t = -\frac{N_i(P_0 - F)}{N_i(P_1 - P_0)}$$

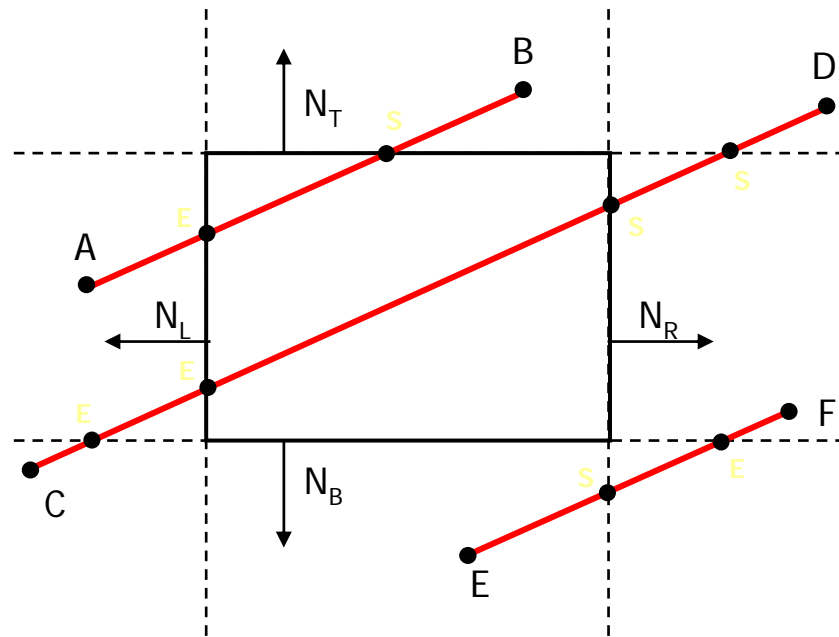


- Hay que controlar que el denominador no se haga cero:
  - Siempre se cumple que  $N_i \neq 0$
  - Siempre se cumple que  $(P_1 - P_0) \neq 0$
  - Puede ocurrir que  $N_i \cdot (P_1 - P_0) = 0$
  - En ese caso, la línea es paralela al borde y no existe intersección



# Algoritmo de recorte de Cyrus-Beck

- Aplicando la fórmula obtenemos los 4 valores de  $t$  que indican las intersecciones de la línea con los 4 bordes.
- ¿Cómo identificamos cuáles son las dos correctas?
- Los valores de  $t$  fuera del rango  $(0,1)$  se descartan
- Cada valor de  $t$  se etiqueta como entrante ( $t_E$ ) o saliente ( $t_S$ ), según entre o salga hacia el lado donde se encuentra la ventana
- ¿Cómo saberlo?



- Mirando el signo de  $N_i \cdot (P_1 - P_0)$ 
  - Si es negativo  $\rightarrow$  punto entrante
  - Si es positivo  $\rightarrow$  punto saliente
- La solución viene dada por el tramo de línea entre el  $P_E$  más alto y el  $P_S$  más bajo
- Si  $t_E > t_L \rightarrow$  la línea no se dibuja
- Una vez obtenidos los valores de  $t$ , se sustituyen en la ecuación paramétrica para obtener las coordenadas  $(x,y)$  de los puntos

# Algoritmo de recorte de Cyrus-Beck

- Es muy rápido calcular si es entrante o saliente: el producto escalar ya estaba calculado!
- Para implementarlo se usan dos variables,  $t_E$  y  $t_S$ , donde se va almacenando el  $t_E$  más grande y el  $t_S$  más pequeño que vayamos encontrando
- Al acabar los cálculos con los 4 bordes, se comparan los  $t_E$  y  $t_S$  finales, y se decide si dibujar o no

```
Funcion Cyrus_Beck (...)  
Para cada borde de la ventana  
    Calcular t  
    Si t es entrante y t > te entonces te = t  
    Si t es saliente y t < ts entonces ts = t  
Fin Para  
Si te < ts entonces  
    Calcular Pe = P(te)  
    Calcular Ps = P(ts)  
    Pintar linea (Pe, Ps)  
Fin Si
```

# Ejemplo de recorte

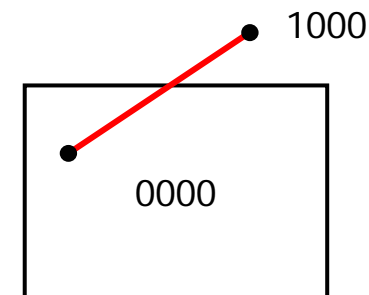
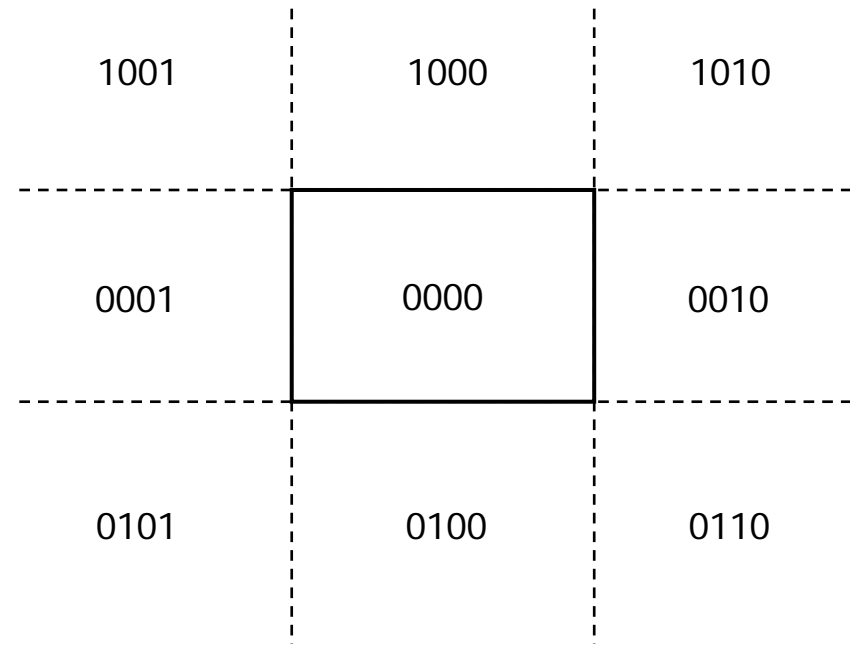
## 2. Recorte 2D.

### Cyrus-Beck

- (a) Explicar el algoritmo de ~~Sutherland - Cohen~~. (1.5 puntos)
- (b) Hacer una traza del algoritmo para recortar las aristas  $[(-1,9)..(4,4)]$ ,  $[(8,7)..(11,8)]$  y  $[(-3,1)..(1,-3)]$  sobre la ventana rectangular cuyas esquinas son  $(0,0)$  y  $(10,6)$ . (2 puntos)

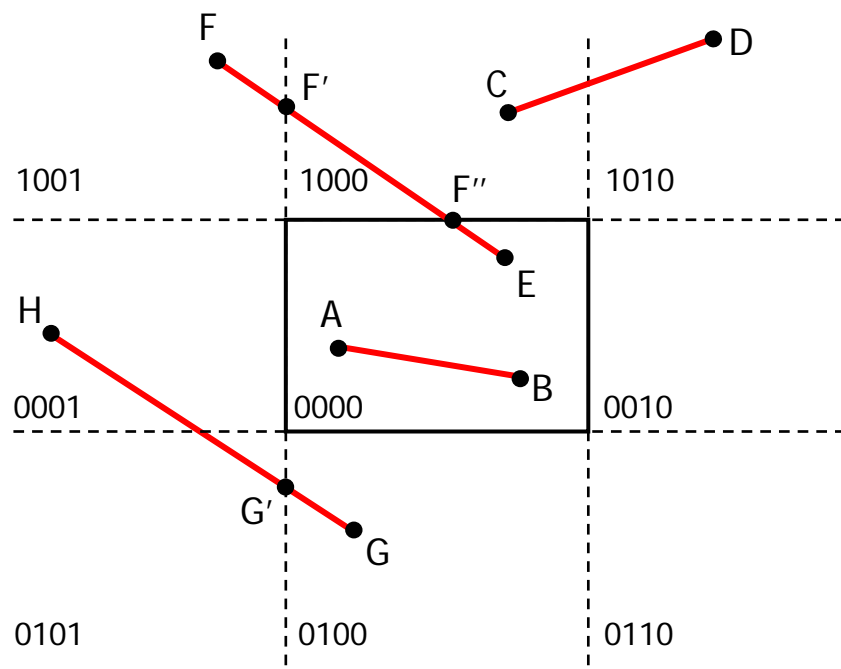
# Algoritmo de recorte de Cohen-Sutherland

- Se ejecuta primero un test inicial para reducir el número de intersecciones a calcular
- A cada extremo de la línea le asignamos un código de 4 bits
- Cada bit indica si el punto está a un lado o a otro de cada borde
  - El primer bit indica si el punto está por encima del borde superior
  - El segundo bit indica si el punto está por debajo del borde inferior
  - El tercer bit indica si el punto está a la derecha del borde derecho
  - El cuarto bit indica si el punto está a la izquierda del borde izquierdo
- Cada bit se calcula mediante una resta
- Por ejemplo, el primer bit viene dado por el signo de la resta  $(y - y_{\max})$



# Algoritmo de recorte de Cohen-Sutherland

- A continuación hacemos una operación AND entre los códigos de ambos extremos
  - Si el AND  $\neq 0 \rightarrow$  la línea es completamente invisible y la descartamos
  - Si el AND  $= 0$  siendo ambos códigos 0000  $\rightarrow$  la línea es totalmente visible y la pintamos íntegra
  - Si no es ninguno de los casos anteriores, la línea la catalogamos como parcialmente visible, y sólo en este caso nos ponemos a calcular intersecciones



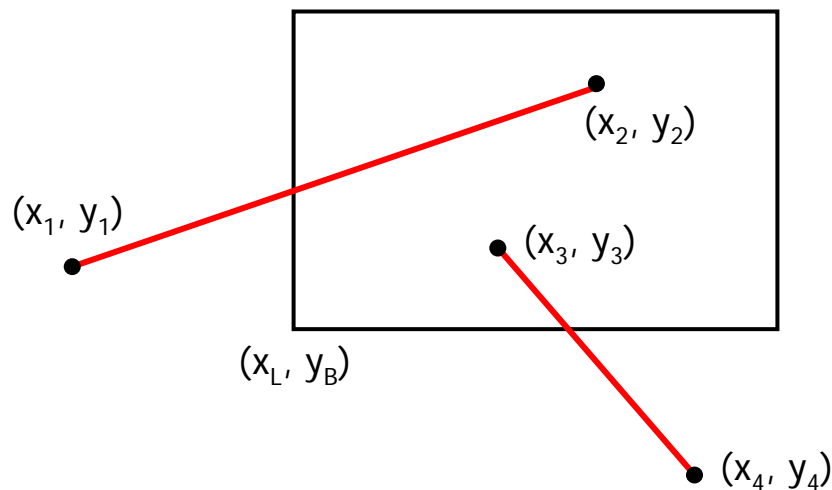
- Línea AB  $\rightarrow$  totalmente visible
- Línea CD  $\rightarrow$  totalmente invisible
- Línea EF  $\rightarrow$  parcialmente visible
  - Calculamos la intersección con uno de los bordes puestos a 1  $\rightarrow F'$
  - Ahora la línea se divide en dos:  $FF'$  y  $EF'$
  - La línea  $FF'$  es invisible  $\rightarrow$  se descarta
  - La línea  $EF'$  es parcialmente visible  $\rightarrow$  calculamos la intersección y obtenemos  $F''$
  - La línea  $F'F''$  es invisible  $\rightarrow$  se descarta
  - La línea  $EF''$  es totalmente visible  $\rightarrow$  se pinta
- Línea HG  $\rightarrow$  parcialmente visible
  - Calculamos la intersección  $\rightarrow G'$
  - La línea  $GG'$  es invisible  $\rightarrow$  se descarta
  - La línea  $HG'$  es invisible  $\rightarrow$  se descarta

# Algoritmo de recorte de Cohen-Sutherland

- Para calcular la intersección con los bordes se utiliza la ecuación de la línea

$$y = y_1 + m(x - x_1), \quad \text{siendo } m = (y_2 - y_1) / (x_2 - x_1)$$

- Para un borde vertical, la ecuación es  $x = x_L$
- Sustituimos en la ecuación de la línea y obtenemos el valor de  $y$



- Para un borde horizontal, la ecuación es  $y = y_B$

- La ecuación de la línea puede ponerse como

$$x = x_3 + (y - y_3) / m$$

# Ejemplo de recorte

## 2. Recorte 2D.

- (a) Explicar el algoritmo de Sutherland - Cohen. (1.5 puntos)
- (b) Hacer una traza del algoritmo para recortar las aristas  $[(-1,9)..(4,4)]$ ,  $[(8,7)..(11,8)]$  y  $[(-3,1)..(1,-3)]$  sobre la ventana rectangular cuyas esquinas son  $(0,0)$  y  $(10,6)$ . (2 puntos)

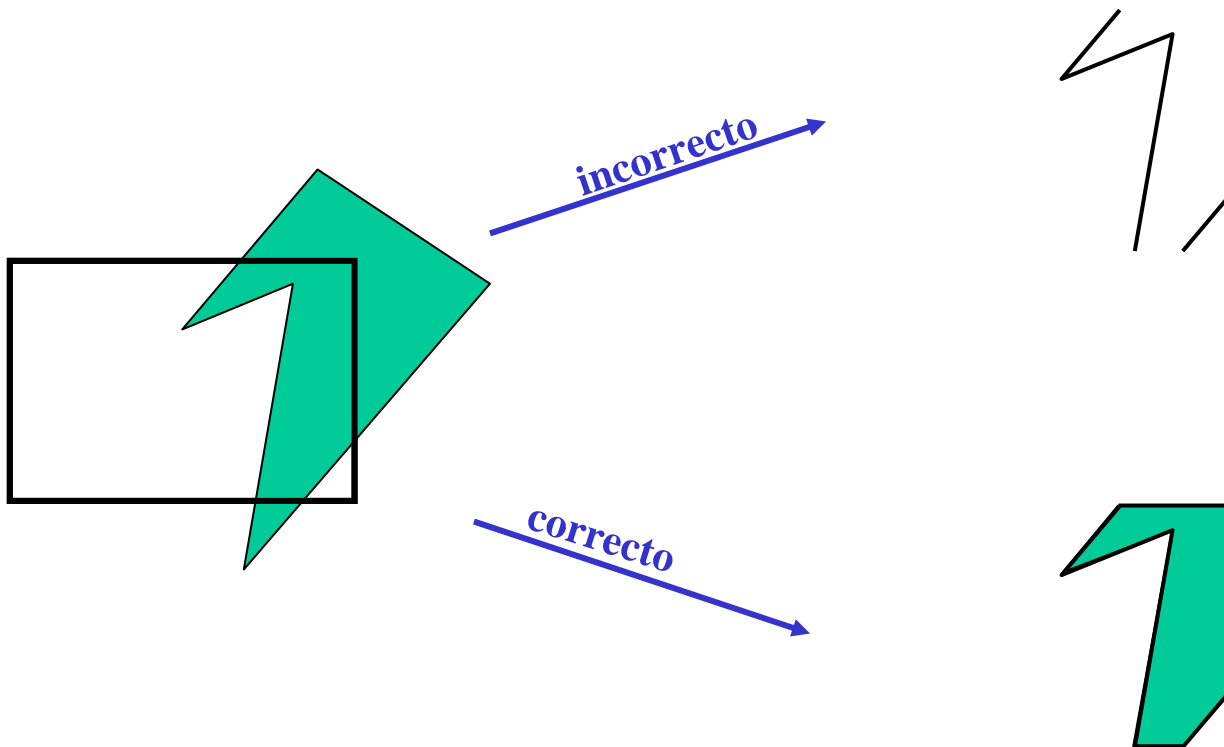
# Comparativa de ambos algoritmos

- El algoritmo *Cyrus-Beck* es más eficiente que *Cohen-Sutherland* porque sólo calcula las coordenadas cartesianas  $(x,y)$  al final
- Funciona muy bien cuando la mayoría de las líneas cae en el interior de la ventana de recorte
- El algoritmo *Cohen-Sutherland* es mejor cuando la mayoría de las líneas cae fuera de la ventana, porque se rechazan en el test inicial



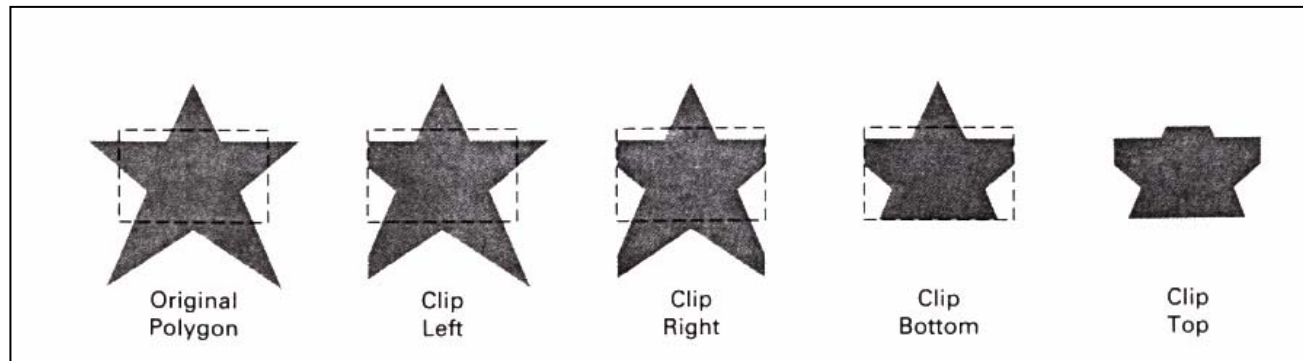
# Recorte de polígonos

- Para recortar polígonos no basta sólo con recortar sus aristas
- Lo que hace falta es un algoritmo que genere una o más áreas cerradas que puedan rellenarse si se desea
- Es decir, queremos la secuencia de vértices que represente al polígono recortado

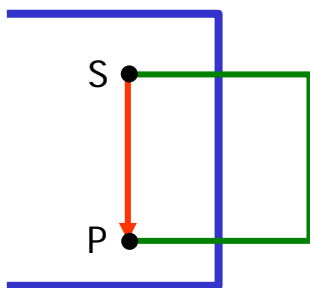


# Algoritmo de recorte de Sutherland-Hodgeman

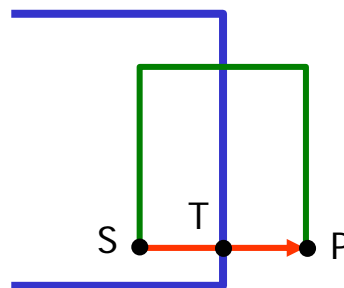
- En cada iteración vamos recortando el polígono frente a uno de los bordes



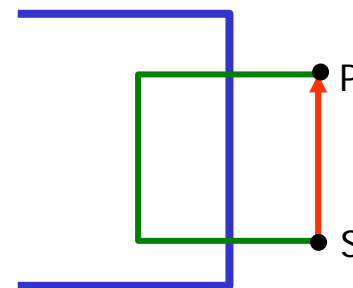
- Vamos recorriendo la lista de vértices del polígono, y se genera una lista de salida
- Existen 4 casos posibles:



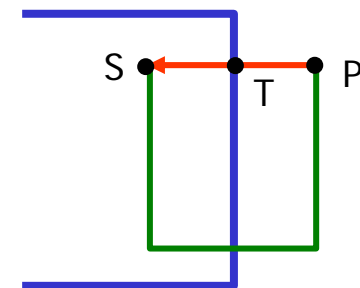
Se añade P  
a la lista



Se añade T  
a la lista

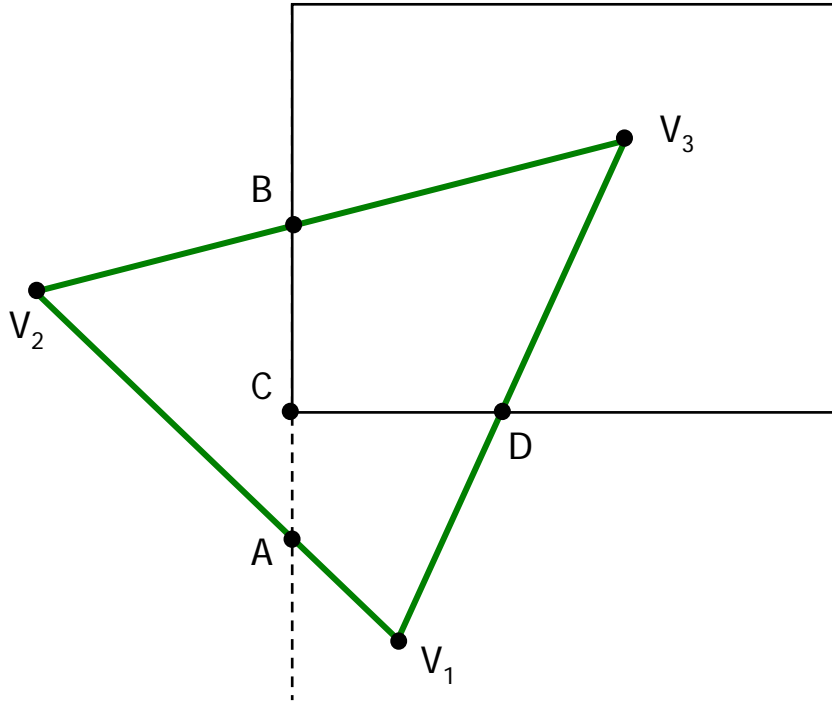


No se añade  
ningún vértice



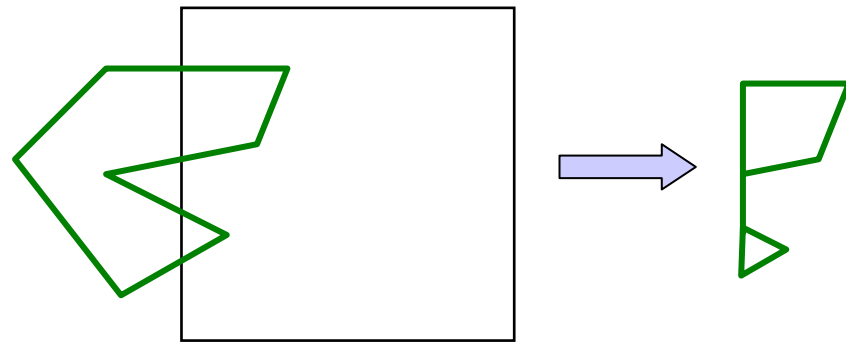
Se añaden T  
y P a la lista

# Ejemplo



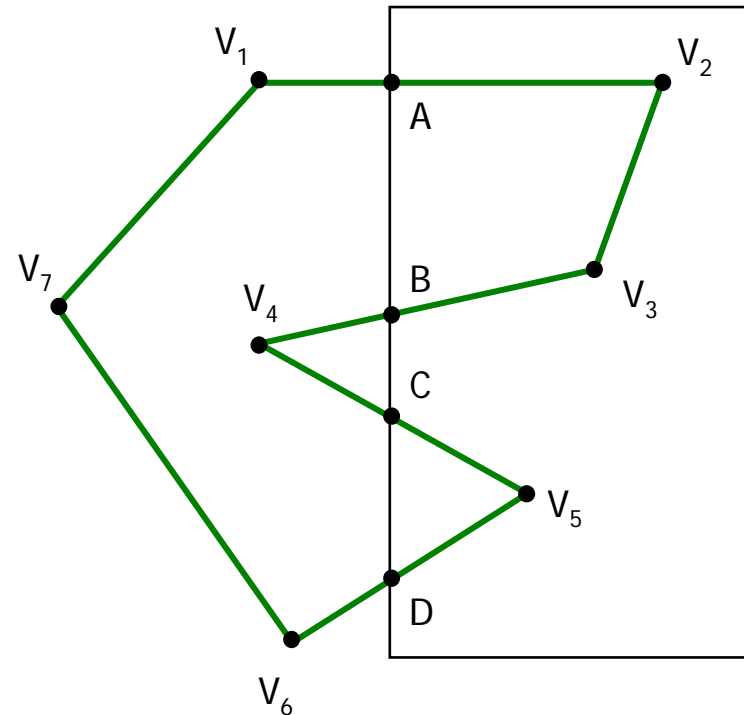
- Se hace por separado para cada borde porque si no no se sabría si estamos dentro o fuera
- Borde izquierdo:  $V_1, A, B, V_3$
- Borde derecho:  $V_1, A, B, V_3$
- Borde inferior:  $C, B, V_3, D$
- Borde superior:  $C, B, V_3, D$

- Existen problemas con los polígonos cóncavos en algunas orientaciones  $\rightarrow$  aparecen líneas falsas!



# Algoritmo de recorte de Weiler-Atherton

- Arregla el problema de los polígonos cóncavos. ¿Cómo?
- El problema del algoritmo anterior es que devuelve un único polígono de salida
- La solución consiste en poder devolver más de un polígono
- En lugar de viajar por los vértices del polígono, a veces nos moveremos por los bordes de la ventana
- Las reglas para decidir el movimiento son:
  - Si en una arista viajamos de fuera a dentro, seguimos la arista del polígono
  - Si la dirección es de dentro a fuera, seguimos por el borde de la ventana
- Comenzaremos siempre en un punto exterior (si todos son interiores, el polígono es completamente visible)



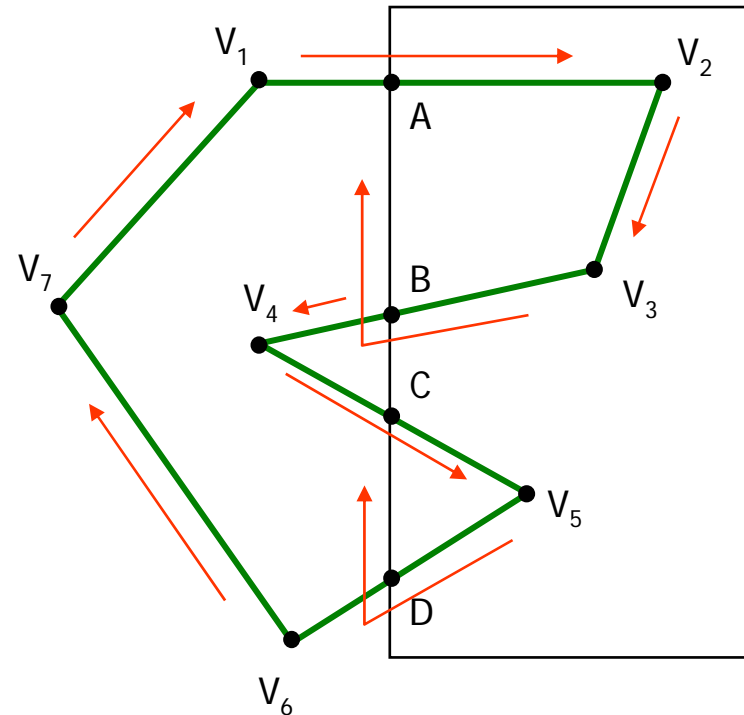
# Algoritmo de recorte de Weiler-Atherton

- Comenzamos con  $V_1$
- Como cruzamos de fuera hacia adentro, nos quedamos con la intersección  $A$
- Al llegar a  $B$  (de dentro hacia fuera) unimos  $B$  con  $A$  y creamos el polígono

$\{ A, V_2, V_3, B \}$

- Continuamos a partir de  $B$
- Como cruzamos de fuera hacia adentro, nos quedamos con la intersección  $C$
- Al llegar a  $D$  (de dentro hacia fuera) unimos  $D$  con  $C$  y creamos el polígono

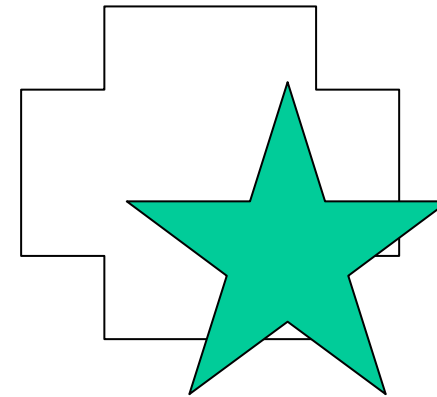
$\{ C, V_5, D \}$



# Corolario

- A veces se necesita un recorte externo (recortar lo que caiga dentro de la ventana).
- Por ejemplo, en operaciones de cortar y pegar
- El algoritmo es similar, pero quedándonos con la parte de fuera

- Cuando la ventana no es rectangular, o el objeto es curvo, se complican un poco los cálculos, pero también se puede conseguir



- Cuando el número de objetos es muy grande, se suele hacer una etapa previa de *Bounding Rectangle*

- Figura 1: completamente invisible
- Figura 2: completamente visible
- Figura 3: hay que calcular el recorte

