

# Computación Gráfica

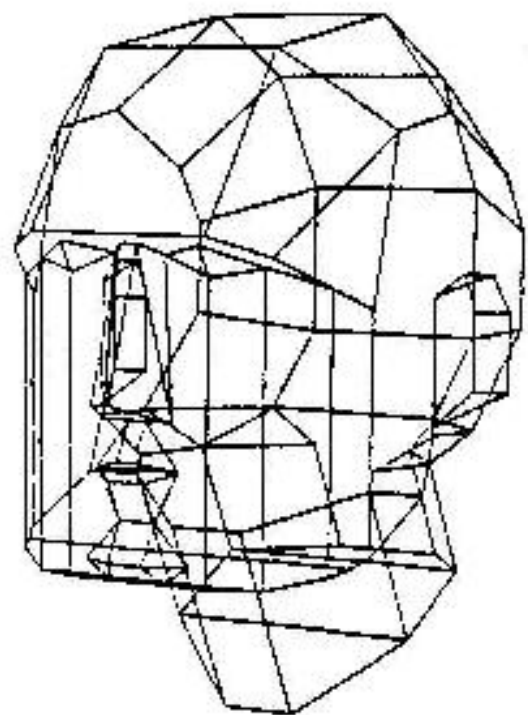
Eduardo Fernández

# Determinación de superficies visibles

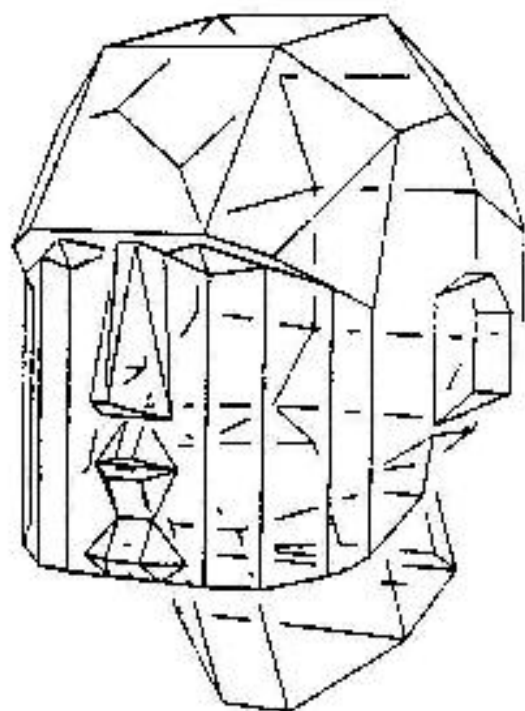
Basado en: **Capítulo 13**

Del Libro: **Introducción a la Graficación  
por Computador**

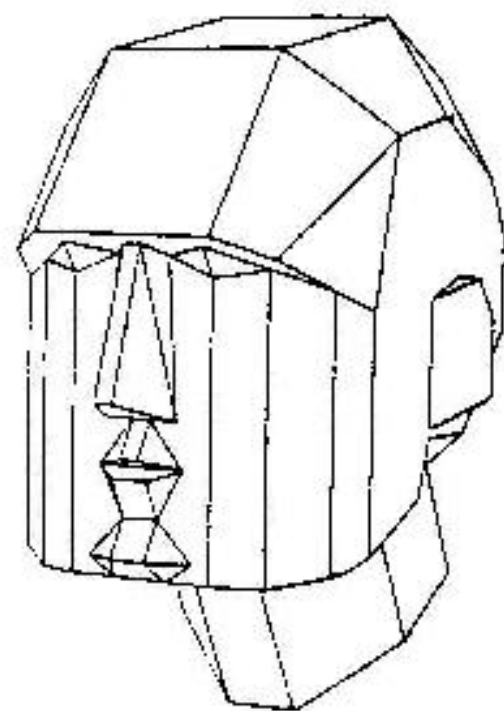
*Foley – Van Dam – Feiner – Hughes - Phillips*



(a)



(b)



(c)

# Dos métodos fundamentales

## 1.- Algoritmo de precisión de imagen:

**For** (cada pixel de la imagen) {

Determinar el objeto más cercano al observador  
que es atravesado por el rayo proyector a  
través del pixel;

Dibujar el pixel con el color apropiado;

}

# Dos métodos fundamentales

## 2.- Algoritmo de precisión de objeto:

**For** (cada objeto del mundo) {

Determinar aquellas partes del objeto cuya vista  
no está obstruida por otras partes del mismo  
objeto o por otro objeto;

Dibujar esas partes con el color apropiado;

}

# Diferencias

- En precisión de objeto no se considera la resolución de la pantalla para los cálculos => el dibujo en pantalla es el último paso.
- En precisión de imagen, al ampliar una imagen hay que rehacer todos los cálculos.
- Los algoritmos de precisión de objeto se hicieron inicialmente para sistemas gráficos vectoriales.
- Los algoritmos de precisión de imagen se hicieron para sistemas gráficos de barrido.
- Los algoritmos más recientes combinan la precisión de imagen y la precisión de objeto.

# Técnicas generales para algoritmos eficientes

1. Coherencia
2. Transformación en perspectiva
3. Extensiones y volúmenes acotantes
4. Eliminación de caras posteriores
5. Partición espacial
6. Jerarquía

# 1.- Coherencia

**Coherencia = similitud local, continuidad, etc.**

**Motivación de la coherencia: ahorrar cálculos.**

**Hay 8 tipos de coherencia identificados:**

1. Coherencia de objetos
2. Coherencia de caras
3. Coherencia de aristas
4. Coherencia de aristas implicadas
5. Coherencia de líneas de barrido
6. Coherencia de áreas
7. Coherencia de profundidad
8. Coherencia de cuadros



## 1.- Coherencia

### 1. Coherencia de objetos

Si un objeto **A** está separado de otro **B**, no es necesario fijarse en la superposición de cada parte de **A** con cada parte de **B**

### 1. Coherencia de caras

Las propiedades de las superficies varían suavemente sobre una misma cara. Se pueden hacer cálculos incrementales.

### 1. Coherencia de aristas

Una arista cambia de visibilidad, solo cuando cruza detrás de otra arista (visible) o penetra en una cara visible.

## 1.- Coherencia

### 4. Coherencia de aristas implicadas

Si una cara penetra a otra, se genera una nueva arista (la línea de intersección). Esta se puede determinar a partir de 2 ptos de intersección.

### 4. Coherencia de líneas de barrido

Entre una línea de barrido y otra, los tramos visibles de los objetos difieren muy poco.

### 4. Coherencia de áreas

Píxeles adyacentes, en general se corresponden con una misma cara visible.

## 1.- Coherencia

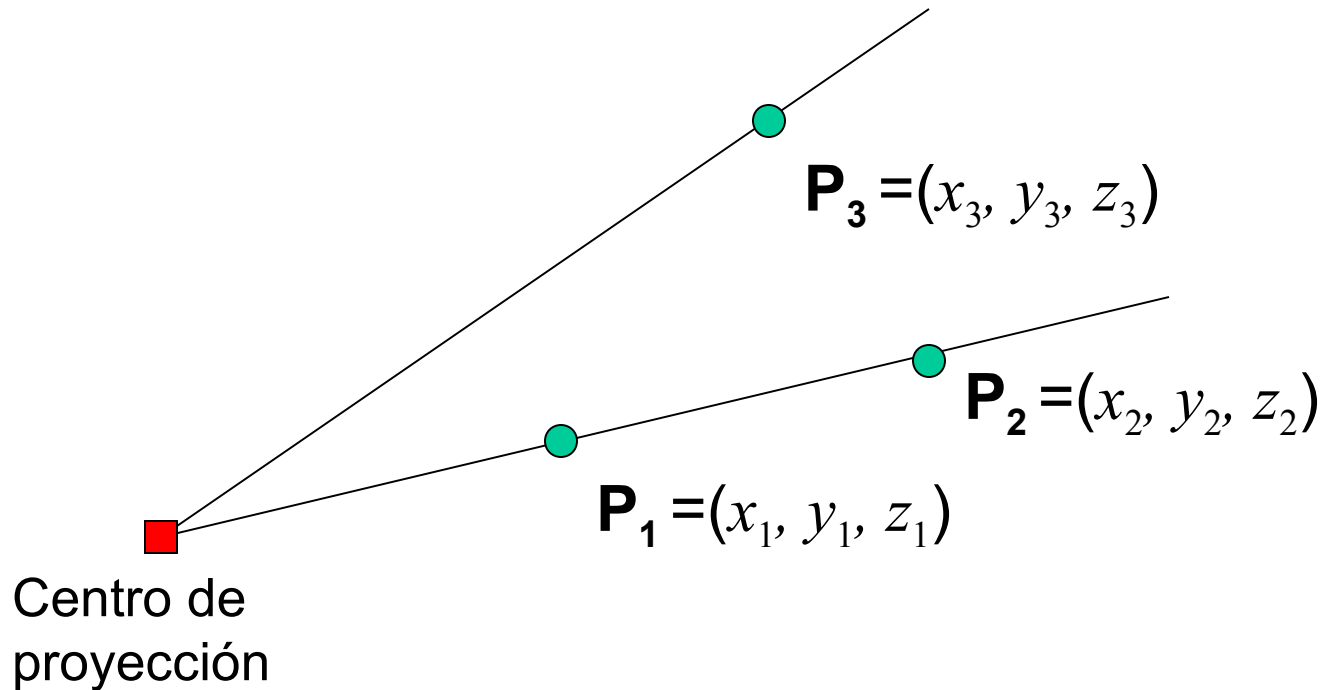
### 7. Coherencia de profundidad

- Partes cercanas de una misma superficie están muy cerca en profundidad.
- Superficies distintas en el mismo lugar de la pantalla están a mayor separación de profundidad.

### 8. Coherencia de cuadros

Las imágenes del mismo ambiente en dos instantes sucesivos en el tiempo casi siempre serán bastante similares. Los calculos realizados para una imagen se pueden reutilizar en la siguiente.

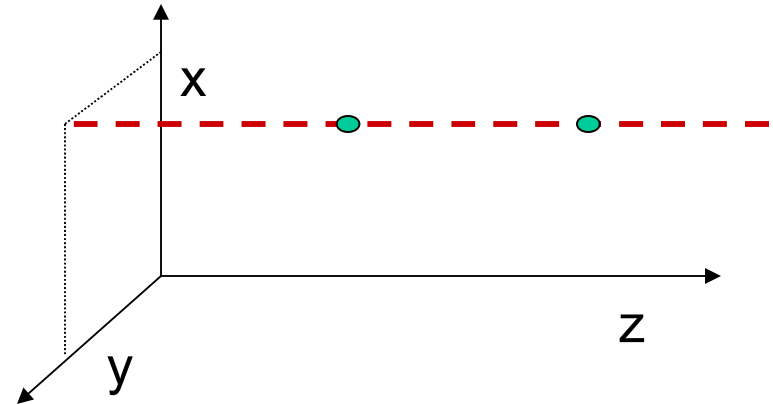
## 2.- Transformación de perspectiva



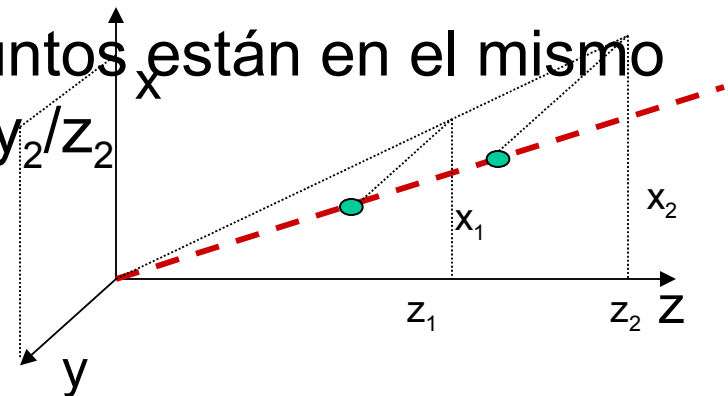
Se hacen transformaciones de normalización, para que los rayos proyectores sean paralelos al eje  $z$  (**proyección paralela**), o para que los rayos proyectores emanen del origen (**proyección en perspectiva**).

## 2.- Transformación de perspectiva

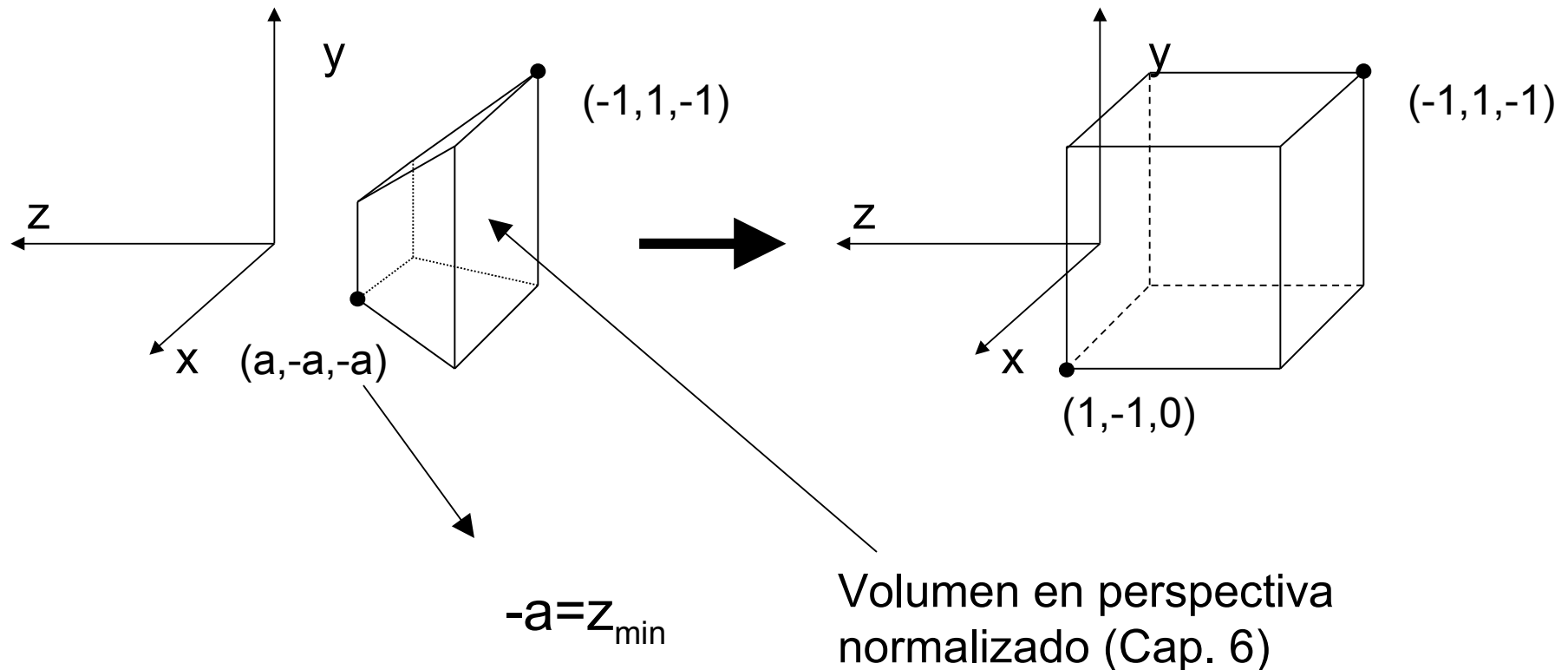
- Proyección paralela: puntos están en el mismo proyector si  $x_1 = x_2$ ,  $y_1 = y_2$



- Proyección de perspectiva: puntos están en el mismo proyector si  $x_1/z_1 = x_2/z_2$ ,  $y_1/z_1 = y_2/z_2$



## 2.- Transformación de perspectiva

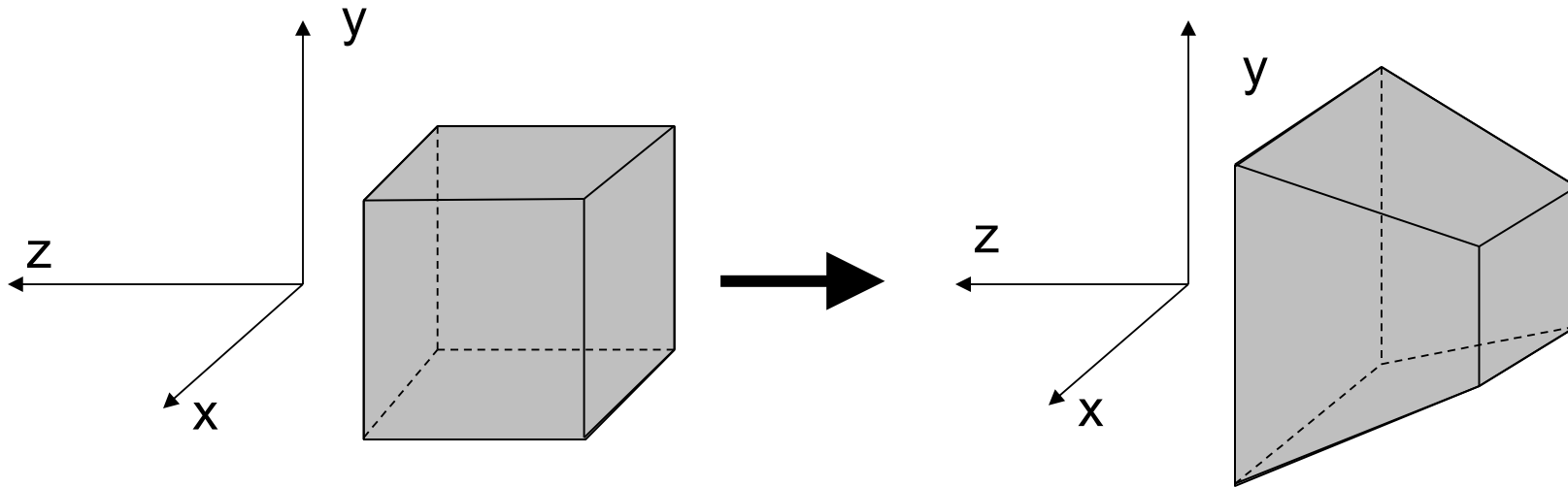


## 2.- Transformación de perspectiva

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1 + z_{\min}} & \frac{-z_{\min}}{1 + z_{\min}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

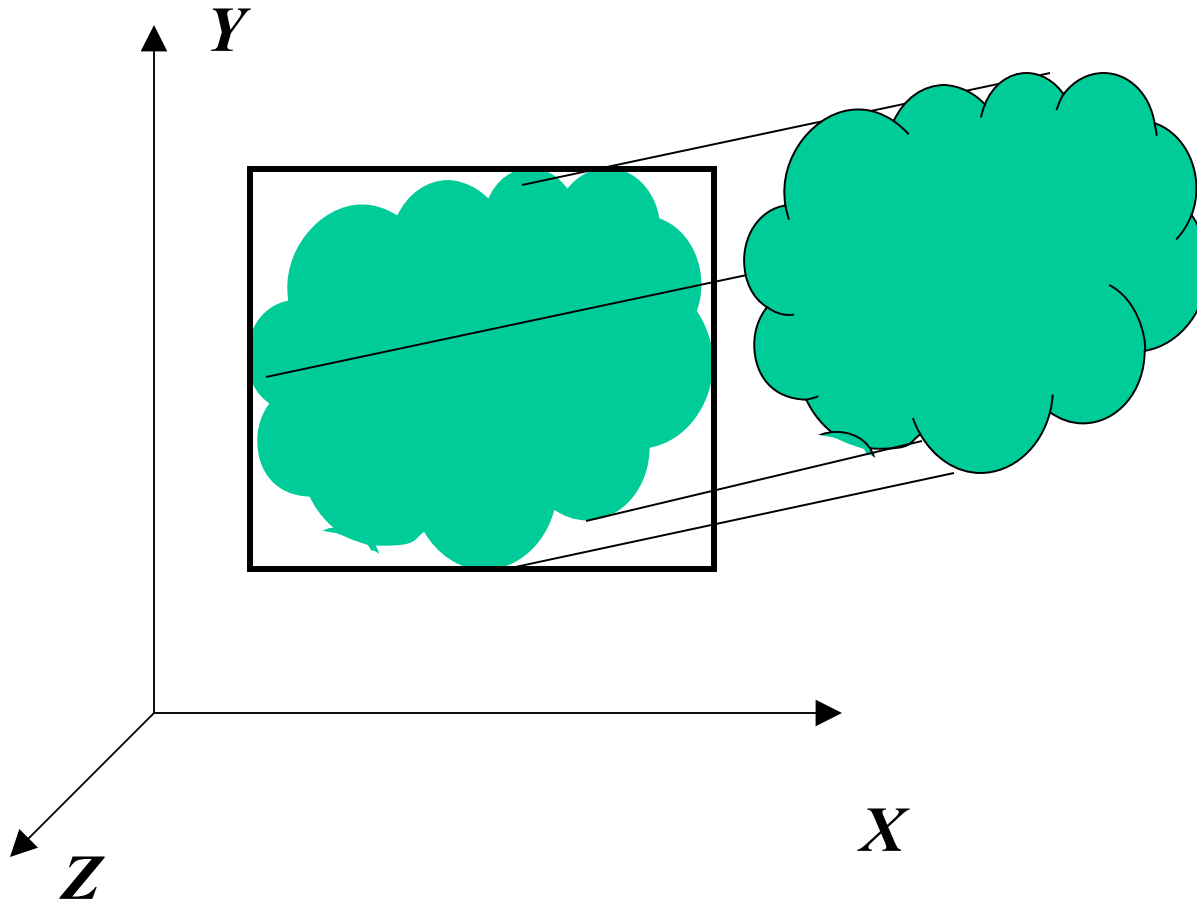
$$z_{\min} \neq -1$$

## 2.- Transformación de perspectiva

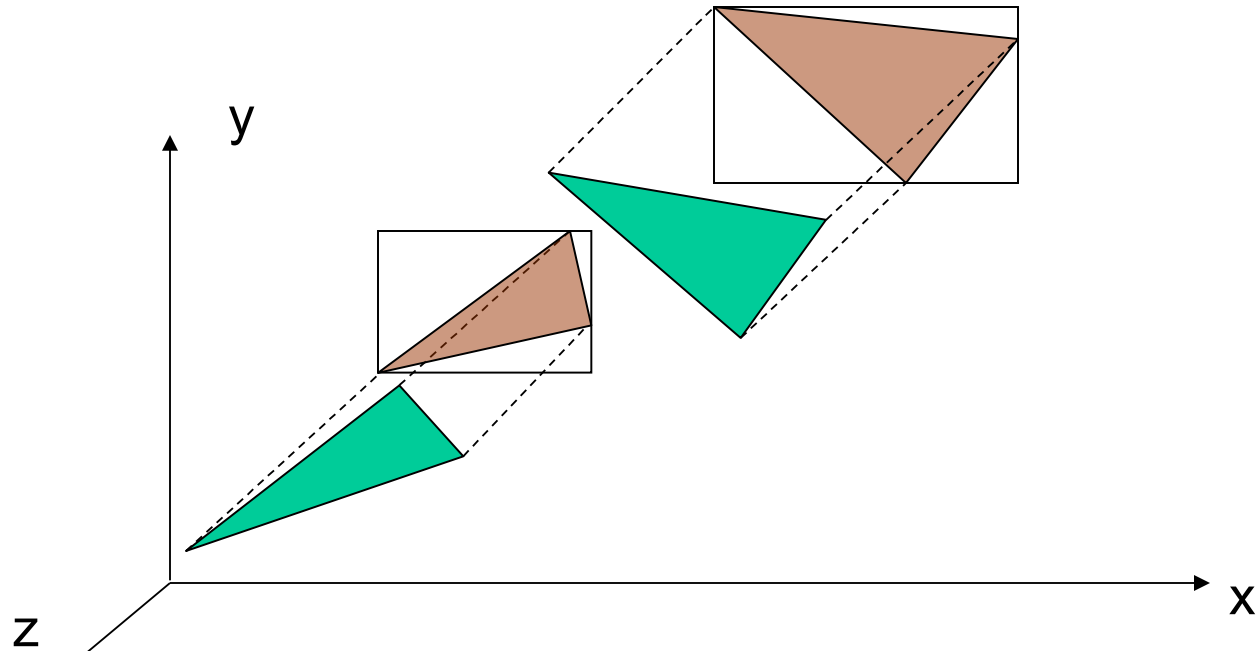




### 3.- Extensiones y volúmenes acotantes



## 3.- Extensiones y volúmenes acotantes

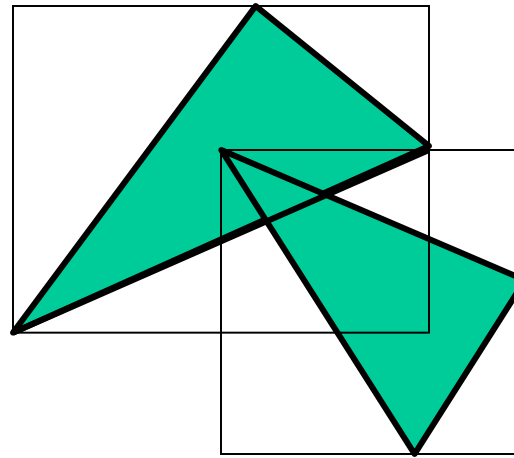
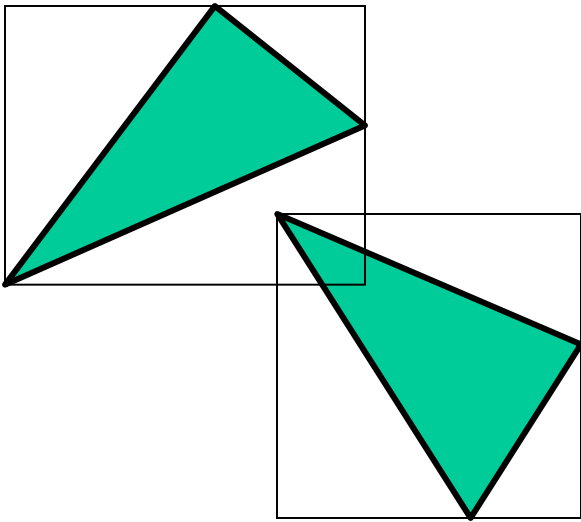


### Prueba minmax

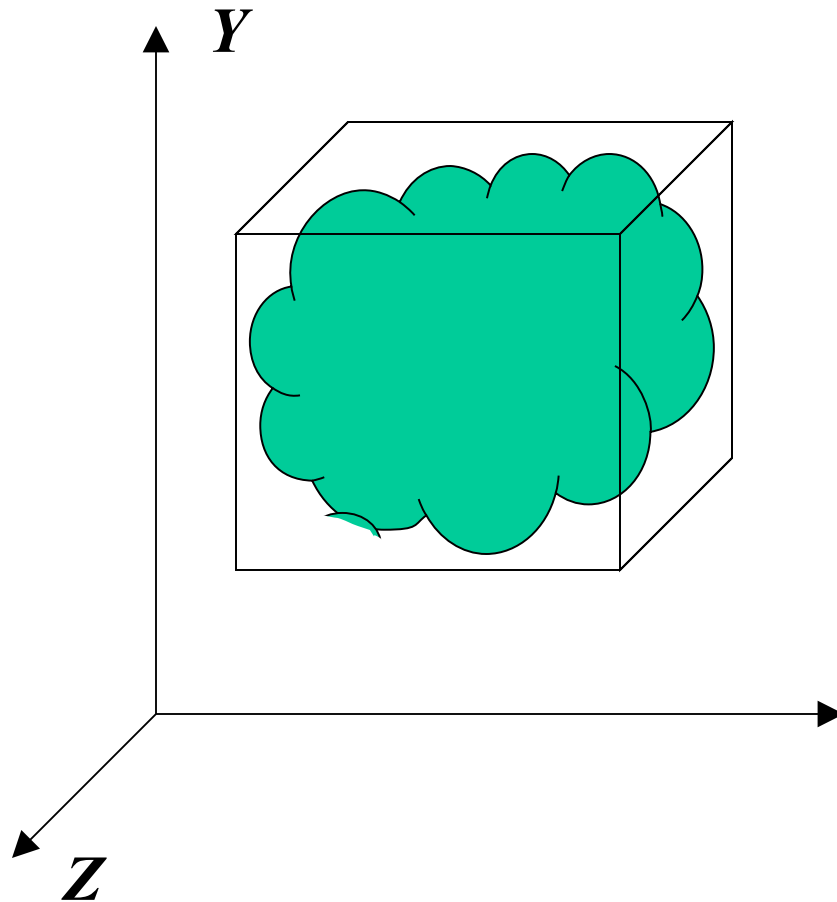
No hay superposición si:

$$\begin{aligned} &(((x_{\max 2} < x_{\min 1}) \text{ or } (x_{\max 1} < x_{\min 2})) \text{ or} \\ &((y_{\max 2} < y_{\min 1}) \text{ or } (y_{\max 1} < y_{\min 2}))) \end{aligned}$$

### 3.- Extensiones y volúmenes acotantes



### 3.- Extensiones y volúmenes acotantes



Caja acotante, o  
volumen acotante.

Prueba minmax

La prueba incluye el  
control de z

$((z_{\max 2} < z_{\min 1}) \text{ or } (z_{\max 1} < z_{\min 2}))$

### 3.- Extensiones y volúmenes acotantes

Si hay otros volúmenes acotantes, ¿cómo se los elige?

$$T = bB + oO$$

$T$  = Costo de los test de intersección de un objeto.

$b$  = Número de veces que el volumen acotante es testeado.

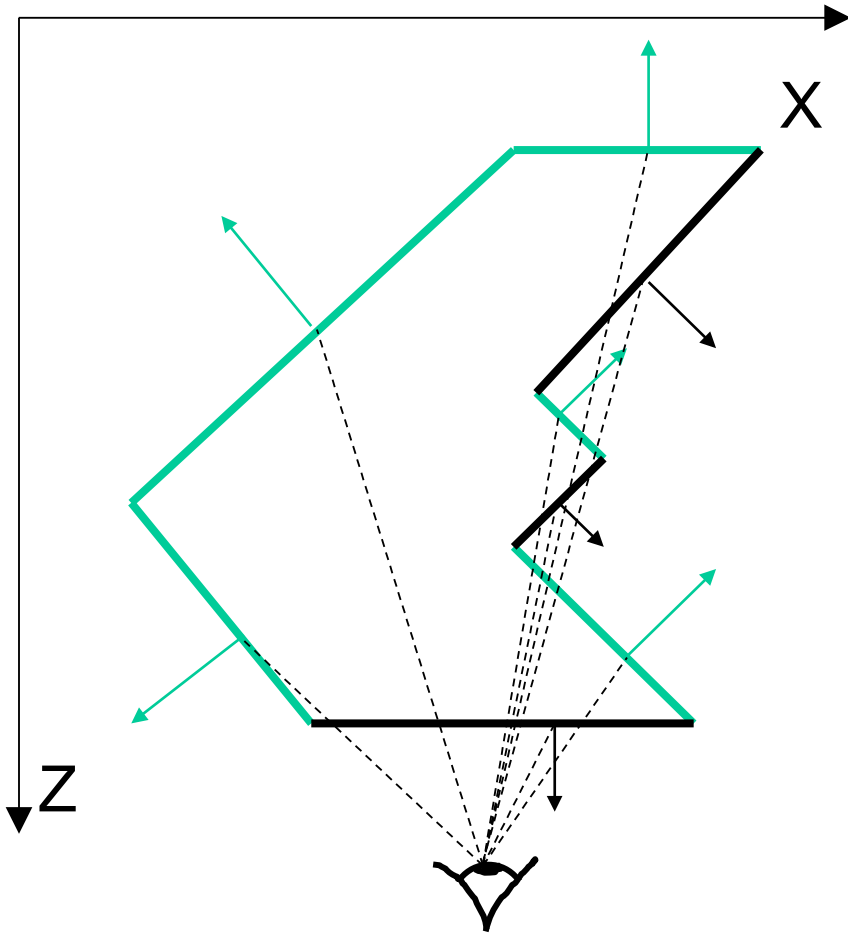
$B$  = Costo de calcular un test de intersección entre 2 v.a.

$o$  = Número de veces que el objeto es testeado.

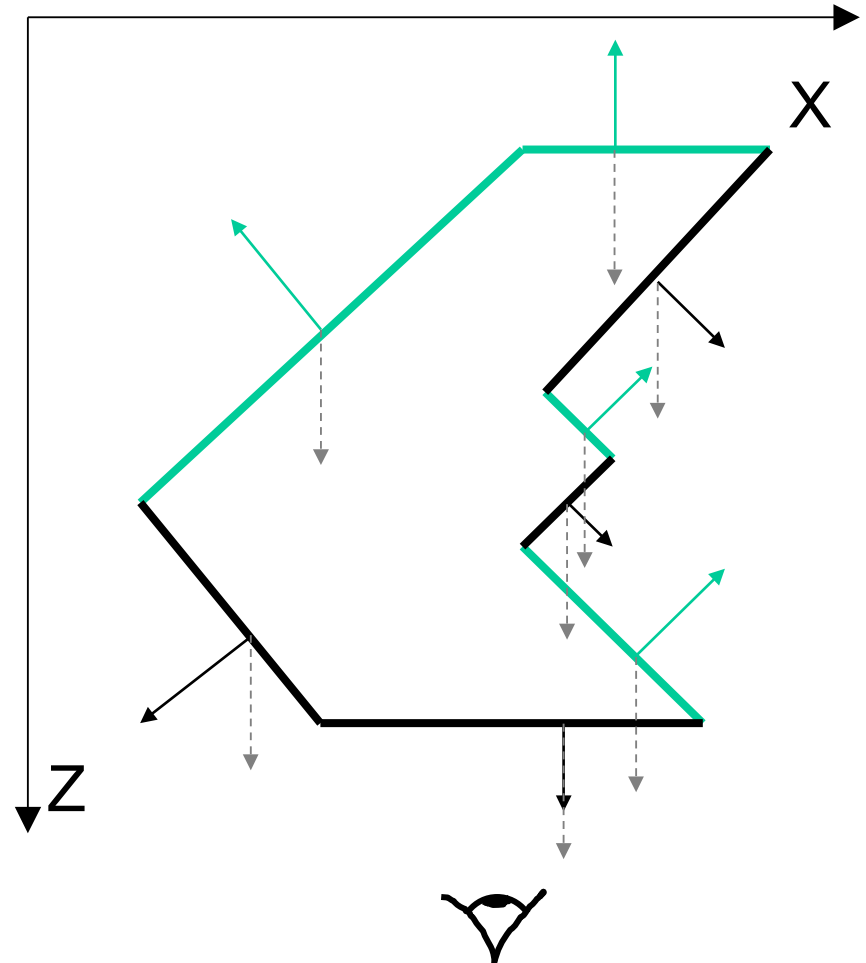
$O$  = El costo de calcular una intersección entre 2 objetos.

## 4.- Eliminación de caras posteriores

Perspectiva



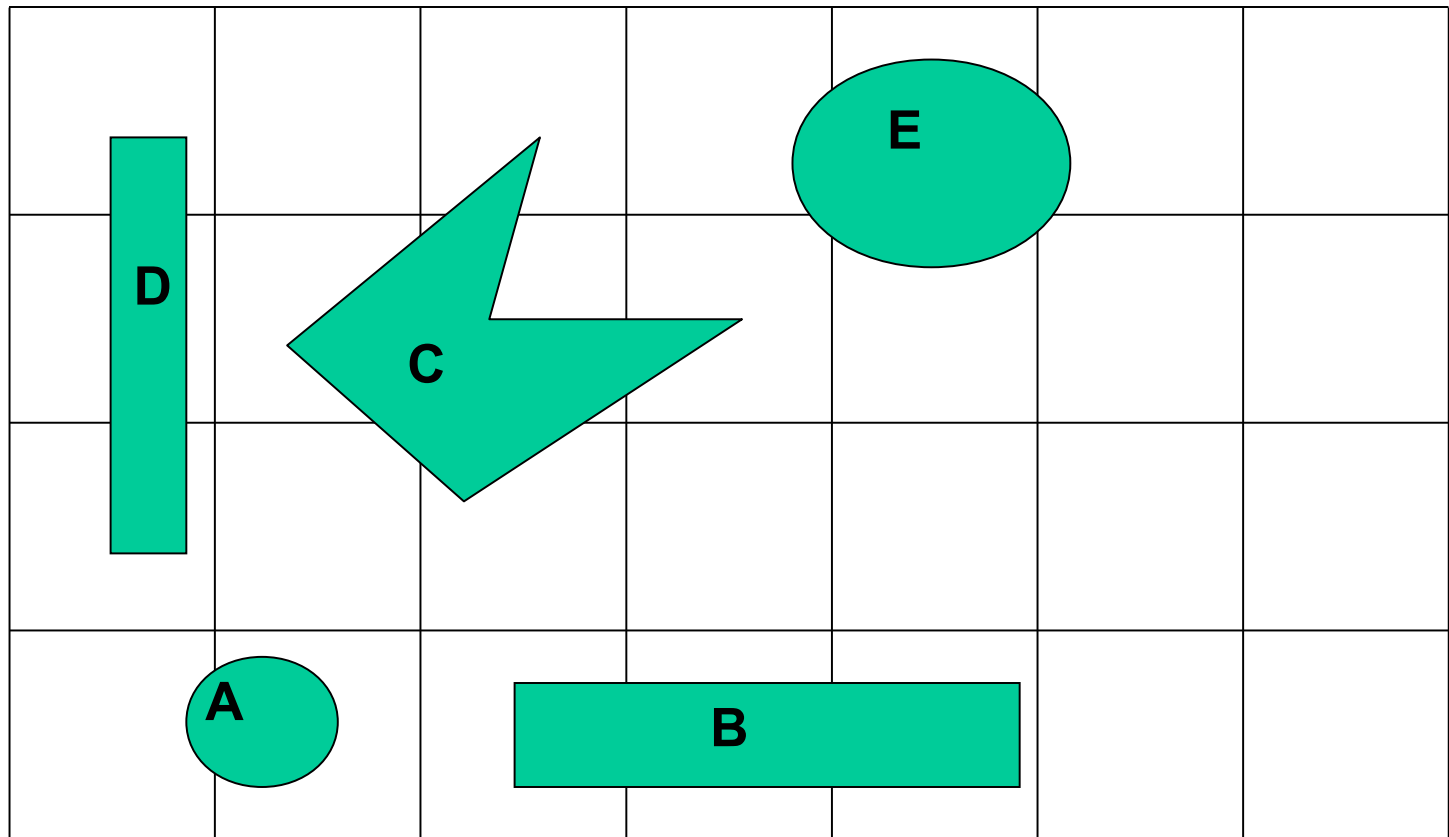
Paralela



## 5.- Partición Espacial

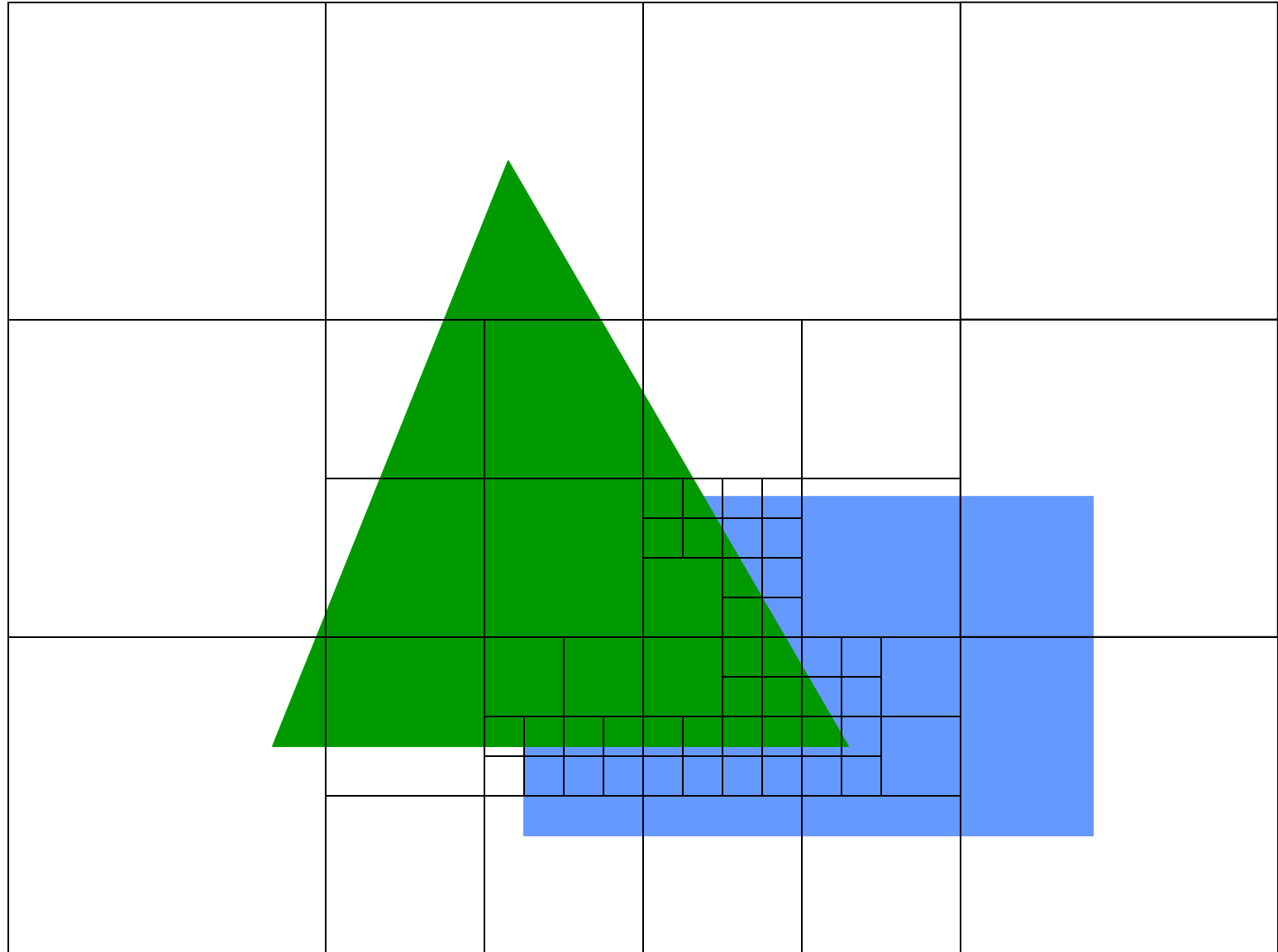
- Se divide un problema grande en otros más pequeños.
- Se asigna objetos a grupos espacialmente coherentes.
  - Por ej.: se divide el plano de proyección en una malla rectangular , y se determina en qué sección está cada objeto
  - La partición puede ser adaptable (si hay objetos distribuidos de forma desigual)
    - Por ej.: Estructura de árboles de octantes y cuadrantes.

## 5.- Partición Espacial

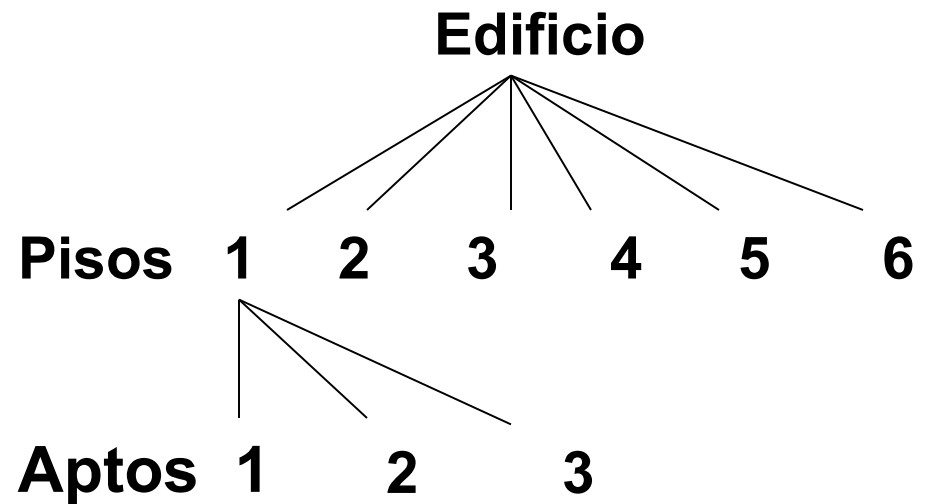
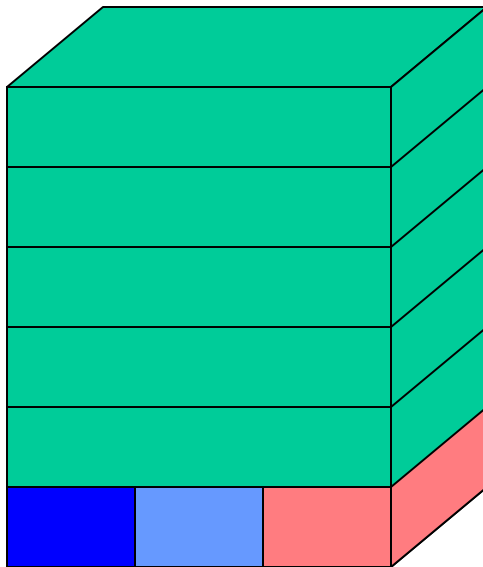




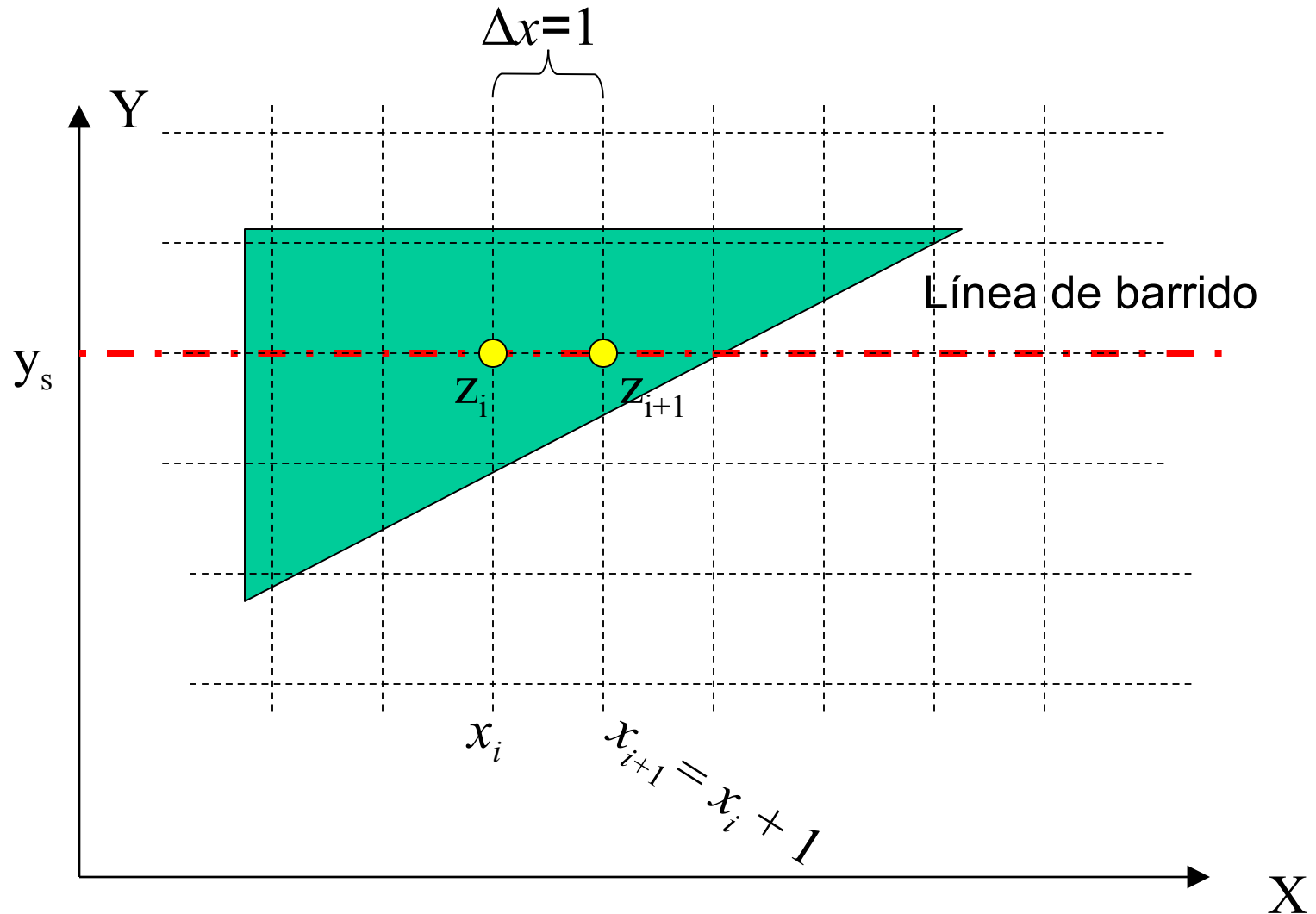
## 5.- Partición Espacial



## 6.- Jerarquía



# Algoritmo de z-buffer



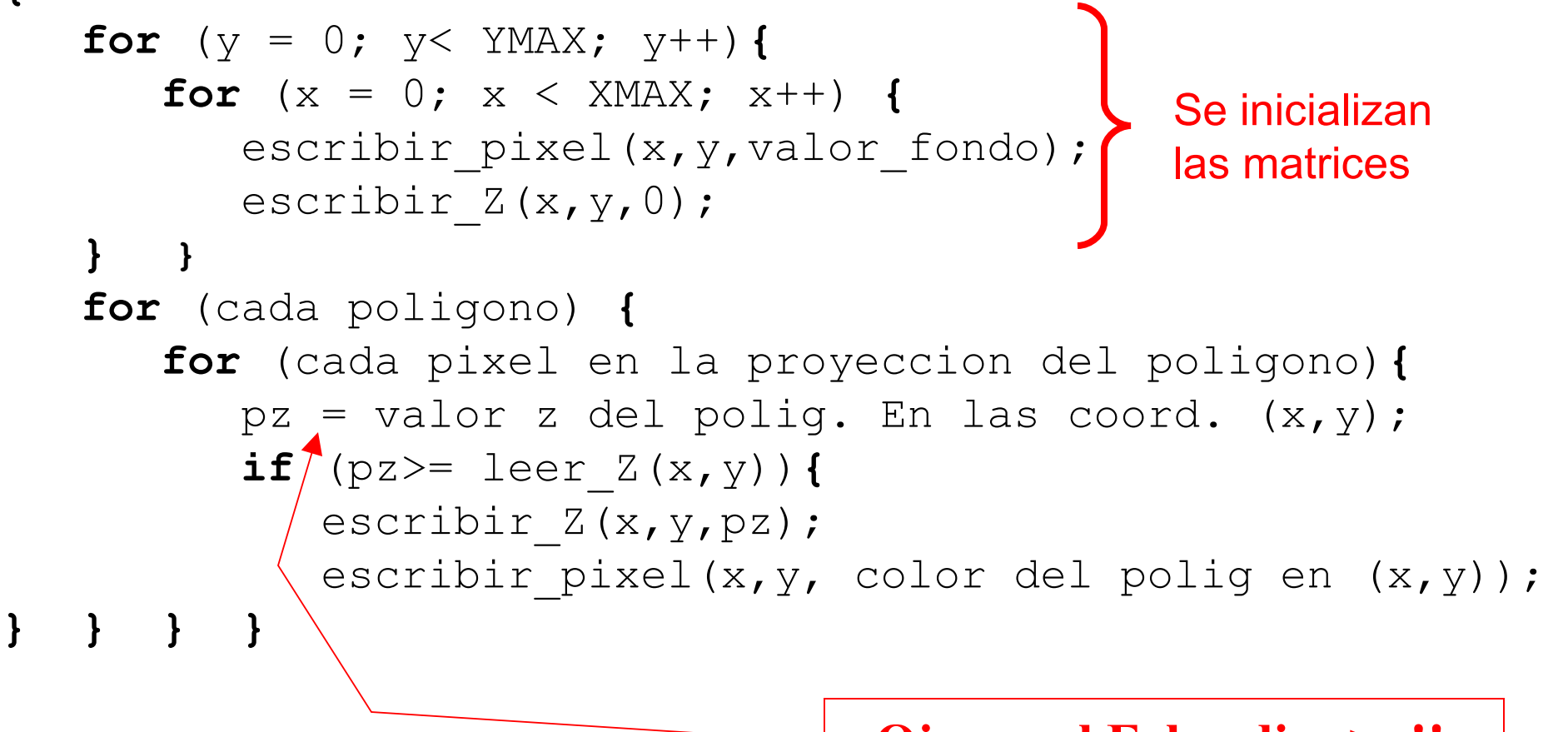
# Algoritmo de z-buffer

Cálculo de profundidad de un plano.

- $Ax + By + Cz + D = 0$
- $z = (-D - Ax - By)/C = (-D - By)/C - x A/C$
- $z_i = (-D - By_s)/C - x_i A/C$
- $z_{i+1} = (-D - By_s)/C - (x_i + 1) A/C = z_i - A/C$

# Algoritmo de z-buffer

```
void memoria_z
int pz;    /*la z del polígono en las coord de pixel(x,y)*/
{
    for (y = 0; y < YMAX; y++) {
        for (x = 0; x < XMAX; x++) {
            escribir_pixel(x,y,valor_fondo);
            escribir_Z(x,y,0);
        }
    }
    for (cada poligono) {
        for (cada pixel en la proyeccion del poligono) {
            pz = valor z del polig. En las coord. (x,y);
            if (pz >= leer_Z(x,y)) {
                escribir_Z(x,y,pz);
                escribir_pixel(x,y, color del polig en (x,y));
            }
        }
    }
}
```



The diagram illustrates the initialization and main loop of the z-buffer algorithm. A red curly brace on the right side groups the nested loops for y and x, with the text "Se inicializan las matrices" (The matrices are initialized) in red. A red arrow points from the "if" condition in the main loop to a red-bordered box at the bottom right containing the text "¡¡Ojo, en el Foley dice >=!!" (Beware, in Foley it says >=!!).

¡¡Ojo, en el Foley dice  $\geq$ !!

# Algoritmo de z-buffer

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

+

5	5	5	5	5	5	5	
5	5	5	5	5	5		
5	5	5	5	5			
5	5	5	5				
5	5	5					
5	5						
5							

=

5	5	5	5	5	5	5	0
5	5	5	5	5	5	0	0
5	5	5	5	5	0	0	0
5	5	5	5	0	0	0	0
5	5	5	0	0	0	0	0
5	5	0	0	0	0	0	0
5	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

# Algoritmo de z-buffer

5	5	5	5	5	5	5	0
5	5	5	5	5	5	0	0
5	5	5	5	5	0	0	0
5	5	5	5	0	0	0	0
5	5	5	0	0	0	0	0
5	5	0	0	0	0	0	0
5	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

+

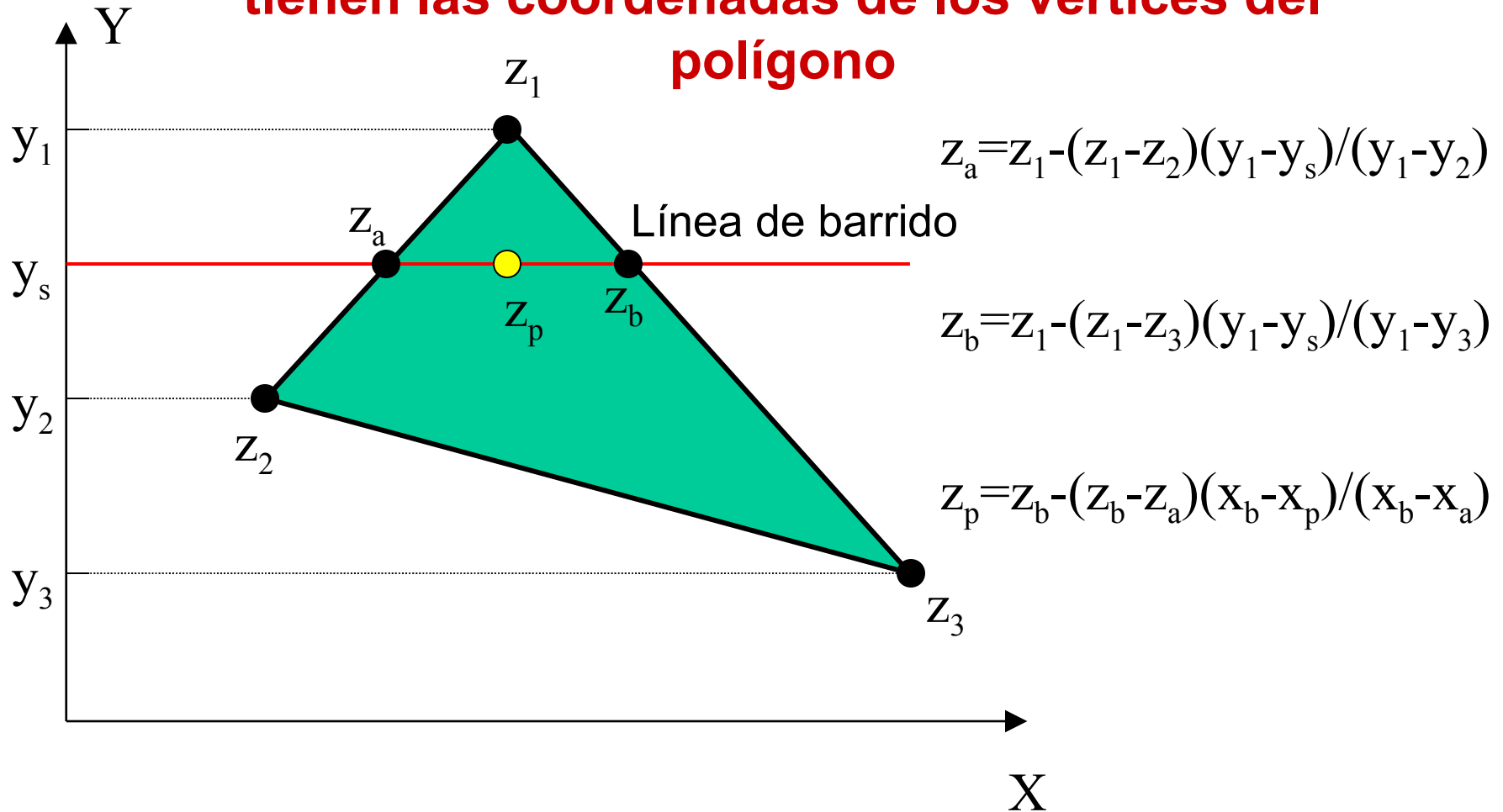
3							
4	3						
5	4	3					
6	5	4	3				
7	6	5	4	3			
8	7	6	5	4	3		

=

5	5	5	5	5	5	5	0
5	5	5	5	5	5	0	0
5	5	5	5	5	0	0	0
5	5	5	5	0	0	0	0
6	5	5	3	0	0	0	0
7	6	5	4	3	0	0	0
8	7	6	5	4	3	0	0
0	0	0	0	0	0	0	0

# Algoritmo de z-buffer

Si no se ha determinado el plano pero se tienen las coordenadas de los vértices del polígono





# Algoritmos de línea de barrido

## **Tabla de aristas** (aristas horizontales son ignoradas)

1. La coordenada ***X*** del extremo con menor coordenada ***Y***.
2. La coordenada ***Y*** del otro extremo de la arista.
3. El incremento ***X***,  $\Delta x$ , que se usa para pasar de una línea de rastreo a la siguiente.
4. El número de identificación del polígono.

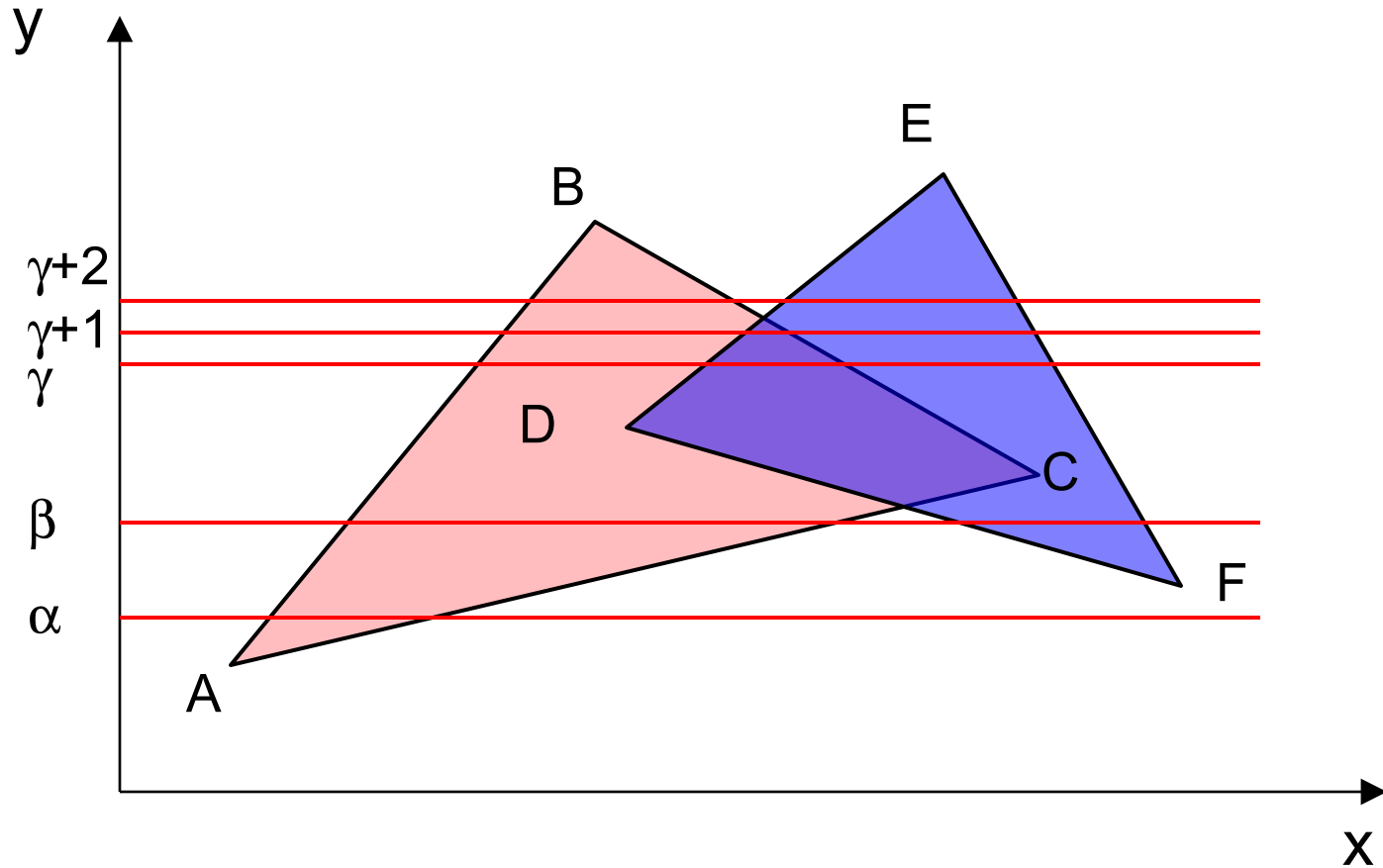
## **Tabla de polígonos**

1. Coeficientes de la ecuación del plano
2. Información del sombreado o color para el polígono
3. Bandera booleana de entrada-salida, con valor inicial ***falso***.

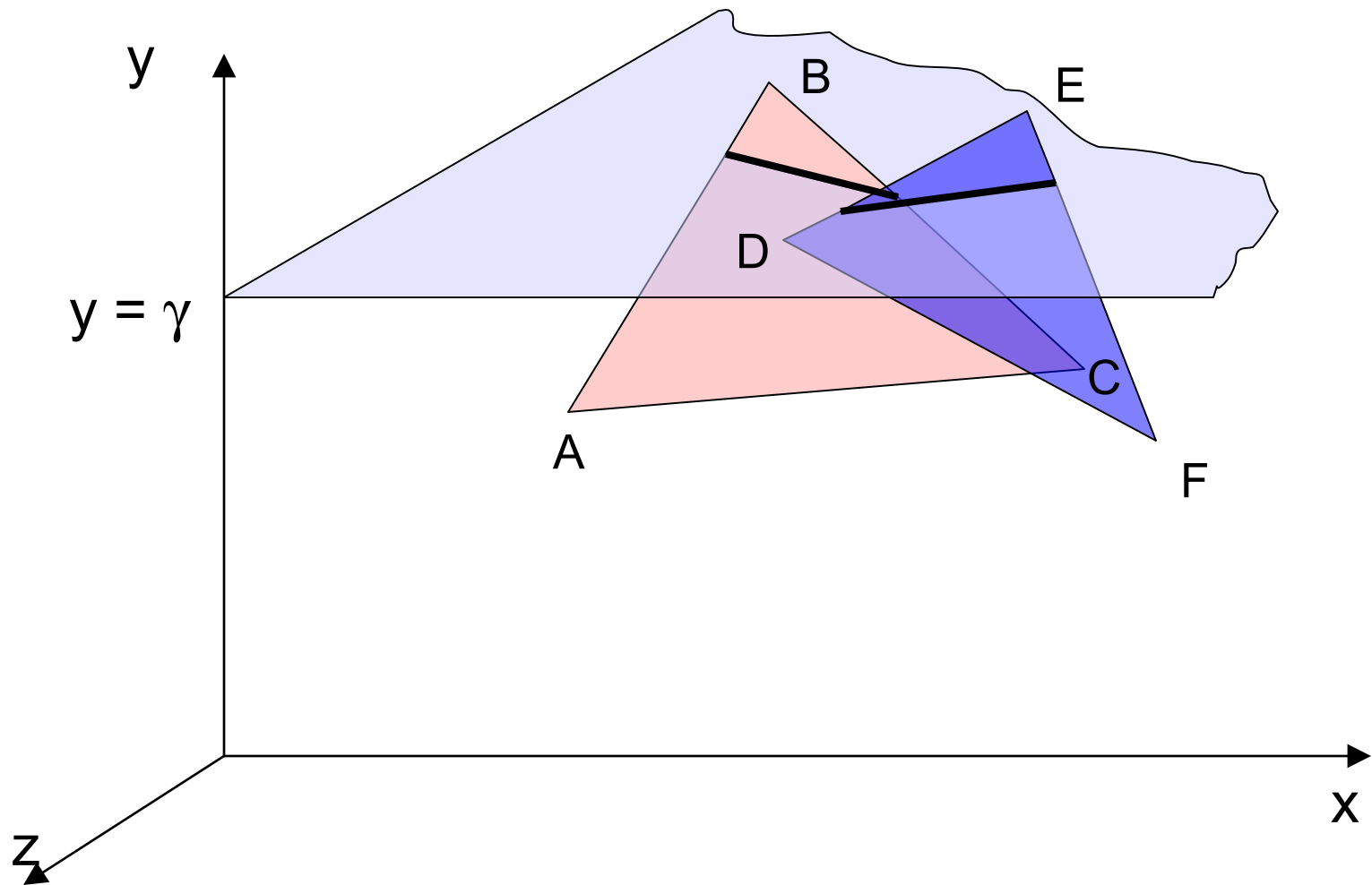
## **Tabla de aristas activas (AET)**

Contiene las aristas que intersectan la línea de rastreo actual.  
Las aristas están en orden creciente de  $x$ .

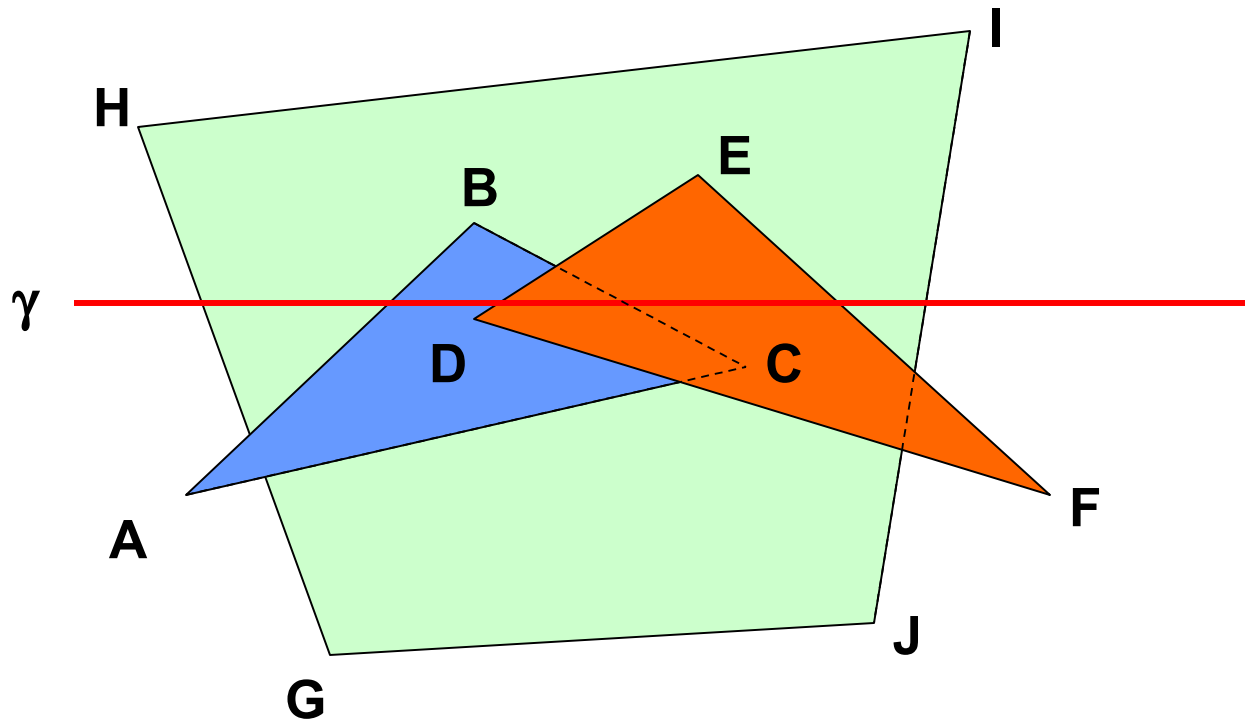
# Algoritmos de línea de barrido



# Algoritmos de línea de barrido

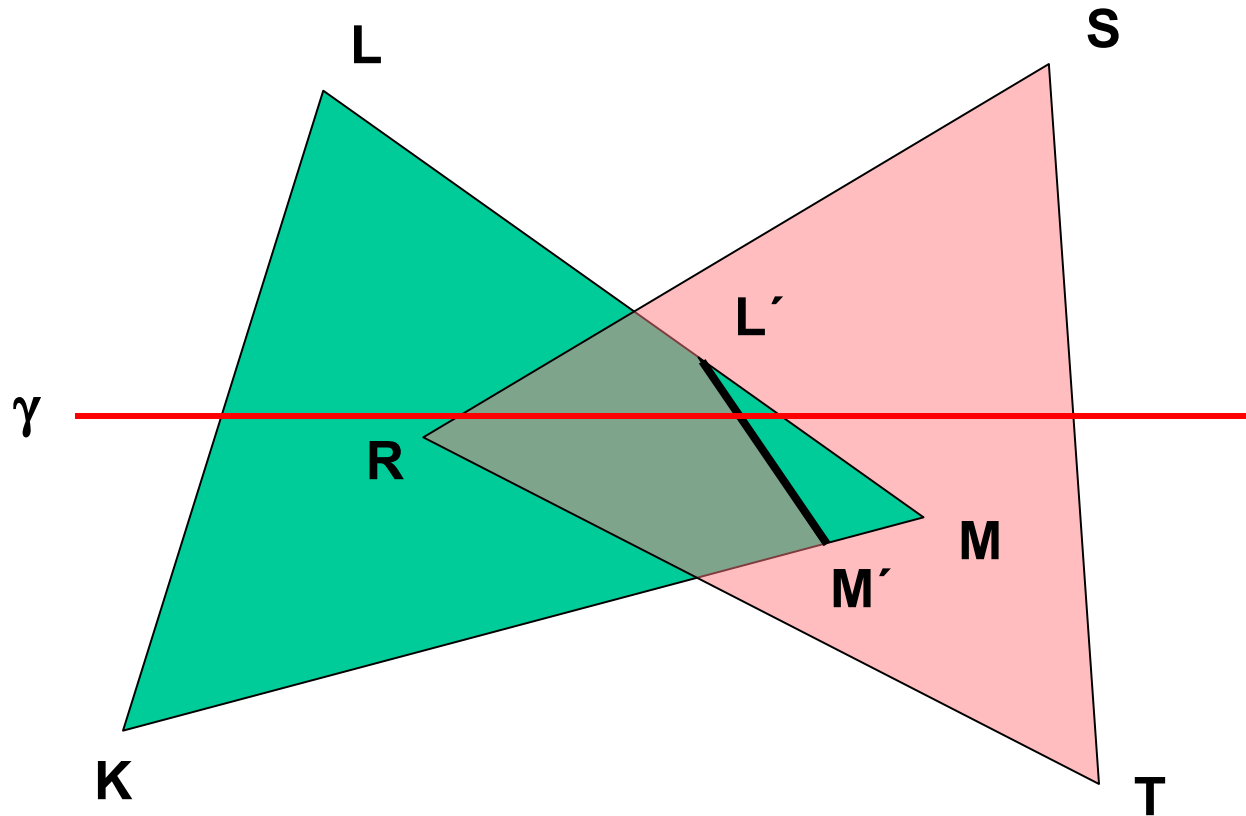


# Algoritmos de línea de barrido



Al no haber polígonos penetrantes, no es necesario realizar cálculo de profundidad al pasar  $\gamma$  por  $\overline{BC}$

# Algoritmos de línea de barrido



Para que el algoritmo funcione bien, el polígono KLM se debe dividir en 2.

# Algoritmos de línea de barrido

**Una variante utiliza la coherencia de profundidad**

**Si los polígonos no se penetran, si en la AET están las mismas aristas y en el mismo orden, entonces no han ocurrido cambios en las relaciones de profundidad y por tanto no es necesario realizar nuevos cálculos de profundidad.**

# Algoritmos de línea de barrido

(previamente se debe asignar un valor al fondo)

## **Seudocódigo de el algoritmo (ampliado para superficies poligonales o más generales)**

Añadir superficies a la tabla de superficies;

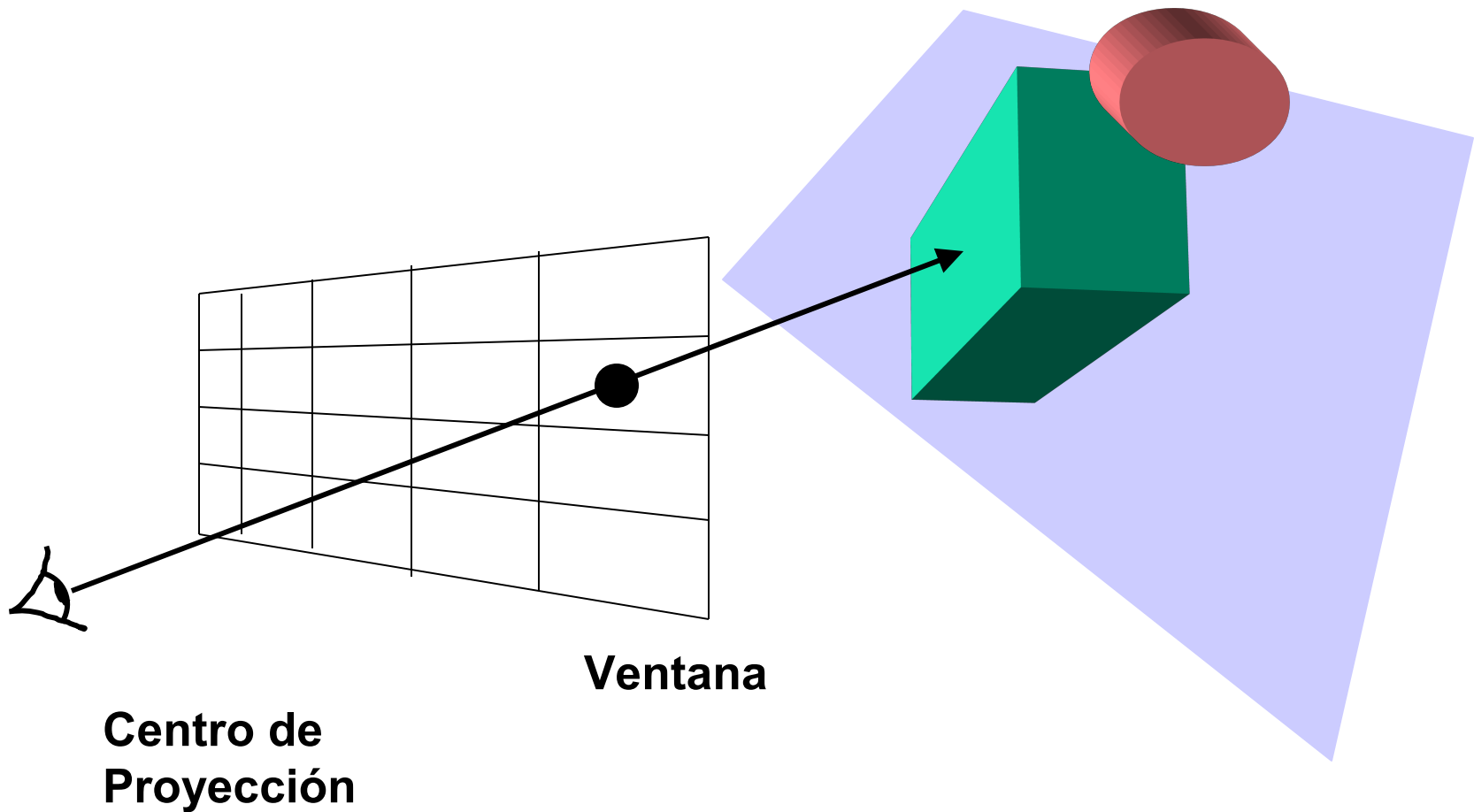
Asignar valores iniciales a la tabla de superficies activas;

**For** (*cada línea de barrido*) {  
    actualizar la tabla de superficies activas;

**For** (*cada pixel en la línea de barrido*) {  
        determinar las superficies que se proyectan al pixel;  
        encontrar la más cercana de estas superficies;  
        en el pixel, determinar el matiz de la sup. más  
        cercana;

    }  
}

# Traza de rayos en superficies visibles





# Traza de rayos en superficies visibles

## Seudocódigo para un algoritmo

Seleccionar el centro de proyección y una ventana en el plano de vista;

```
For ( cada línea de rastreo en la imagen ) {  
    For ( cada pixel en la línea de rastreo ) {  
        hallar el rayo centro de proyección por el pixel;  
        For ( cada objeto en la escena ) {  
            if (  $\exists \cap(\text{objeto}, \text{rayo})$  y es el más cercano hasta ahora)  
                registrar intersección y nombre del objeto;  
        }  
        asignar color del pixel corresp. al del objeto más cercano;  
    }  
}
```

## Cálculo de Intersecciones

Rayo que pasa por  $(x_0, y_0, z_0)$  y  $(x_1, y_1, z_1)$   
se determina por:

$$x = x_0 + t(x_1 - x_0), \quad y = y_0 + t(y_1 - y_0), \quad z = z_0 + t(z_1 - z_0)$$

Si  $\Delta x = x_1 - x_0 \Rightarrow x = x_0 + t \Delta x$

Idem con  $y, z \Rightarrow$

$x = x_0 + t \Delta x,$	$y = y_0 + t \Delta y,$	$z = z_0 + t \Delta z$
-------------------------	-------------------------	------------------------

## Cálculo de Intersecciones

Esfera con centro  $(a,b,c)$  y radio  $r$ :

$$(x-a)^2 + (y-b)^2 + (z-c)^2 = r^2 .$$

Su intersección con el rayo es:

$$(x_0 + t\Delta x - a)^2 + (y_0 + t\Delta y - b)^2 + (z_0 + t\Delta z - c)^2 = r^2$$

y luego de operar da la ecuación cuadrática:

$$(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 + 2[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)]t + (x_0 - a)^2 + (y_0 - b)^2 + (z_0 - c)^2 - r^2 = 0$$

## Cálculo de Intersecciones

Plano :

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

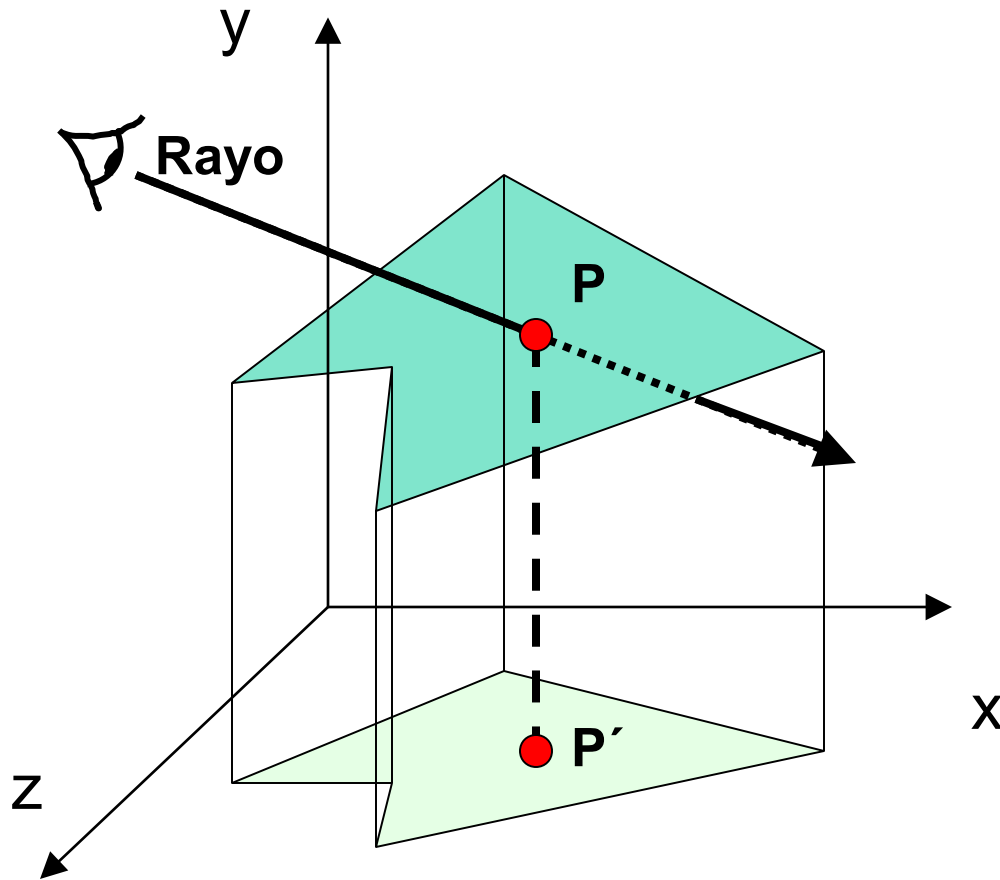
Su intersección con el rayo es:

$$A(x_0 + t\Delta x) + B(y_0 + t\Delta y) + C(z_0 + t\Delta z) + D = 0$$

=>

$$t = -(Ax_0 + By_0 + Cz_0 + D) / (A\Delta x + B\Delta y + C\Delta z + D)$$

# Cálculo de Intersecciones



Conviene proyectar sobre el plano que dé la proyección más larga. Este es el correspondiente a la variable cuyo coeficiente en la ecuación del plano del polígono tenga el mayor valor absoluto

## Consideraciones de eficiencia

La versión sencilla del algoritmo determina la intersección de cada rayo con cada polígono

=>  $1024\text{píxeles} \times 1024\text{píxeles} \times 100\text{objetos} =$   
100M cálculos de intersección

Entre 75% al 95% del tiempo está dedicado al cálculo de las intersecciones.

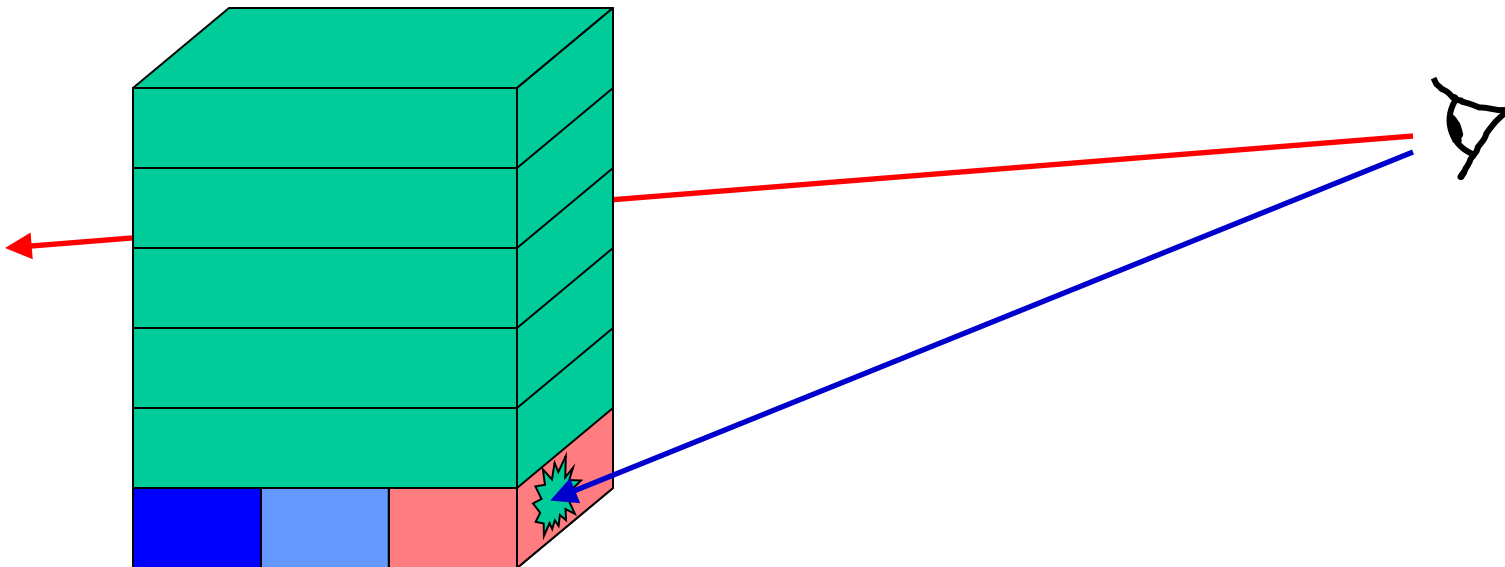
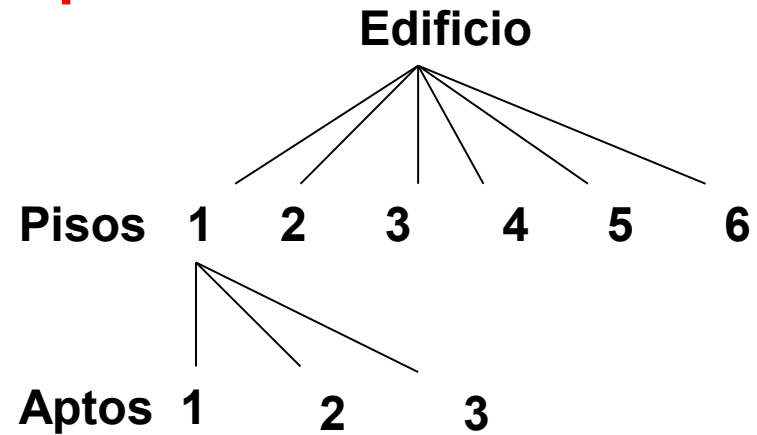
=> es importante dar estrategias que aceleren los cálculos

## Consideraciones de eficiencia:

# Optimiz. del cálculo de las intersecciones

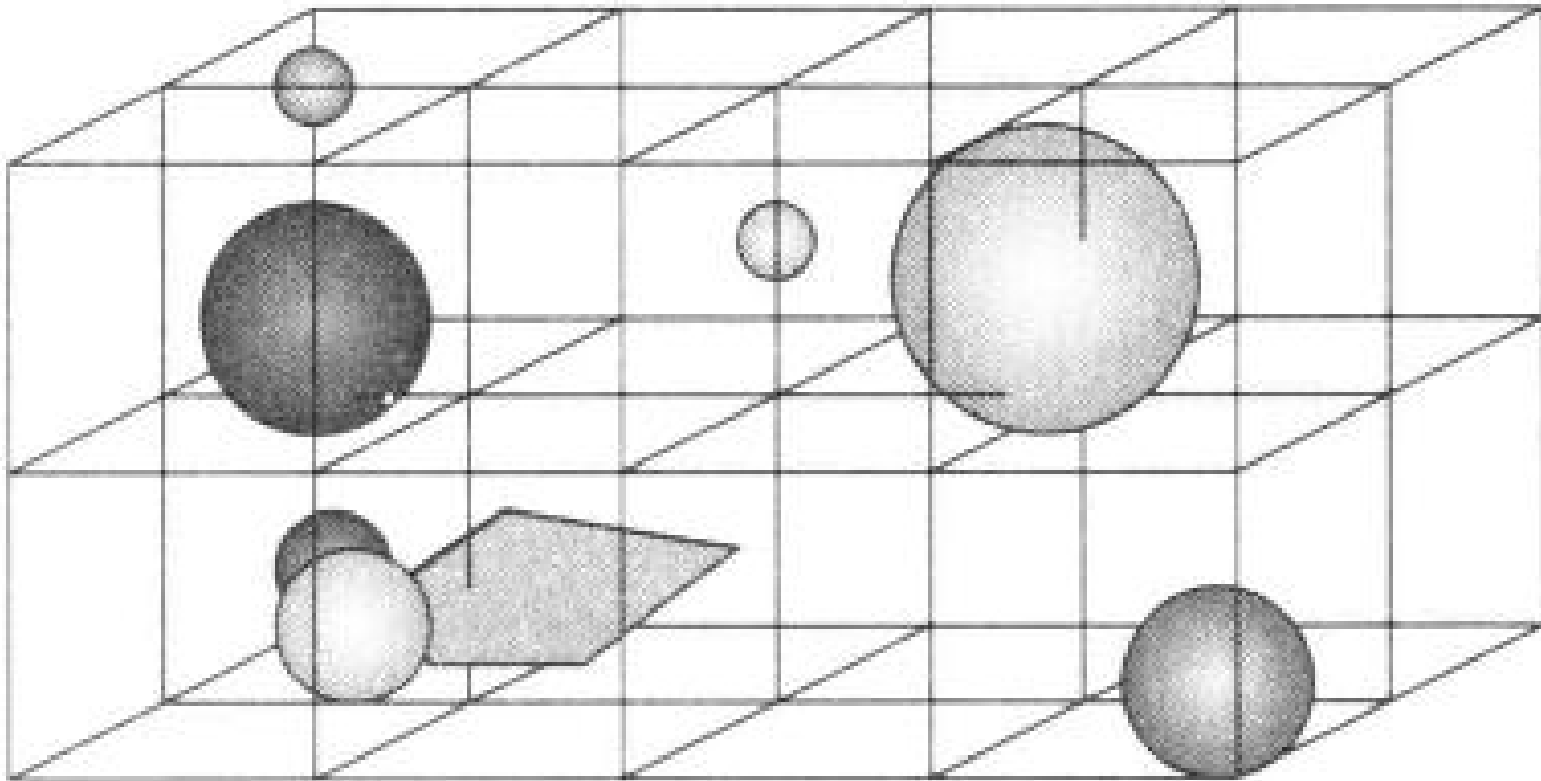
- Hallar constantes en las ecuaciones de intersección objeto-rayo.
- Hacer que el rayo coincida con el eje  $z$  y aplicar la misma transformación geométrica a los objetos (simplifica el cálculo de la  $\cap$  y permite determinar el objeto más cercano con un ordenamiento basado en  $z$ ).
- Volúmenes acotantes simplifican el cálculo de las intersecciones.

## Consideraciones de eficiencia: Jerarquía



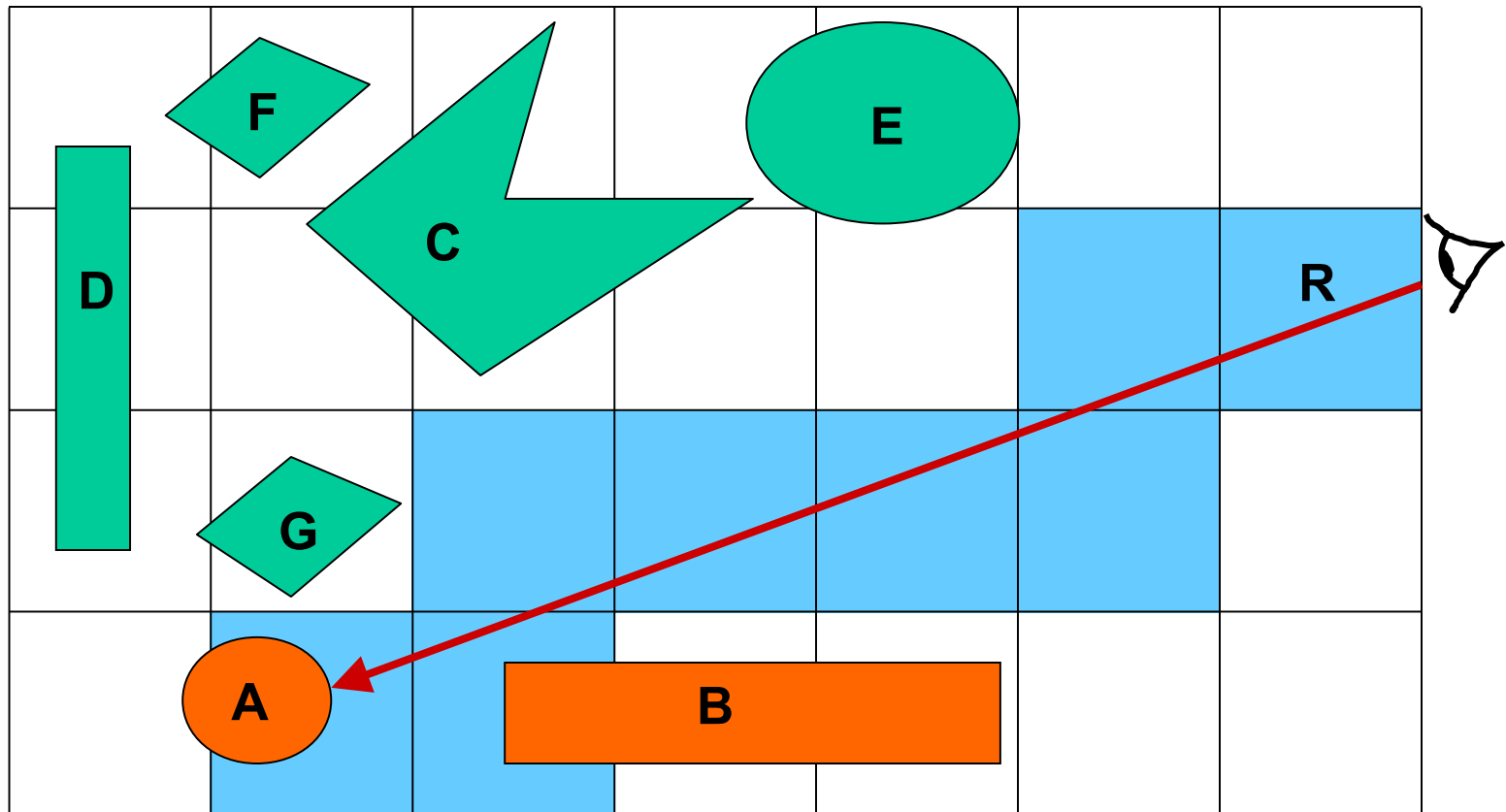


## Consideraciones de eficiencia: Partición Espacial

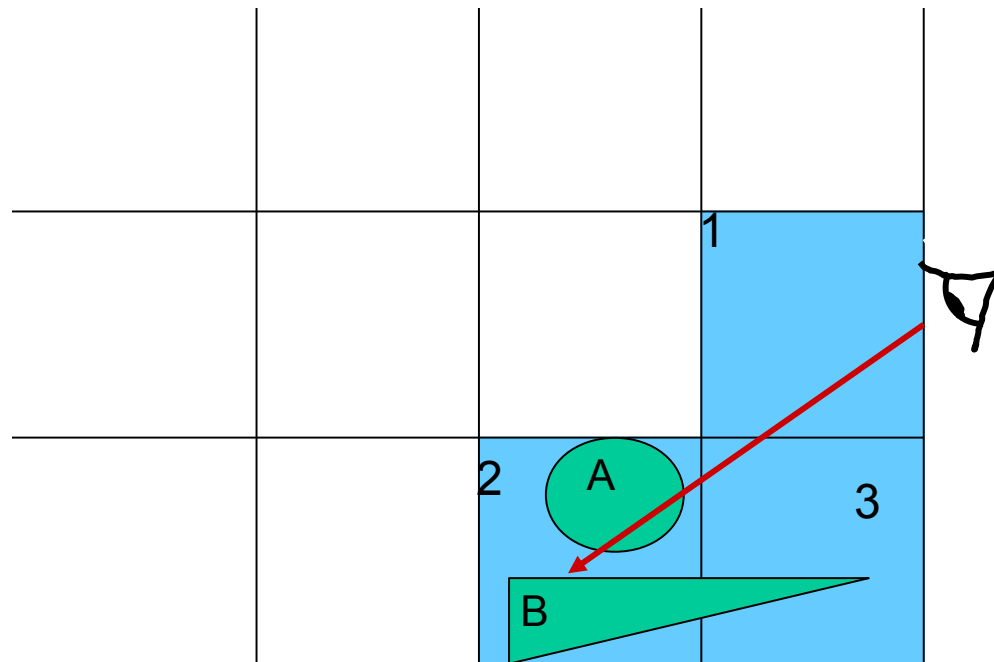


La escena se divide en una malla regular con volúmenes de igual tamaño.

# Consideraciones de eficiencia: Partición Espacial



## Consideraciones de eficiencia: Partición Espacial



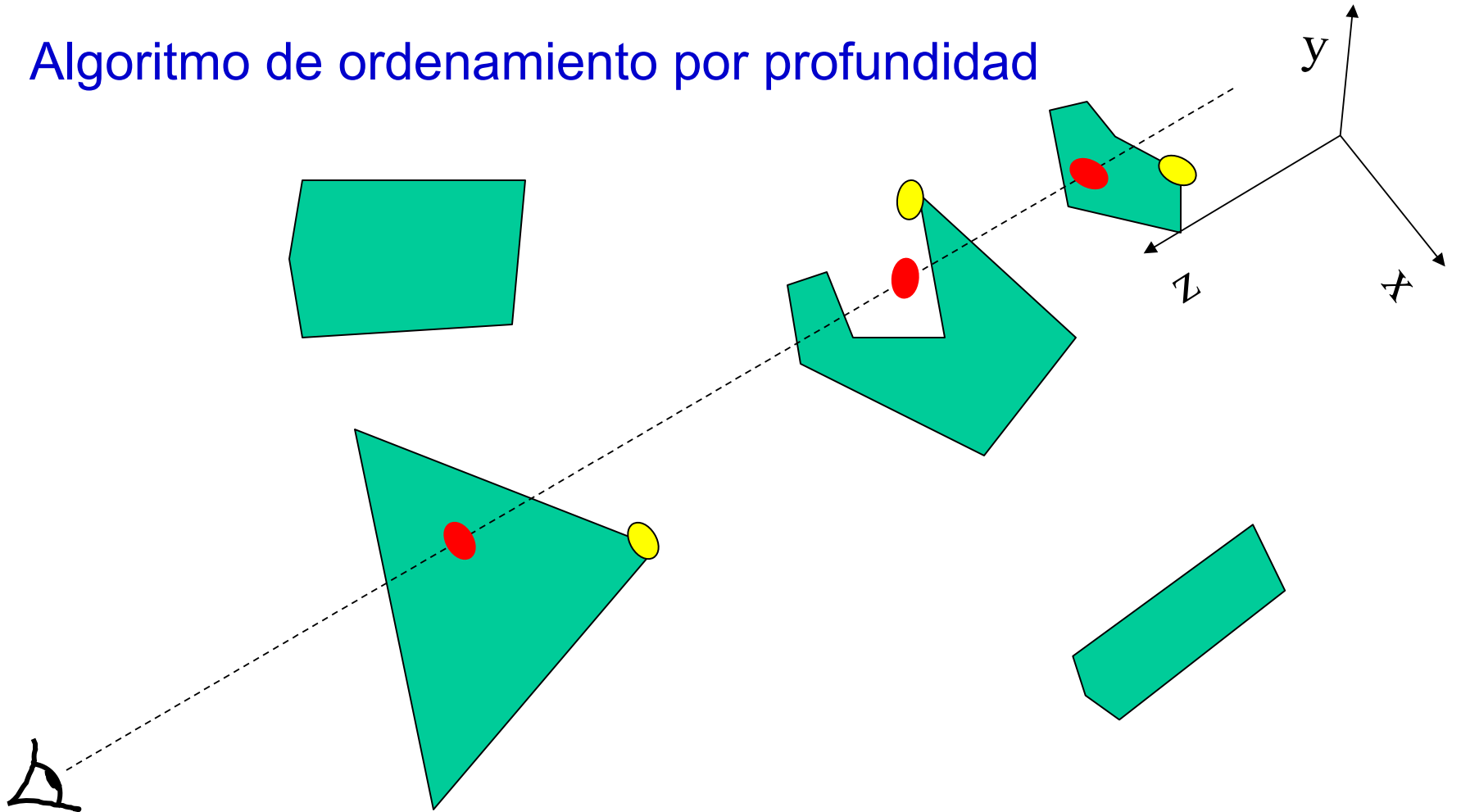
# Algoritmos de prioridad de listas

## Algoritmo de ordenamiento por profundidad (o algoritmo del pintor:

- Ordenar todos los polígonos de acuerdo con la menor coordenada  $z$  de cada uno.
- Resolver las ambigüedades cuando las extensiones  $z$  se superponen. Dividir polígonos de ser necesario
- Discretizar cada polígono en orden ascendente de la menor coordenada  $z$

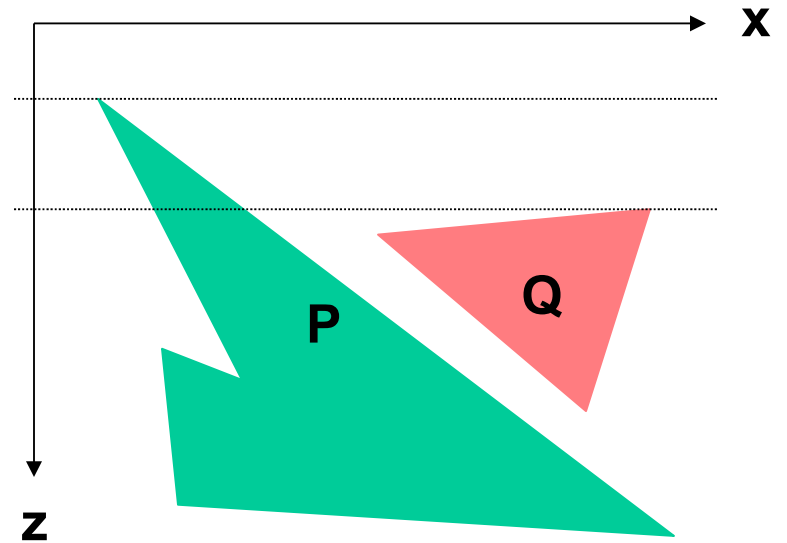
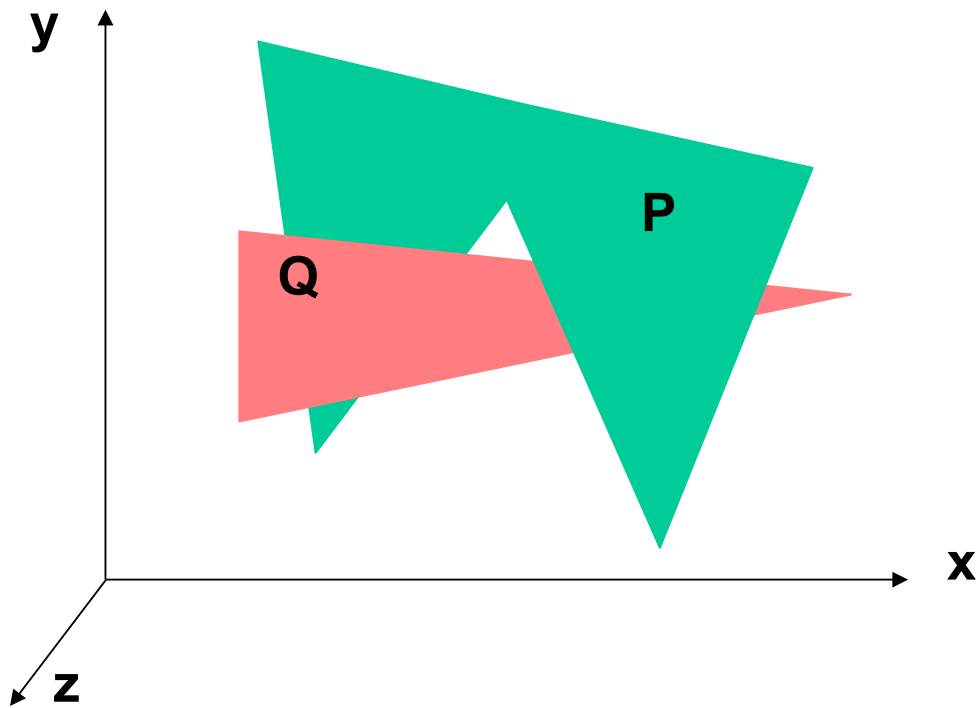
# Algoritmos de prioridad de listas

## Algoritmo de ordenamiento por profundidad



● Menor coordenada z

# Algoritmos de prioridad de listas



# Algoritmos de prioridad de listas

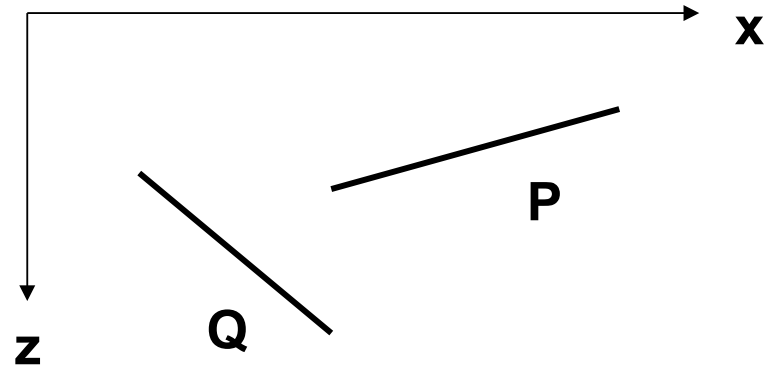
**P** es el polígono a pintar. Me fijo en los polígonos **Q** cuyas extensiones **z** se superpongan. Para cada **Q** hago 5 preguntas en orden creciente de complejidad.

Si alguna pregunta es verdadera  $\forall \text{ } Q \Rightarrow$

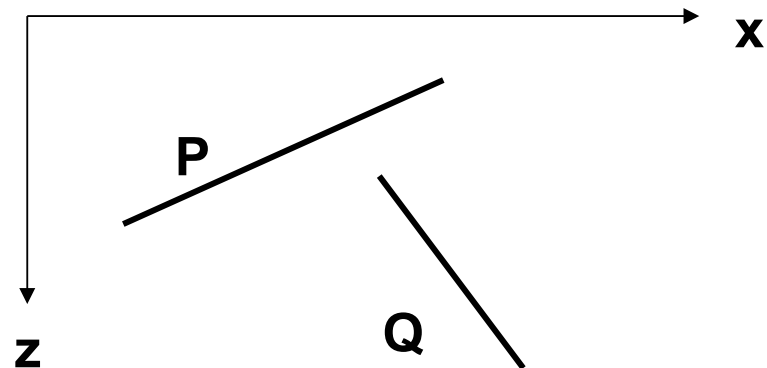
**P** no se superpone con  $\{Q\}$

- 1.- ¿No se superponen las extensiones **x** de los polígonos?
- 2.- ¿No se superponen las extensiones **y** de los polígonos?
- 3.- ¿Está todo **P** y el punto de observación en semiespacios distintos del plano de **Q**?
- 4.- ¿Está todo **Q** y el punto de observación en el mismo semiespacio del plano de **P**?
- 5.- ¿No se superponen las proyecciones de los polígonos en el plano (**x,y**)?

# Algoritmos de prioridad de listas



Tiene éxito la prueba 3



Tiene éxito la prueba 4



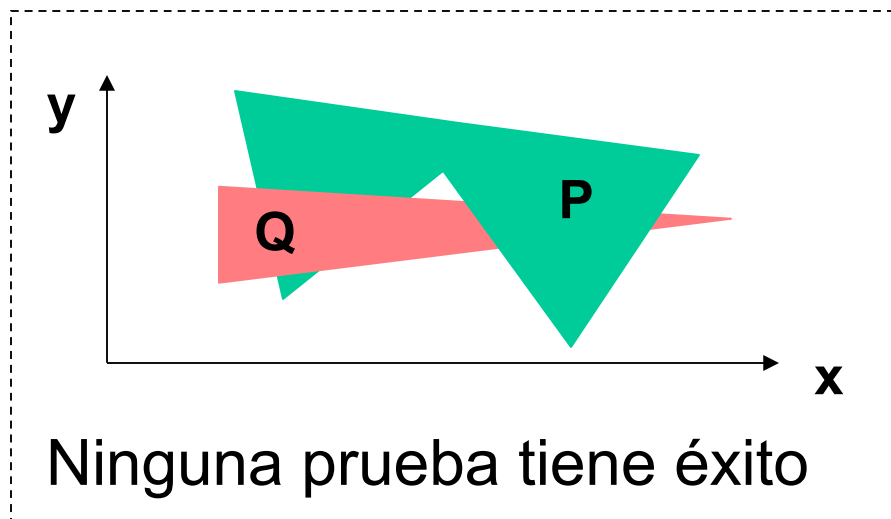
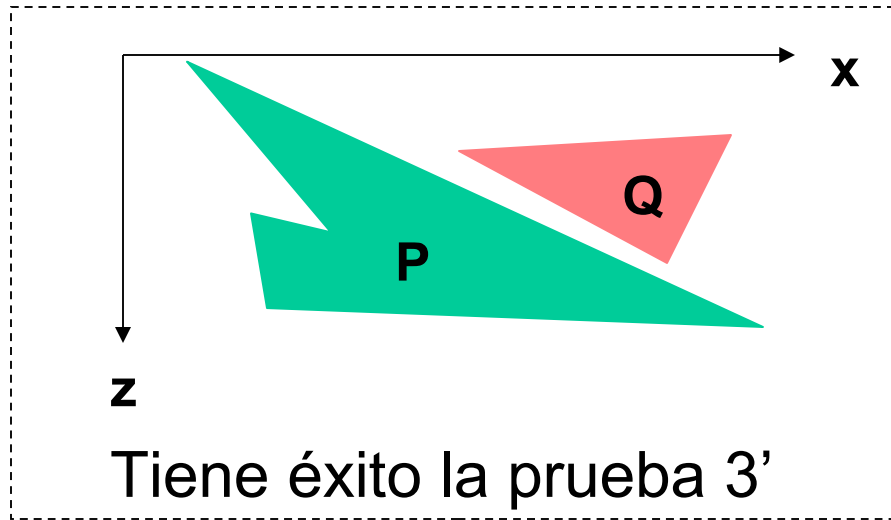
# Algoritmos de prioridad de listas

En el caso en que las 5 pruebas anteriores fracasen, se realizan dos pruebas extras, que en el caso de que alguna sea verdadera, se debe discretizar **Q** antes que **P**.

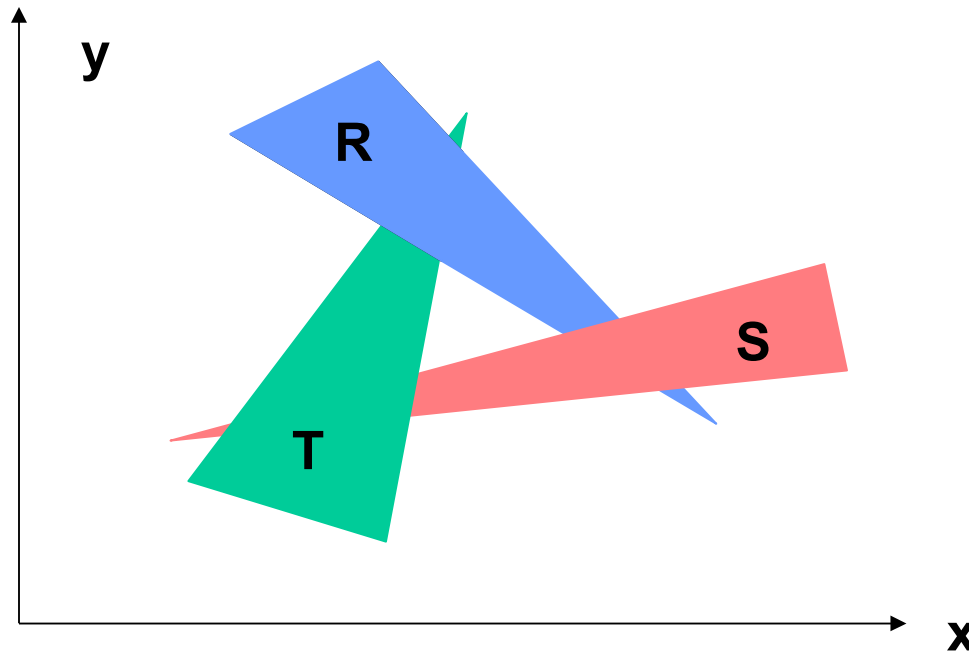
- 3' ¿Están **Q** (por completo) y el punto de observación en distintos semiespacios del plano de **P**?
- 4' ¿Están **P** (por completo) y el punto de observación en el mismo semiespacio del plano de **Q**?

Si todo esto fracasa, entonces hay que dividir **P** o **Q**, eliminar el polígono original y agregar los nuevos polígonos a la lista.

# Algoritmos de prioridad de listas



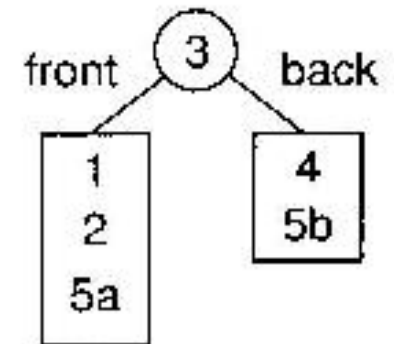
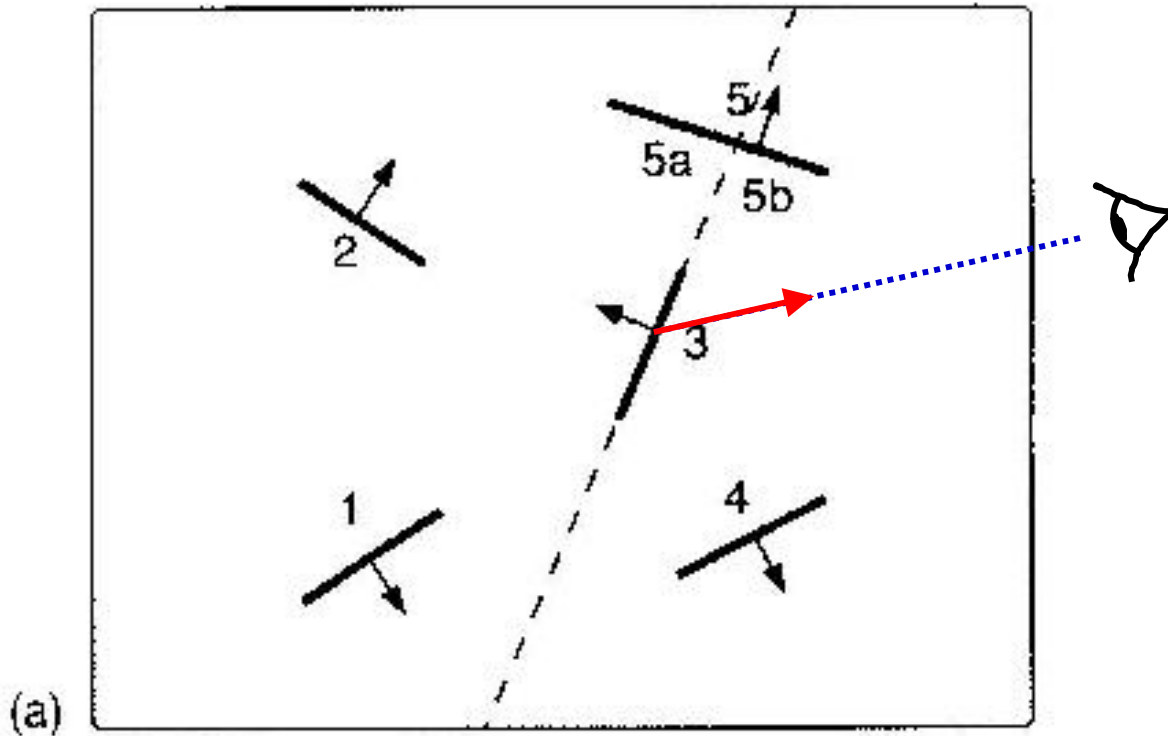
# Algoritmos de prioridad de listas



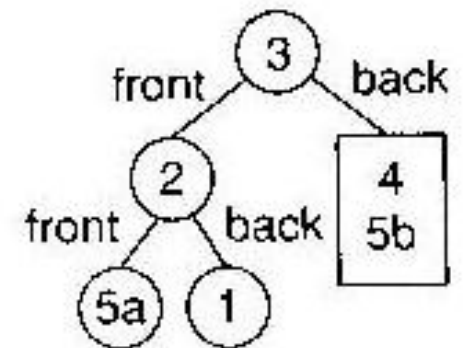
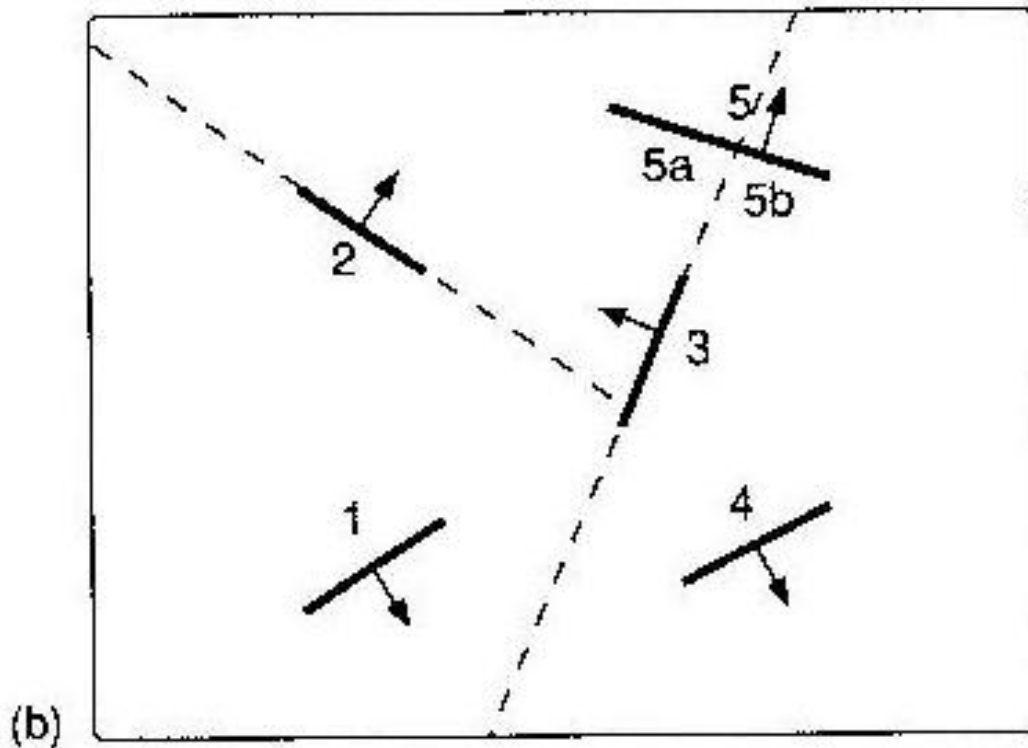
## Caso que genera ciclos.

Solución: marcar al polígono que se mueva al final de la lista. Si fracasan las 5 pruebas y el que sería **Q** ya está marcado, entonces no se hacen las 3' y 4', sino que se divide **P** o **Q**.

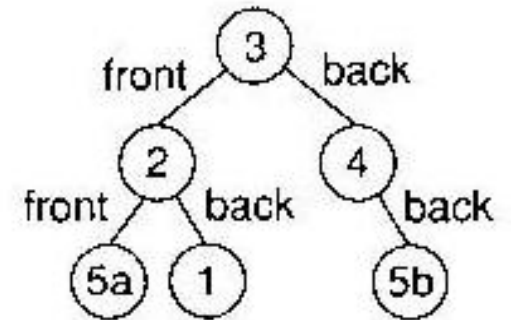
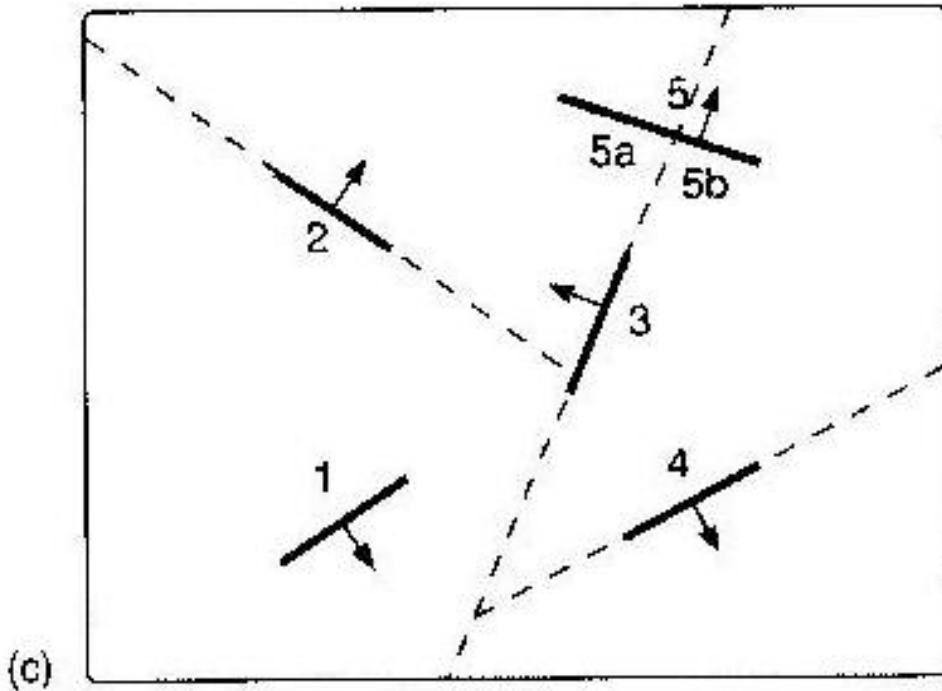
# Árboles binarios para partición del espacio (BSP tree)



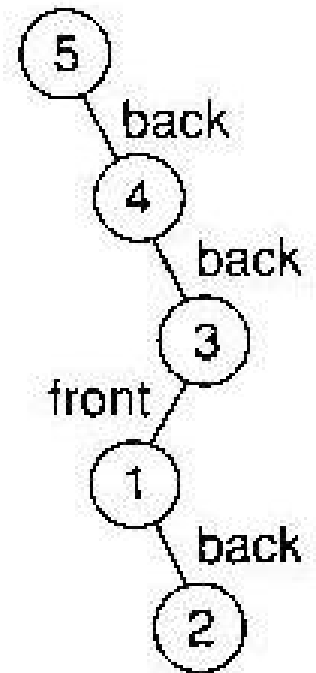
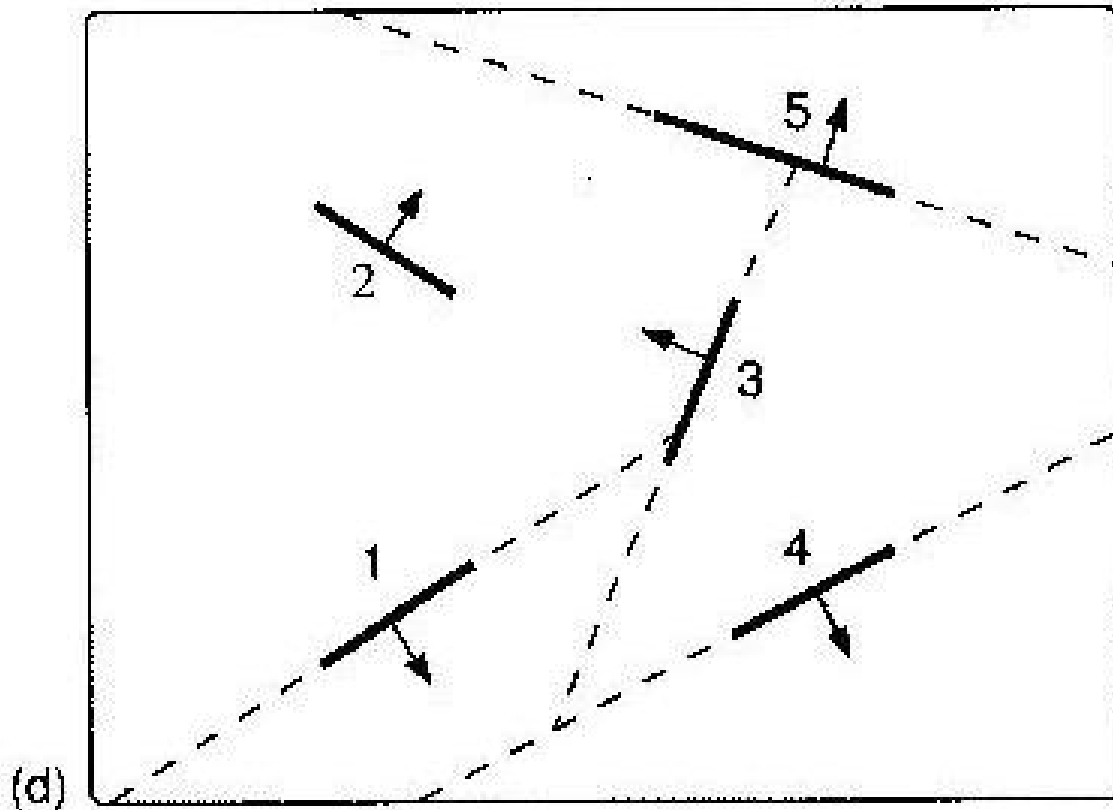
# Árboles binarios para partición del espacio (BSP tree)



# Árboles binarios para partición del espacio (BSP tree)

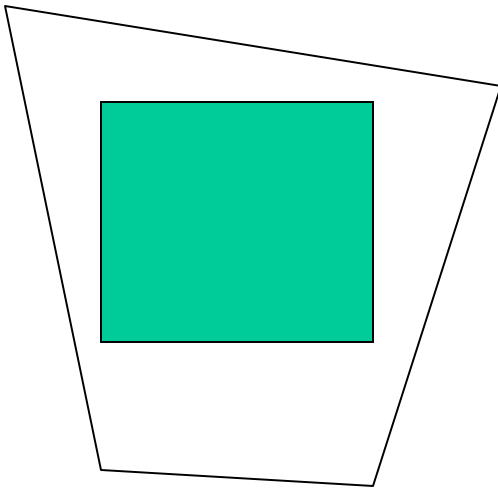


# Árboles binarios para partición del espacio (BSP tree)

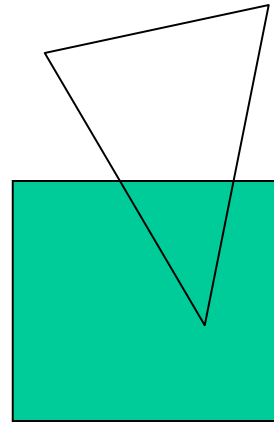


# Algoritmos de subdivisión de área

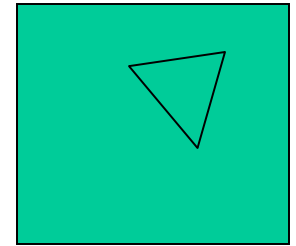
## Algoritmo de Warnock



Pol. Circundante

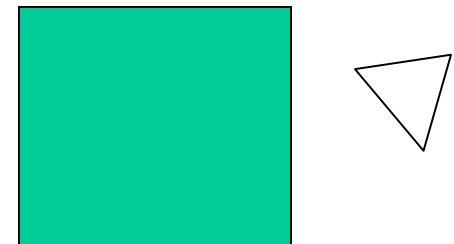


Pol. Intersecante



Pol. Contenido

Relación entre ***Polígono*** y  
***Elemento de Área***



Pol. Disjunto

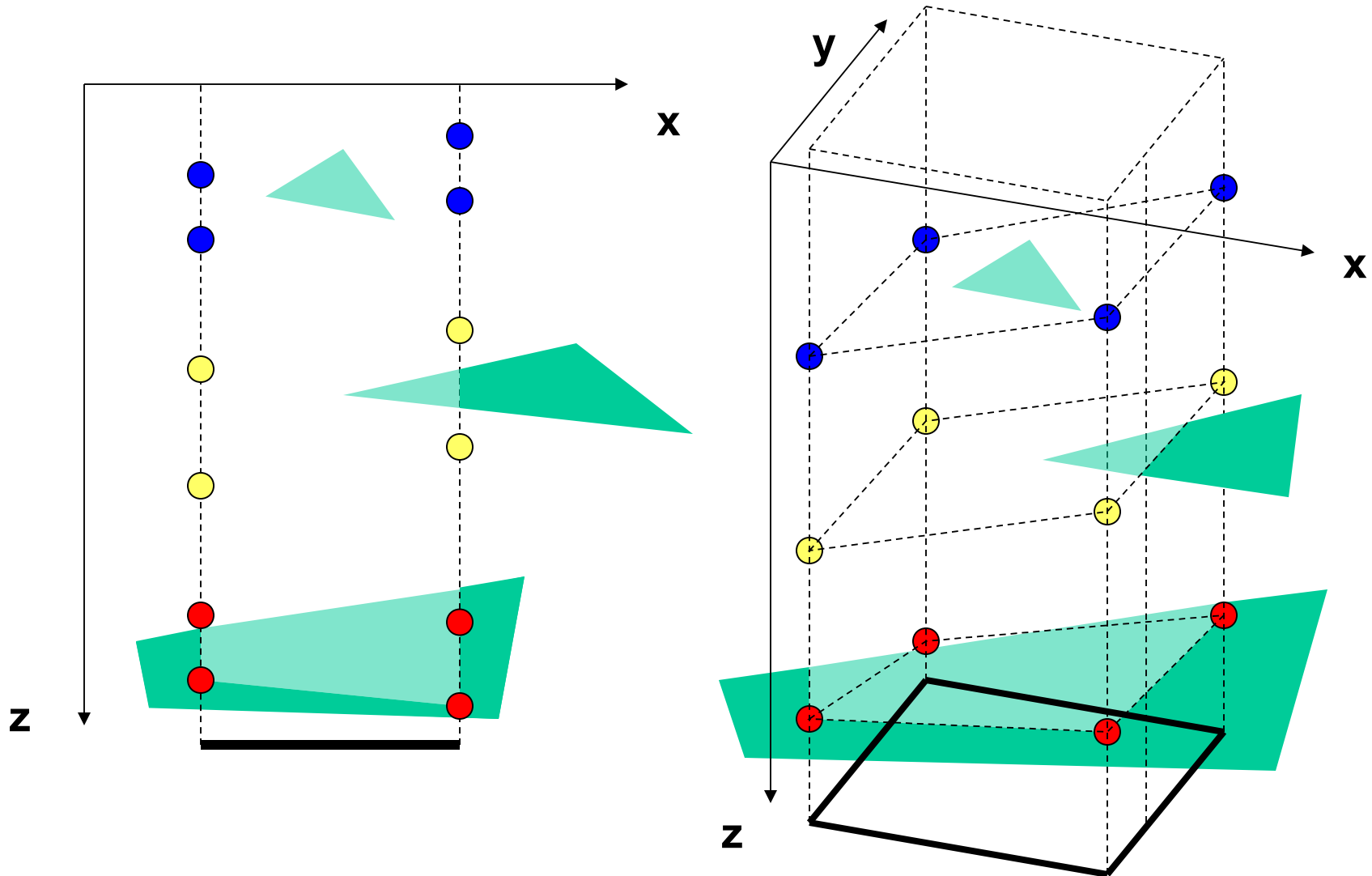


# Algoritmo de Warnock

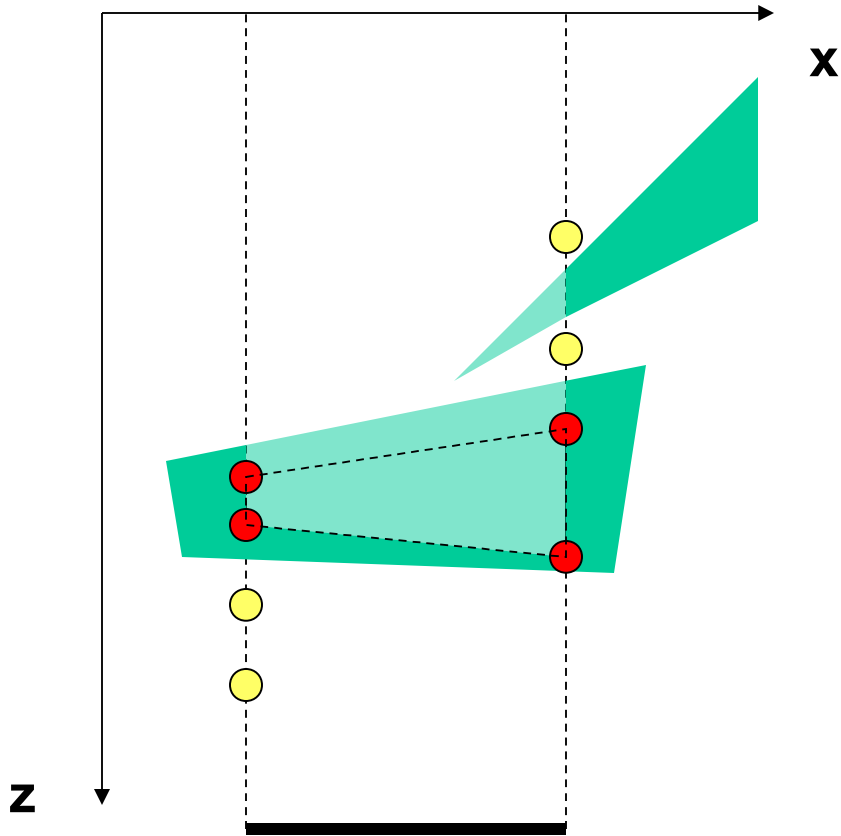
Para decidir sobre pintar o subdividir un área hay 4 casos:

- 1.- Todos los polígonos son disjuntos respecto al área. Con el color de fondo se puede pintar el área.
- 2.- Sólo hay un polígono intersecante o contenido.
- 3.- Hay un solo polígono que interseca al área y es circundante.
- 4.- Hay varios polígonos intersecantes, uno de ellos es circundante y está enfrente de los demás polígonos. Esto se controla con los valores  $z$  de los polígonos en los 4 vértices del área.

# Algoritmo de Warnock



# Algoritmo de Warnock



En este caso se subdivide el área.

La reacción es diferente al algoritmo de ordenamiento por profundidad, ya que en este caso no sería necesario subdividir.

# Algoritmo de Warnock

