

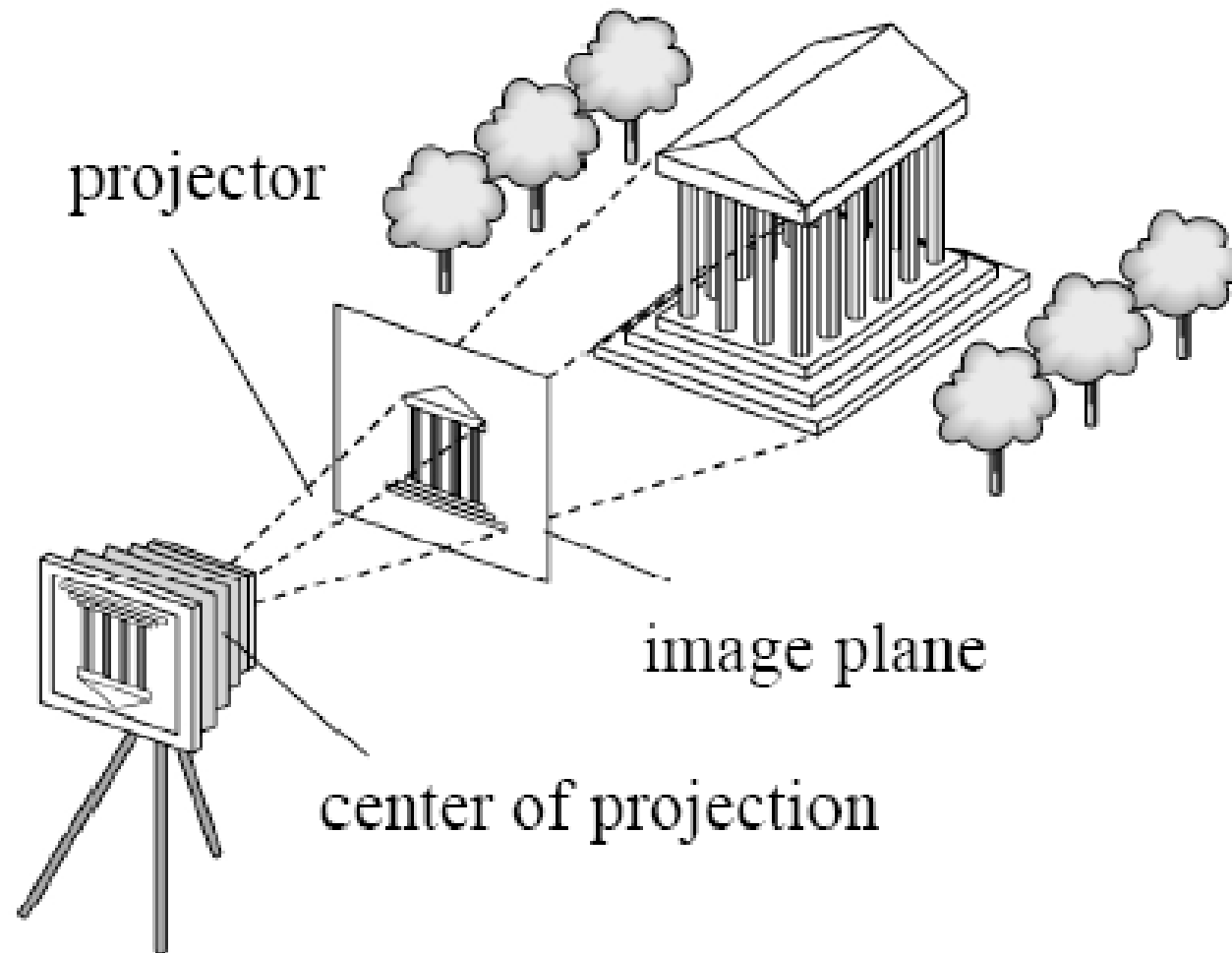
Superficies Visibles

Prof. Fernández et al. (Universidad de la República de Uruguay) -
<http://www.fing.edu.uy/inco/cursos/compgraf/>

Prof. Möller et al. Universidad Simon Fraser
<http://www.cs.sfu.ca/~torsten/Teaching/Cmpt361>

Capitulo 7 – Angel Edward
Capitulo 13, 15 – Foley et al.

Motivación

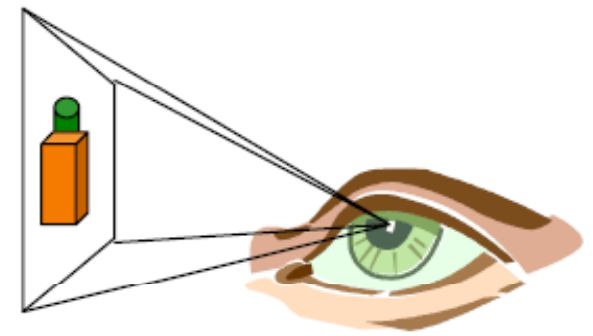
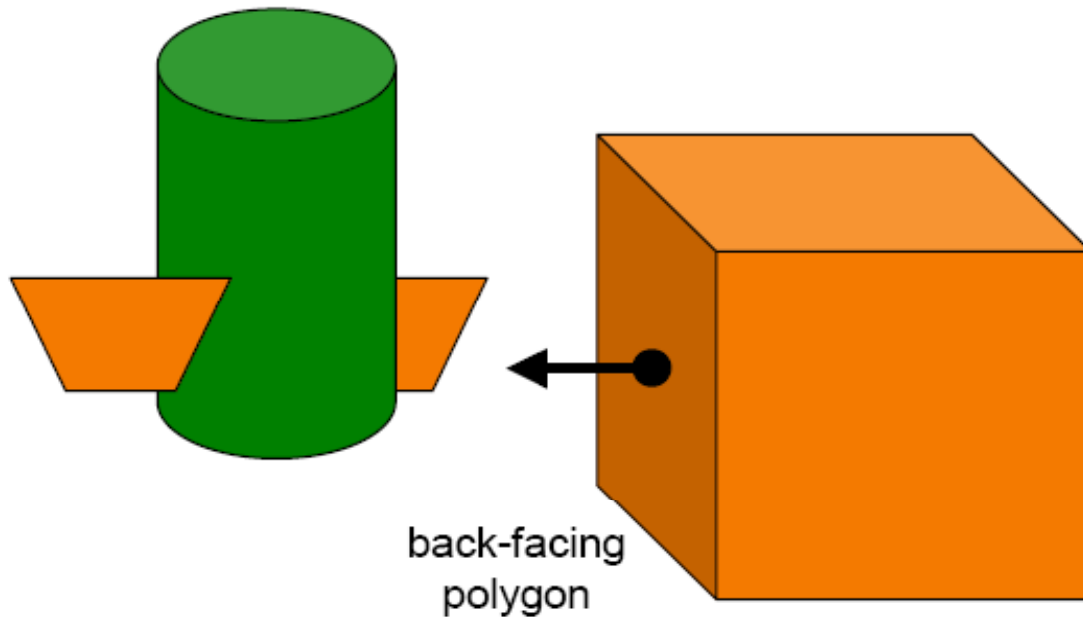


Imaging model adopted by 3D computer graphics

Motivación

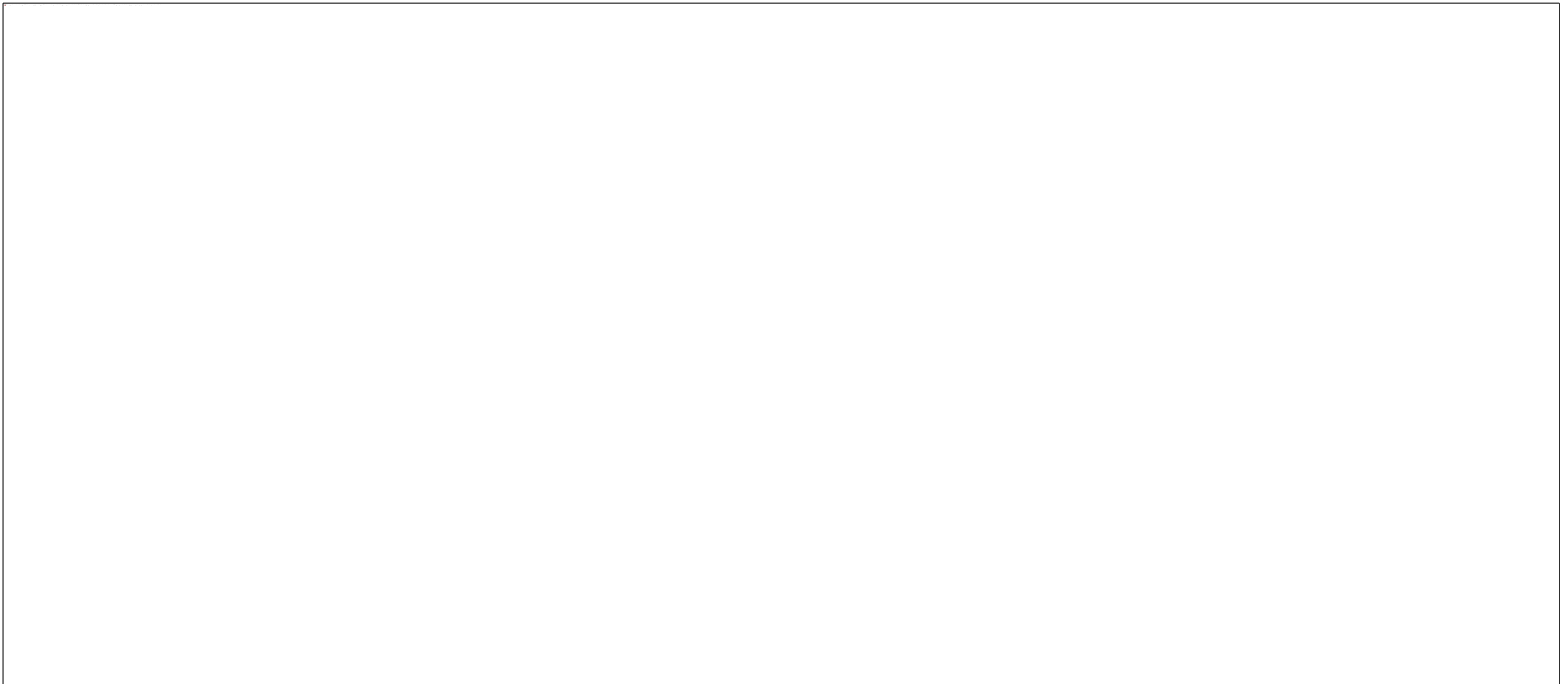


- Superficie en sentido opuesto
- Superficie ocluida
- Superficies superpuestas
- Superficies intersectadas

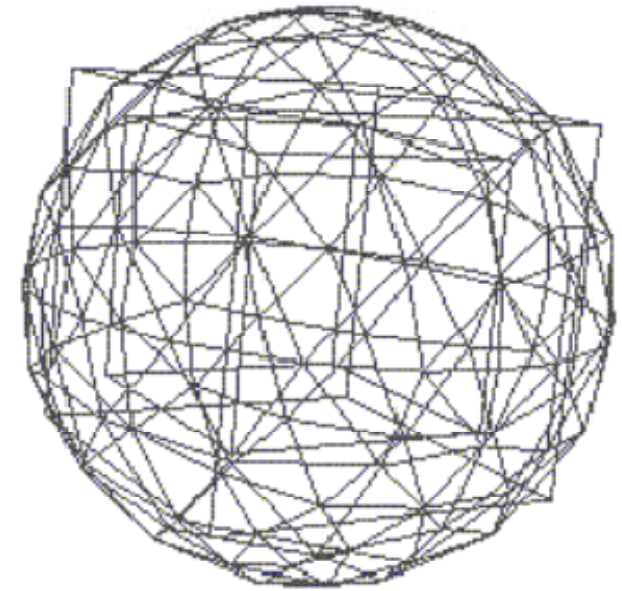
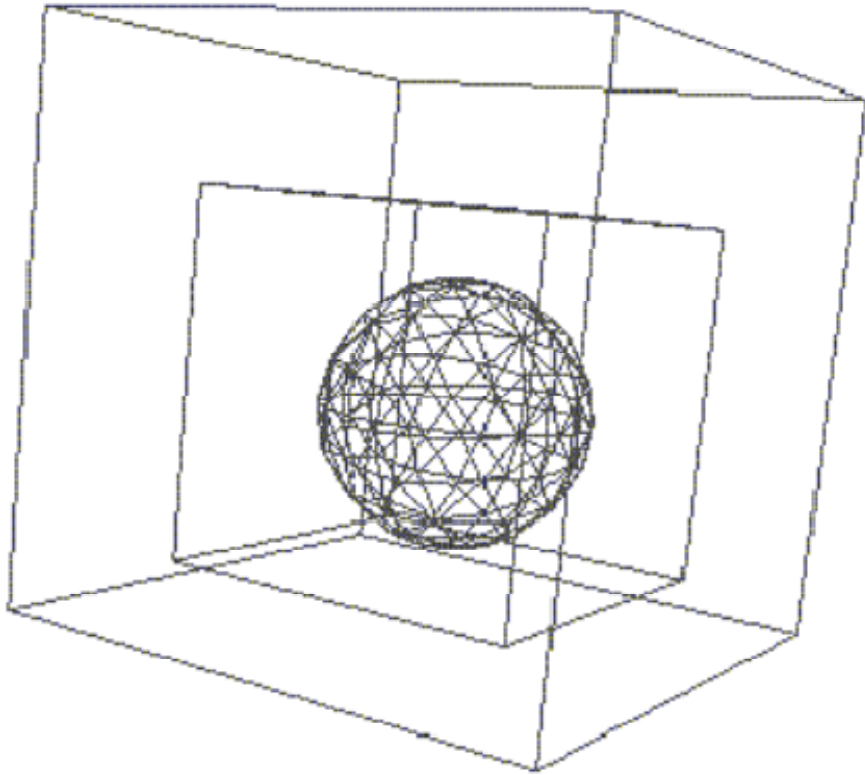




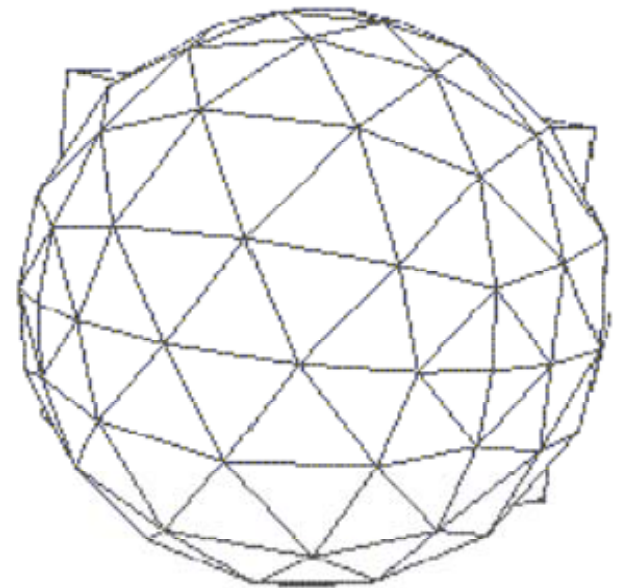
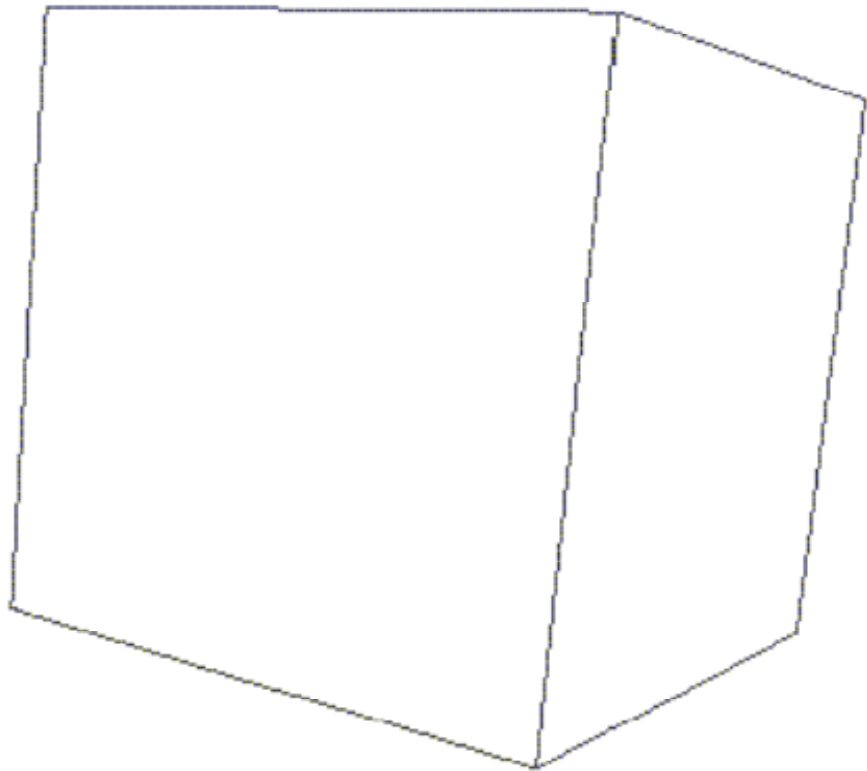
- ❑ Estudio de algoritmos para facilitar la visualización de las superficies visibles.



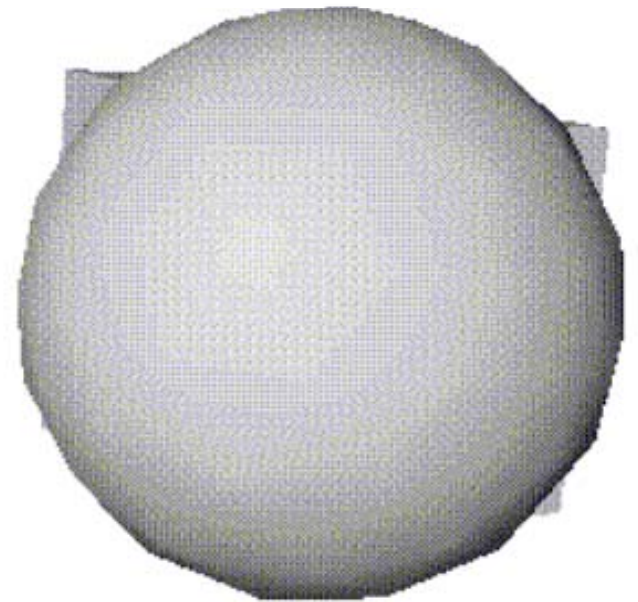
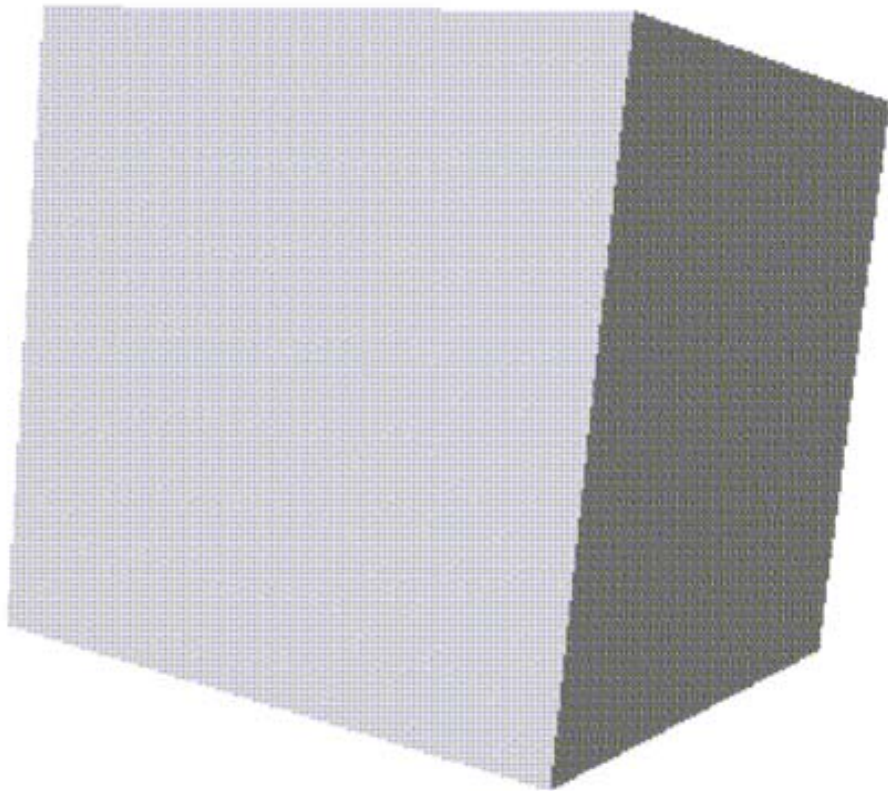
Motivación



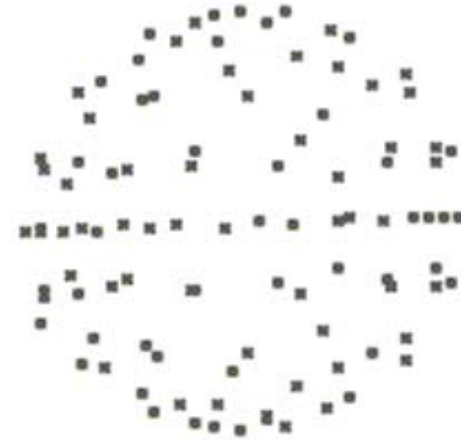
Motivación



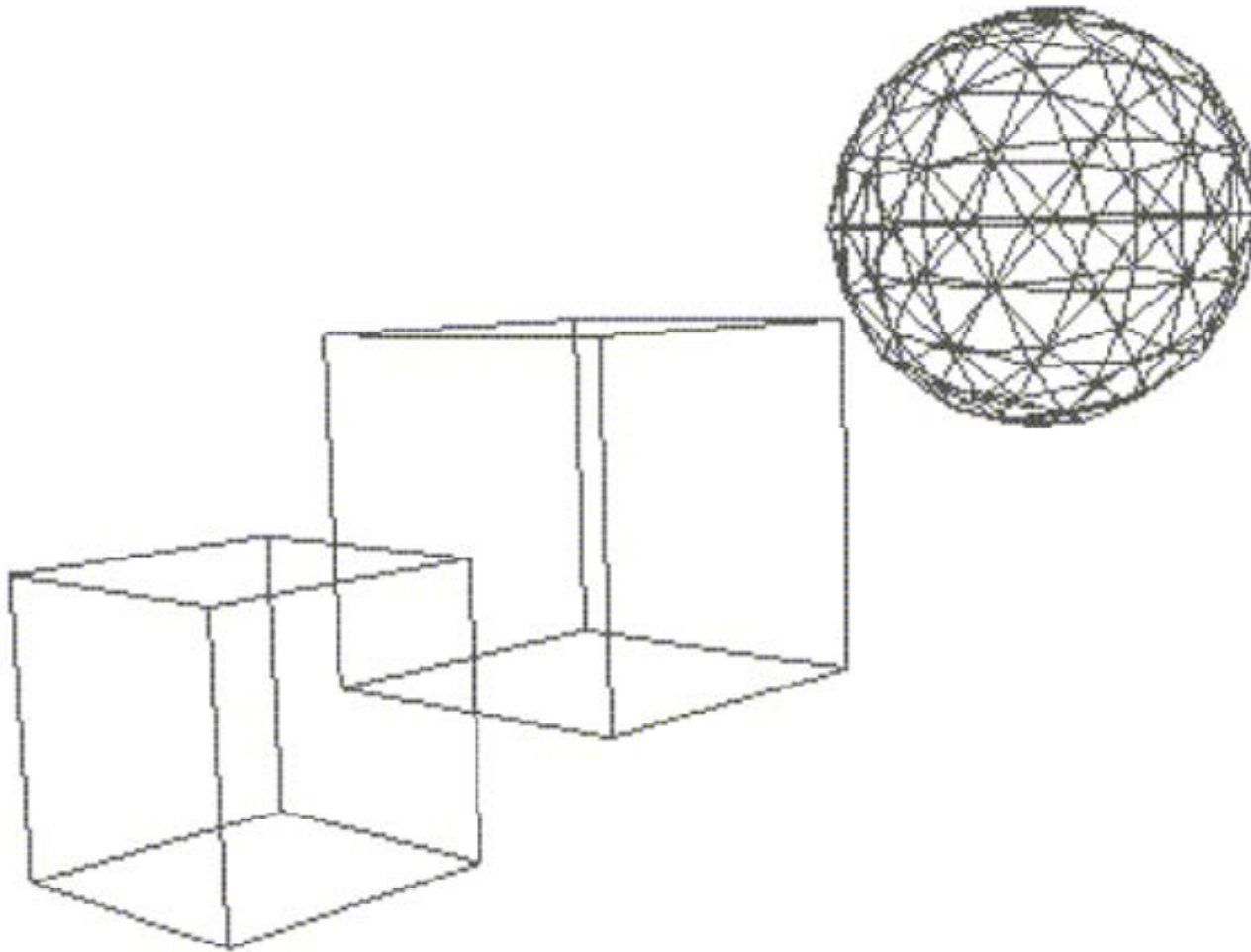
Motivación



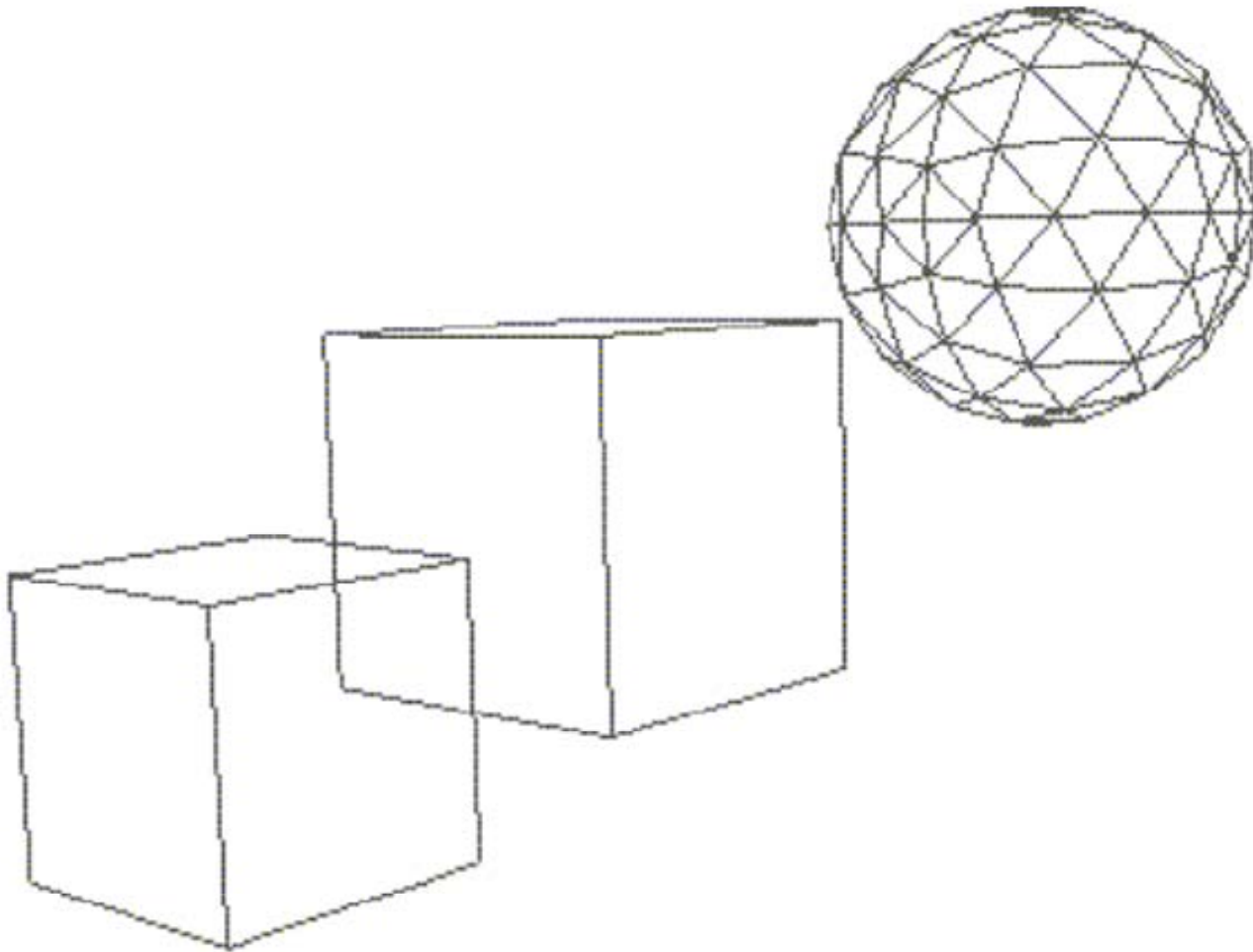
Motivación



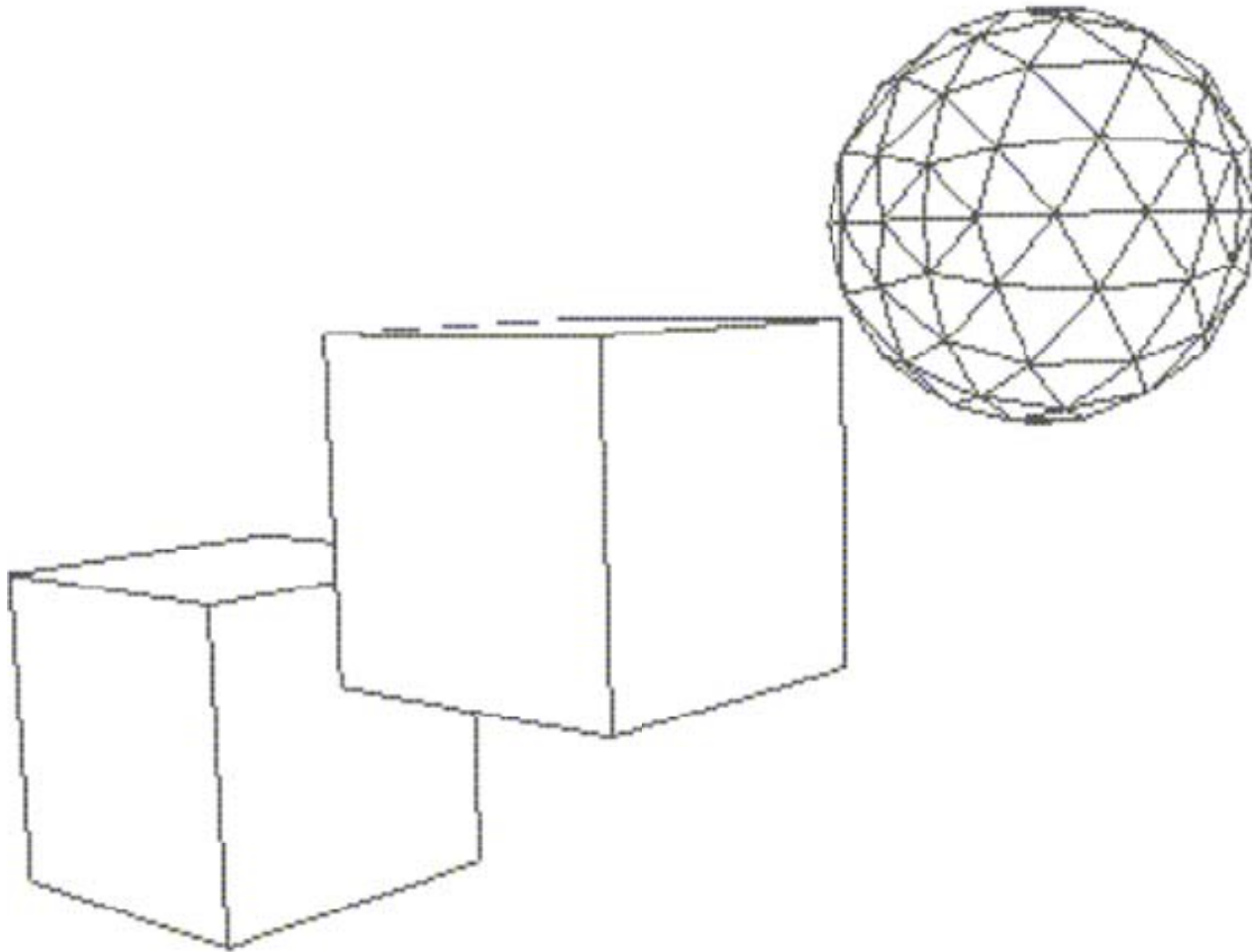
Motivación



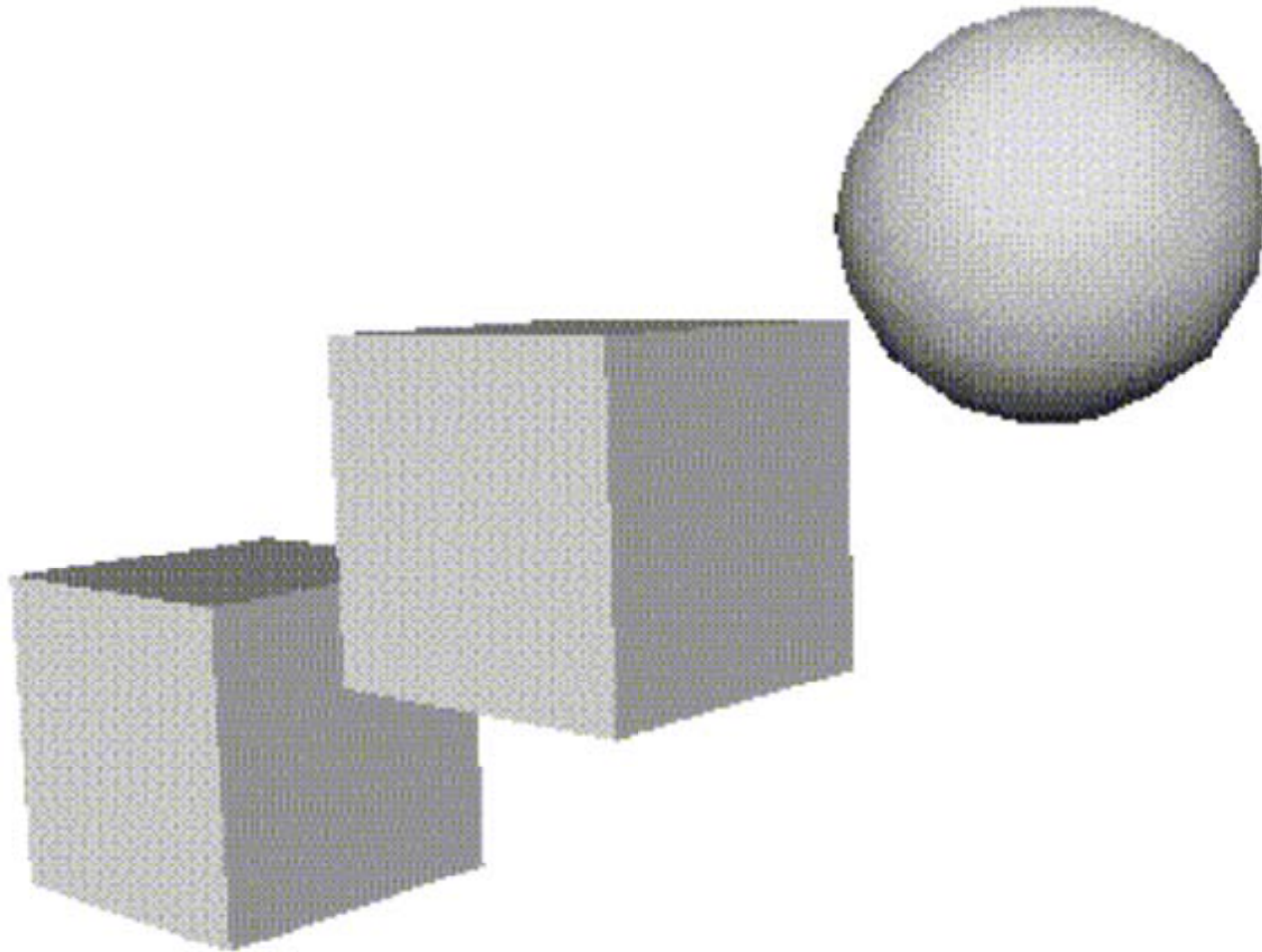
Motivación



Motivación



Motivación





- ❑ Realismo. El ocultar objetos no visibles permite crear una escena más realista
- ❑ Menos ambigüedad. Persepción de profundidad
- ❑ Eficiencia. El tiempo del renderizado es *time consuming*, por lo tanto no deberíamos malgastar tiempo en el renderizado de objetos no visibles en la escena

Clasificación de los algoritmos



- ❑ Eliminación de superficies escondidas (HSR – hidden surface removal) vs. Determinación de superficies visibles (VSD – visible surface determination)
 - ❑ HSR. Identifica que cosas no se van a ver
 - ❑ VSD. Identifica que cosas se van a ver
- ❑ Presición de imagen
 - ❑ Determinar el color del pixel basado en lo que es visible (ray tracing, z-buffering)
- ❑ Precisión de objeto

Precisión de imagen

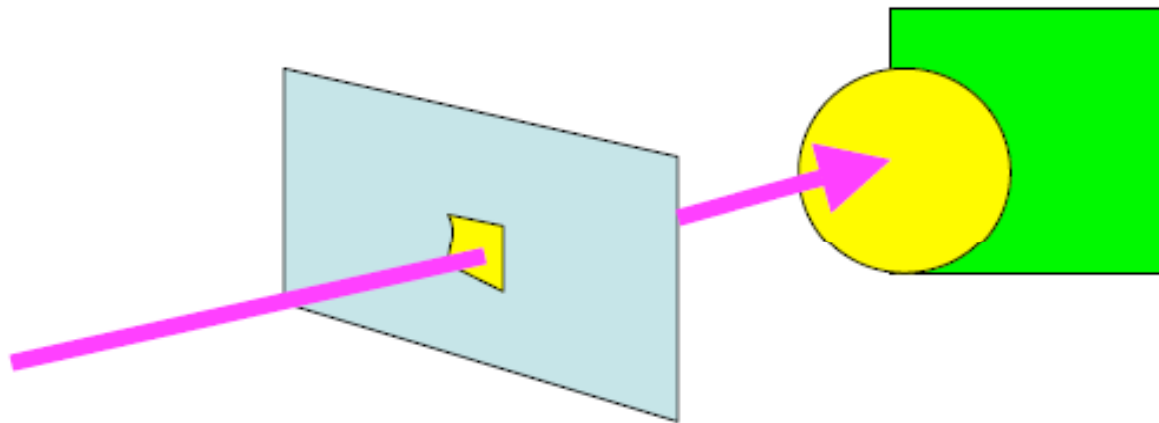


For (cada píxel de la imagen) {

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

Dibujar el píxel con el color apropiado;

}

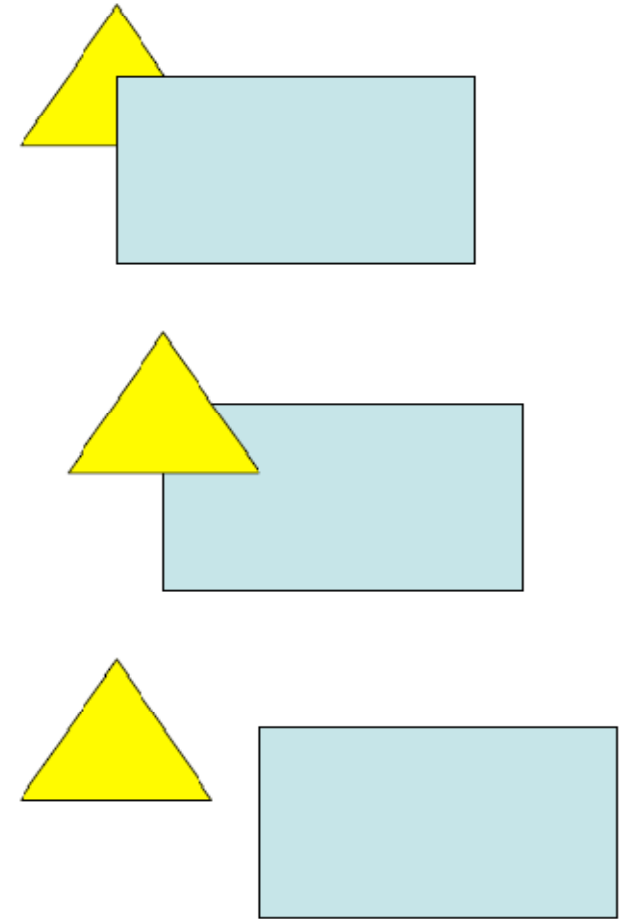


Este algoritmo depende del dispositivo utilizado y el tamaño de la ventana. Al cambiar la cantidad de píxeles se deben rehacer los cálculos.

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;
}



Este algoritmo es independiente del dispositivo de impresión. Si bien toda imagen termina en una pantalla o impresora, el despliegue de la misma se hace en una etapa posterior, cuando ya se conoce qué es lo que se va a dibujar y lo que no.



Precisión de Imagen.

En precisión de imagen, al ampliar una imagen hay que rehacer todos los cálculos.

Los algoritmos de precisión de imagen se hicieron para sistemas gráficos de barrido.

Precisión de Objeto

En precisión de objeto no se considera la resolución de la pantalla para los cálculos => el dibujo en pantalla es el último paso.

Los algoritmos de precisión de objeto se hicieron inicialmente para sistemas gráficos vectoriales.

Los algoritmos más recientes combinan la precisión de imagen y la precisión de objeto.

Técnicas de visibilidad eficientes



- Coherencia
- Transformación en perspectiva
- Extensiones y volúmenes acotantes
- Eliminación de caras posteriores
- Partición espacial
- Jerarquía

Coherencia



Coherencia = similitud local, continuidad, etc.

Motivación de la coherencia: ahorrar cálculos.

Hay 8 tipos de coherencia identificados:

- Coherencia de objetos
- Coherencia de áreas
- Coherencia de caras
- Coherencia de aristas
- Coherencia de aristas implicadas
- Coherencia de líneas de barrido
- Coherencia de profundidad
- Coherencia de cuadros

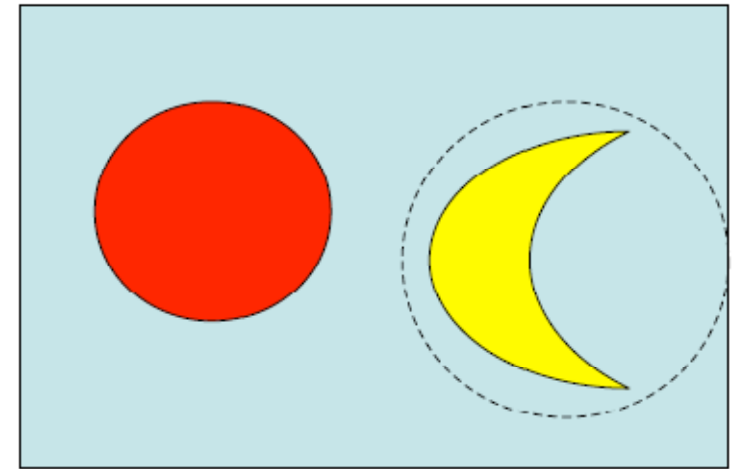
Aprovechar la coherencia para acelerar los cálculos, significa aprovechar las propiedades de los objetos, para que los cálculos ya realizados se aprovechen y permitan ahorrar el tiempo total utilizado para el cálculo de imágenes y estructuras intermedias de datos.

Coherencia



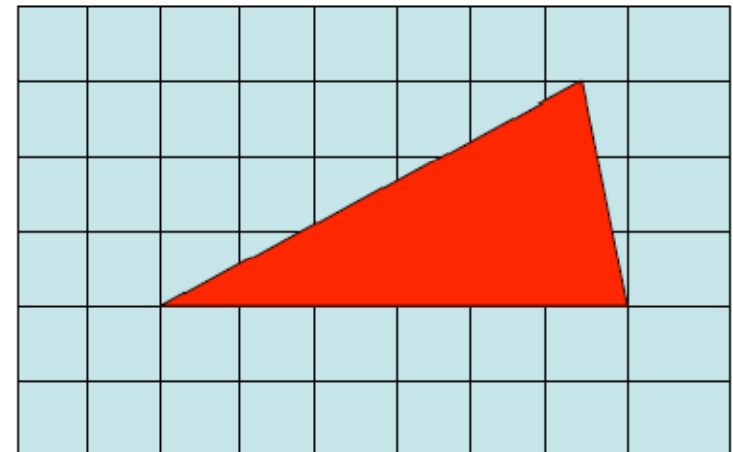
❑ Coherencia de objetos

- ❑ Si un objeto A no se superpone con otro B, no es necesario fijarse en la superposición de cada parte de A con cada parte de B.



❑ Coherencia de áreas

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





- ❑ Coherencia de caras
 - ❑ Las propiedades de las superficies varían suavemente sobre una misma cara. Se pueden hacer cálculos incrementales.
 - ❑ Si ya se calculó la distancia entre un punto de una cara plana y el observador, el punto siguiente en una línea de rastreo se puede calcular con un simple incremento del resultado anterior.
- ❑ Coherencia de aristas
 - ❑ Una arista puede hacerse visible, sólo cuando cruza detrás de otra arista (visible) o penetra en una cara visible.
 - ❑ Hay métodos en los que se divide en dos la cara que penetra a otra por la línea de intersección, para simplificar los algoritmos.



- ❑ 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 puntos de intersección.

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

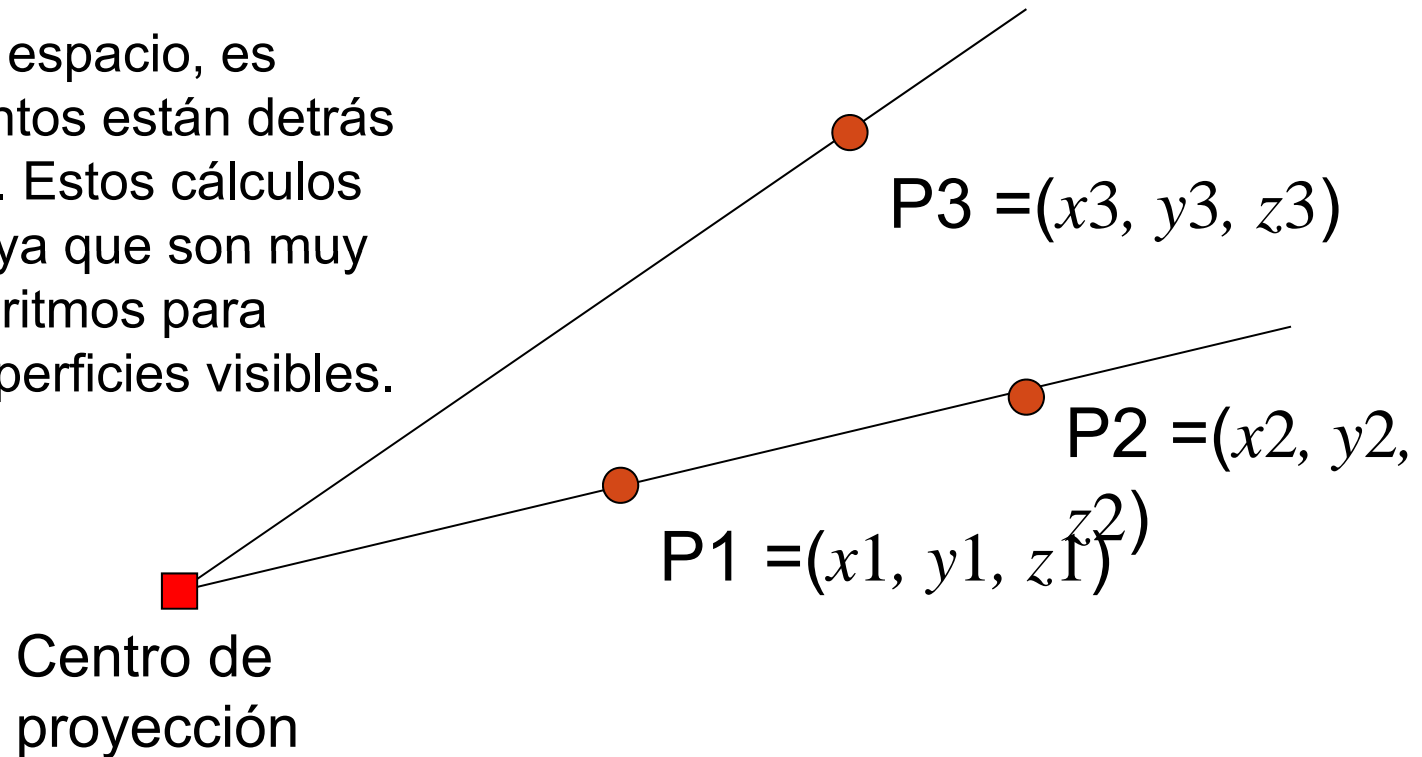


- ❑ Coherencia de profundidad
 - ❑ Partes cercanas de una misma superficie están muy cerca en profundidad.
 - ❑ Superficies distintas en el mismo lugar de la pantalla están generalmente a mayor separación de profundidad.
- ❑ Coherencia de cuadros
 - ❑ Las imágenes del mismo ambiente en dos instantes sucesivos en el tiempo casi siempre serán bastante similares. Los cálculos realizados para una imagen se pueden reutilizar en la siguiente.

Transformación de perspectiva



Dado un punto en el espacio, es bueno saber qué puntos están detrás según el observador. Estos cálculos hay que acelerarlos ya que son muy comunes en los algoritmos para determinación de superficies visibles.

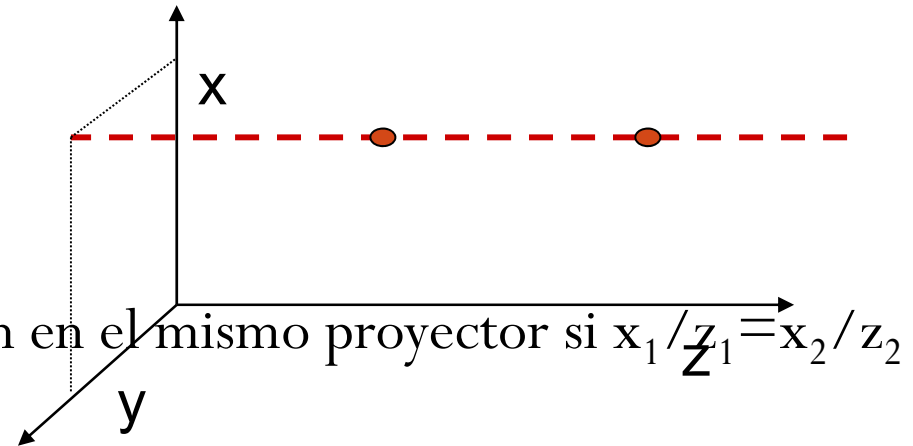


Se hacen transformaciones de normalización, para que los rayos proyectores sean paralelos al eje z (proyección paralela), o para que los rayos proyectores emanen del origen (proyección en perspectiva).

Transformación de perspectiva



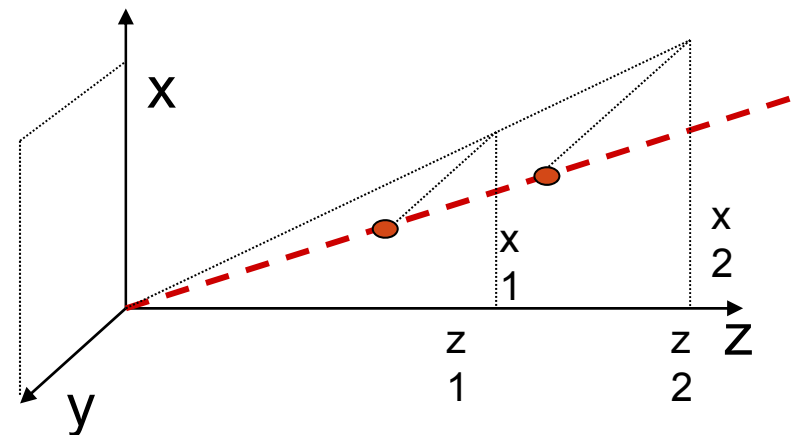
- Proyección paralela: puntos están en el mismo proyector si $x_1 = x_2$, $y_1 = y_2$



Estas equivalencias
facilitan enormemente los
cálculos

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

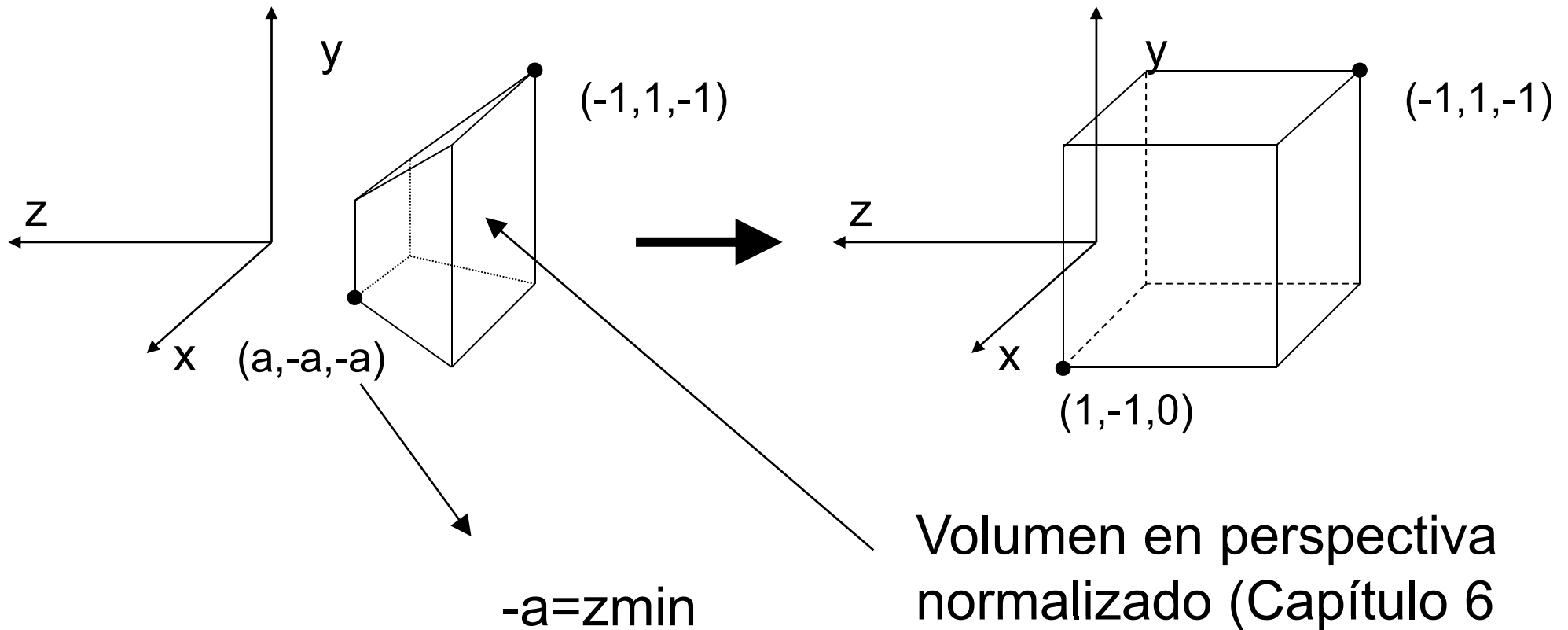
Si le dan a elegir entre ambos tipos de equivalencias, ¿cuál elegiría?. ¿Hay mucha diferencia entre incluir o no divisiones? ¿Cómo haría para trabajar con proyección de perspectiva y no incluir divisiones en las equivalencias?



Transformación de perspectiva



Se realiza una transformación en la que la proyección de perspectiva se convierte en una transformación paralela.



Volumen en perspectiva
normalizado (Capítulo 6
del Foley – Van Dam)

Transformación de perspectiva



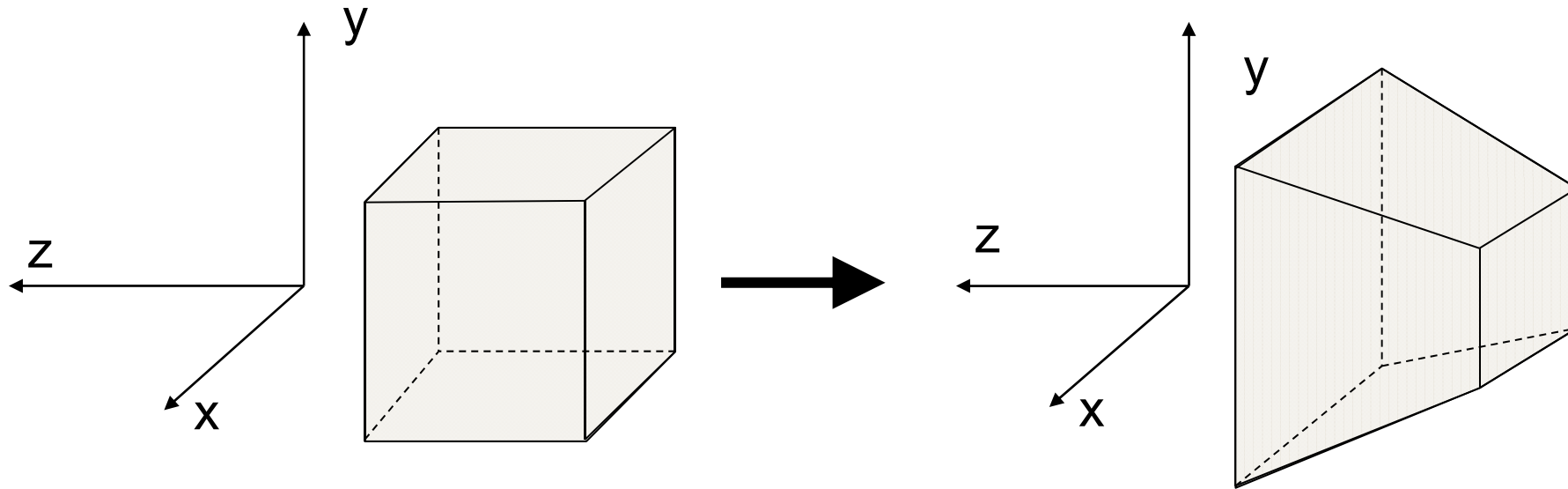
Al multiplicar las coordenadas homogeneas de puntos por la matriz M , estos se pasan de la proyección canónica en perspectiva a la proyección canónica paralela.

Compruebe que el punto $(a, -a, -a)$ se transforma en $(1, -1, 0)$.

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

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

Transformación de perspectiva

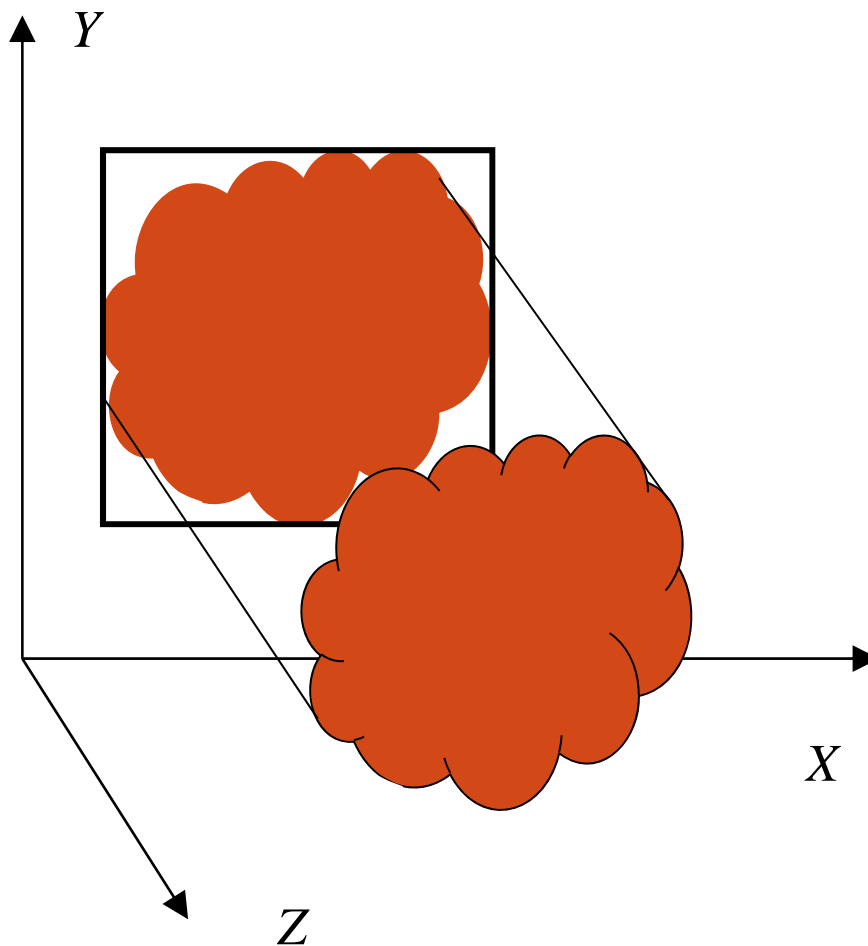


El cubo de la izquierda, al pasar de la proyección en perspectiva a la paralela, se transformará en la pirámide truncada de la derecha.



- Para simplificar muchos cálculos, en lugar de trabajar con objetos complejos, se trabajan con volúmenes acotantes más sencillos, que sirven de una primera aproximación.
- En el caso de la determinación de superficies visibles, en lugar de ver si un objeto oculta a otro total o parcialmente, se ve si el volumen acotante del primer objeto oculta total o parcialmente al volumen acotante del segundo objeto. Se supone que dichos cálculos son mucho más sencillos de realizar. Si llega a detectarse ocultamiento, entonces sí se comienza a ver si esos volúmenes se ocultan.

Extensiones y volúmenes acotantes

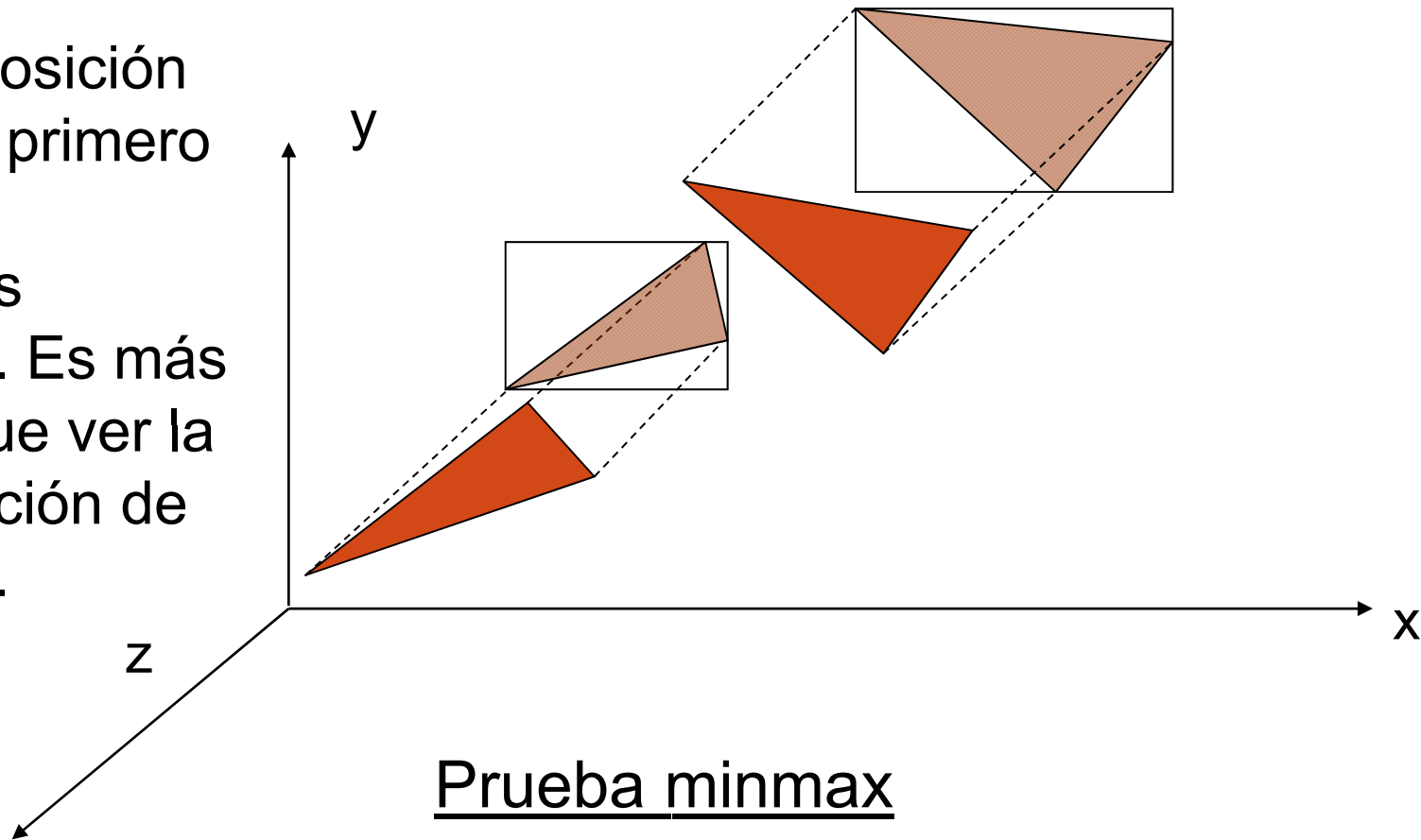


- ❑ El volumen complejo se proyecta en el plano XY .
- ❑ Luego, se puede generar una extensión al área proyectada.
- ❑ Si la proyección es canónica paralela, hallar este tipo de extensiones (cotas mínimas y máximas en x e y de la proyección), equivale a hallar las cotas mínimas y máximas en x e y de las coordenadas de los puntos del objeto.

Extensiones y volúmenes acotantes



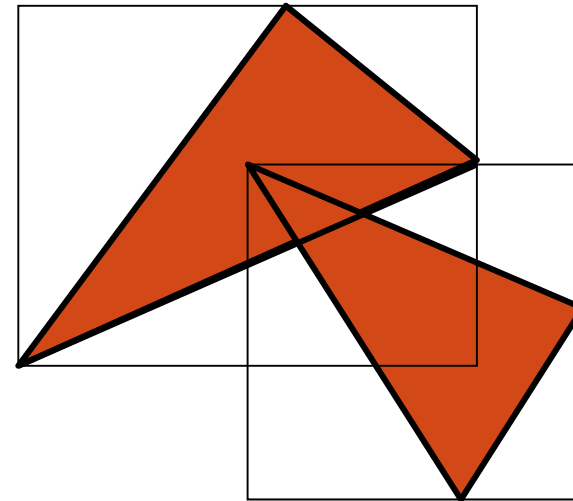
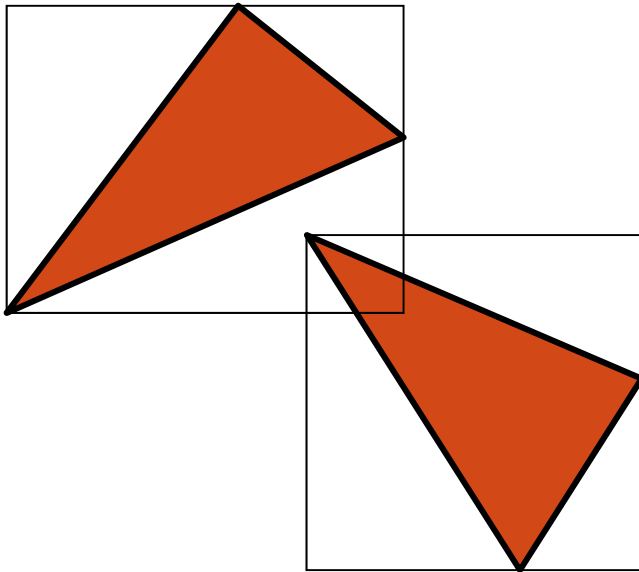
La superposición se realiza primero con los volúmenes acotantes. Es más sencillo que ver la superposición de triángulos.



No hay superposición si:

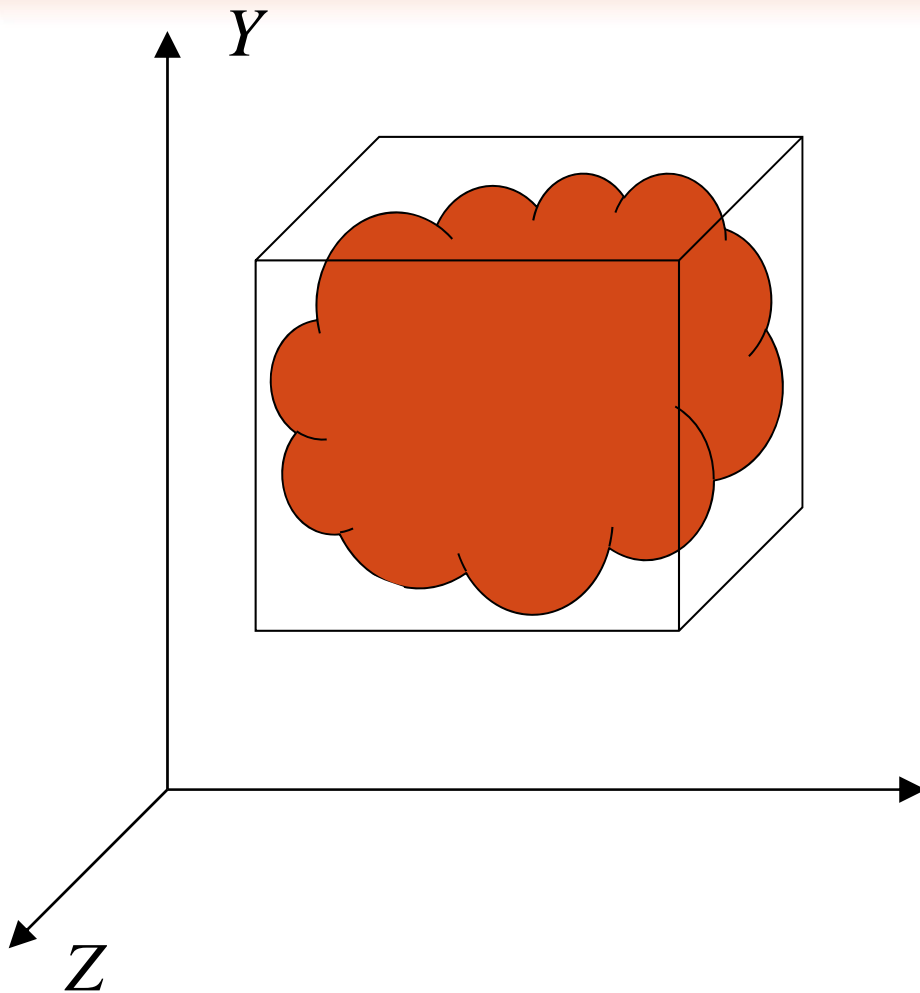
$$\left((x_{\max 2} < x_{\min 1}) \text{ or } (x_{\max 1} < x_{\min 2}) \right) \text{ and} \\ \left((y_{\max 2} < y_{\min 1}) \text{ or } (y_{\max 1} < y_{\min 2}) \right)$$

Extensiones y volúmenes acotantes



En el caso de la izquierda, las extensiones acotantes se intersecan pero no así las proyecciones de los objetos. Esto indica que no alcanza con este test, aunque sí sirve descartar el testeo directo de las proyecciones en la mayoría de los casos. Para objetos complejos esto implica un ahorro enorme en los cálculos.

Extensiones y volúmenes acotantes



Caja acotante, o
volumen acotante.

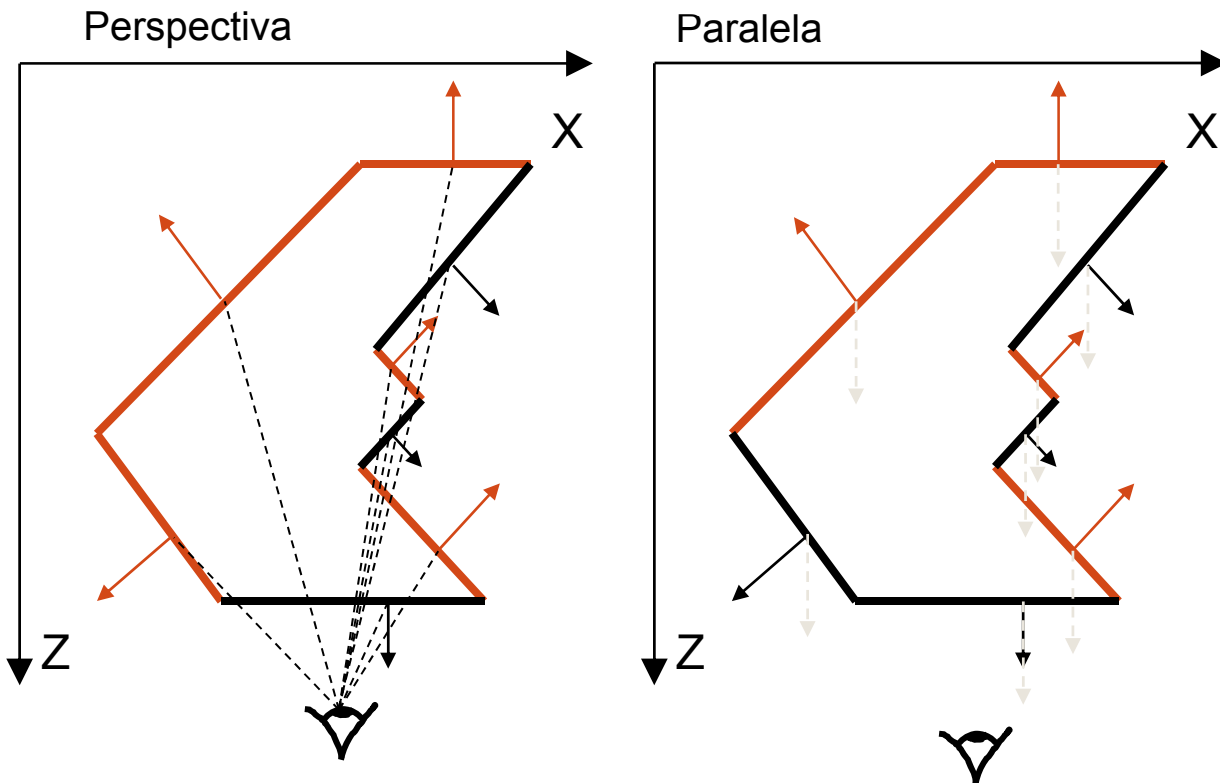
Prueba minmax

La prueba anterior
incluye el control de z
 $((z_{\max 2} < z_{\min 1}) \text{ or } (z_{\max 1} < z_{\min 2}))$

X

Este es un tipo de volumen muy utilizado, en el que las caras son paralelas a los ejes. Podrían haber otros volúmenes acotantes como esferas, elipsoides, etc.

Eliminación de caras posteriores



Según el tipo de proyección, las caras posteriores pueden ser diferentes para un mismo objeto y una misma ubicación del plano de proyección.

En un objeto cerrado, aproximadamente la mitad de las caras son posteriores (o sea, la normal se aleja del observador).

Las caras posteriores no son visibles, por haber siempre otra cara más cerca del observador.

Determinar las caras posteriores es sencillo, alcanza con calcular el ángulo entre la normal a la cara y el rayo de proyección. Si es mayor a 90° , la cara es posterior y no se dibuja.

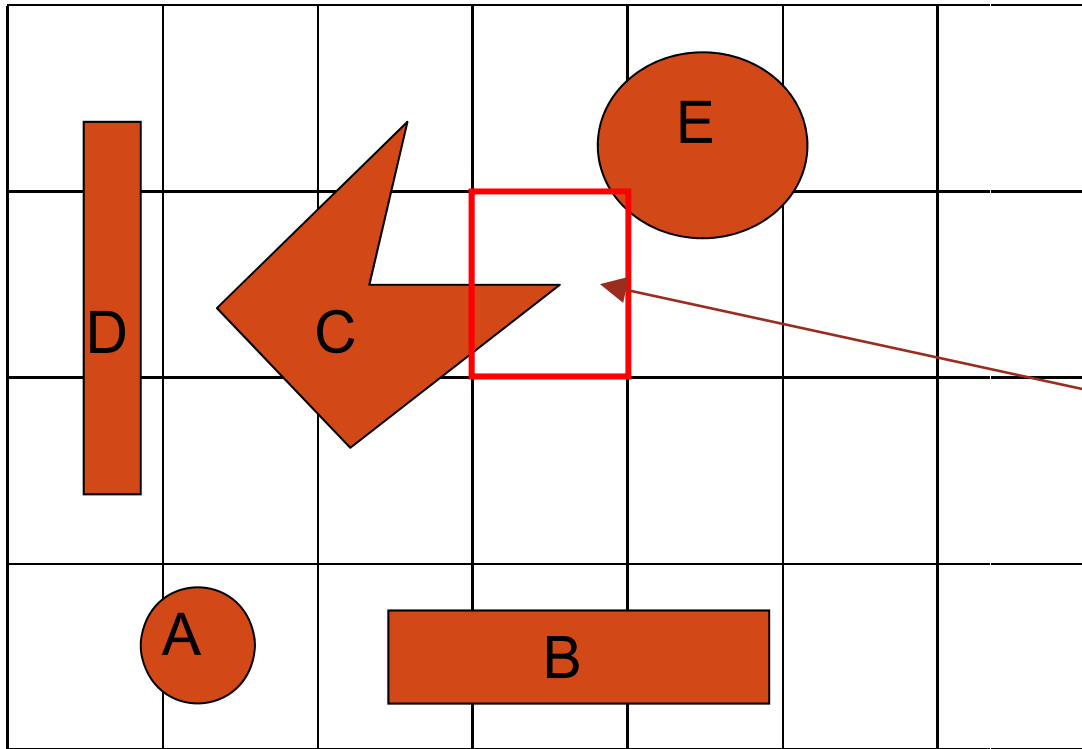
En estos dibujos, las caras posteriores se dibujan verdes.

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 (Octrees) y cuadrantes (Quadtrees o BSP-tree).

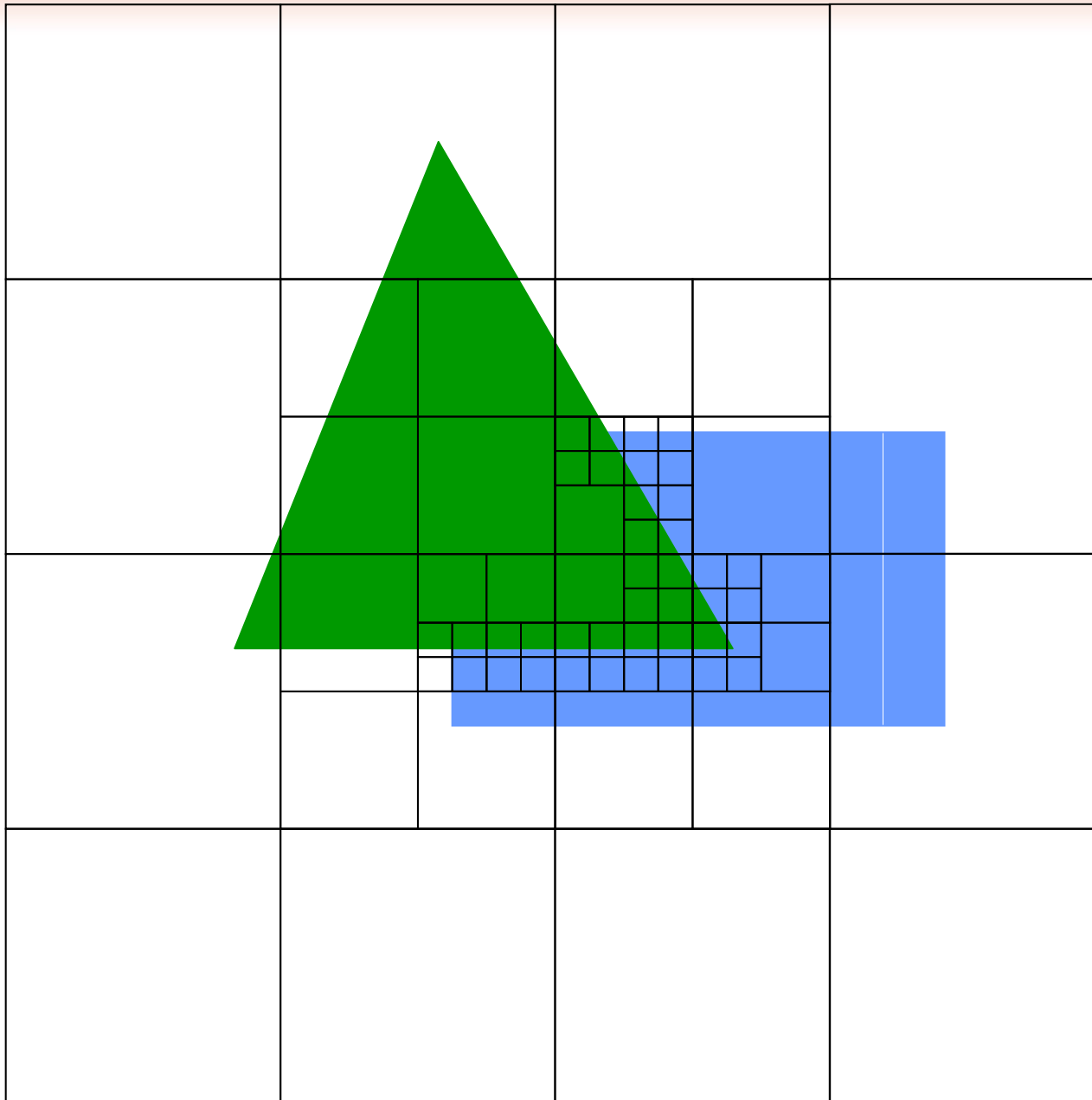
Partición Espacial



Aquí solo me preocupó de ver la superposición de aquellos objetos que caen en la misma celda.

En este dibujo, solamente C y E potencialmente se podrían llegar a superponer.

Partición Espacial

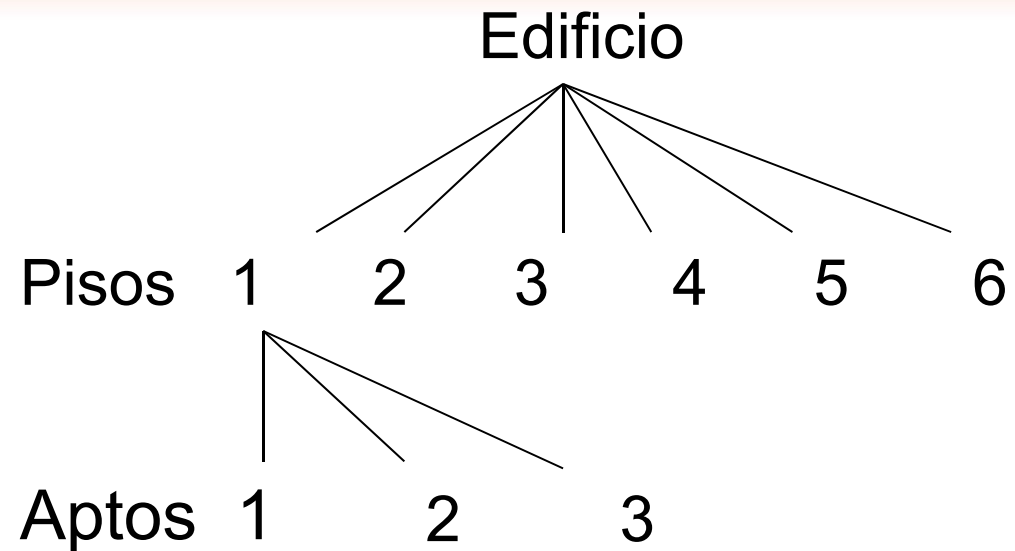
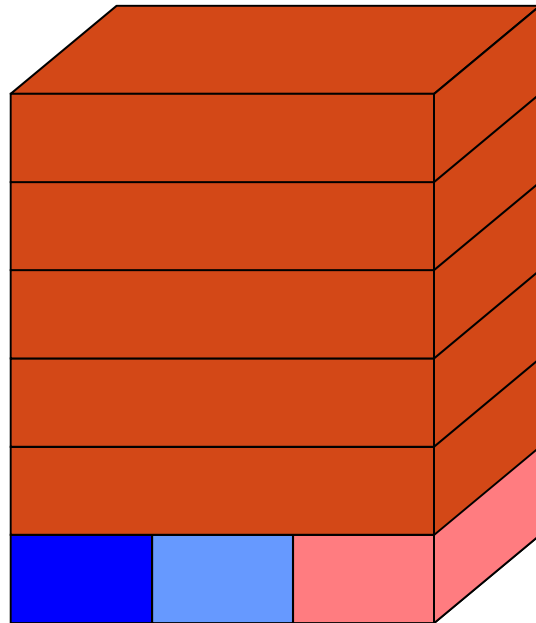


Una estructura de quadtree que facilita el trabajo y el tratamiento de las superficies visibles.

Este tipo de estructuras pueden servir para determinar qué partes de un objeto superponen a qué partes de otro objeto.

Se divide el área en 4, y luego se sigue subdividiendo si se cumple alguna propiedad (en este caso es más de un objeto por cuadro). Al final se detiene la subdivisión por algún criterio de parada. Esta estructura se puede asociar a un árbol con cuatro hijos por padre.

Jerarquía



En esta metodología primero me fijo si el edificio oculta o es ocultado (total o parcialmente) por otro objeto (un automóvil). Si la respuesta es negativa, me despreocupo. Si la respuesta es afirmativa, entonces comienzo a fijarme Sólo me fijo cuáles de los pisos ocultan otros objetos cuando la respuesta es afirmativa.



- ❑ Basado en imagen
 - ❑ Z-buffer (coherencia de profundidad)
 - ❑ Rastreo de línea (coherencia de rastreo de línea y profundidad)
 - ❑ Warnock (subdivisión de areas) – (coherencia de area y subdivisión espacial)
- ❑ Basado en el objeto
 - ❑ Orden de profundidad (región acotada)
 - ❑ Particionamiento de espacio binario (BSP)



- ❑ Es uno de los más simples y más usados
- ❑ Facilmente implementablea nivel de hardware
 - ❑ En OpenGL

```
glutInitDisplay(GLUT_RGB | GLUT_DEPTH)
glEnable(GL_DEPTH_TEST)
```
 - ❑ El *depth buffer* almacena la profundidad del objeto más cercano al punto (x,y) de pantalla
- ❑ Trabaja basado en la imagen

Algoritmo de z-buffer

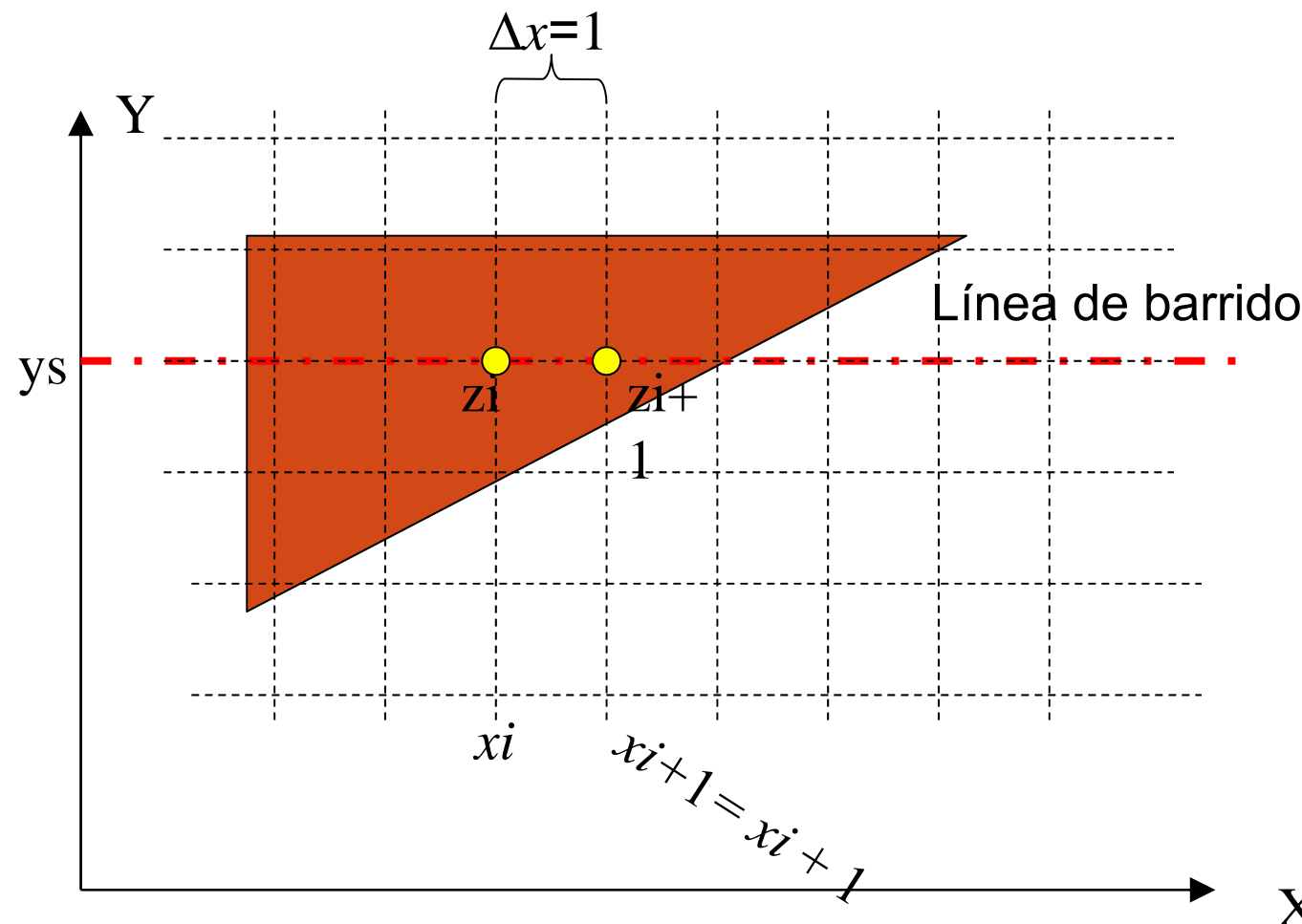


Este algoritmo está implementado en la mayoría de las tarjetas gráficas y forma parte de las técnicas utilizadas por las bibliotecas gráficas (opengl, DirectX)

Se crea una matriz que contiene los valores z de la superficie visible para cada píxel.

Aquí se ve una representación de la matriz. Se recorren las líneas de barrido para calcular los z .

Como el triángulo pertenece a un plano. Dado x e y es fácil calcular el z .



Algoritmo de z-buffer



Cálculo de profundidad de un plano.

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

Calculado el valor z de un plano en un píxel, se puede calcular fácilmente el valor de z en el punto siguiente de la línea de barrido,

¿Qué tipo de coherencia se puede utilizar aquí para acelerar los cálculos?

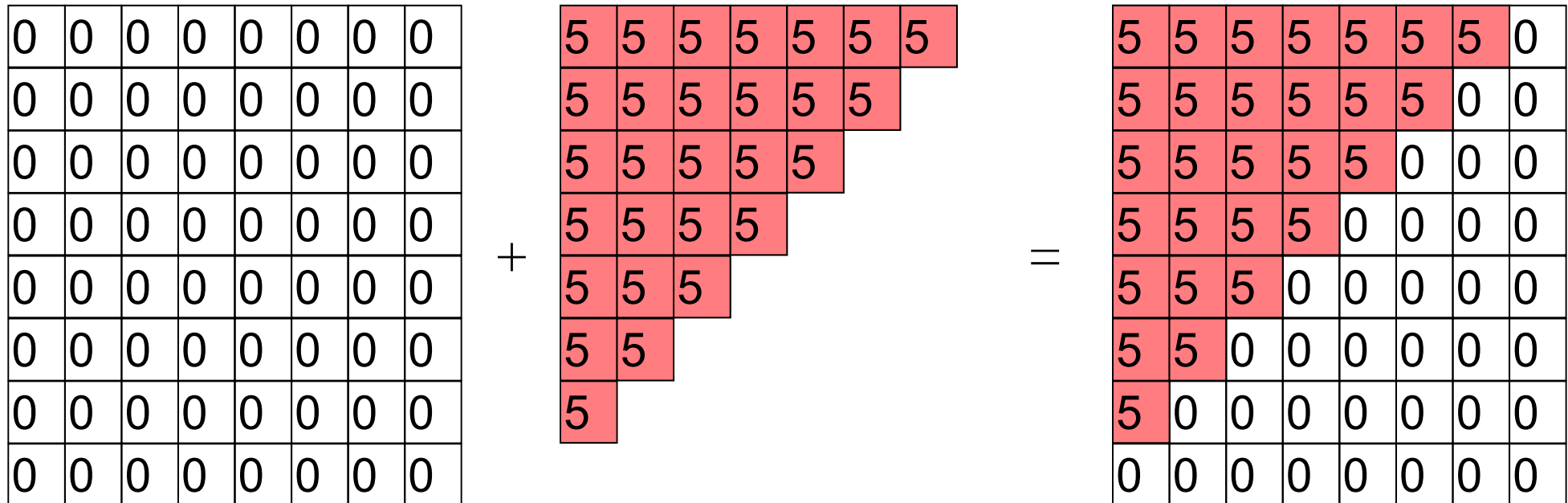
Algoritmo de z-buffer



```
void memoria_z
int pzi; /*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));
            }
        }
    }
}
```

Se inicializan las matrices

Algoritmo de z-buffer



Dado el z buffer vacío, tiene un cero asociado a cada celda (y pixel).

Luego, se recorre un primer triángulo paralelo al plano z. Por tanto, todas las celdas tienen el mismo valor.

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

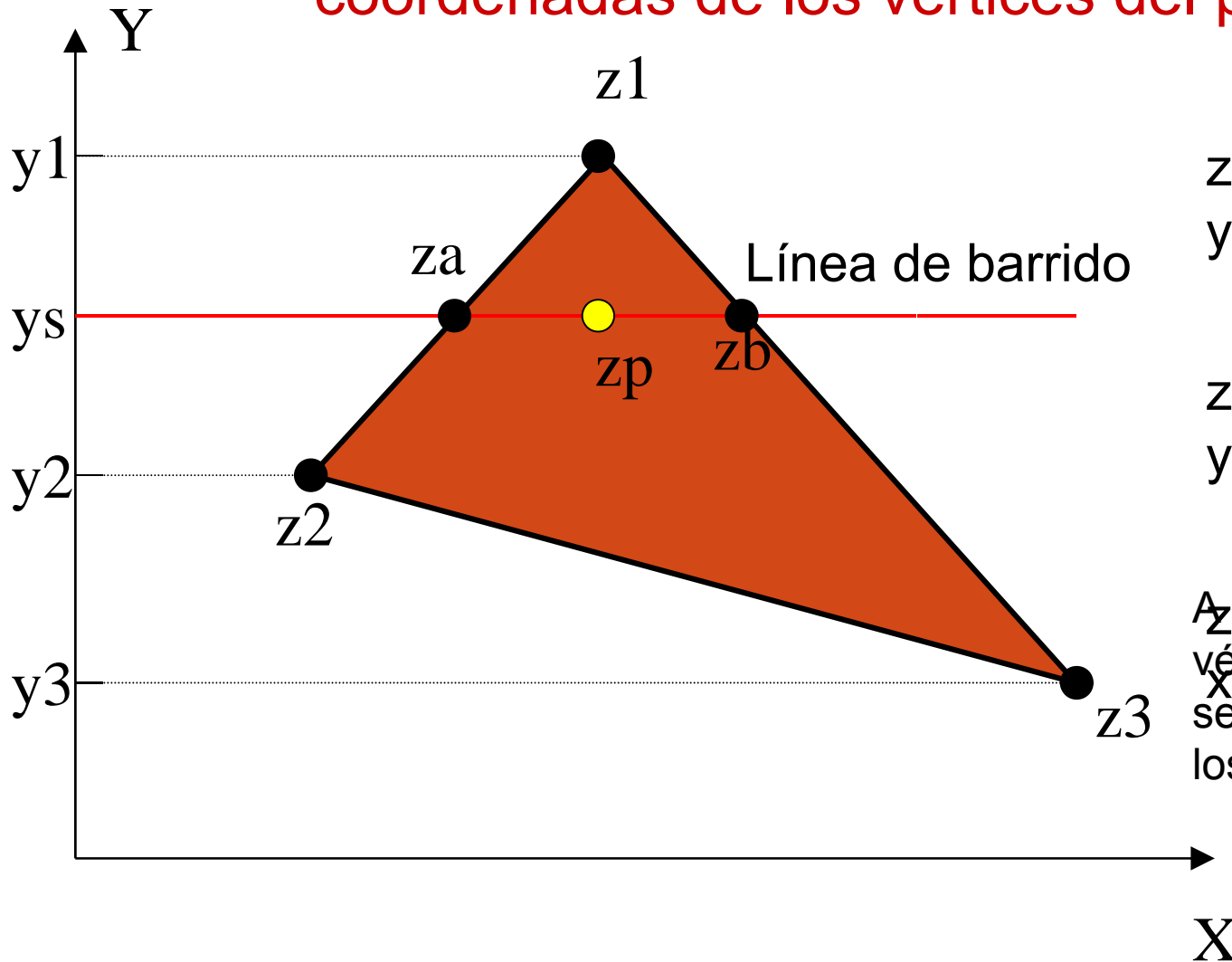
Luego, se recorre un segundo triángulo. Se compara esos valores con los ya existentes. Si en el triángulo hay valores mayores que los equivalentes en el z-buffer, entonces se ingresan dichos valores al z-buffer.

¿Considera que es práctico calcular el z de cada píxel para cada polígono? ¿Por qué será el algoritmo elegido para ser implementado en las tarjetas gráficas?

Algoritmo de z-buffer



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



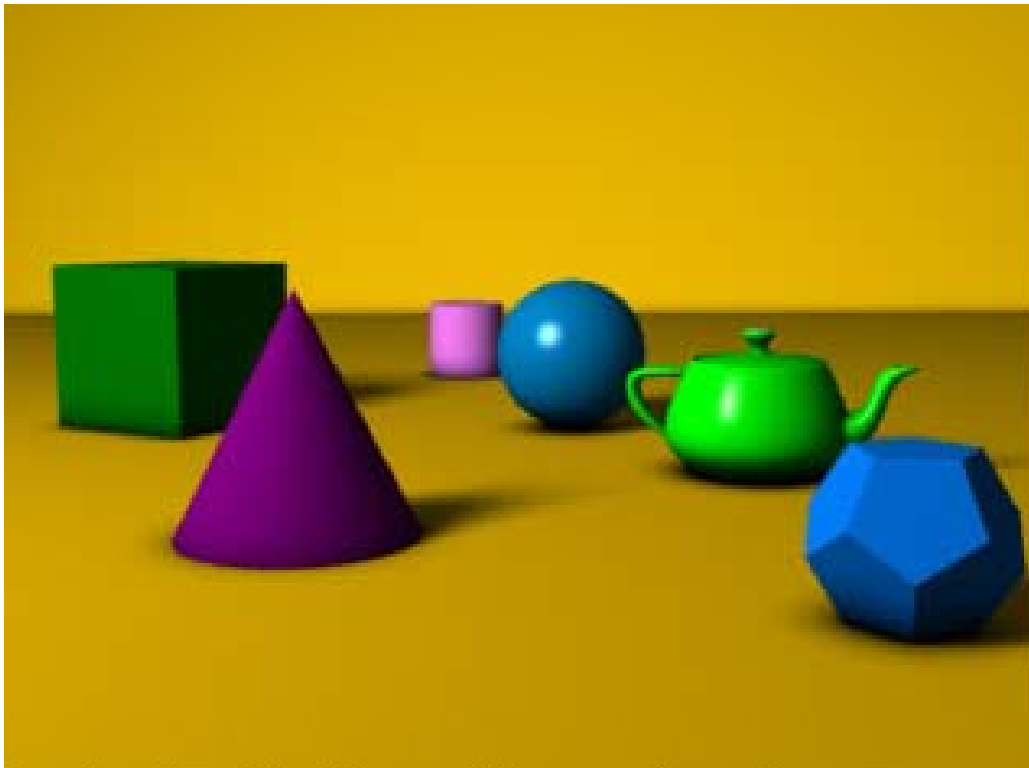
$$z_a = z_1 - (z_1 - z_2)(y_1 - y_s) / (y_1 - y_2)$$

$$z_b = z_1 - (z_1 - z_3)(y_1 - y_s) / (y_1 - y_3)$$

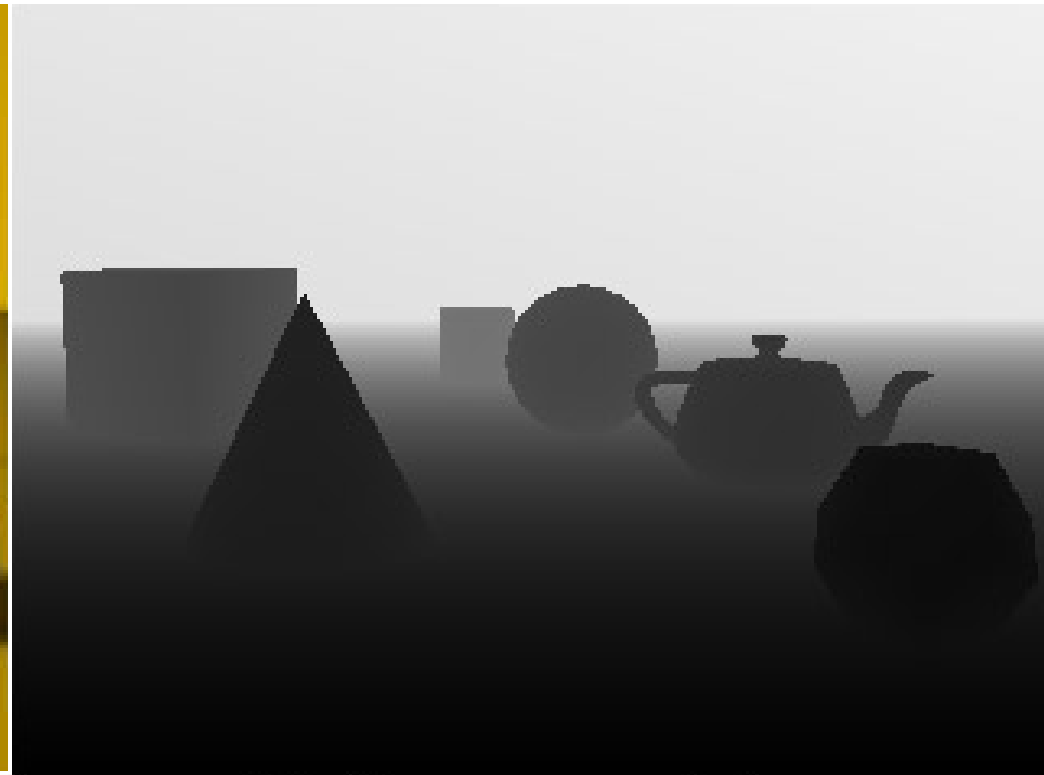
A partir de los valores z de los vértices (los cuales se disponen), se pueden calcular los valores de los puntos interiores al triángulo.

¿Cómo aceleraría los cálculos de este tipo a lo largo de una línea de barrido?

Z-buffer



A simple three dimensional scene



Z-buffer representation

Z-buffer



- ❑ Puede pintar el mismo pixel varias veces
 - ❑ Pintar solo despues que haya pasado la prueba del z-buffer
- ❑ Se puede pintar el inverso de la escena
- ❑ Requiere de una capacidad de memoria bastante grande.

Algoritmos de línea de barrido



Tabla de aristas (aristas horizontales son ignoradas)

- La coordenada X del extremo con menor coordenada Y.
- La coordenada Y del otro extremo de la arista.
- El incremento X, Δx , que se usa para pasar de una línea de rastreo a la siguiente.
- El número de identificación del polígono.

Tabla de aristas activas (AET)

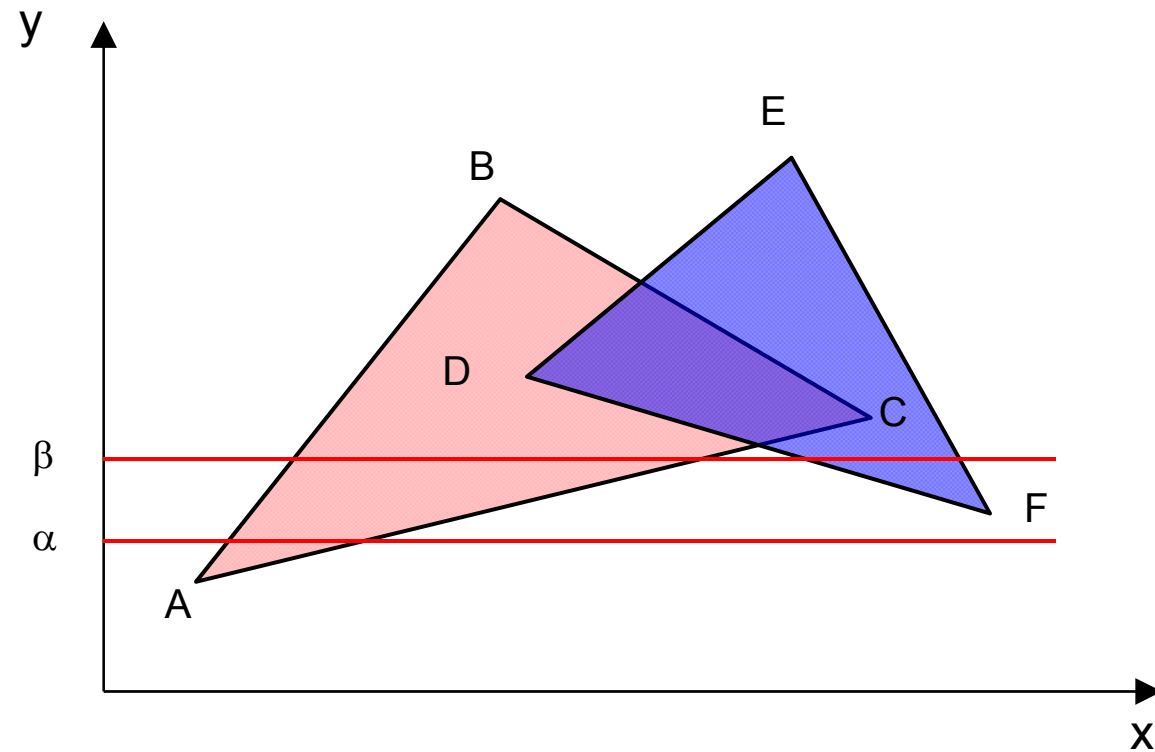
- Contiene las aristas que intersecan la línea de rastreo actual. Las aristas están en orden creciente de x.

Tabla de polígonos

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

Se generan estas tablas, y el algoritmo recorre la pantalla por líneas de barrido.

Algoritmos de línea de barrido

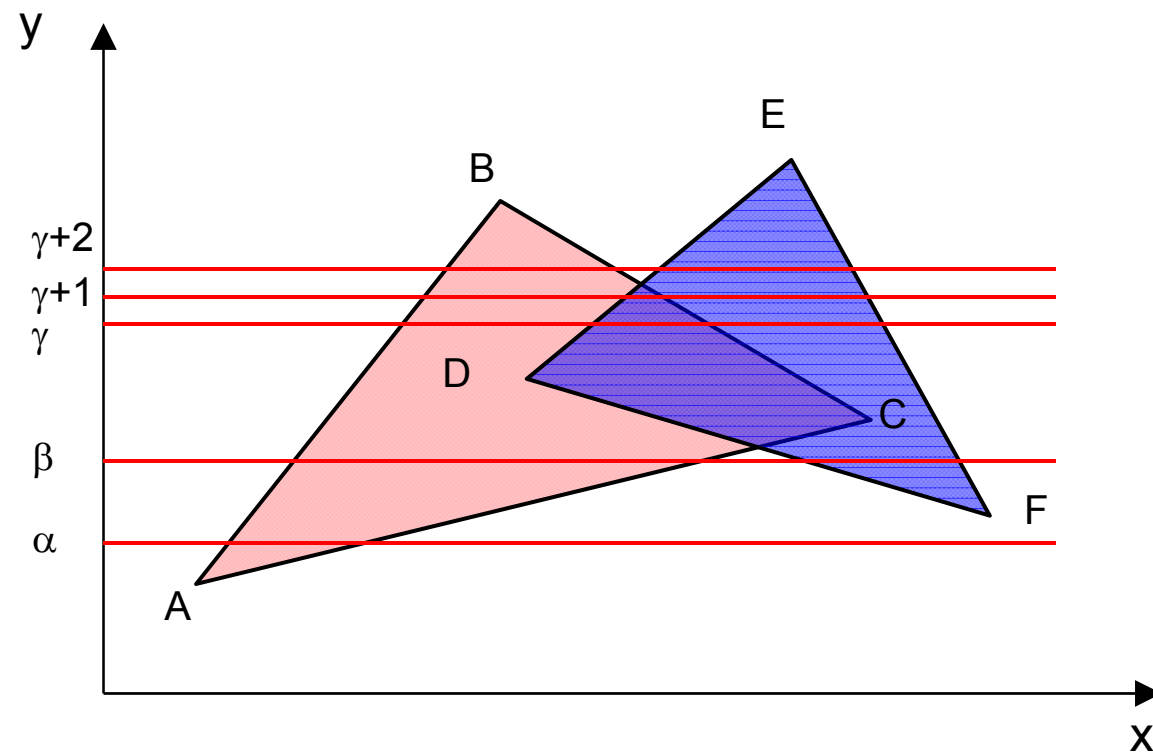


Dada una línea de rastreo, se calculan las aristas que la intersecan y se las ubica de forma ordenada (izquierda a derecha) en la AET.

En el caso de α , AET primero tiene a AB. A partir del punto de intersección, se comienza a pintar el polígono ABC y se pone en *verdadero* la bandera del polígono ABC.

AET luego tiene a AC. A partir de ese punto, se pone en *falso* la bandera y se deja de pintar.

Algoritmos de línea de barrido



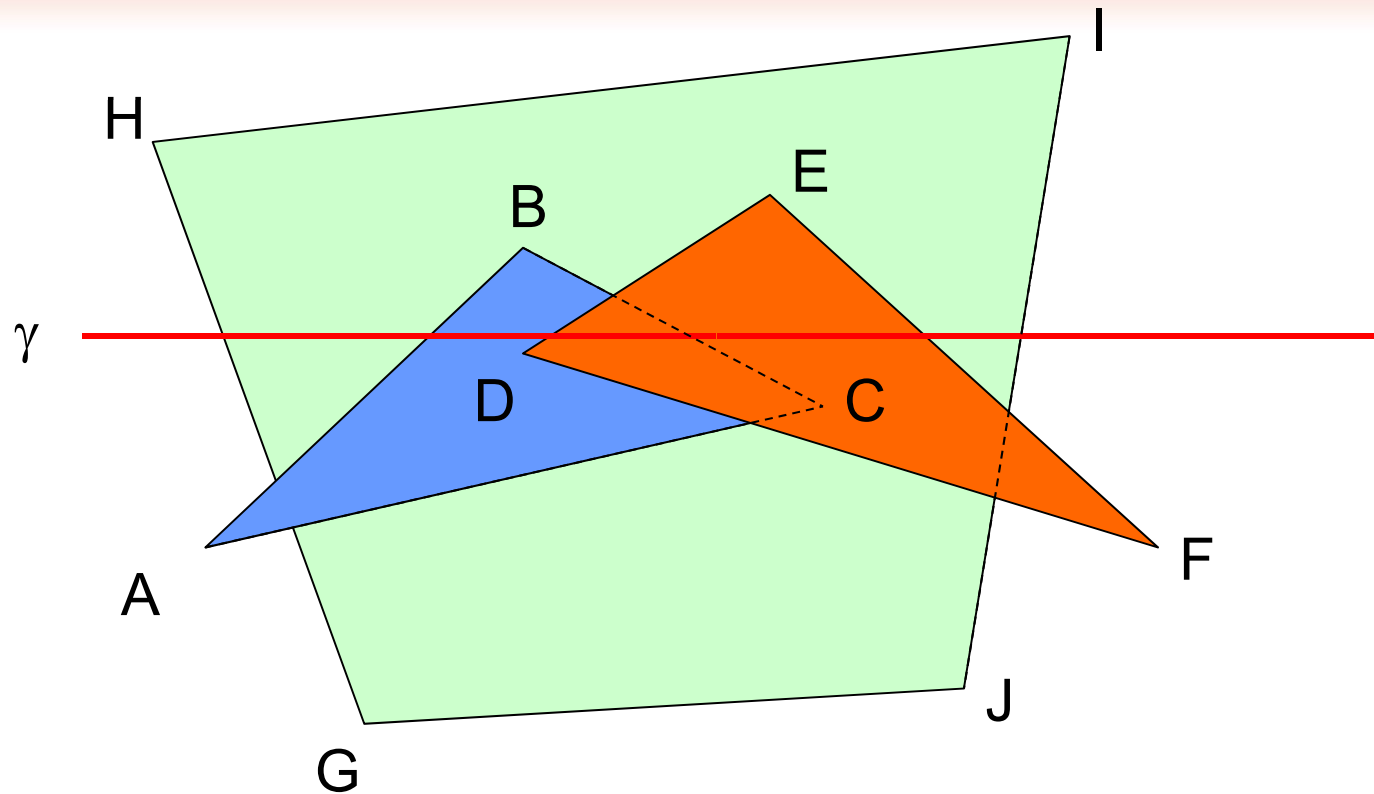
Se aprovechan los resultados de una línea de barrido para la siguiente línea.

En la línea de barrido γ , al llegar a la arista DE, hay que calcular qué polígono está por delante.

En ese punto hay 2 banderas en *verdadero*, pero sólo se dibuja un polígono.

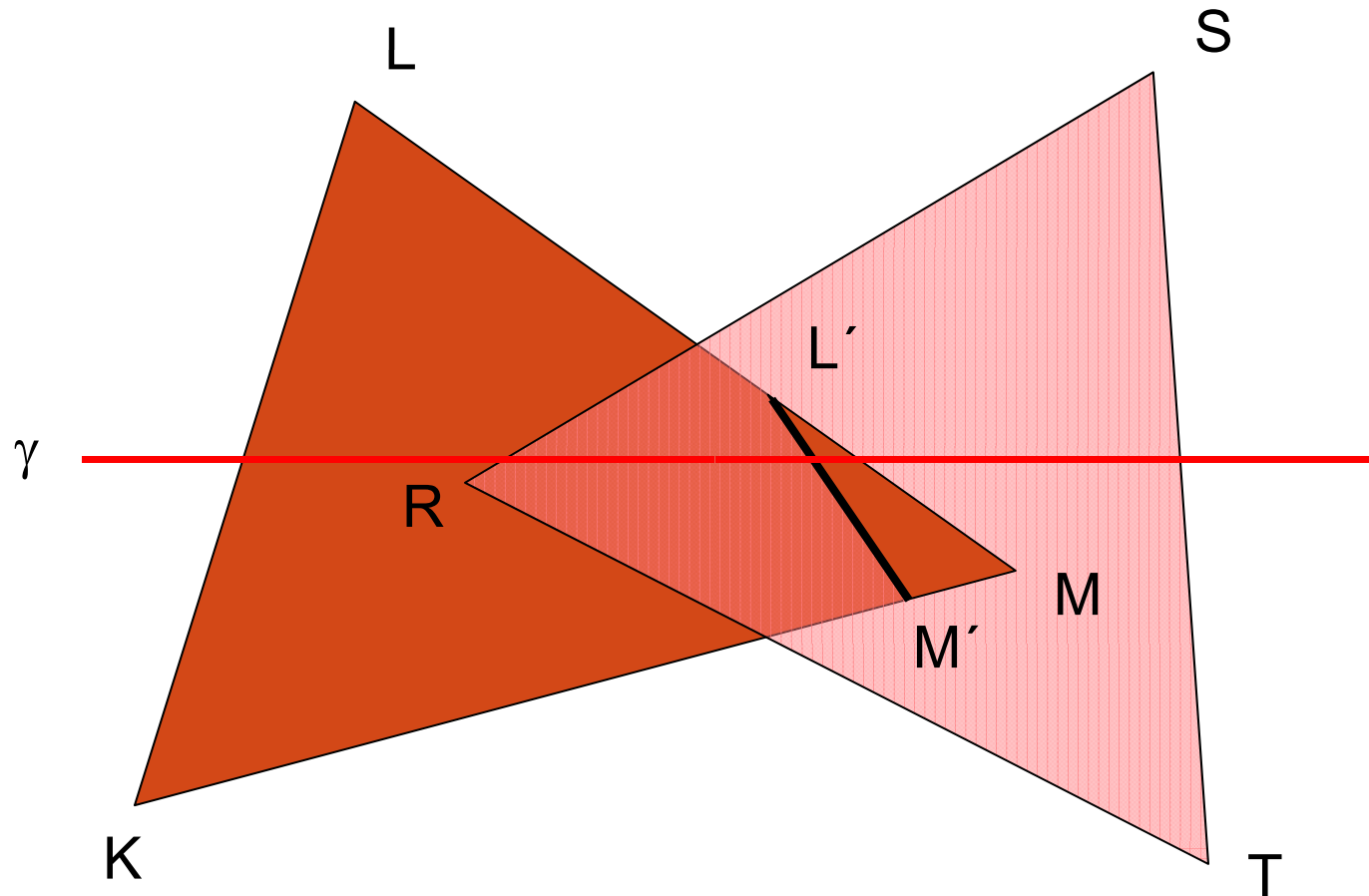
Los cálculos realizados para γ sirven para $\gamma+1$, dado que las aristas en la Tabla de Aristas Activas (AET) son las mismas en ambos casos y están en el mismo orden.

Algoritmos de línea de barrido



Al no haber polígonos interpenetrantes, no es necesario realizar cálculos de profundidad al pasar γ por BC. La elección hecha sobre qué superficie es visible entre DE y BC sirve para todos los demás líneas de barrido en que DE está antes de BC (de izquierda a derecha).

Algoritmos de línea de barrido



Aquí hay polígonos interpenetrantes. Para que el algoritmo funcione bien, el polígono KLM se debe dividir en 2.

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 píxel en la línea de barrido*) {

 determinar las superficies que se proyectan al píxel;

 encontrar la más cercana de estas superficies;

 en el píxel, determinar el color de la sup. más cercana;

 }

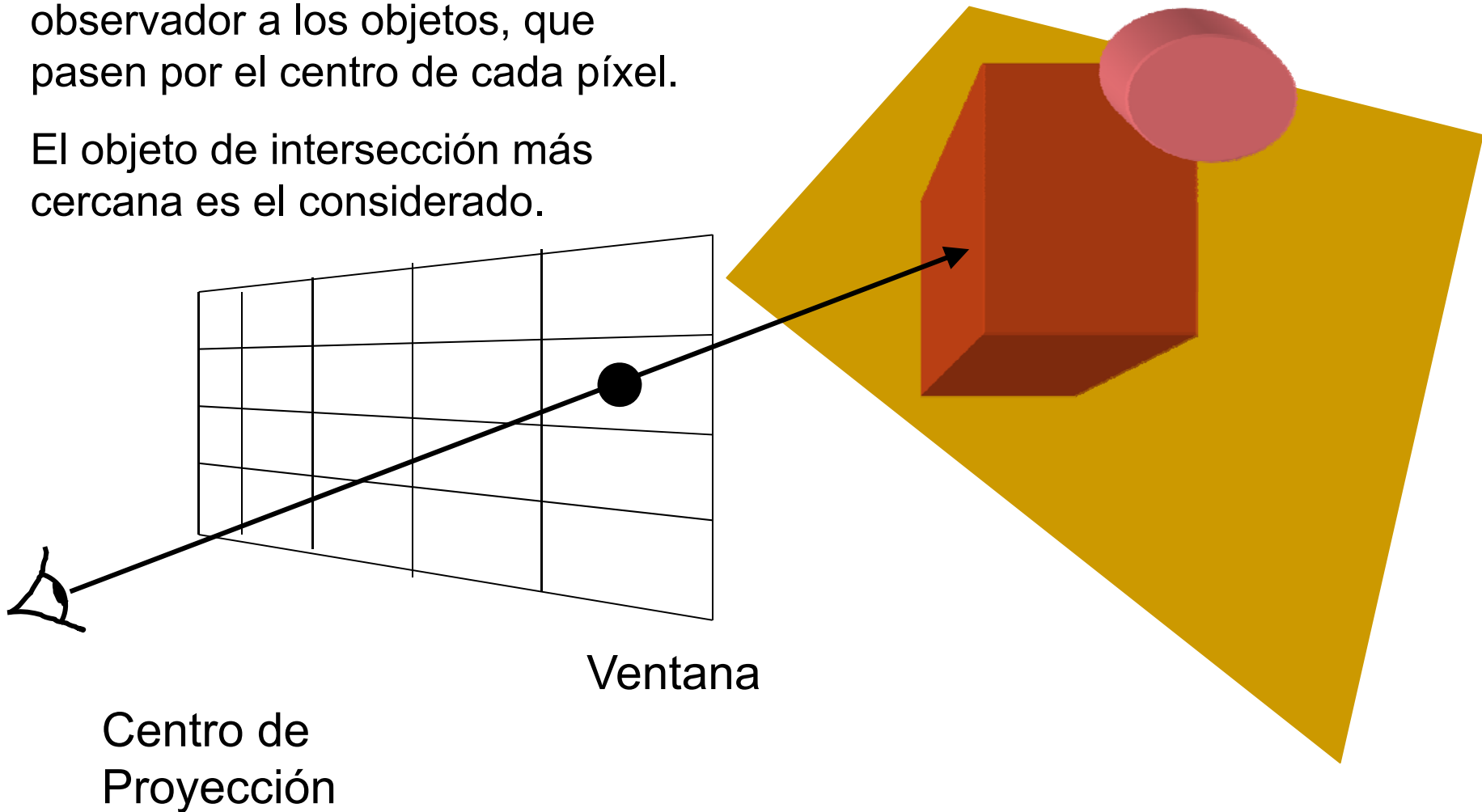
}

Traza de rayos en superficies visibles



Se basa en lanzar rayos desde el observador a los objetos, que pasen por el centro de cada píxel.

El objeto de intersección más cercana es el considerado.



Traza de rayos en superficies visibles



Seudocódigo

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

```
For ( cada línea de rastreo en la imagen ) {  
    For ( cada píxel en la línea de rastreo ) {  
        hallar el rayo centro de proyección por el píxel;  
        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 píxel corresp. al del objeto más cercano;  
    }  
}
```

¿Qué diferencia hay entre este algoritmo y el Z-buffer?

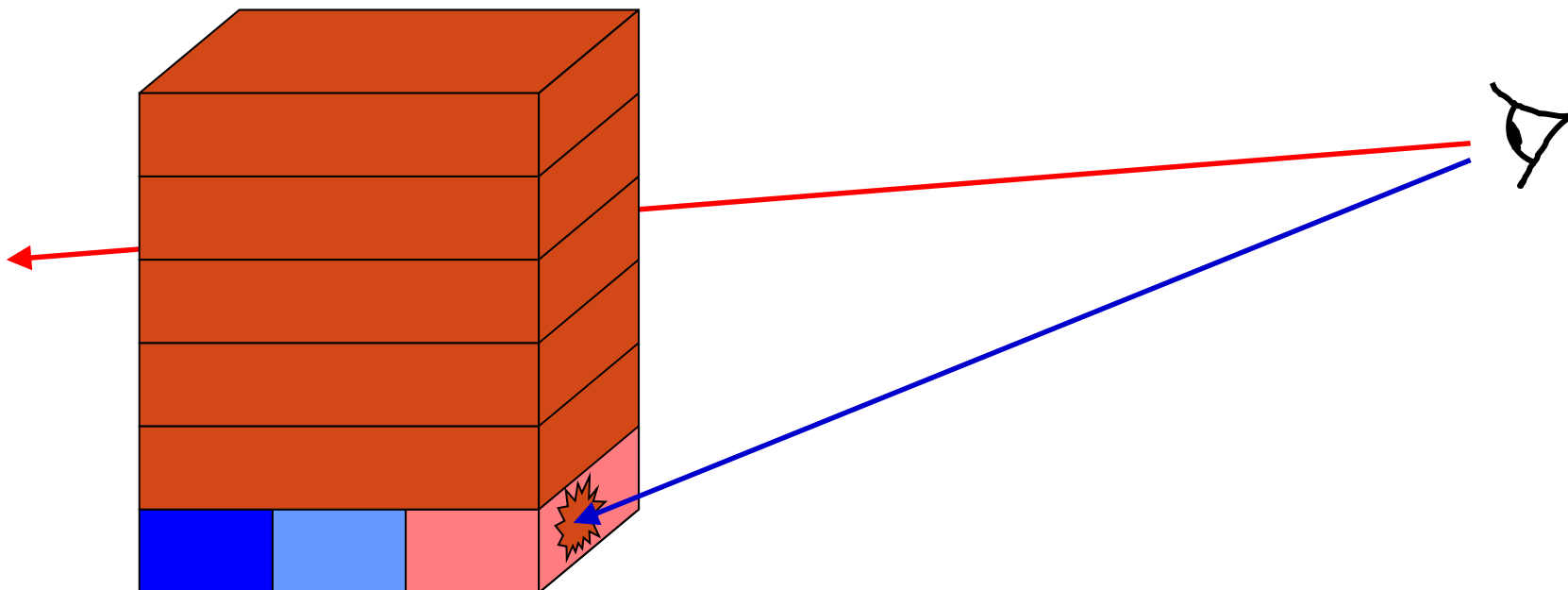
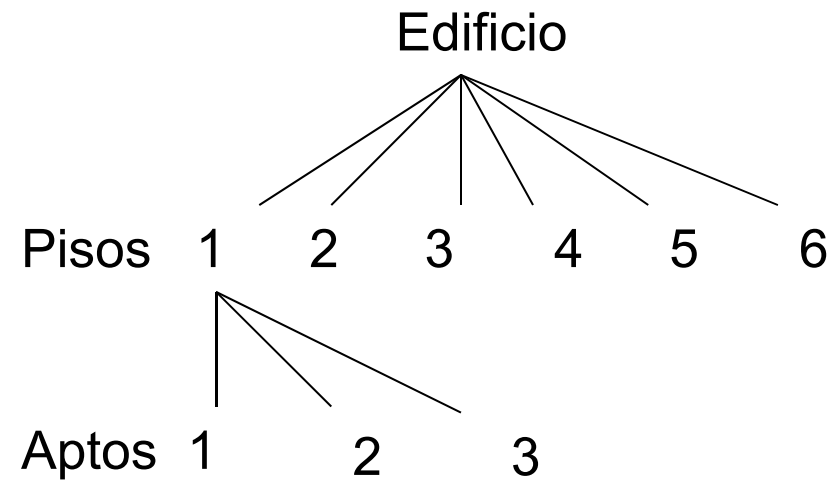


- ❑ Hallar constantes en las ecuaciones de intersección objeto-rayo.
- ❑ Hacer que el rayo coincida con el eje z y aplicar la misma transformación geométrica a los objetos (simplifica el cálculo de la \cap y permite determinar el objeto más cercano con un ordenamiento basado en z).
- ❑ Volúmenes acotantes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

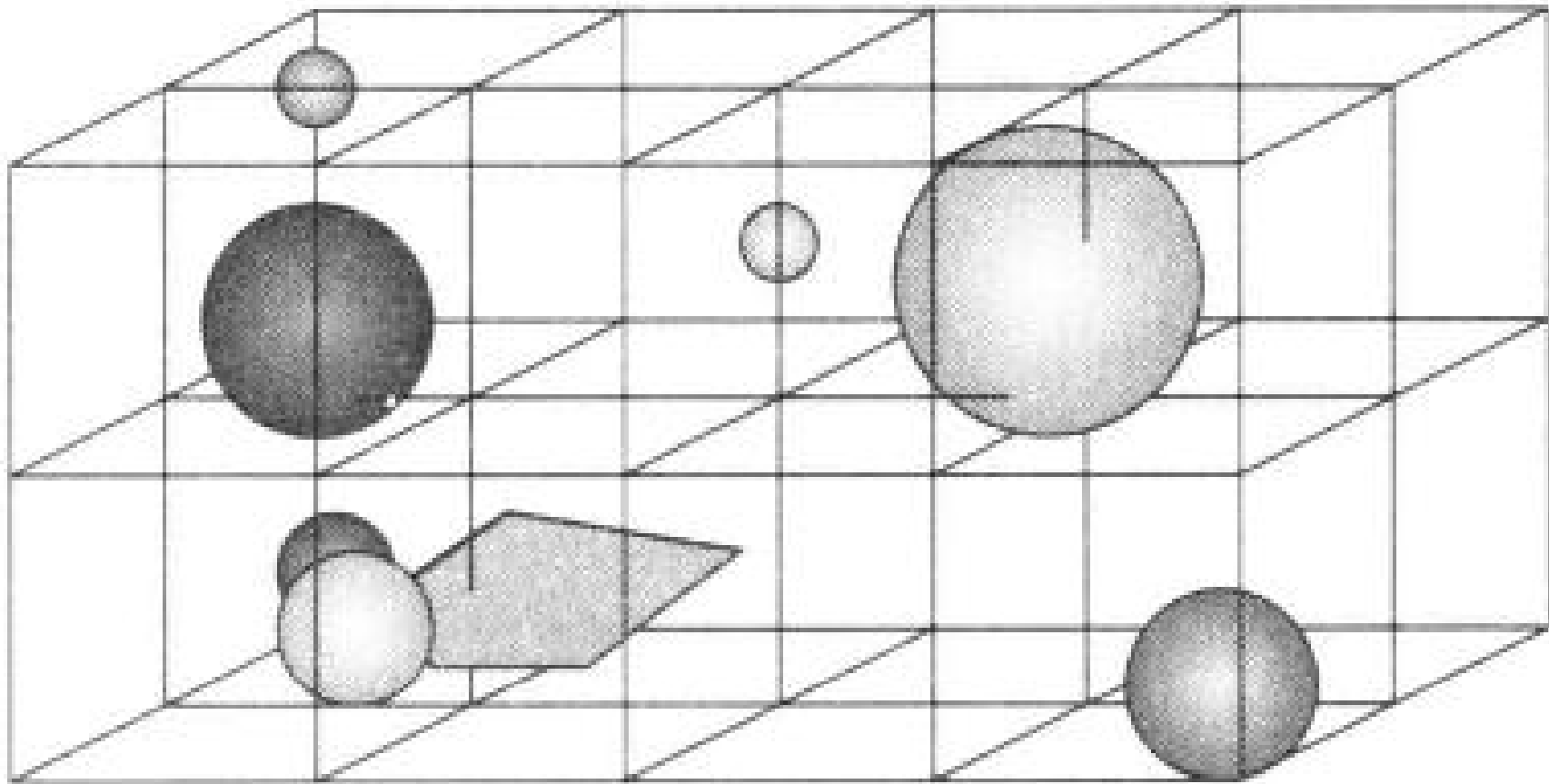
Consideraciones de eficiencia: Jerarquía



Se interseca el rayo con la raíz. Sólo si da positivo se sigue con los hijos.

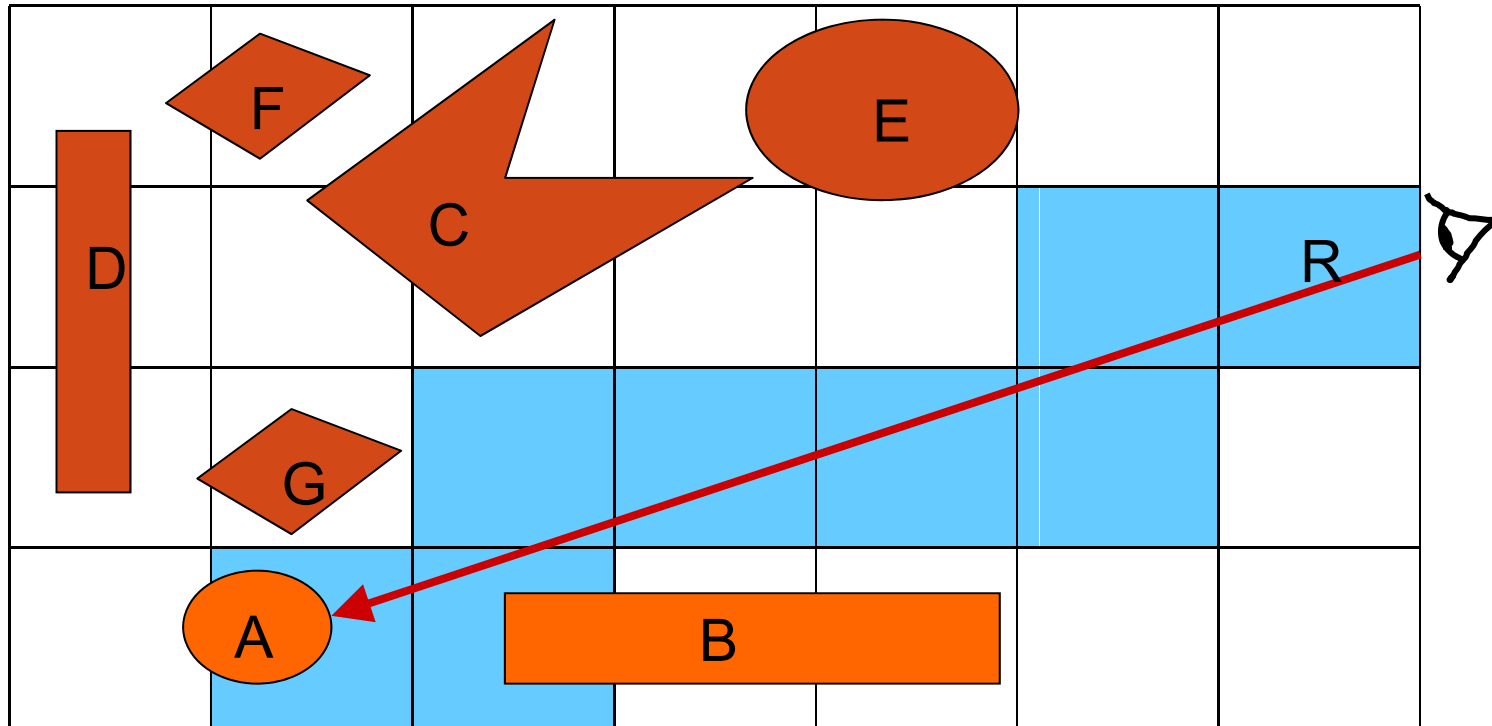


Partición Espacial



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

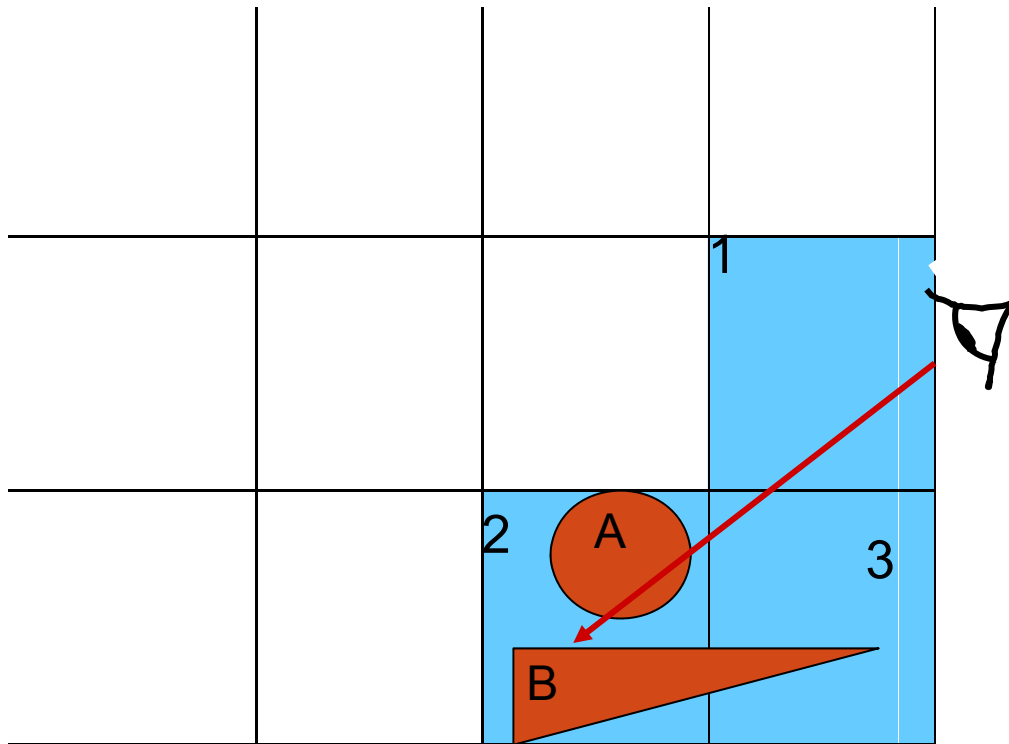
Partición Espacial



Se ve qué celdas son atravesadas por el rayo, y sólo se hace control de intersección del rayo con aquellos objetos que estén contenidos en las celdas consideradas.

Se recorren las celdas en orden creciente de distancia.

Consideraciones de eficiencia: Partición Espacial



Primero se recorre la celda 1 (no hay objeto).

Luego la celda 3. Solamente está el objeto B que interseca al rayo, pero fuera de la celda, por lo cual no se considera.

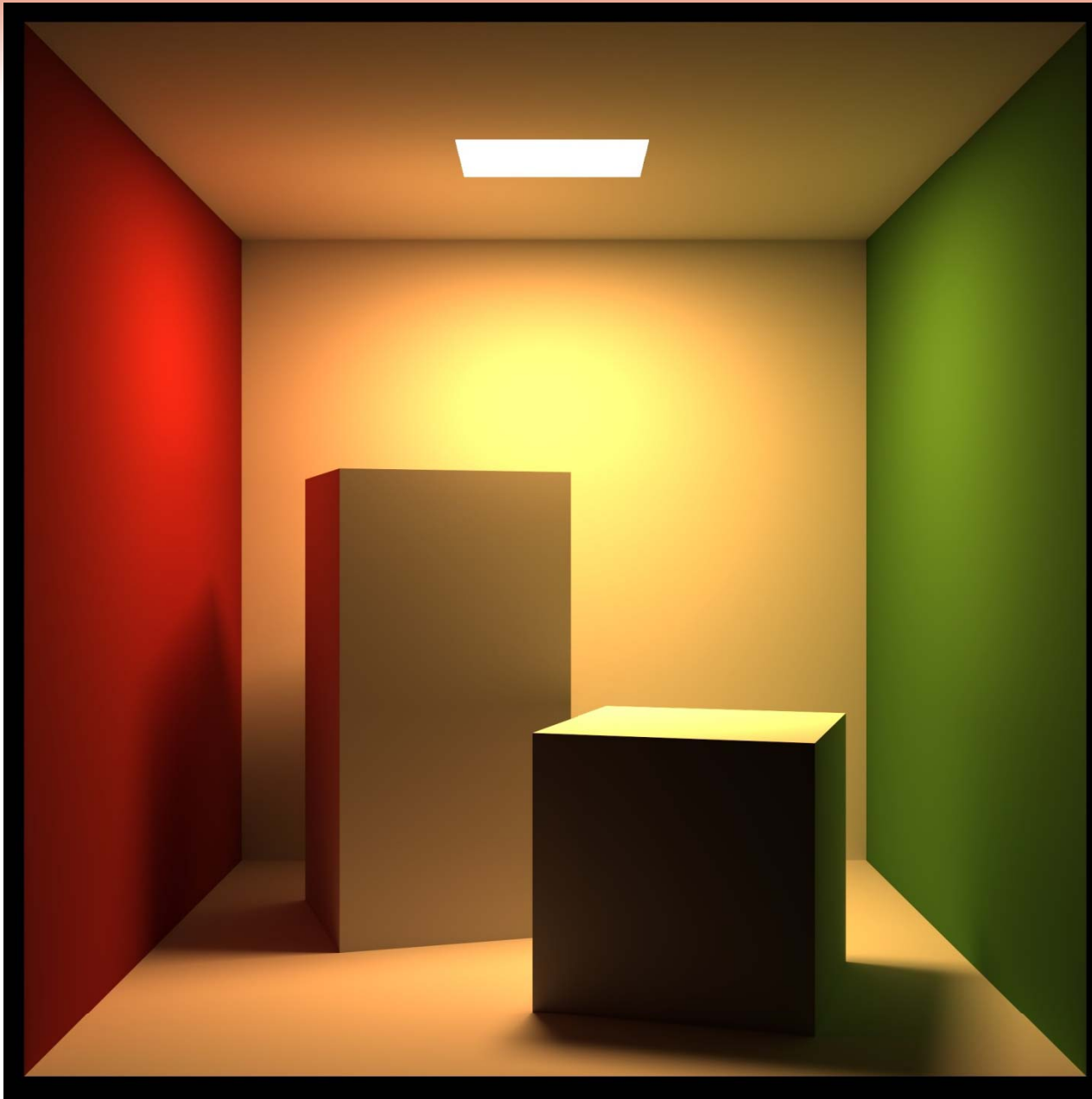
Luego la celda 2. Están los objetos A y B, ambos intersecan al rayo pero A es el más cercano.

Ray-tracing



http://www.lightandmatter.com/html_books/5op/ch01/figs/computer-

Ray-tracing



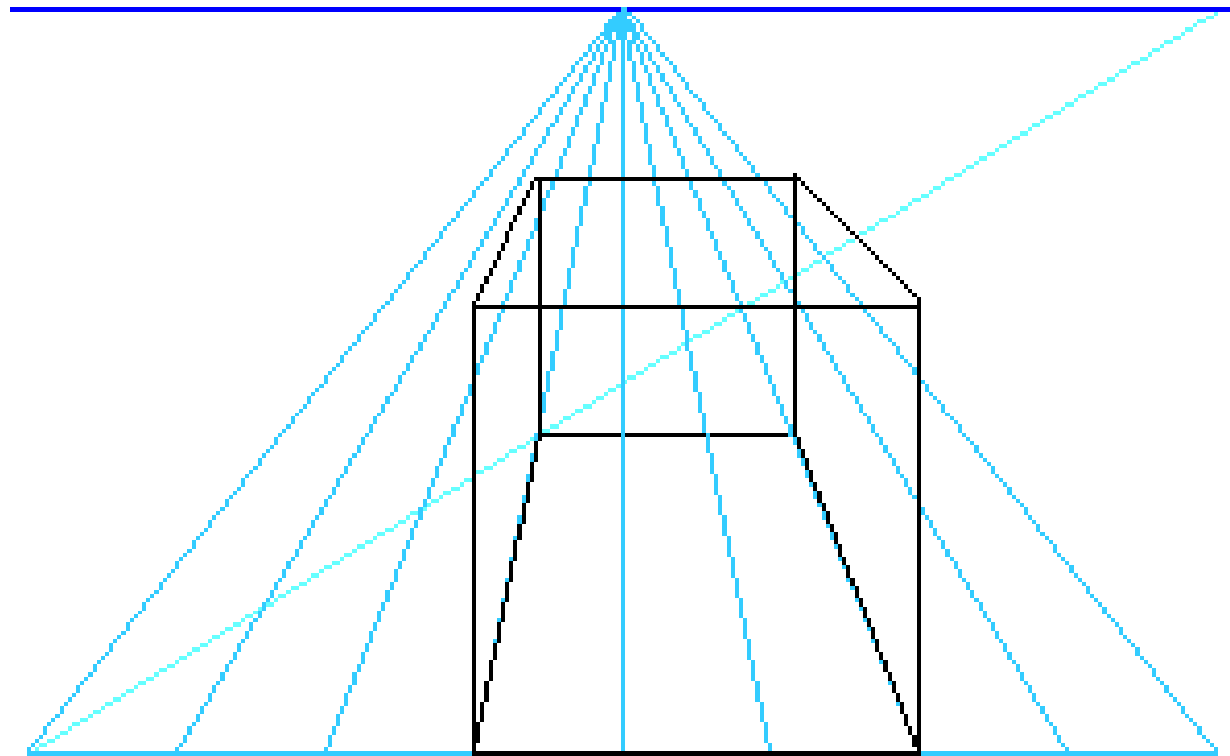
<http://www.cs.utah.edu/~sparker/images/seurat/bigcbox.jpg>



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

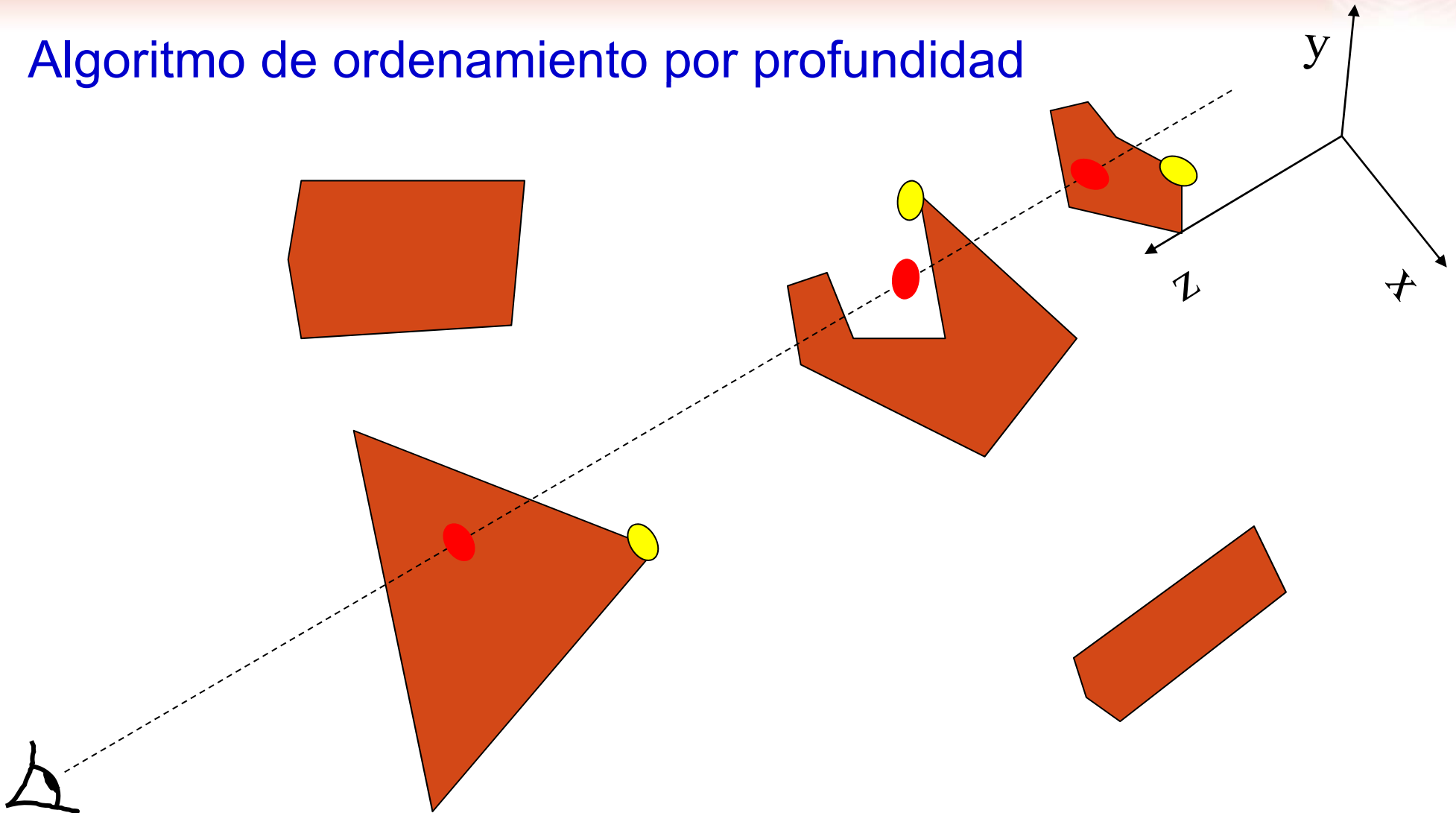
Painter's Algorithm



University of Utah, Dpto de Matemáticas, Prof. Andrejs Treibergs
<http://www.math.utah.edu/~treiberg/Perspect/Perspect.htm>



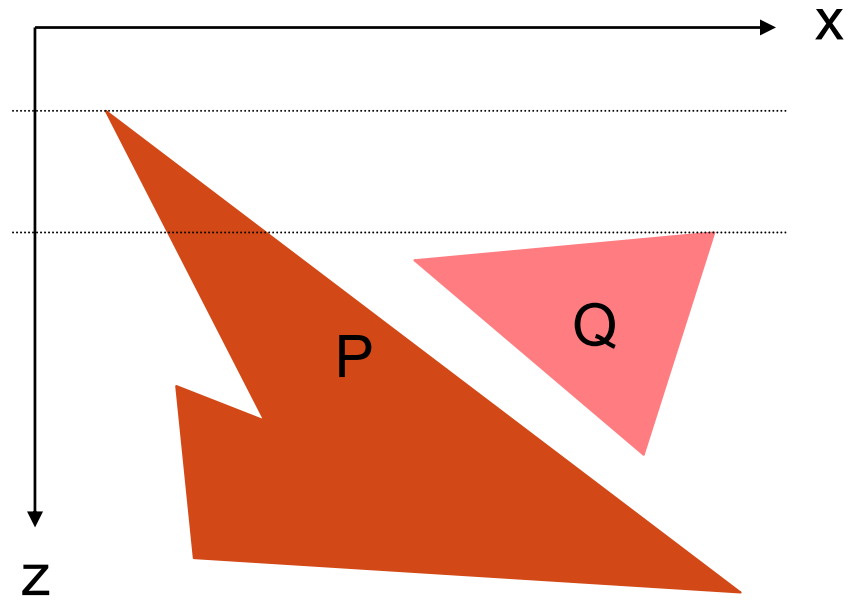
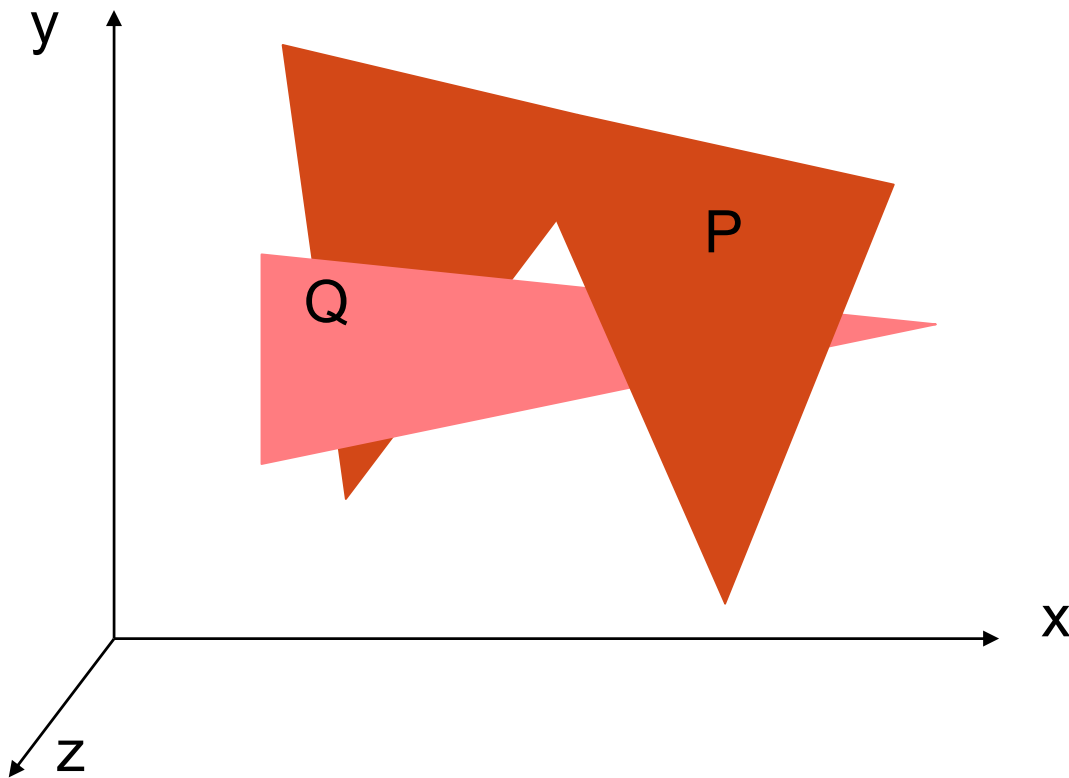
Algoritmo de ordenamiento por profundidad



¿En qué orden se deben dibujar los polígonos?

● 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 sobrepongan. Para cada Q hago 5 preguntas en orden creciente de complejidad.

Si alguna pregunta es verdadera $\forall Q \Rightarrow$

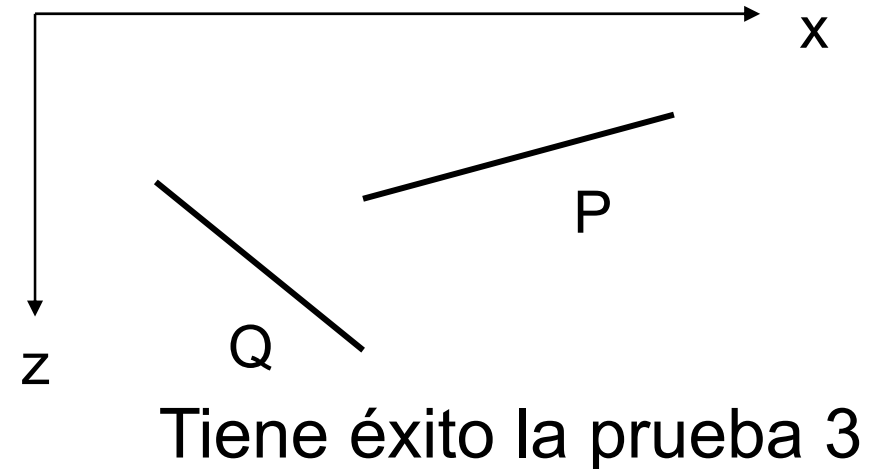
P no se superpone con $\{Q\}$

- 1.- ¿No se sobreponen las extensiones x de los polígonos?
- 2.- ¿No se sobreponen 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 sobreponen las proyecciones de los polígonos en el plano (x,y) ?

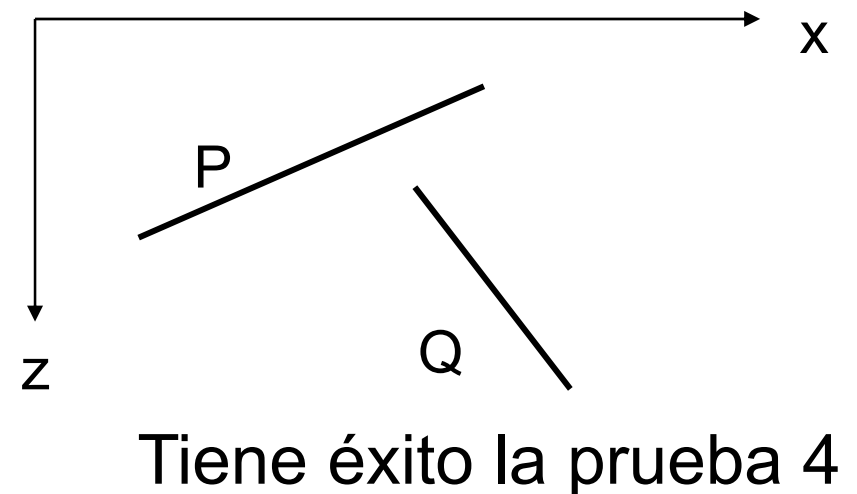
Algoritmos de prioridad de listas



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 ?



Algoritmos de prioridad de listas

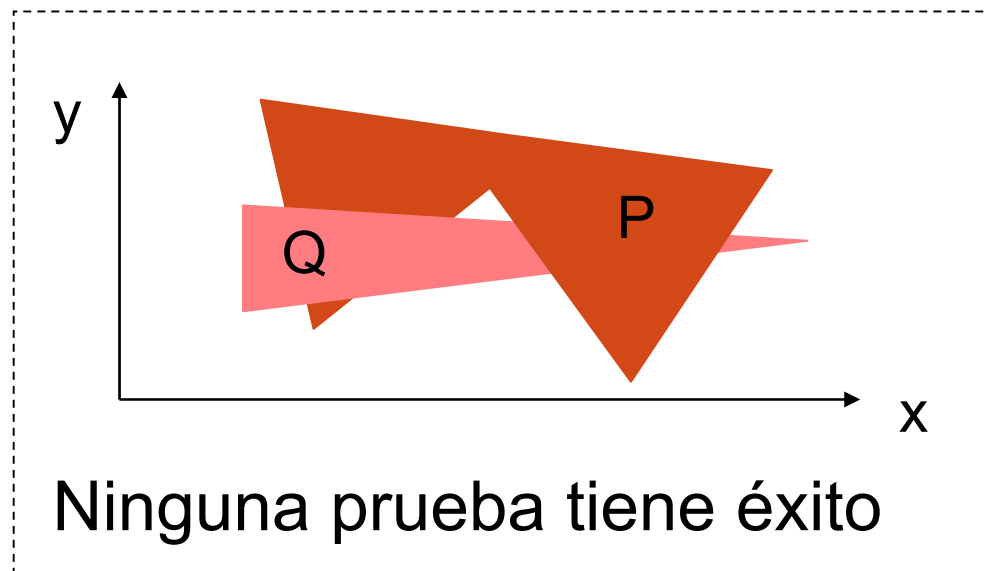
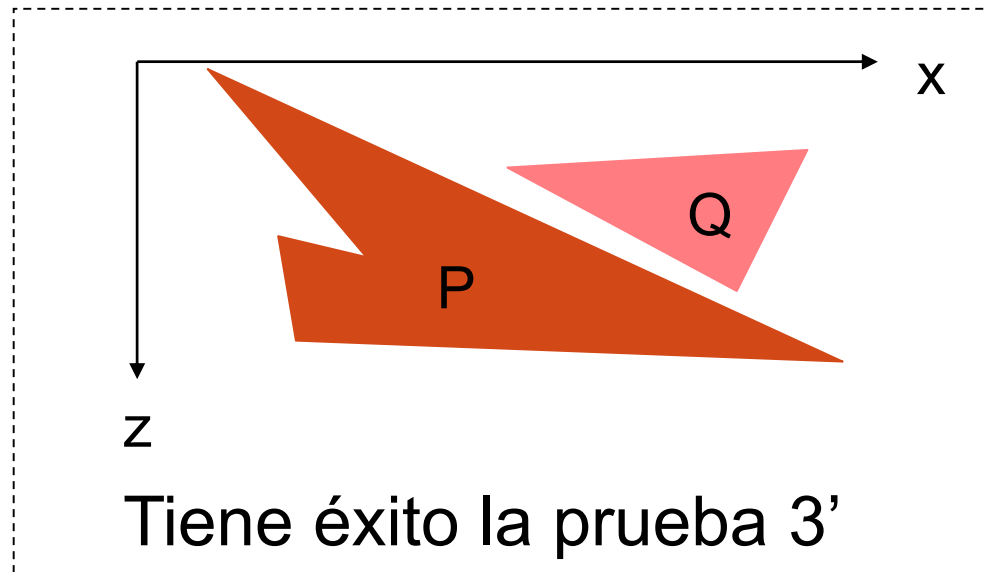


En el caso en que las 5 pruebas anteriores fracasen, se realizan dos pruebas extras, que en el caso de que alguna sea verdadera, se debe discretizar Q antes que P .

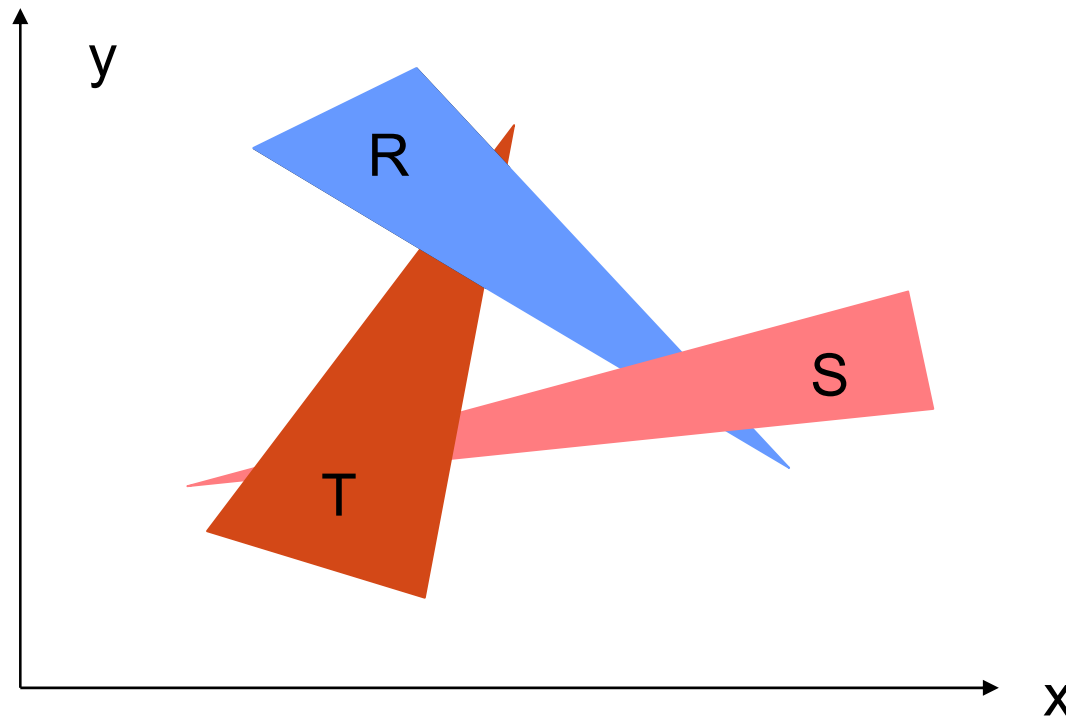
- 3' ¿Están Q (por completo) y el punto de observación en distintos semiespacios del plano de P ?
- 4' ¿Están P (por completo) y el punto de observación en el mismo semiespacio del plano de Q ?

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



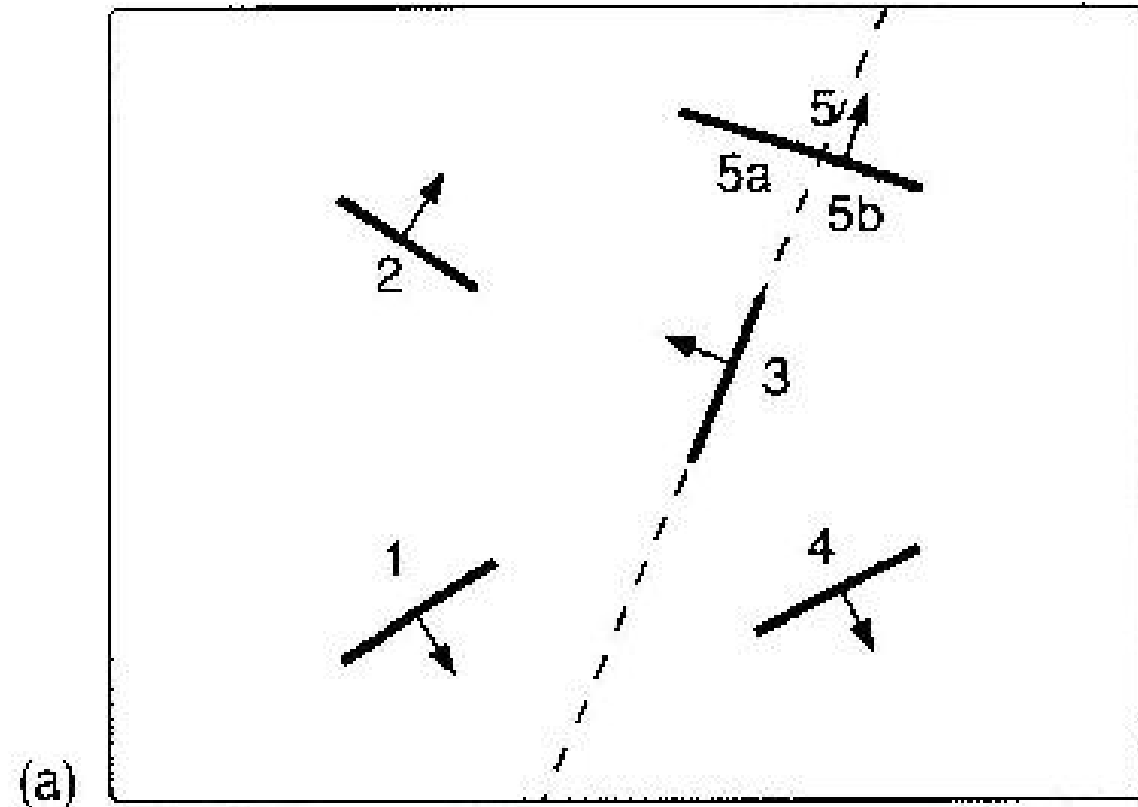
Aquí la única posibilidad es subdividir alguno de los polígonos.



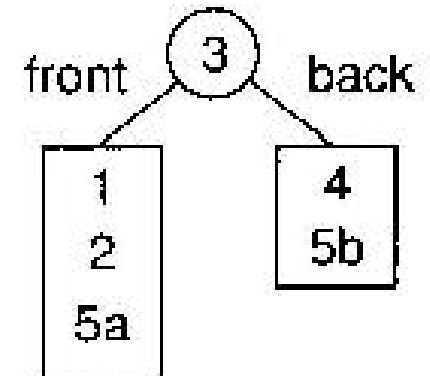
Caso que genera ciclos.

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)



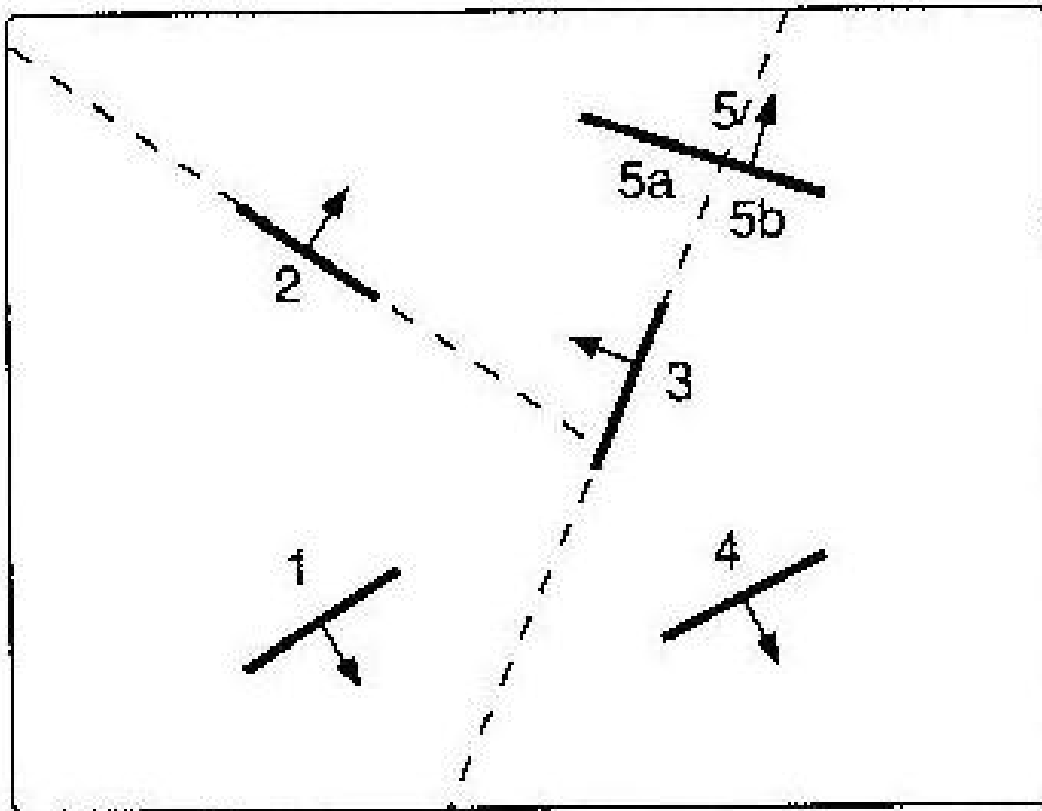
Algoritmo utilizado en el videojuego Doom.



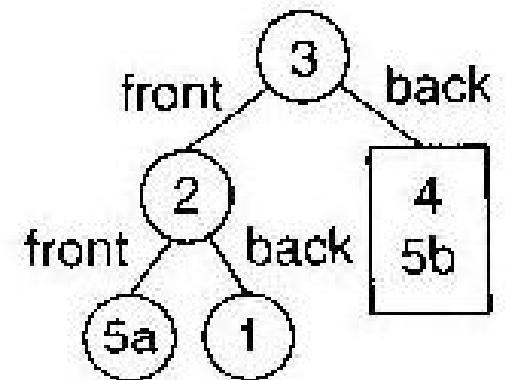
Aquí se tienen 5 objetos. Se toma uno como raíz y se subdivide el espacio en dos (front y back). Paralelamente se genera una estructura de árbol donde se especifican qué objetos están en qué semiespacio.

Legendary Doom



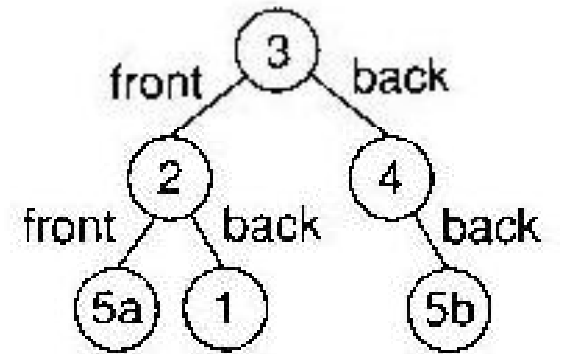
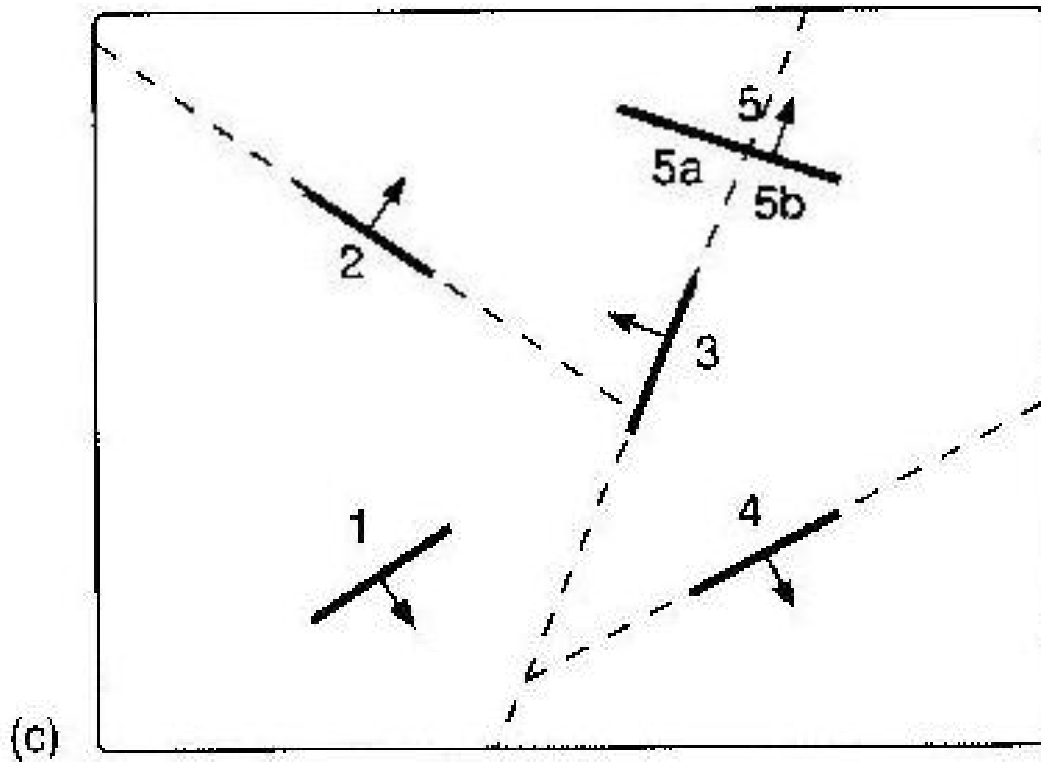


(b)



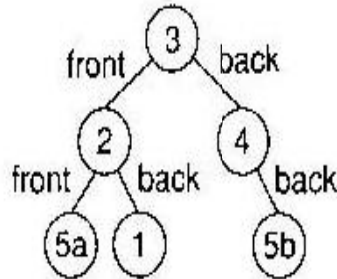
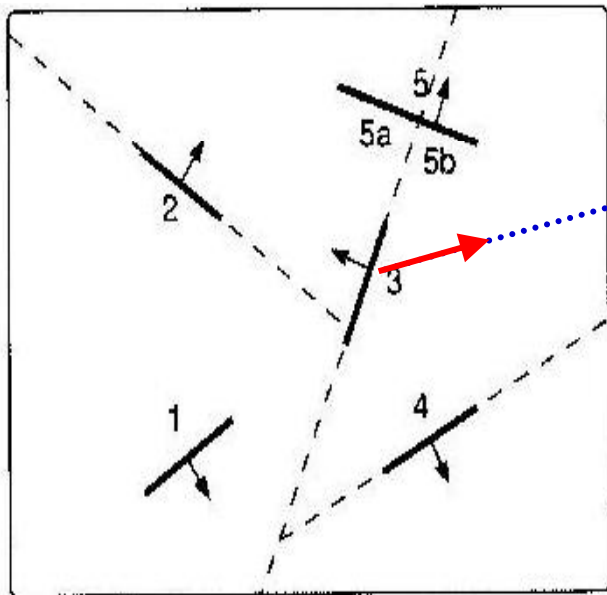
Luego, en uno de los semiespacios se hace la misma tarea. Se toma un elemento raíz y se subdivide el semiespacio en dos. Paralelamente se construye la estructura de árbol. Se continua hasta que haya un objeto por nodo.

Árboles binarios para partición del espacio (BSP tree)



Posible subdivisión espacial y árbol final.

Árboles binarios para partición del espacio (BSP tree)

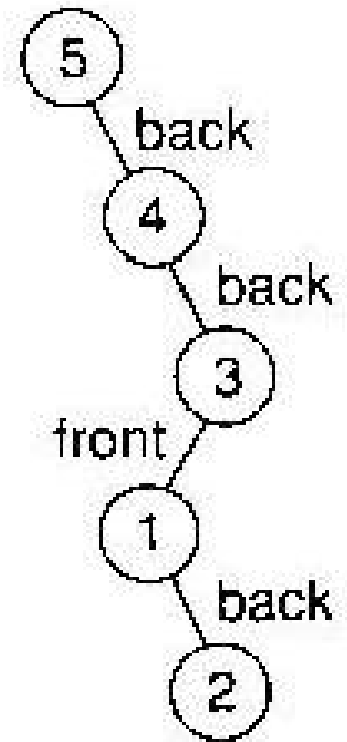
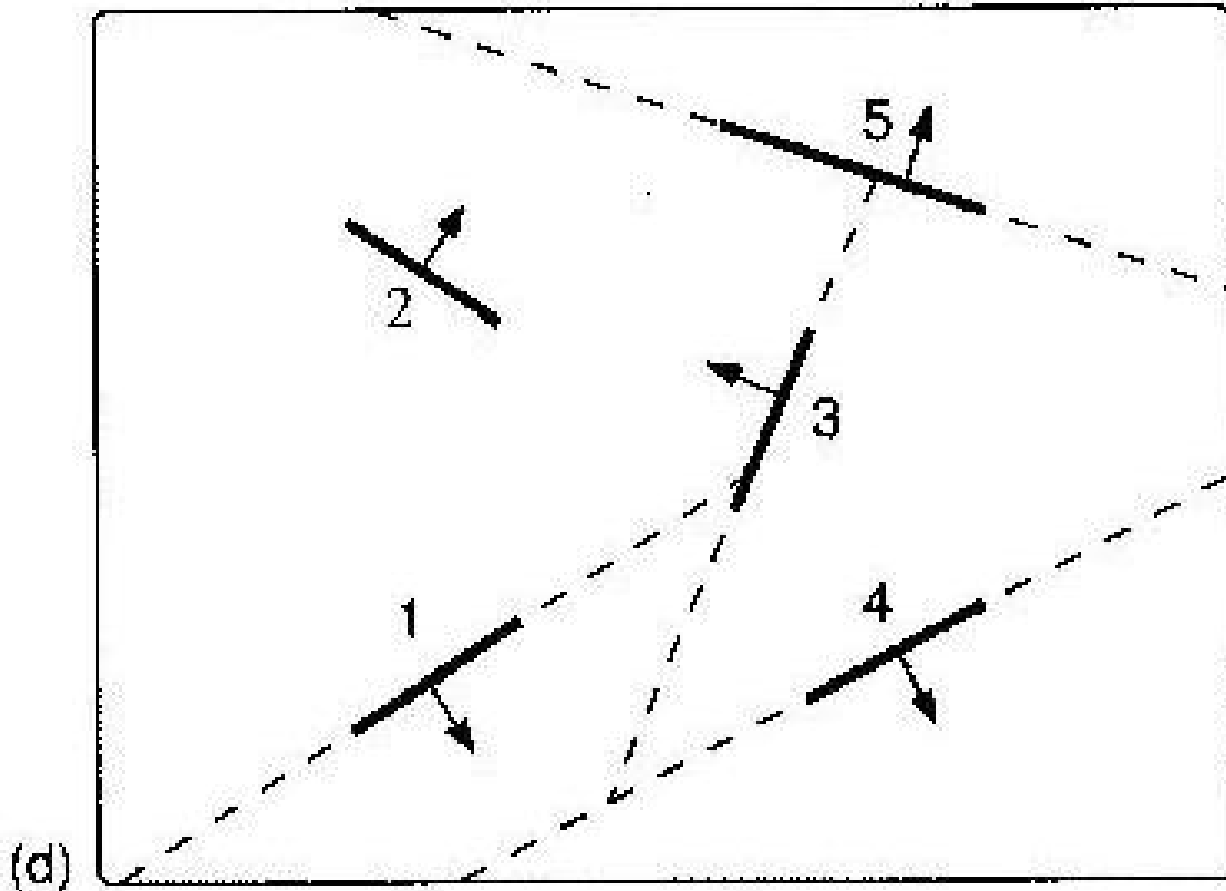


Luego de construido el árbol, lo que interesa es que rápidamente se puedan dibujar los polígonos en el orden correcto según la posición del observador.

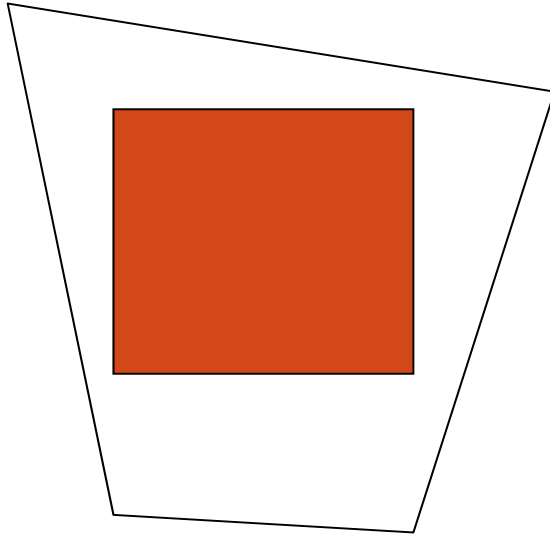
Se recorre el árbol recursivamente. Se comienza por la raíz. Como el observador está detrás del polígono 3 (observe la normal) entonces el orden correcto es: recorrer primero el subárbol front, luego pintar el polígono raíz y luego recorrer el subárbol back.

Siga el algoritmo y verá que se pintan todos los polígonos en el orden correcto.

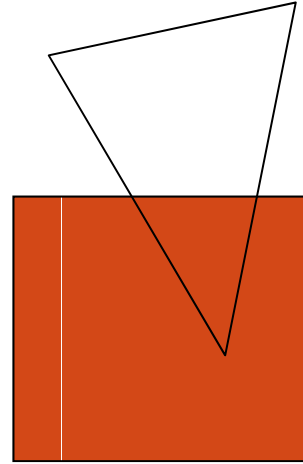
El algoritmo es de orden n , luego de hallado el árbol.



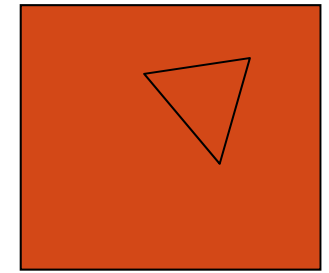
Otra posible posible subdivisión espacial y árbol final.



Pol. Circundante

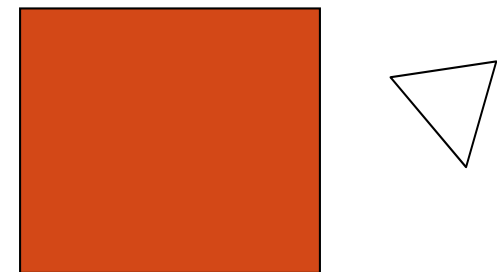


Pol. Intersecante



Pol. Contenido

Existen estas 4 posibles relaciones entre *Polígono* y *Elemento de Área* (o *trozo de ventana*)



Pol. Disjunto

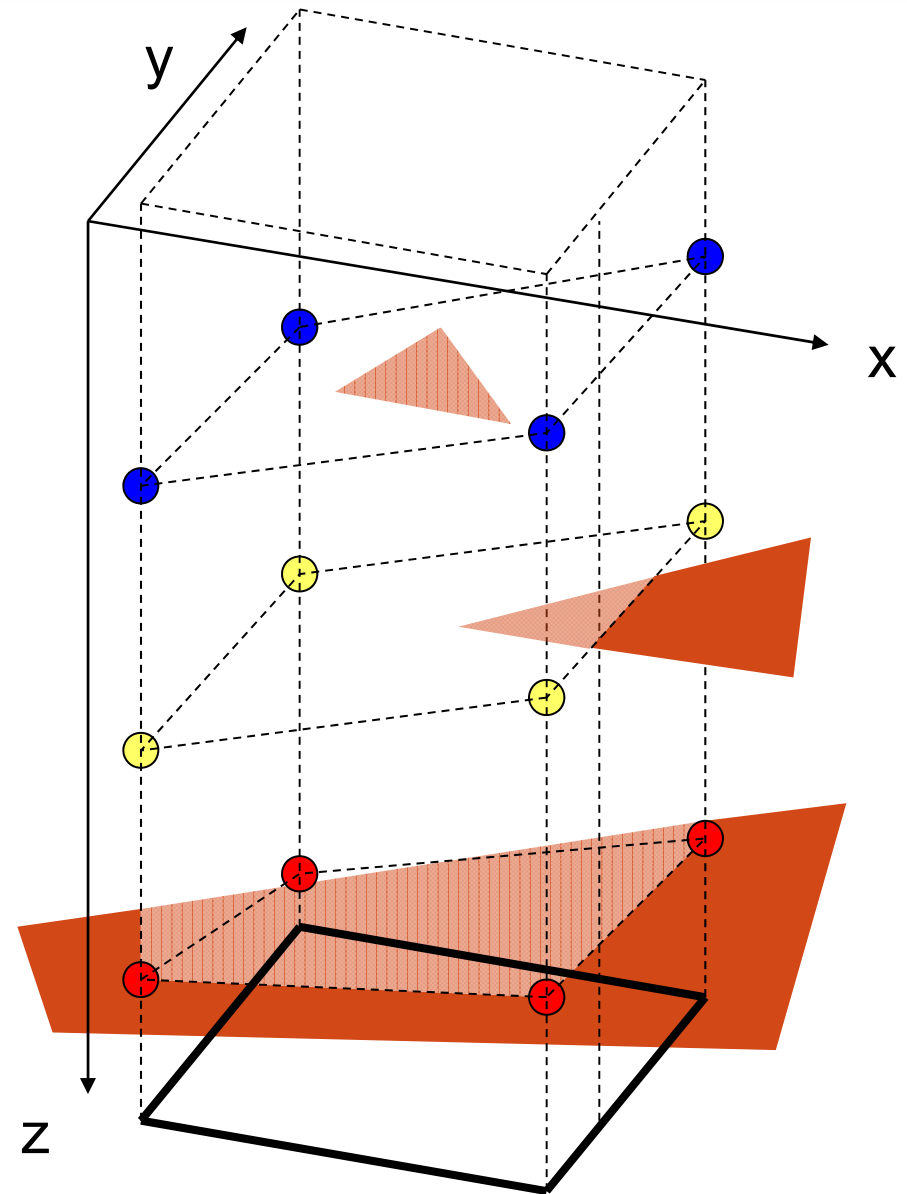
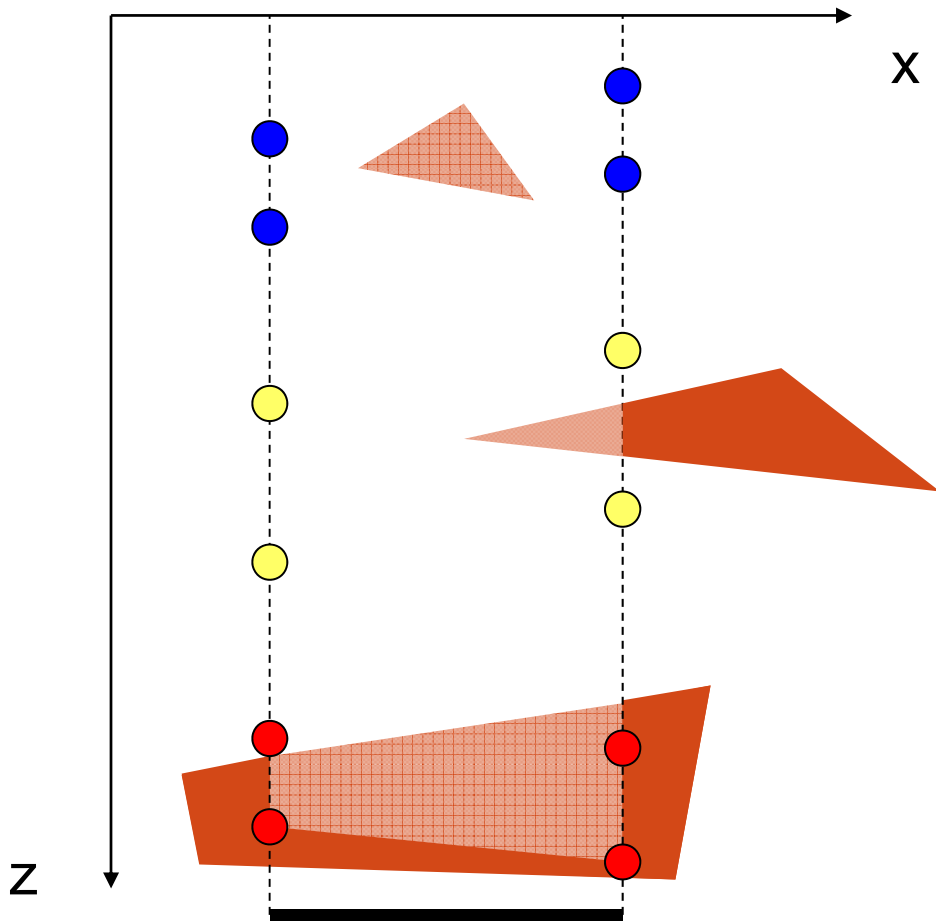
Algoritmo de Warnock



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

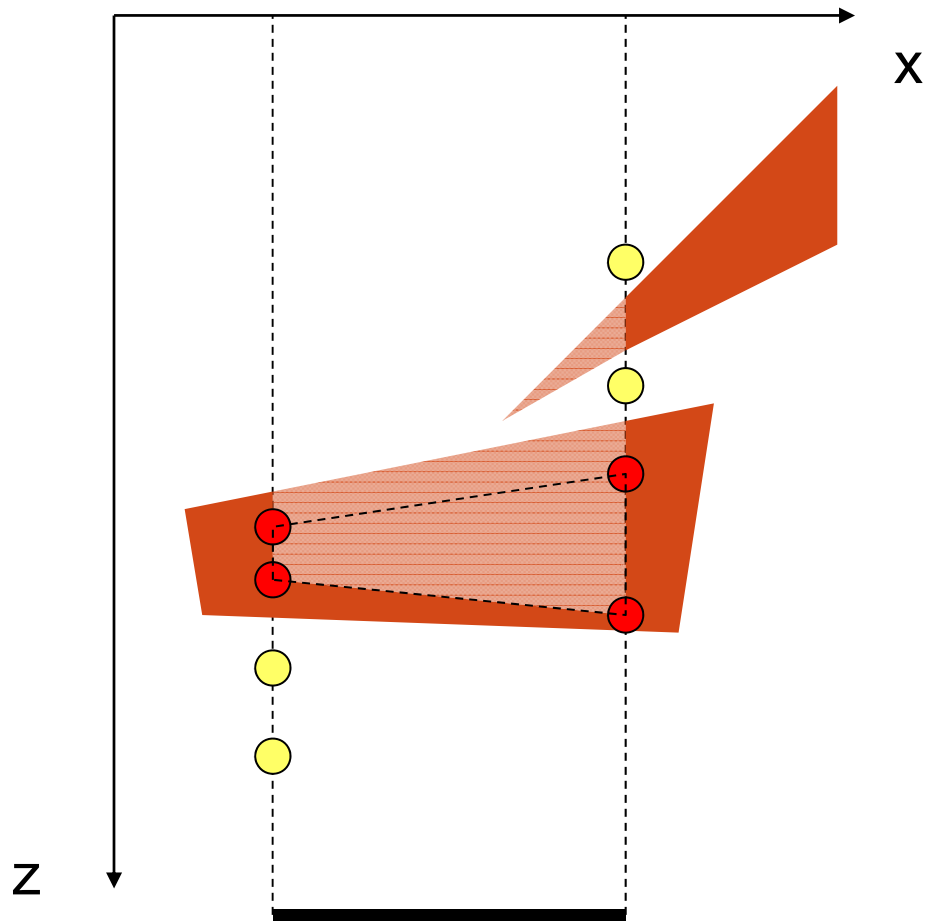
- 1.- Todos los polígonos son disjuntos respecto al área. Con el color de fondo se puede pintar el área.
- 2.- Sólo hay un polígono intersecante o contenido.
- 3.- Hay un solo polígono que interseca al área y es circundante.
- 4.- Hay varios polígonos intersecantes, uno de ellos es circundante y está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

Algoritmo de Warnock



Aquí se muestran los puntos de intersección de cada polígono respecto a los 4 vértices del área proyectada.

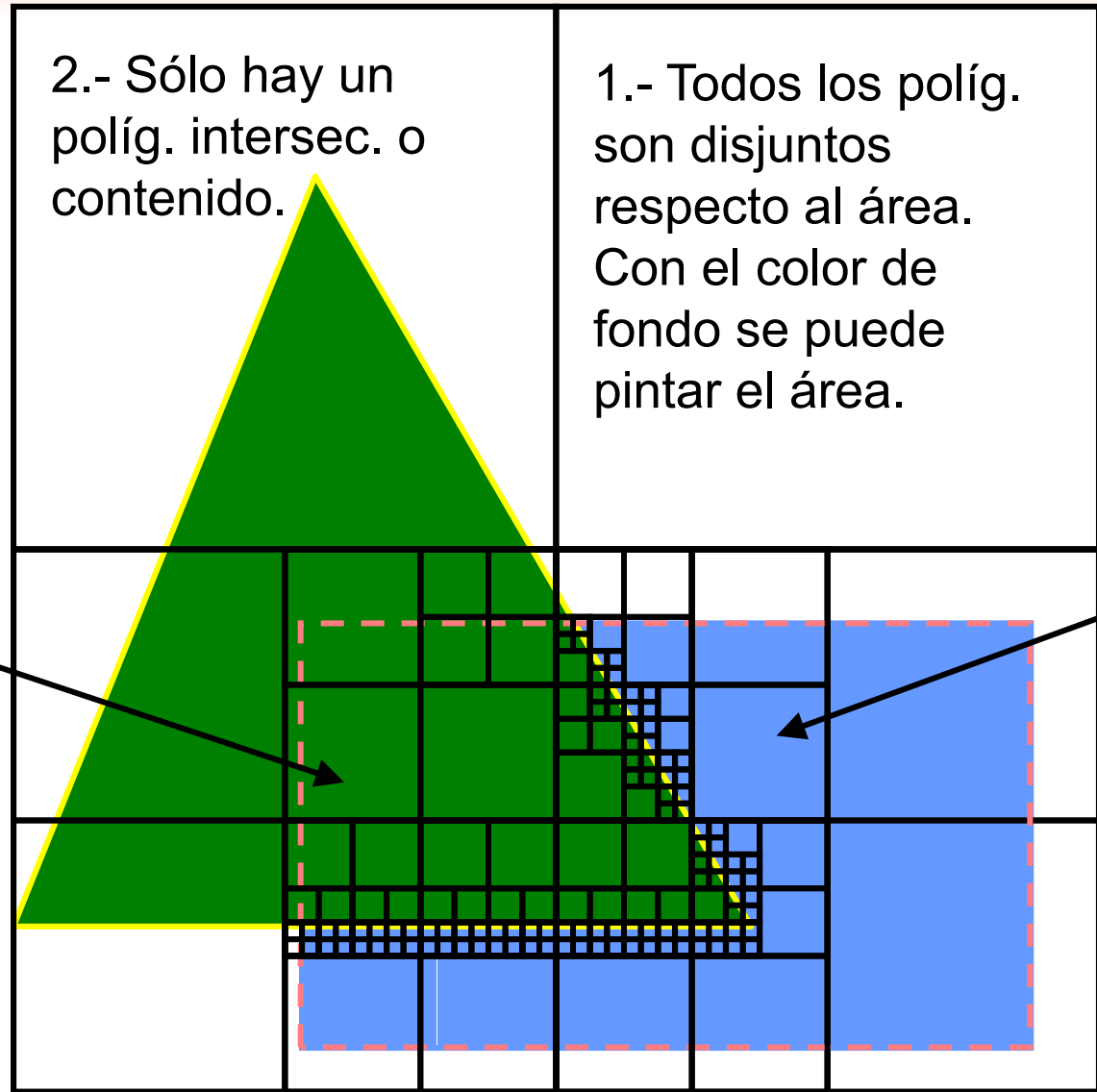
Algoritmo de Warnock



En este caso se subdivide el área.

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

Algoritmo de Warnock



2.- Sólo hay un políg. intersec. o contenido.

1.- Todos los políg. son disjuntos respecto al área. Con el color de fondo se puede pintar el área.

3.- Hay un solo políg. que interseca al área y es circundante.

4.- Hay varios políg. intersec., uno de ellos es circundante y está enfrente de los demás políg. Esto se controla con los valores z de los políg. en los 4 vértices del área.