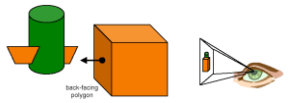
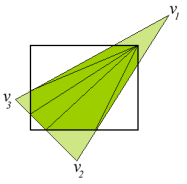
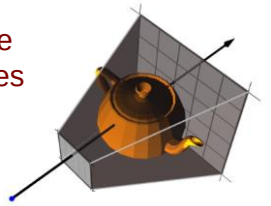


Determinación de Superficies Visibles





Lab. de Visualización y Computación Gráfica
Dpto. de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Introducción

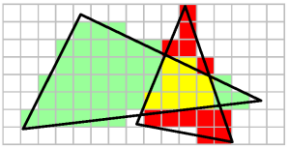
Pantalla 3D

C. 3D de Pantalla

Cara Oculta
Rasterización

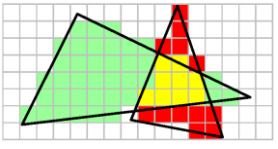
Ventana

C. Ventana



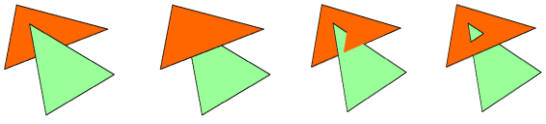
Introducción

Cada pixel tiene un determinado lugar en el frame buffer.



¿Qué triángulo gana?

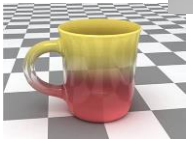
¿O cuál gana parcialmente?

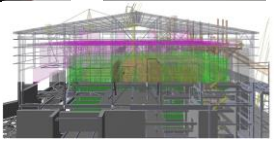


Introducción

La idea es mantener las superficies visibles.

- Típicamente sólo vemos la superficie que está más cerca del ojo
- Excepción: transparencia



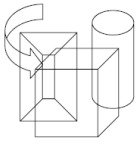


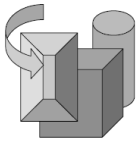
Introducción

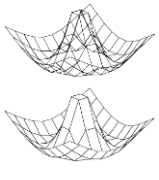
Definición

Dado un conjunto de objetos 3D y una especificación del sistema de vista se quiere determinar qué líneas o superficies de los objetos son visibles.

Una superficie puede estar ocluida por otros objetos o por sí misma (auto oclusión).



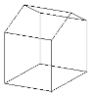


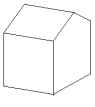


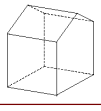
Introducción

Nota Histórica

El problema se presentó por primera vez para el rendering de wireframes. La solución se denominó *eliminación de la línea oculta*.







- Las líneas mismas no ocultan superficies. Las líneas pueden ser aristas de superficies opacas que tapan otras superficies
- Algunas técnicas muestran segmentos como punteados.

Introducción

Para la determinación de superficies visibles hay 3 tipos de algoritmos:

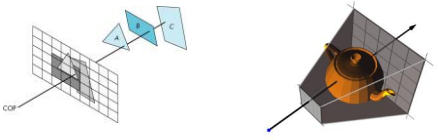
- *Algoritmos simples* que testean visibilidad (culling): determinan las caras que no pueden verse desde el observador
  - – Los resultados deben someterse a alguno de los algoritmos mencionados a continuación. Por ej. back-face culling, clipping del volumen de vista canónico.
- *Precisión de la Imagen*: resuelven la visibilidad en puntos discretos de la imagen
  - – Muestrean el modelo y luego resuelven la visibilidad. Por ej. raytracing o Z-buffer y los scan-line con buffers de profundidad .
- *Precisión del Objeto*: resuelven la visibilidad para todas las posibles direcciones desde un determinado punto de vista
  - – En primera instancia resuelven la visibilidad y luego muestrean los resultados. Son independientes de la dirección de vista o de la densidad de muestreo. Por ej. ordenamiento 3-D en profundidad, árboles BSP.

Introducción

Propuesta en el espacio de la Imagen

Para *cada pixel en la imagen*

- Determinar el objeto más cercano al observador que es atravesado por el proyector a través del *pixel*
- Dibujar el *pixel* del color apropiado

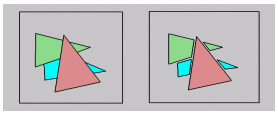


Introducción

Propuesta en el espacio del Objeto

Para *cada objeto en el mundo*

- Determinar las partes del objeto que no se obstruyen por otras partes del mismo o de cualquier otro objeto
- Dibujar esas partes del objeto del color apropiado



Introducción

¿Qué puede hacerse para resolver el problema general de determinación de CO?

- Los algoritmos requieren operaciones potencialmente costosas
  - Determinar, para un proyector y un objeto o para la proyección de dos objetos, si hay intersección y dónde.
  - Calcular, para el conjunto de objetos, el que está más cerca del observador y que por lo tanto es visible.
- Las operaciones costosas deben realizarse
  - Eficientemente.
  - La menor cantidad de veces posible.

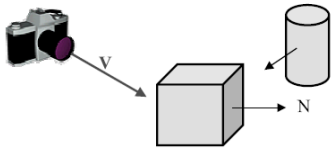
Introducción

Detección de la *cara de atrás (back face culling)*

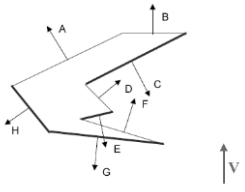
En un objeto volumétrico no se ven las *caras de atrás*.

Éstas pueden identificarse usando el signo del producto escalar VN

¿En qué coordenadas está representada N?

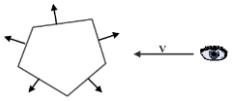


Introducción



- Polígonos cuya normal apunta hacia atrás A, B, D, F
- Polígonos cuya normal apunta hacia delante C, E, G, H

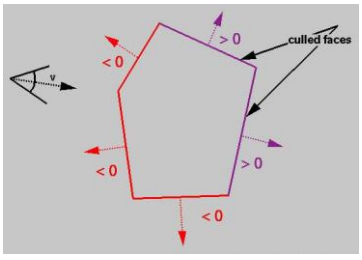
Introducción



- Tres posibilidades
  - → V.N>0 cara de atrás
  - → V.N<0 cara de adelante
  - → V.N=0 cara en la línea de vista
- En objetos convexos, la detección de la cara de atrás resuelve el problema de superficie visible
- Esta detección se aplica fácilmente a objetos poliédricos convexos
- En un objeto cualquiera la cara frontal puede ser visible, invisible o parcialmente visible.

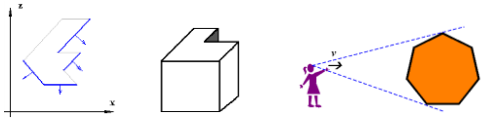
Introducción

Esta eliminación de objetos o partes de los mismos que no se ven se denomina *back face culling*



Introducción

El producto escalar de la normal a una cara y el vector desde el observador a la cara permite determinar las caras no visibles por el observador



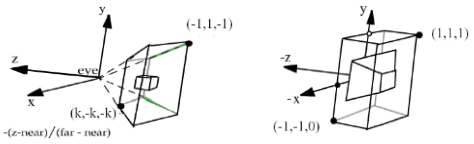
¿Es necesario este cálculo?

En OpenGL simplemente podemos habilitar el culling pero puede no trabajar correctamente sobre objetos que no son convexos.

Introducción

Transformación Perspectiva

La transformación perspectiva preserva la relación de profundidad, la planaridad de los planos y la rectitud de las líneas y realiza el *acortamiento de perspectiva*

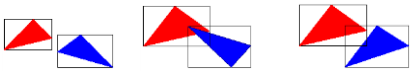


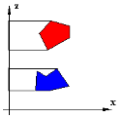
¿Cuándo debe realizarse la comparación en profundidad?

Introducción

Extensión y volúmenes limitantes

Si luego de la proyección perspectiva las extensiones no se solapan, los objetos tampoco lo hacen. Si las mismas se solapan, los objetos pueden solaparse.





Las extensiones pueden limitar los objetos mismos, son los volúmenes limitantes (bounding volumes).

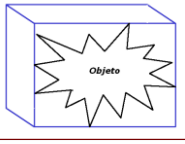
Introducción

Extensión y volúmenes limitantes

La idea de los volúmenes limitantes es limitar cada objeto complejo con objetos más simples:
- Por ej. Esferas, cubos, prismas.

Si el volumen limitante no es visible, tampoco lo será el objeto que está dentro de él.

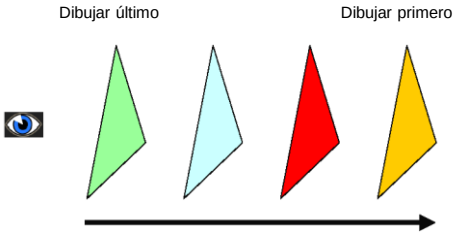
También pueden ponerse varios objetos dentro de un volumen; esto es más eficiente.



Algoritmos de Cara Oculta

Algoritmo del Pintor

Renderizar los polígonos de atrás hacia adelante de modo que los mismos se pinten unos encima de otros.



Algoritmo del Pintor

Desarrollado por Newell, Newell y Sancha (1972).

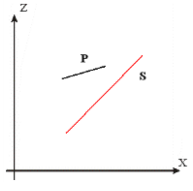
Dado que la estrategia es trabajar de atrás hacia delante, se debe encontrar una forma de ordenar los polígonos en profundidad (z), y luego dibujarlos en ese orden.

Algoritmo:

- Ordenar todos los polígonos en profundidad de acuerdo al z más lejano (la mayor o menor coordenada z de cada polígono, de acuerdo al sistema de coordenadas con que se trabaje)
- Hacer la conversión scan dibujando primero el polígono más lejano y luego trabajar hacia el punto de vista (como un pintor)

Algoritmo del Pintor

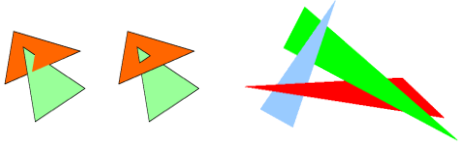
¿Hay algún problema con este algoritmo?



El algoritmo no trabaja bien en este caso.

Algoritmo del Pintor

¿En estos casos?



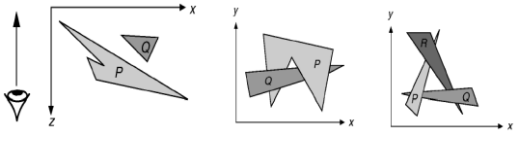
Hay problemas cuando se tienen:
- Intersecciones
- Ciclos

Se resuelve pero el costo es alto y el algoritmo no es bueno.

Además se requiere un ordenamiento.

Algoritmo Depth Sort

Maneja los errores que se producen debido al ordenamiento en z

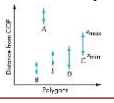


Algoritmo Depth Sort

- Ordenar todos los polígonos de acuerdo a la coordenada z más lejana (espacio del objeto)
- Resolver *cualquier ambigüedad*, partiendo los polígonos si es necesario
- Dibujar todos los polígonos de atrás hacia adelante

- Requiere que primero se ordenen los polígonos
  - Cálculo del orden de O(n log n) para el ordenamiento
  - Cada polígono no está necesariamente delante o detrás de todos los demás.

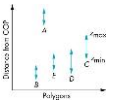
- El ordenamiento de los polígonos trabaja primero con los casos fáciles y luego con los difíciles.



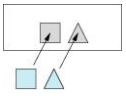
Algoritmo Depth Sort

Casos fáciles

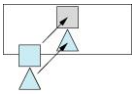
- Uno está detrás de todos los demás
  - Se puede renderizar



- Los polígonos se solapan en z pero no se solapan ni en x ni en y
  - Se pueden renderizar independientemente



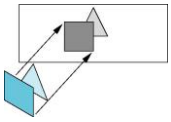
No hay solapamiento en x



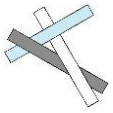
No hay solapamiento en y

Algoritmo Depth Sort

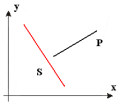
Casos difíciles

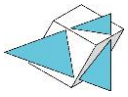


Solapamiento en todas las direcciones pero un polígono puede estar completamente de un lado del otro.



Solapamiento cíclico



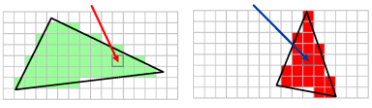


Penetración

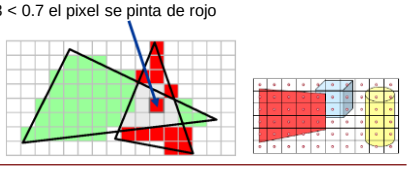
Z-Buffer

El buffer de Profundidad: Algoritmo Z-Buffer

z = 0.7 z = 0.3



Como 0.3 < 0.7 el pixel se pinta de rojo



Z-Buffer

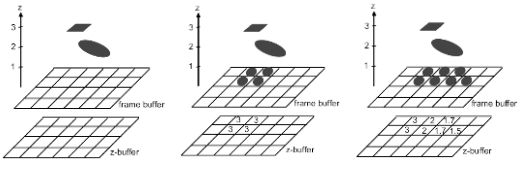
El z-buffer es un buffer 2D del mismo tamaño que la imagen

1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.1	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.1	0.1	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.2	0.2	0.3	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.3	0.3	0.4	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.3	0.4	0.4	0.5	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.4	0.4	0.5	0.5	0.5	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.4	0.5	1.0	1.0	1.0

Z-Buffer

Desarrollado por Catmull (1974). Es uno de los más simples para implementar tanto en software como en hardware.

El *frame buffer* almacena los colores. Se inicializa con el color del fondo. Se mantiene un *Z-buffer* de la misma resolución que el *frame buffer* para mantener la profundidad de cada pixel. Se inicializa con el valor z correspondiente al plano lejano *(far)*.



Z-Buffer

Algoritmo *Z-Buffer*

Para cada polígono p

Para cada pixel(x, y) en la proyección de p

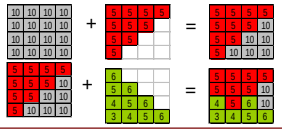
$z_p \leftarrow$ coordenada z de p en (x, y)

si $z_p < z_{buffer}$ (x, y)

entonces {z_p está más cerca}

setearPixel (x, y, color) en *frame buffer*

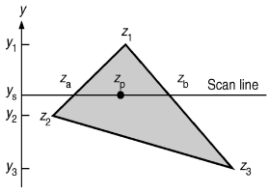
$z_{buffer}(x, y) \leftarrow z_p$



Z-Buffer

¿Cómo calcular el color y el valor de z en cada punto del polígono?

Por una cuestión de velocidad, la pseudo profundidad z_p podemos calcularla por interpolación.



$$z_a = z_1 - (z_1 - z_2)\frac{y_1 - y_s}{y_1 - y_2}$$

$$z_b = z_1 - (z_1 - z_3)\frac{y_1 - y_s}{y_1 - y_3}$$

$$z_p = z_b - (z_b - z_a)\frac{x_b - x_p}{x_b - x_a}$$

Tanto la coordenada z de cada punto del polígono, como el color (Gouraud) podemos calcularlo incrementalmente.

Z-Buffer

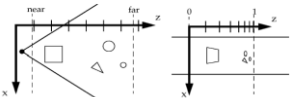
Ventajas

- Puede utilizarse para cualquier tipo de objeto si se puede determinar el valor de z y la iluminación en cada punto de la proyección; observar que también es adecuado para objetos con superficies curvas ya que encuentra la superficie más cercana basándose en un test punto por punto.
- Los polígonos no tienen que compararse en un orden predeterminado: no es necesario un ordenamiento en *z*.
- No se requiere algoritmo de intersección.
- Sólo considera un polígono a la vez aunque la oclusión es un problema global.
- Fácil de implementar. La simplicidad también se traslada al hardware y se usa en la mayoría de las estaciones de trabajo 3-D y las tarjetas gráficas, aún las más baratas.
- Los requerimientos de memoria se pueden superar si la imagen se convierte por zonas.

Z-Buffer

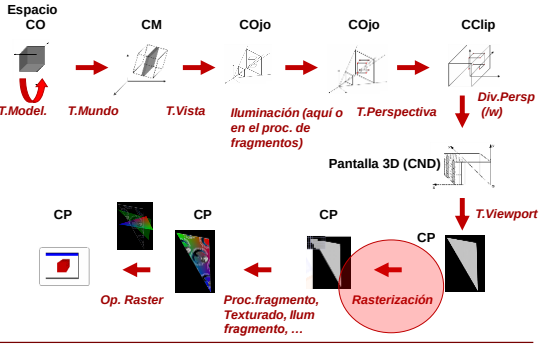
Desventajas

- Puede pintar el mismo pixel varias veces y esto es caro.
- Tiene un problema de precisión debido al acortamiento en perspectiva
  - los objetos originalmente lejos de la cámara, terminan teniendo entre sí menores diferencias en los valores en z que los que estaban cerca
  - la información en profundidad pierde precisión rápidamente, esto puede originar artefactos para objetos distantes



- No se pueden renderizar transparencias directamente.
- No se puede aplicar tratamientos de anti-aliasing directamente.
  - requiere conocer todos los polígonos involucrados en un determinado pixel

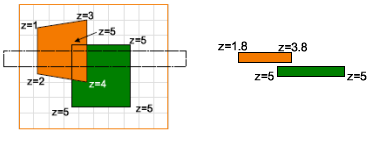
Pipeline de OpenGL



Scan Line

Es una extensión del algoritmo de rasterizado ya que, mientras se avanza a lo largo de una línea de scan, permite

- determinar las superficies que se proyectan a un pixel
- elegir la más cercana de un determinado subconjunto de éstas



Scan Line

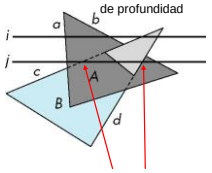
Se deben crear dos tablas:

- Una tabla de lados para todos los lados no horizontales de todos los polígonos
  - Ordenados por la menor coordenada y
  - En cada y ordenados por la menor coordenada x
- Una tabla de polígonos
  - Información de cada lado

x	$y_{máx}$	Δx	ID

  - Información de cada polígono

ID	Ec. Plano	Info sombreado	D/F



linea i: no es necesaria información de profundidad

linea j: se necesita información de profudidad porque se está dentro de más de un polígono

Scan Line

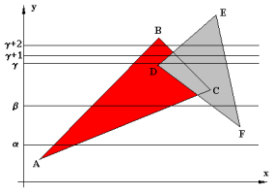
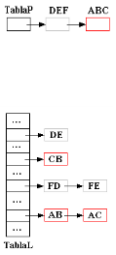
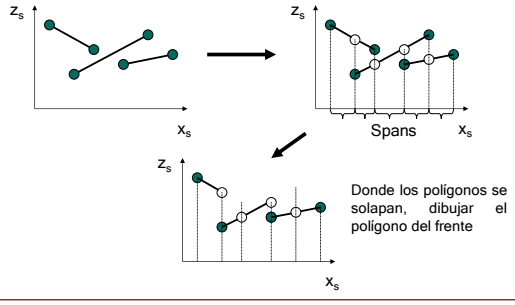


Tabla de linea de scan

Linea de scan	Entradas
α	AB AC
β	AB AC FD FE
$\gamma, \gamma+1$	AB DE CB FE
$\gamma+2$	AB CB DE FE



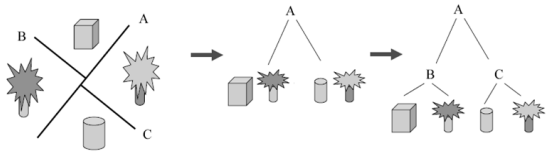
Scan Line



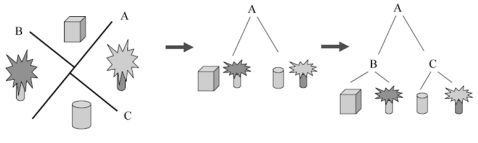
Donde los polígonos se solapan, dibujar el polígono del frente

Árboles BSP

Provee subdivisión espacial y orden de dibujado



Árboles BSP



Para mostrar correctamente cualquier polígono, mostrar todos los polígonos que estén del *lado más lejano* del polígono (relativos al punto de vista), luego ese polígono, y luego todos los polígonos que estén más cercanos.

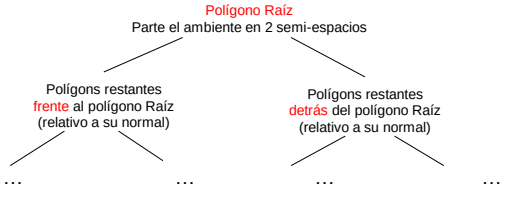
... pero, ¿cómo mostrar correctamente los polígonos que están de un determinado lado? Elegir un polígono y procesar el resto de la misma forma.

• Se debe realizar un paso de preprocesamiento independiente del punto de vista para tener bajo tiempo de ejecución cuando se cambia el punto de vista. Esto implica tiempo y espacio extras.

Árboles BSP

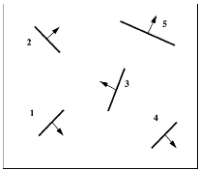
Desarrollado por Fuchs, Kedem y Naylor (1980)

Extremadamente eficiente para calcular las relaciones de visibilidad entre un grupo estático de polígonos 3D vistos desde un punto de vista arbitrario.

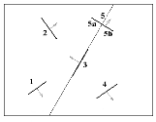


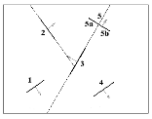
Árboles BSP

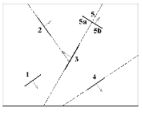
- La tarea que es independiente del punto de vista debe realizarse cada vez que la escena cambia. Esto se realiza entonces del siguiente modo:
  - subdividir recursivamente la escena en una jerarquía de semiespacios de acuerdo al plano determinado por un determinado polígono seleccionado.
  - construir un árbol BSP representando esta jerarquía
  - cada polígono seleccionado será la raíz de un subárbol
- Desarrollaremos el siguiente ejemplo:



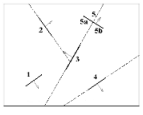
Árboles BSP



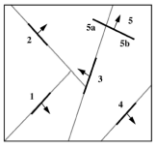


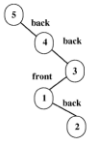


Árboles BSP

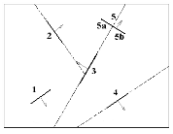


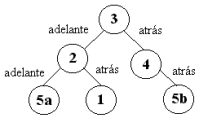
Un árbol BSP alternativo:





Árboles BSP

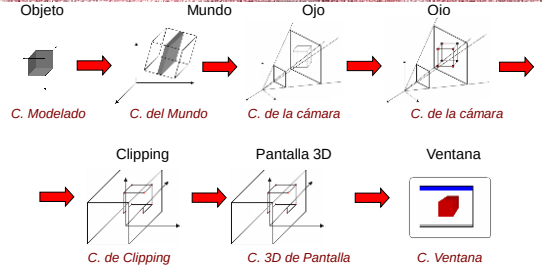




Algoritmo Mostrar_BSP(A)

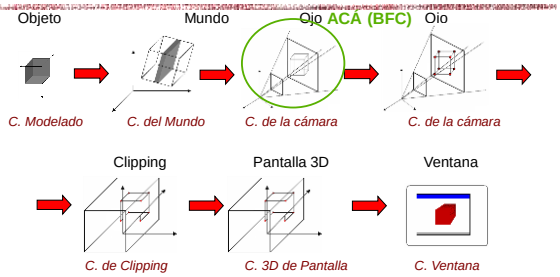
Si el árbol no está vacío
  Si el observador está frente a la raíz
  {Mostrar $H_{atrás}$, raíz, $H_{adelante}$}
    Mostrar_BSP(A^.$H_{atrás}$)
    MostrarPolígono(A ^. Raíz)
    Mostrar_BSP(A ^. $H_{adelante}$)
  sino
  {Mostrar $H_{adelante}$, raíz, $H_{atrás}$}
    Mostrar_BSP(A ^. $H_{adelante}$)
    MostrarPolígono(A ^. Raíz)
    Mostrar_BSP(A ^. $H_{atrás}$)

CO en el Pipeline 3D



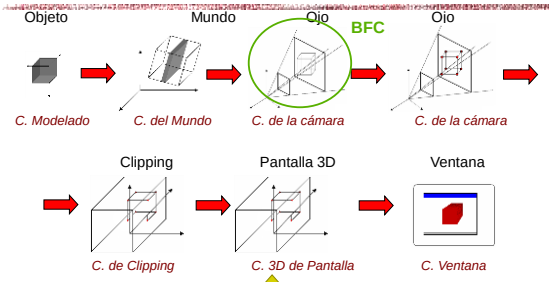
¿Dónde se realiza el culling?
¿Dónde se remueven las superficies ocultas?

CO en el Pipeline 3D



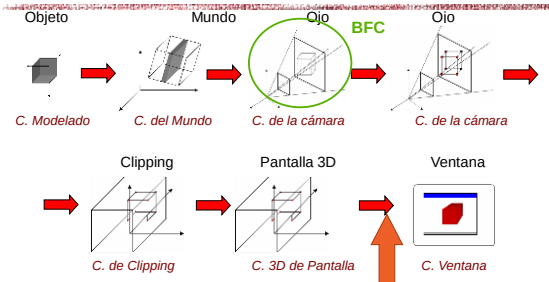
¿Dónde se realiza el culling?
¿Dónde se remueven las superficies ocultas?

CO en el Pipeline 3D



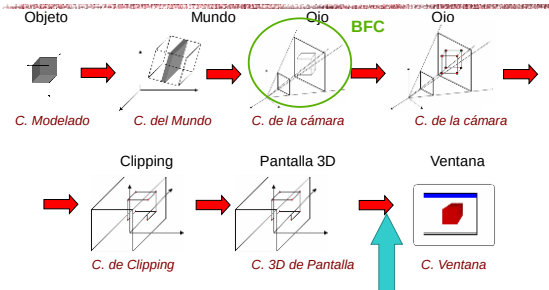
¿Dónde se remueven las superficies ocultas?
En el caso del Depth Sort

CO en el Pipeline 3D



¿Dónde se remueven las superficies ocultas?
En el caso del z-Buffer **En la rasterización**

CO en el Pipeline 3D



¿Dónde se remueven las superficies ocultas?
En el caso del Scan Line **En la rasterización**

Bibliografía

Angel, E., Shreiner, D. *Interactive Computer Graphics: A top-down approach with shader-based OpenGL*, Addison Wesley, 6th. Ed., 2011.

Buss, S., *3-D Computer Graphics, A Mathematical Introduction with OpenGL*, Cambridge University Press, New York, 2003.

Foley, J., van Dam, A., Feiner, S. y Hughes, J., *Computer Graphics. Principles and Practice*, Addison Wesley, 1992, 2nd Edition (Cap. 15)

Hearn, D., Baker, M.P., Computer Graphics, C Version, Prentice Hall Inc., 2003, 3rd Edition.

Hill, F. Jr, Kelley, S., Computer Graphics Using OpenGL, Prentice Hall, 3rd Ed., 2006.

Watt, A., *3D Computer Graphics,* Addison-Wesley Publishing Company, 1999.

Sutherland, I., Sproull, R., Schumacker, R., *A Characterization of Ten Hidden-Surface Algorithms*, ACM Computing Surveys, Vol. 6, No. 1, March 1974.

Shreiner, D., The Khronos OpenGL ARB Working Group, *OpenGL Programming Guide: The Official Guide to Learning OpenGL,* Versions 3.0 and 3.1 7th Edition, 2009.