Frustum Culling

fundamentos en computación gráfica

esmitt ramírez j.

esmitt.ramirez@ciens.ucv.ve

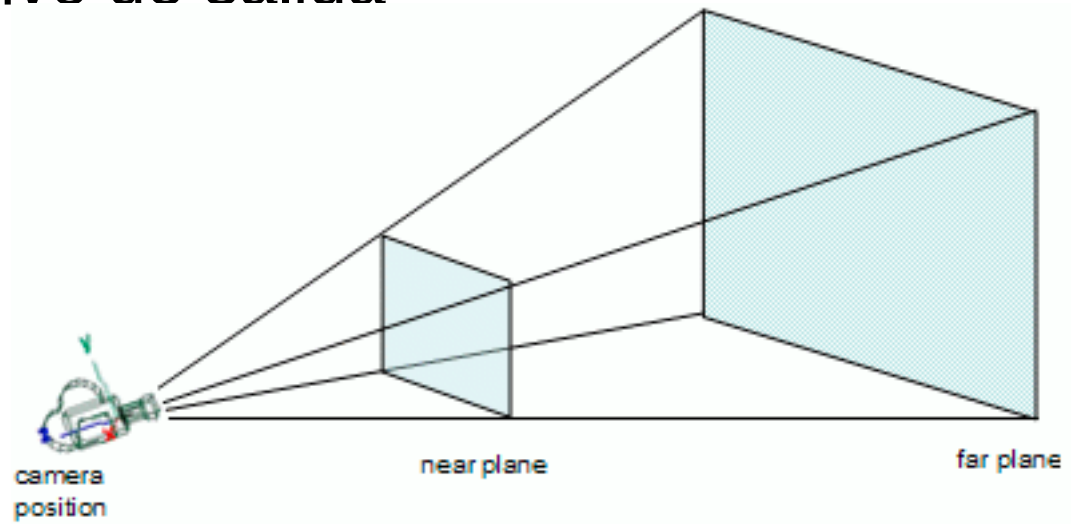
marzo 2009

Puntos a tratar

- Introducción
- Estructura del *Frustum*
- *Frustum Culling*
- ¿Cómo obtener el *Frustum volume*? *(desde el punto de vista geométrico)*
- Implementación

Introducción

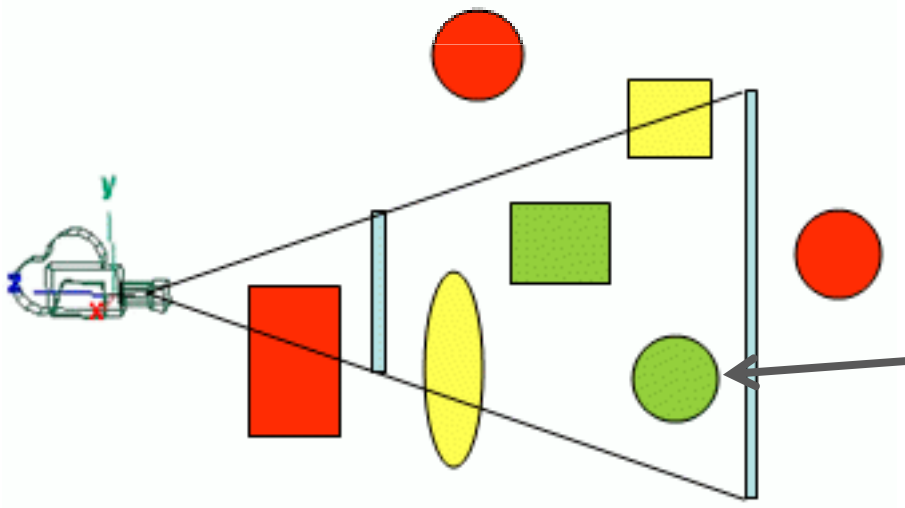
- Una cámara dentro de una escena generalmente emplea: gluPerspective y gluLookAt
- Objetivo: Determinar que es visible o no
- El *View Frustum* es el volumen que contiene TODO objeto que sea potencialmente visible en el dispositivo de salida



Frustum

- Existen objetos:
  1. Visibles
  2. Parcialmente visibles
  3. No visibles

La idea es identificar los objetos del tipo 1 y tipo 2



No debe ser renderizado
¿Por qué?

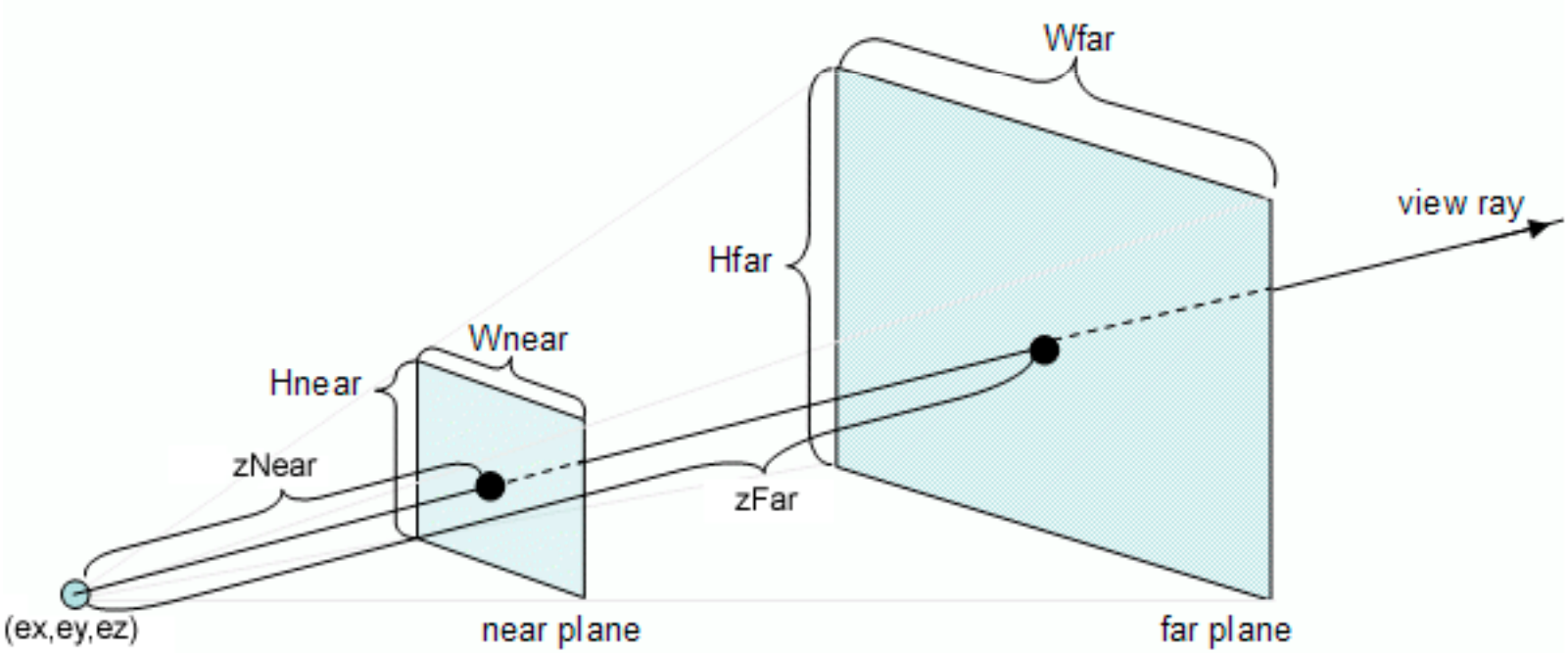
En ocasiones, el *Frustum Culling* no es necesario implementarlo

Estructura del *Frustum*

- Asumiremos que la proyección es perspectiva y la cámara emplea *gluLookAt,* tenemos entonces:
  - gluPerspective(fov, ratio, zNear, zFar)
  - gluLookAt(ex,ey,ez,ax,ay,az,ux,uy,uz)
- La “punta” de la pirámide es la posición del ojo definida por e = (ex,ey,ez)
- El rayo de visión puede ser calculado con dirección d = a – e, donde a = (ax,ay,az)
- Los planos *near y far*, son perpendiculares al rayo de visión, y ubicados a distancias zNear y zFar respectivamente

Estructura del *Frustum*

- Las dimensiones de los planos *near* y *far* están regidos por: la distancia entre ellos, el fov (*vertical field of view*) y el ratio (radio aspecto entre los campos de visión horizontal y vertical)



Estructura del *Frustum*

- El alto y ancho del rectángulo formado por los bordes del plano *near*, se definen como:
  - Hnear = 2 * tan(fov / 2) * zNear
  - Wnear = Hnear * ratio
- De la misma forma, aplicándolo en el plano *far*:
  - Hfar = 2 * tan(fov / 2) * zFar
  - Wfar = Hfar * ratio

- El algoritmo de *Culling* consiste en:
  1. Extraer la información del *frustum volume*
  2. Verificar si los objetos de la escena son descartados o no

Frustum Culling

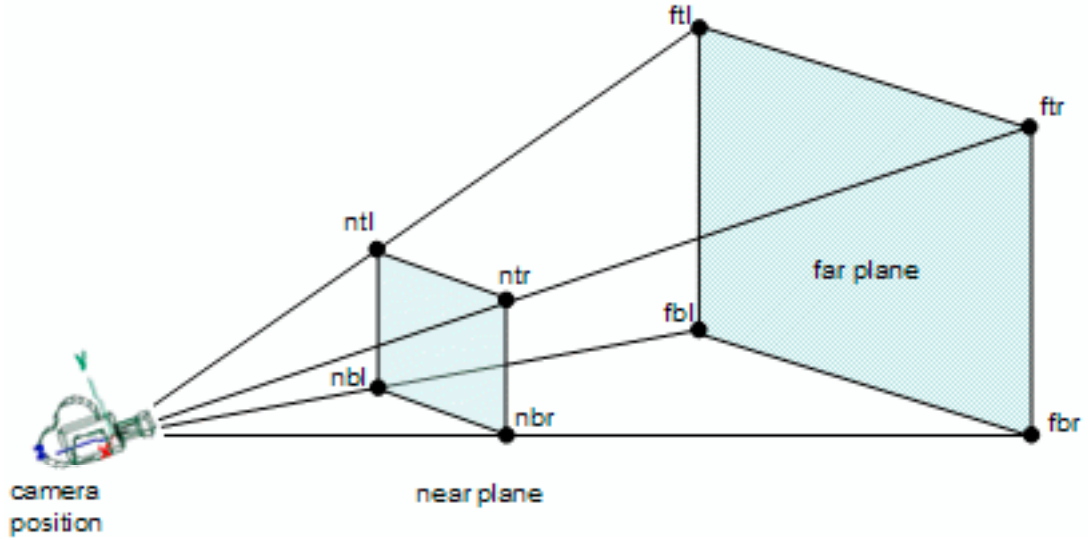
- Cada vez que la cámara se mueve, se debe extraer el volumen del *frustum*

- Algoritmo:
  - For each frame do
    - For each objeto-k do
      - If objeto-k = Tipo 1 o objeto-k = Tipo 2 then
        - render (k);
      - Else
        - descartar k;
      - EndIf
    - EndFor
  - EndFor

¿Cómo obtener el *frustum volume* ?

- *Recordatorio: se está trabajando en el espacio objeto*
- El *frustum volume* está formado por 6 planos: near, far, top, bottom, left y right
- El vector normal de los planos del *frustum*, definirán la ubicación de los objetos: delante o detrás del plano
- Calcular la distancia dist de un punto al plano-i del *frustum*
- Si dist es positivo (está por delante/detrás) de los 6 planos, el punto está dentro; en caso contrario está fuera

¿Cómo obtener el *frustum volume* ?

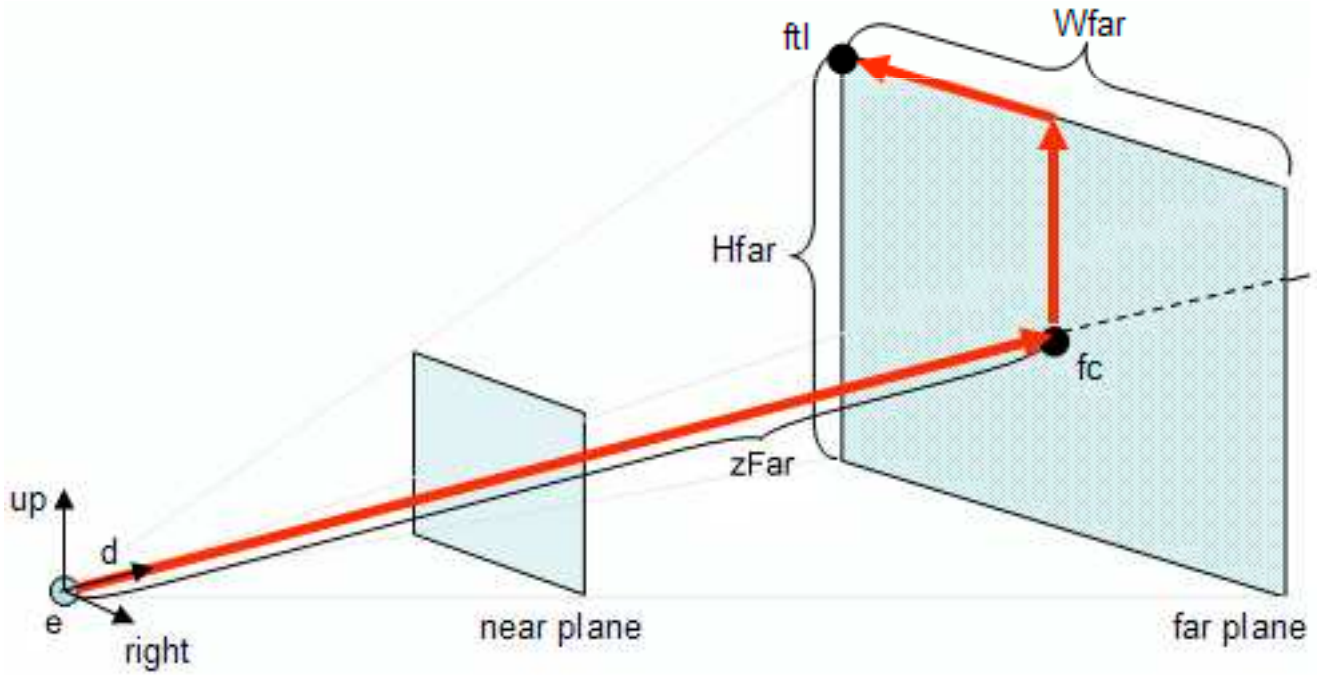


- [n | f] [b | t] [r | l]; near | far, bottom | top, right o left

- near: nbl, nbr, ntr, ntl
- far: fbl, fbr, ftr, ftl

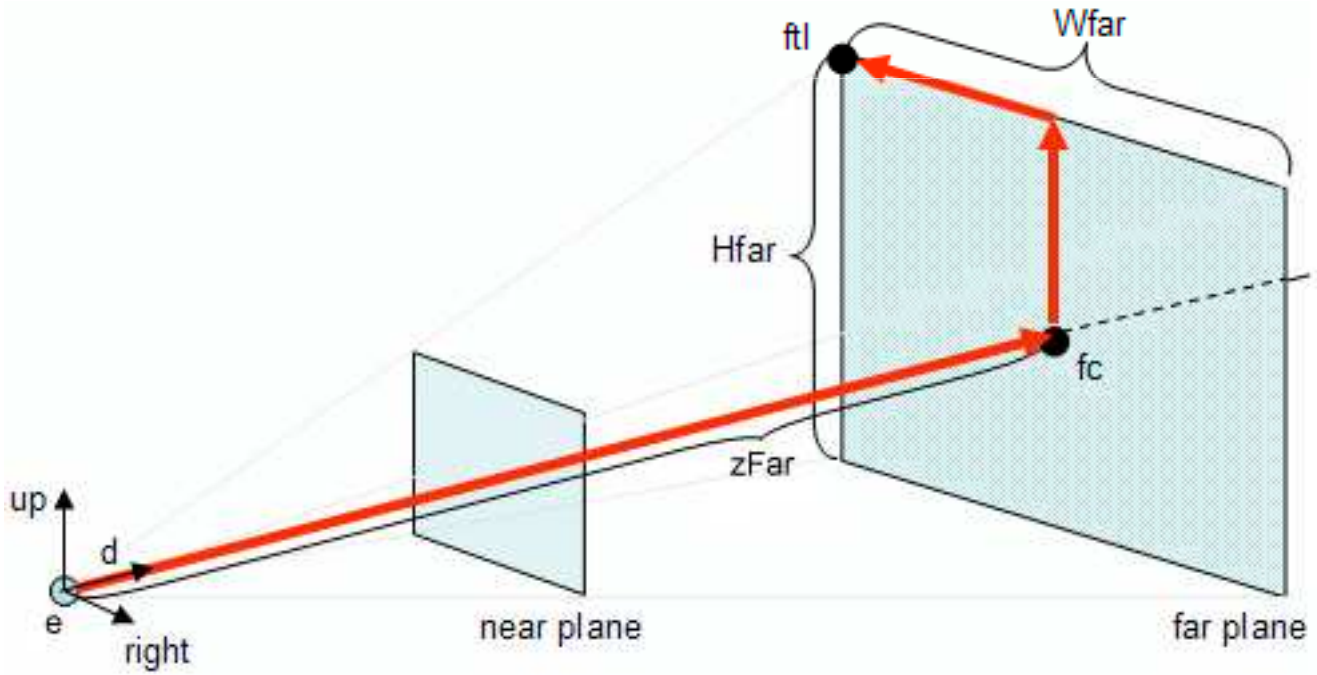
¿Cómo obtener el *frustum volume* ?

- Debemos obtener los 8 puntos de ambos planos
- Empezemos por encontrar el punto ftl del plano far



¿Cómo obtener el *frustum volume* ?

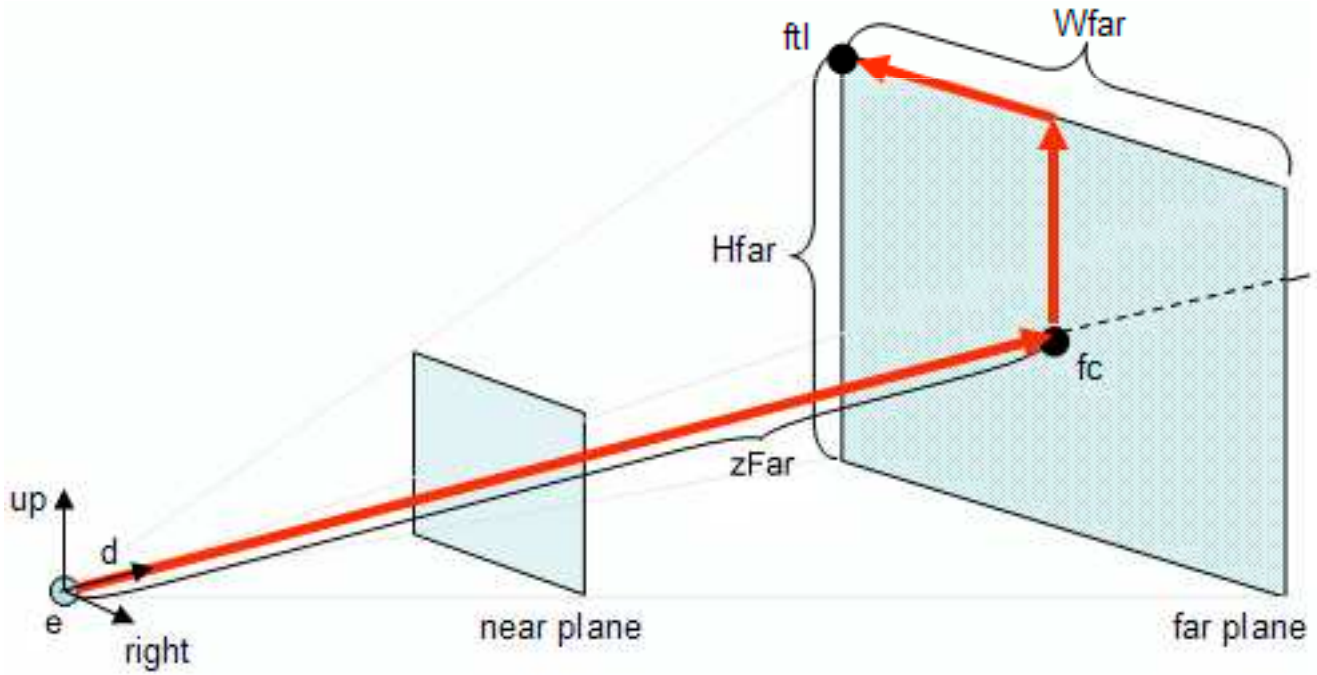
- Necesitamos 2 vectores: up y right
- El vector up es el último parámetro del *gluLookAt*
- El vector right se obtiene por d x up



¿Cómo obtener el *frustum volume* ?

fc = e + (d * zFar);

ftl = fc + (up * Hfar/2) + (-right * Wfar/2);



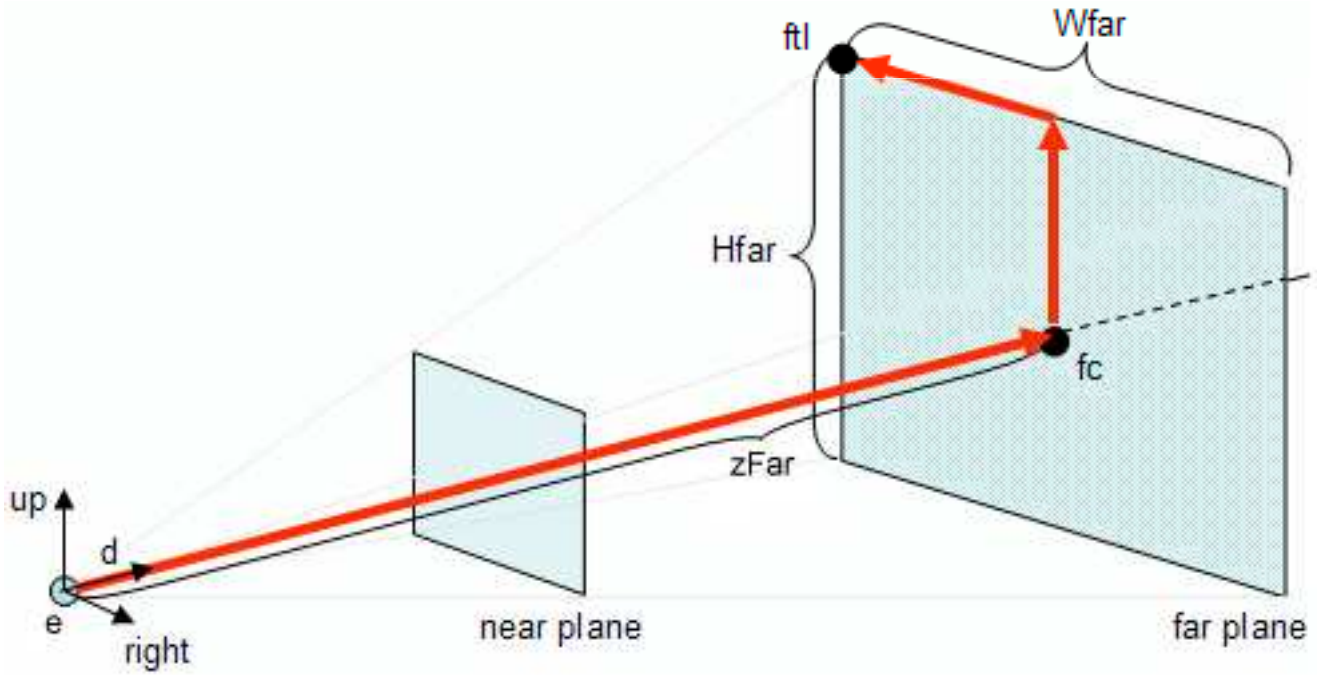
¿Cómo obtener el *frustum volume* ?

- Los otros puntos se calculan de la siguiente forma:

ftr = fc + (up * Hfar/2) + (right * Wfar/2);

fbl = fc + (-up * Hfar/2) + (-right * Wfar/2);

fbr = fc + (-up * Hfar/2) + (right * Wfar/2);



¿Cómo obtener el *frustum volume* ?

- Se aplica el mismo razonamiento para el plano near, y se tiene:

```
nc = e + (d * zNear);
ntl = nc + (up * Hnear/2) + (-right * Wnear/2)
ntr = nc + (up * Hnear/2) + (right * Wnear/2);
nbl = nc + (-up * Hnear/2) + (-right * Wnear/2);
nbr = nc + (-up * Hnear/2) + (right * Wnear/2);
```

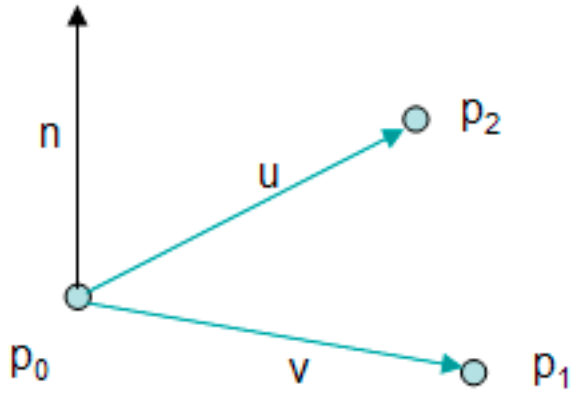
- Ahora, ya tenemos los 8 puntos
- Siguiente paso: construir los planos

¿Cómo obtener el *frustum volume* ?

- Un plano se define como

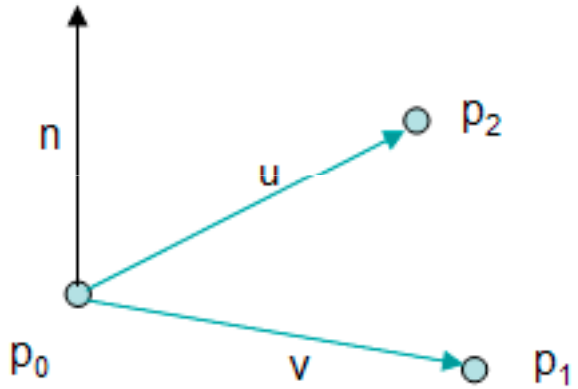
$$Ax + By + Cz + D = 0$$

- Asumiendo 3 puntos, p0, p1, p2 los coeficientes A, B, C; un plano se calcula como:
  - Calcular los vectores v = p1 – p0 y u = p2 – p0
  - Calcular n = u x v
  - Normalizar n
  - Con n = (nx,ny,nz) entonces:
    - A = xn
    - B = yn
    - C = zn
  - -D = Ax + By + Cz



¿Cómo obtener el *frustum volume* ?

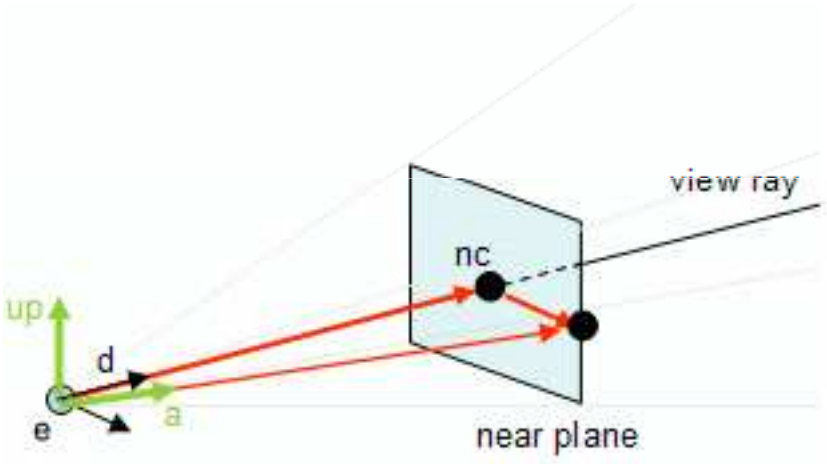
- Ahora, reemplazamos (x,y,z) por un punto en el plano. Por conveniencia tomamos a p0
- Obtenemos entonces D = -n . p0



- Para definir un plano necesitamos solamente un punto y su vector normal !!

¿Cómo obtener el *frustum volume* ?

- El plano near se puede definir con el vector normal d y el punto nc en el plano
- El plano far: vector normal -d y el punto fc



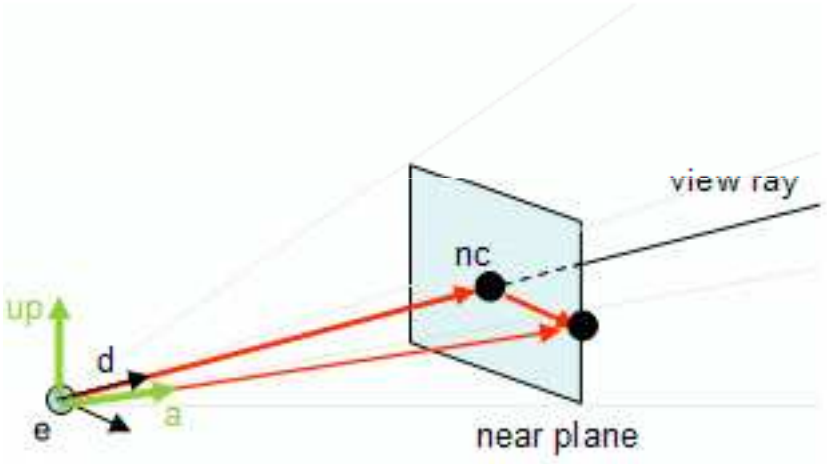
- Los otros planos los calcularemos de la misma forma. Veamos como lo hacemos para el plano right

¿Cómo obtener el *frustum volume* ?

nc = e + (d * zNear);

fc = e + (d * zFar);

a = (nc + right * Wnear / 2) – e;



a.Normalize();

normalRight = up x a;

Implementación

- De la misma forma obtenemos los planos restantes
- Ahora, calcular la distancia de un punto a los planos

```
// return true if is inside, otherwise false
Function pointInFrustum(Vector3f p) : Boolean
   for (i = 0; i < 6; i++)
         if planes[i].distance(p) < 0 then
               return false;
   return true
EndFunction
```

Implementación

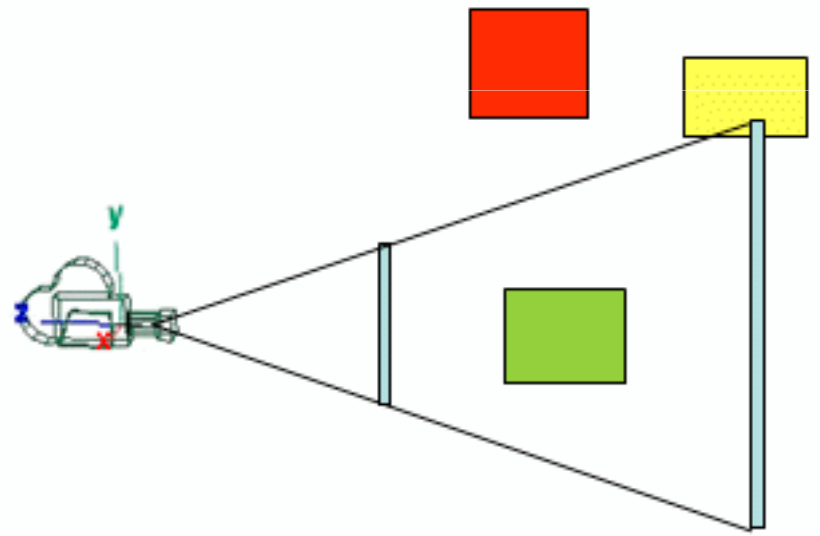
- Todos los vértices de un objeto deben ser probados
- Si TODOS están por fuera, entonces entonces el objeto es descartado
- Supongamos un objeto con una geometría de 100.000 polígonos. Quizás es mejor no hacer el test de culling !!
- Se utilizan los *bounding volumen* de los objetos: *bounding box, bounding sphere*, etc.

Es probable que los polígonos esten fuera del frustum, pero el bounding volume este dentro o viceversa

Implementación

Bounding Boxes - BB

- El 1er enfoque es probar los 8 puntos del BB
- Si todos los puntos del BB están fuera, entonces el objeto está afuera

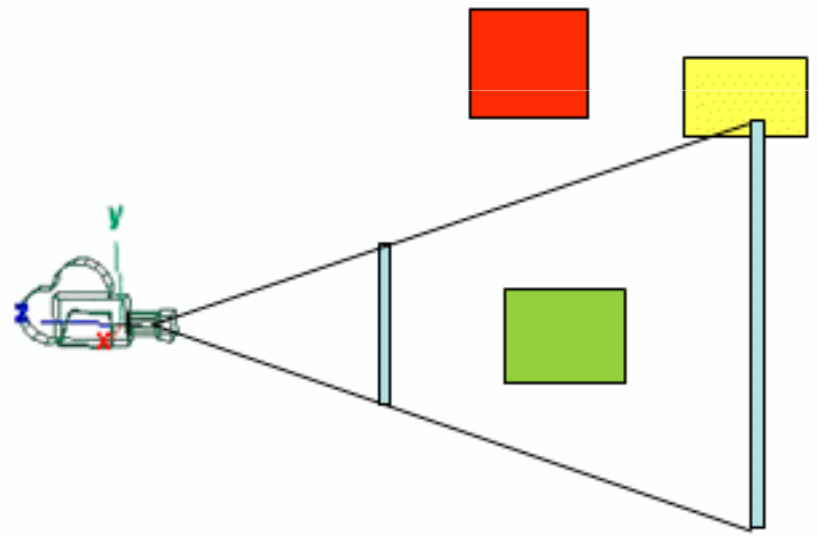


- No siempre funciona (caja amarilla)

Implementación

Bounding Boxes - BB

- Se va a rechazar una caja si y solo si, TODOS los puntos están por fuera para un mismo plano

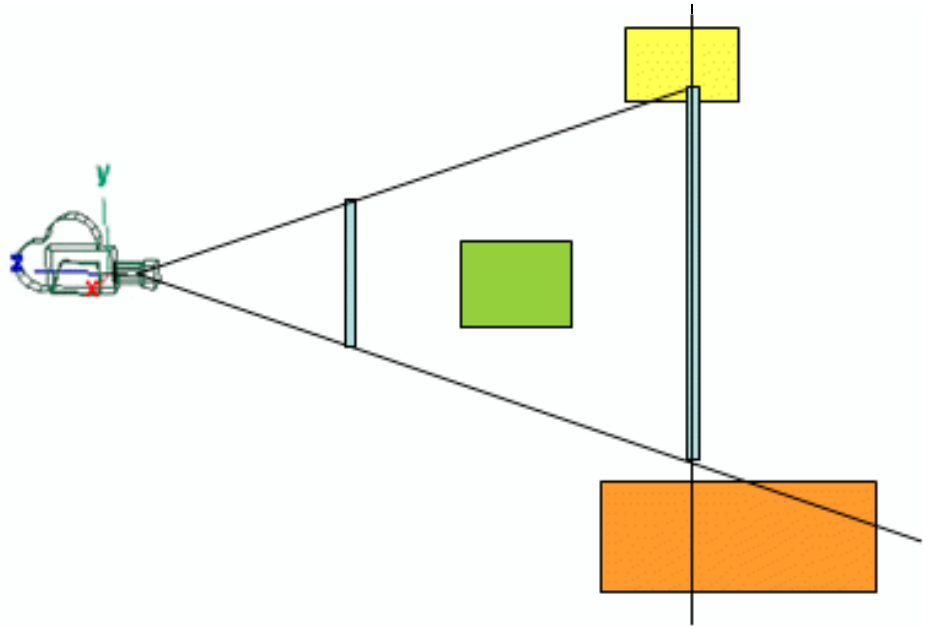


- De esta forma la caja amarilla no es rechazada

Implementación

Bounding Boxes - BB

- La caja naranja, no será rechazada !! Dos posibles soluciones se pueden plantear:
  1. Aceptar estas cajas
  2. Hacer una comprobación más profunda



Implementación

```
Function boxInFrustum(Box bb) : Boolean
    Integer in, out;
    for (i = 0, in = 0, out = 0; i < 6; i++)
            for (k = 0, k < 8 && (in==0 || out ==0); k++)
                    if planes[i].distance(bb.getVertex(k)) < 0 then
                            out = out + 1;
                    else
                            in = in + 1;
            EndFor
            if (in == 0) return false;
            else if(out > 0) return true;
    EndFor
EndFunction
```

Implementación

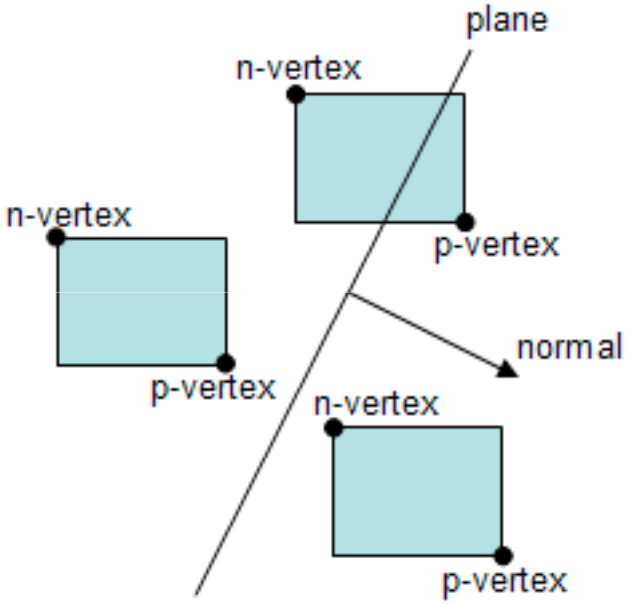
Bounding Sphere - BS

- Una esfera está formada por un centro y radio
- Una esfera se encuentra fuera si el centro está fuera y su distancia al plano es mayor que el radio

```
Function sphereInFrustum(Vector3f c, Float radius) : Boolean
   for (i = 0; i < 6; i++)
         if planes[i].distance(p) < -radius then
               return false;
         else if planes[i].distance(p) < radius then
               return true;
   Endfor
   return true;
EndFunction
```

Implementación

- Existen otras técnicas para detectar si una caja está dentro o no del frustum (e.g. *positive/negative vertex*)



- Los BB pueden ser AABB (*Axis Aligned Bounding Box*), OBB (*Oriented Bounding Box*), k-DOP, entre otros

Tarea

- Averiguar la forma eficiente de calcular distancia (con signo) de un punto al plano