z-buffer-y-transformacion-de-vis...

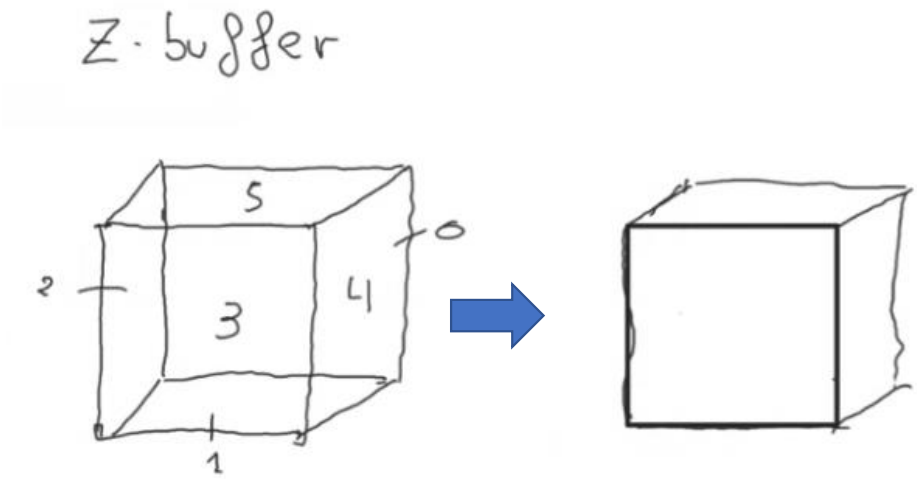
PruebaAlien

Informática Gráfica

3º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Z-buffer

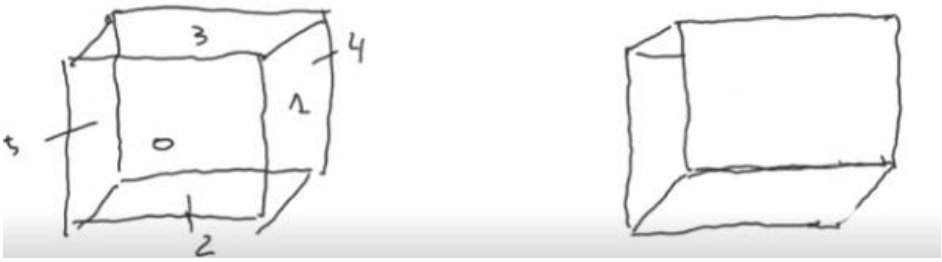


Si lo dibuja en ese orden 0, 1, 2, …, nos dibuja bien el cubo.

Pero ¿y si el orden de las caras es este?

Al dibujar la cara 2 borra parte de la cara 1.

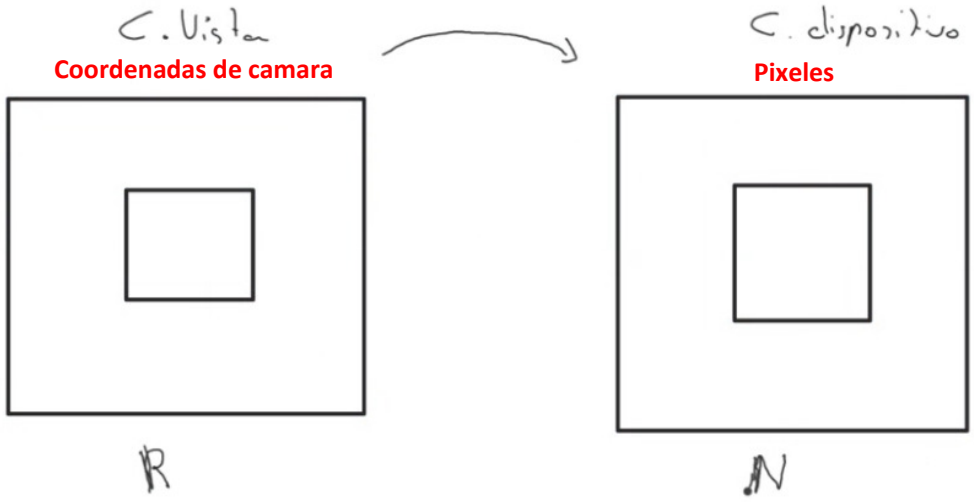
Si seguimos, al dibujar la cara 4, nos borra parte de la cara 0, 1 y 3:



Resultado final:

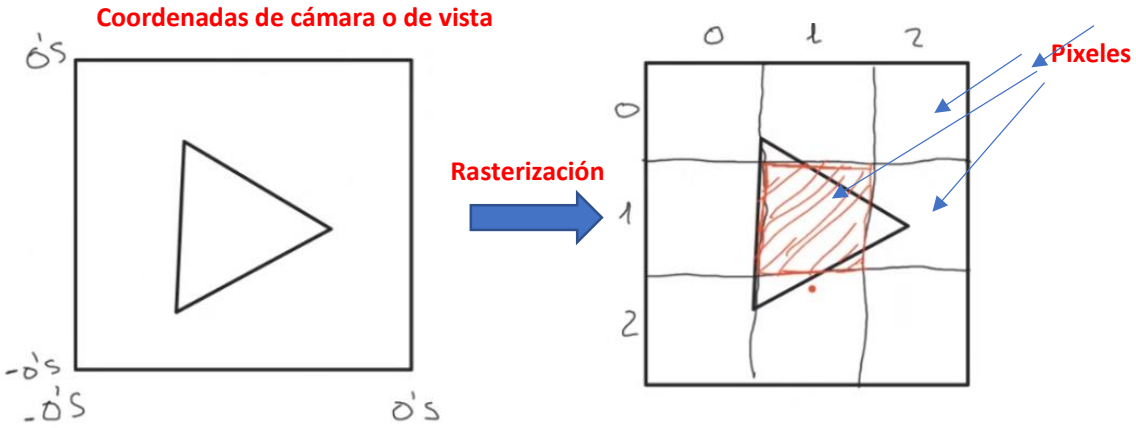
De tal forma que el resultado esta mál.

Entonces para evitar esto, utilizamos un algoritmo de **eliminación de partes ocultas**. Que actualmente lo traen implementado en todos los ordenadores en la tarjeta grafica.



Las coordenadas de vista trabaja con numero reales, mientras que las coordenadas de dispositivo trabaja con números enteros naturales.

Ejemplo Pixeles:



Un pixel se pinta cuando el pixel es ocupado por parte del objeto el 50% o mas.

Con lo cual en este ejemplo, solo pintara el pixel (1,1) el resto no se pintan, ya que no ocupan el 50% o mas.

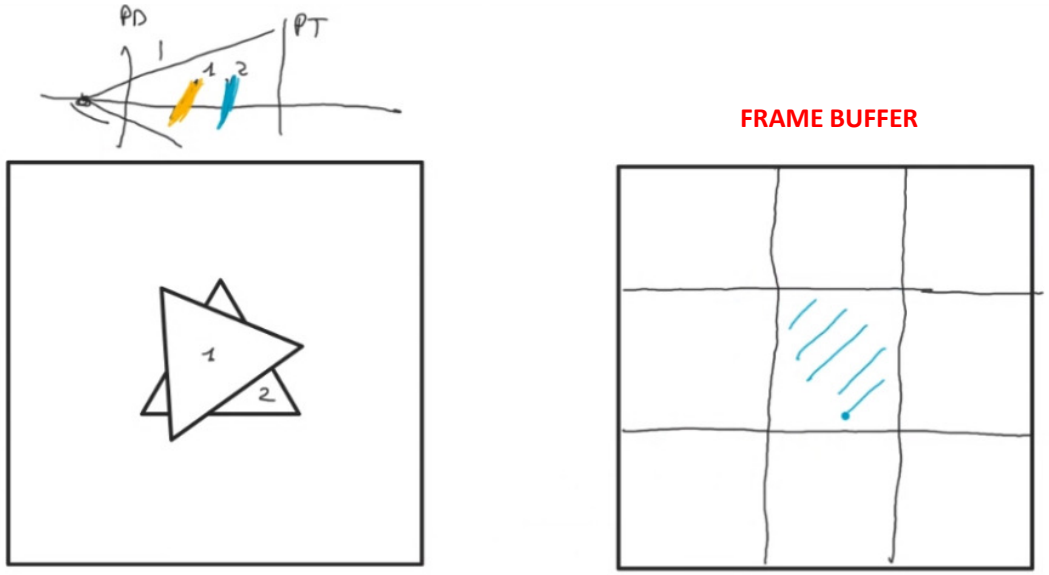
KEEP CALM AND ESTUDIA UN POQUITO

WUOLAH + #QuédateEnCasa

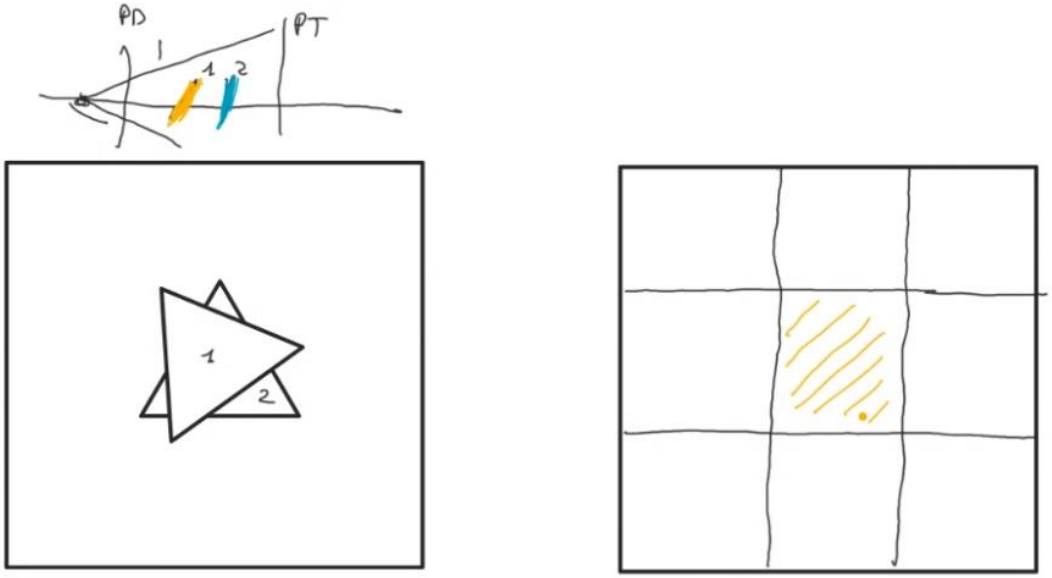
A este proceso se le llama **rasterización**. Que básicamente lo que hace es pasar de continuo a modo discreto.

Otra situación:

Ahora tenemos 2 triangulos, y dibujamos primero el triangulo 2 y luego el triangulo 1. Al dibujar primero el triangulo 2, pinta el pixel del color del triangulo 2.

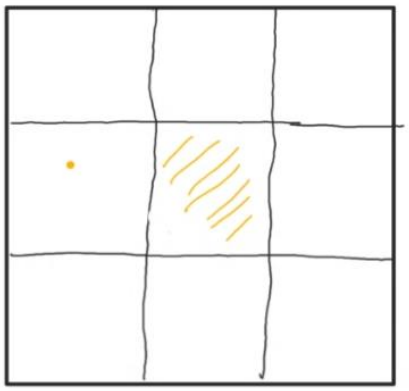


Pero cuando pinta el triangulo 1, el pixel pintado de azul se machaca por el color amarillo del triangulo 1. Pintando como resultado final el pixel amarillo:

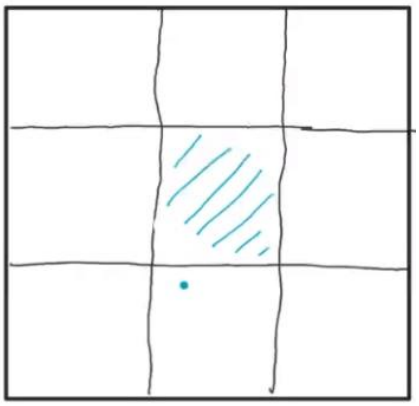


Pero si le cambiamos el orden primero el triangulo 1 y después el triangulo 2

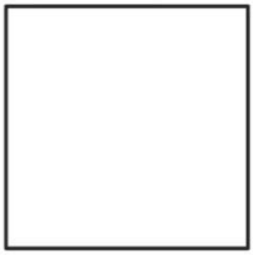
Pintara primero el triangulo 1 en el pixel de color amarillo.

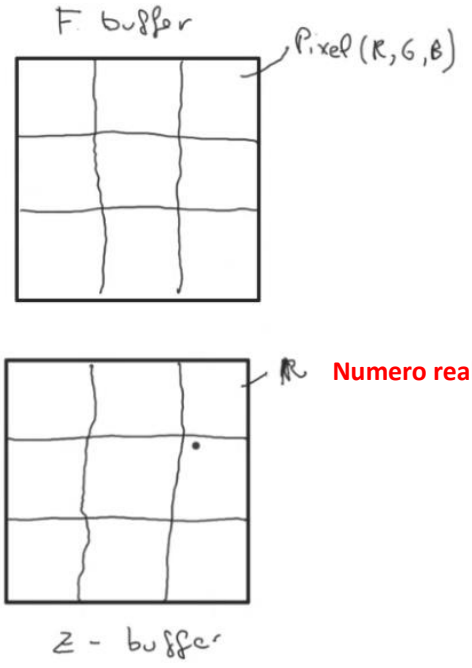


Y después pintara el pixel de color azul por el triangulo 2.



Con lo cual esto es un problema, para solucionarlo independientemente del orden, se utiliza el Z-buffer, que nos permite distinguir entre objetos, a través de la profundidad. El Z-buffer tiene las mismas dimensiones que el FRAME BUFFER.

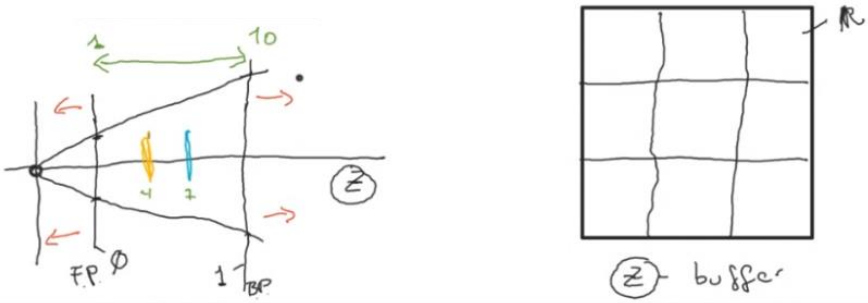




De tal forma que el **frame buffer** guardara el color y el **z-buffer** guardara la profundidad.

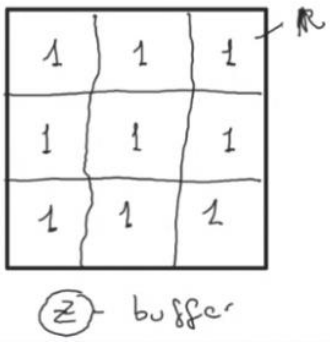
¿Qué es lo que hace el Z-buffer?

Es una forma de distinguir entre objetos, a través de la profundidad, con lo cual guardara la profundida que percibe.



De tal forma que el **z-buffer** contendrá valores de 0 a 1, siendo 0 lo mas cercano al front panel y 1 lo mas cercano al back panel.

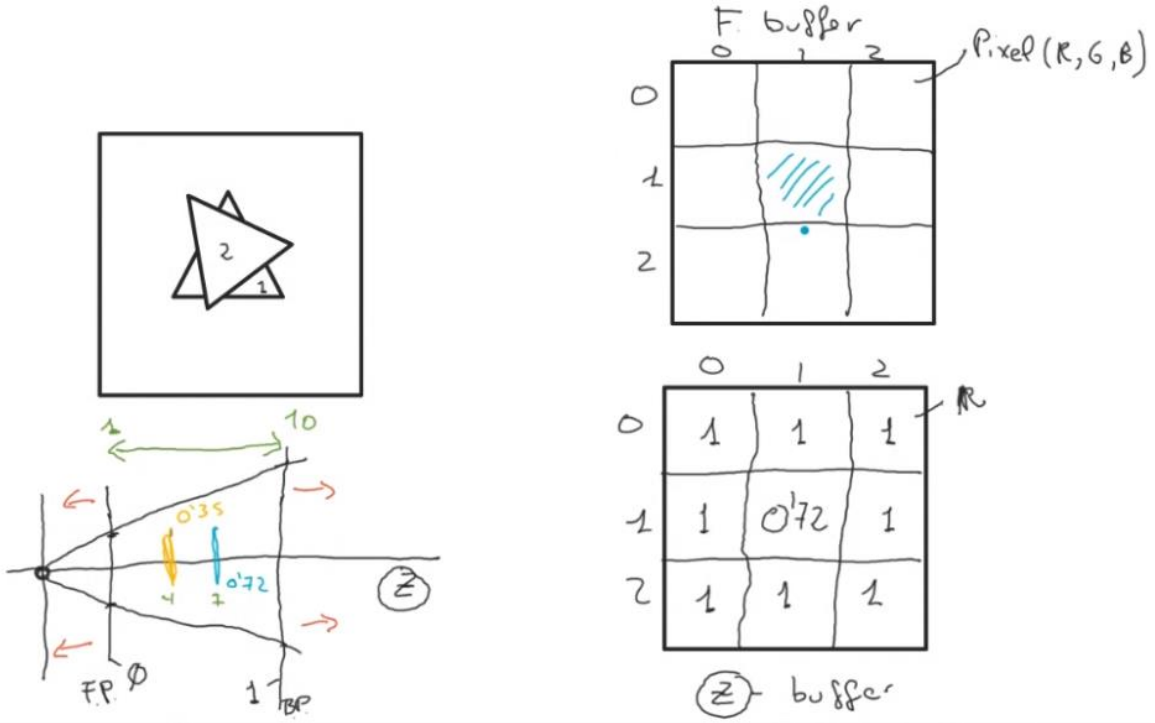
De tal forma que antes de pintar los objetos, pinta el fondo, poniendo como valor inicial en el z-buffer 1.



De tal forma que por ejemplo el **triangulo amarillo** tiene una profundidad 0.35 y el **triangulo azul** una profundidad de 0.72. Suponiendo que el 0 →1 y 1 →10 y que el triangulo amarillo esta en la posición 4 en el eje z y el triangulo azul en la posición 7 en el eje z.

El algoritmo lo que hace primero es ver que pixel le corresponde, por ejemplo el triangulo azul le corresponde el pixel (1,1). Despues mira en el z-buffer en esa misma posición (1,1) y compara el resultado del z-buffer con el valor de profundidad del objeto. En nuestro caso con el triangulo azul (0.72).

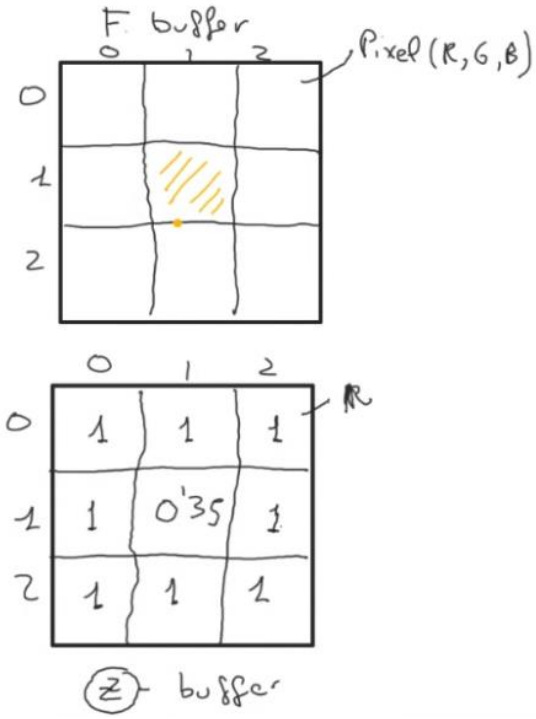
Como 0.72 < 1, cambia el valor de esa posición del z-buffer por el valor 0.72.



Ahora vamos a pintar el triangulo amarillo:

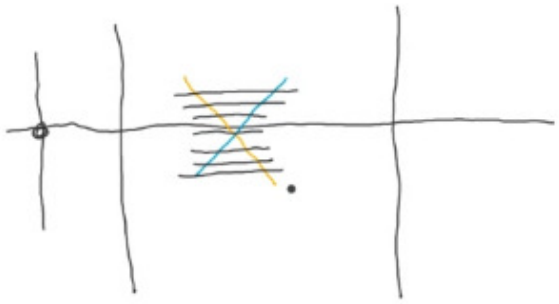
- Comprueba los pixeles que ocupa el objeto
- Y después comprueba los valores de cada posición de cada pixel con la del objeto

Como 0.35 < 0.72, cambia el valor en el z-buffer en la posición (1,1) y pinta el pixel de color amarillo.



Si el orden es al reves el resultado es el mismo, ya que pinta el triangulo amarillo (0.35 < 1) y el triangulo azul no lo pinta porque el valor de profundidad 0.72 > 0.35, con lo cual no actualiza el z-buffer y no lo pinta en esa posición del pixel, si fuera menor el valor de profundidad al que tiene el z-buffer lo dibujaria.

Ejemplo en el que los 2 triangulos esten cruzados, depende de por donde el pixel puede pintarse el triangulo azul o el triangulo amarillo.



Con lo cual funciona el z-buffer.

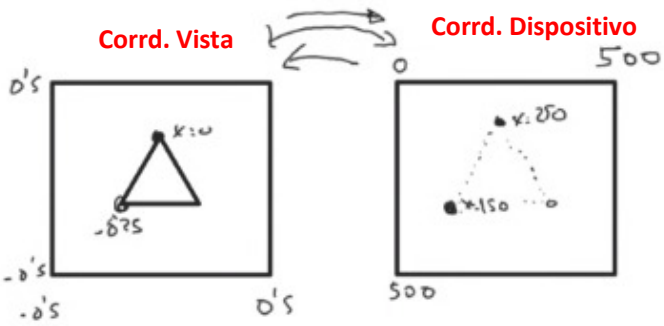
Calcular la recta (pasar de Coordenadas de Vista a Coordenadas de Dispositivo)

Coordenadas Vista: $X_{vista(max)}$, $X_{vista(min)}$

Coordenadas de Dispositivo: $X_{dispositivo(max)}$, $X_{dispositivo(min)}$

$$\boldsymbol{X_d = \frac{\left(X_{dispositivo(max)} - X_{dispositivo(min)}\right) * \left(X - X_{vista(min)}\right)}{X_{vista(max)} - X_{vista(min)}} + X_{dispositivo(min)}}$$

Ejemplo:



$$X_d = \frac{(500 - 0) * \left(X - (-0.5)\right)}{0.5 - (-0.5)} + 0 = \frac{500 * (X + 0.5)}{1}$$

Y ya con esa formula podemos pasar cualquier coordenada x de vista a dispositivo.

Ejemplos:

X = 0;

$$X_d = \frac{500 * (0 + 0.5)}{1} = 250$$

X = -0.25;

$$X = \frac{500 * ((-0.25) + 0.5)}{1} = 500 * 0.25 = 125$$

(pasar de Coordenadas de Dispositivo a Coordenadas de Vista) Es simplemente despejar.

$$X_d = \frac{(X_{dispositivo(max)} - X_{dispositivo(min)}) * (\boldsymbol{X} - X_{vista(min)})}{X_{vista(max)} - X_{vista(min)}} + X_{dispositivo(min)}$$

$$X_d - X_{dispositivo(min)} = \frac{(X_{dispositivo(max)} - X_{dispositivo(min)}) * (\boldsymbol{X} - X_{vista(min)})}{X_{vista(max)} - X_{vista(min)}}$$

$$\left(X_d - X_{dispositivo(min)}\right) * X_{vista(max)} - X_{vista(min)} = \left(X_{dispositivo(max)} - X_{dispositivo(min)}\right) * \left(\boldsymbol{X} - X_{vista(min)}\right)$$

$$\frac{\left(X_d - X_{dispositivo(min)}\right) * X_{vista(max)} - X_{vista(min)}}{\left(X_{dispositivo(max)} - X_{dispositivo(min)}\right)} = \left(\boldsymbol{X} - X_{vista(min)}\right)$$

$$\frac{\left(X_d - X_{dispositivo(min)}\right) * X_{vista(max)} - X_{vista(min)}}{\left(X_{dispositivo(max)} - X_{dispositivo(min)}\right)} + X_{vista(min)} = X$$

Solución final:

$$\boldsymbol{X} = \frac{\left(\boldsymbol{X_d} - \boldsymbol{X_{dispositivo(min)}}\right) * \left(\boldsymbol{X_{vista(max)}} - \boldsymbol{X_{vista(min)}}\right)}{\left(\boldsymbol{X_{dispositivo(max)}} - \boldsymbol{X_{dispositivo(min)}}\right)} + \boldsymbol{X_{vista(min)}}$$

Ejemplo anterior:

$$X = \frac{(X_d - 0) * (0.5 - (-0.5))}{(500 - 0)} + (-0.5) = \frac{X_d * 1}{500} - 0.5 = \frac{X_d}{500} - 0.5$$

$$X_d = 250$$

$$X = \frac{250}{500} - 0.5 = 0$$

$$X_d = 150$$

$$X = \frac{150}{500} - 0.5 = -0.2$$

Resumen:

pasar de Coordenadas de Vista a Coordenadas de Dispositivo

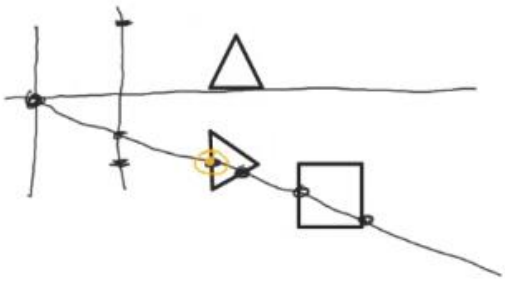
$$\boldsymbol{X_d} = \frac{(\boldsymbol{X_{dispositivo(max)}} - \boldsymbol{X_{dispositivo(min)}}) * (\boldsymbol{X} - \boldsymbol{X_{vista(min)}})}{\boldsymbol{X_{vista(max)}} - \boldsymbol{X_{vista(min)}}} + \boldsymbol{X_{dispositivo(min)}}$$

pasar de Coordenadas de Dispositivo a Coordenadas de Vista

$$\boldsymbol{X} = \frac{(\boldsymbol{X_d} - \boldsymbol{X_{dispositivo(min)}}) * (\boldsymbol{X_{vista(max)}} - \boldsymbol{X_{vista(min)}})}{(\boldsymbol{X_{dispositivo(max)}} - \boldsymbol{X_{dispositivo(min)}})} + \boldsymbol{X_{vista(min)}}$$

Cuando pincho con el ratón un objeto, guarda la posición (x) e (y) y lanza un rayo de proyección desde el centro de proyección a la posición Y que ha pinchado. A continuación, comprueba si algún objeto interseca con el rayo, y se queda con el objeto que esta más cerca del centro de proyección que tenga una intersección con el rayo.

En este ejemplo ha habido 4 intersecciones en 2 objetos:



Para realizar la **transformación de vista** necesito:

- **Vector hacia arriba View UP**
- **Centro de proyección PRP**
- **Posición de la cámara VPN**

Ejemplo (CUBO):

(Teniendo un cubo de lado 1 con sus esquinas en (0,0,0) y (1,1,1))

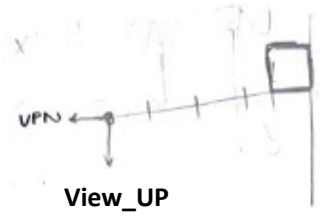
PRP = (0,0,5), VPN = (0,0,1) y View_UP = (0,1,0)

Donde se encuentra:

Como se ve desde la cámara:

PRP = (0,0,5), VPN = (0,0,1) y View_UP = (0,-1,0)

Donde se encuentra:

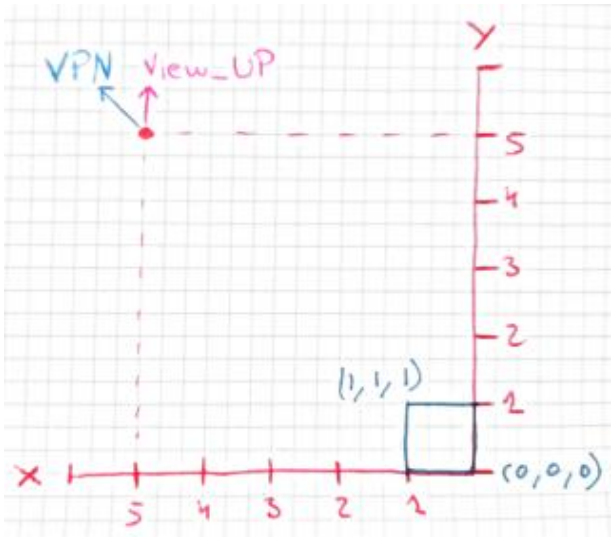


View_UP

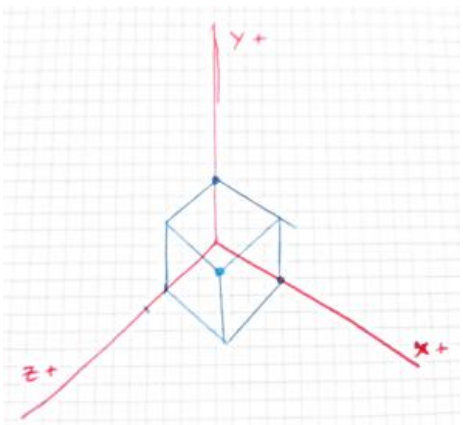
Como se ve desde la cámara:

PRP = (5,5,5), VPN = (1,1,1) y View_UP = (0,1,0)

Donde se encuentra:

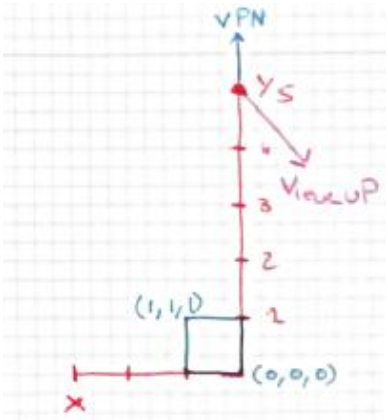


Como se ve desde la cámara:

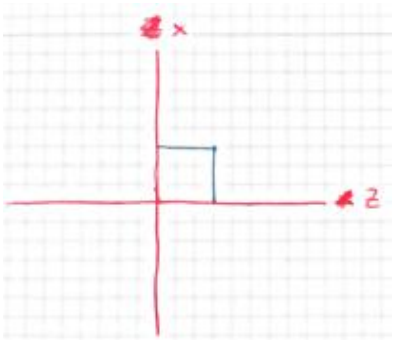


PRP = (0,5,0), VPN = (0,1,0) y View_UP = (1,0,0)

Donde se encuentra:

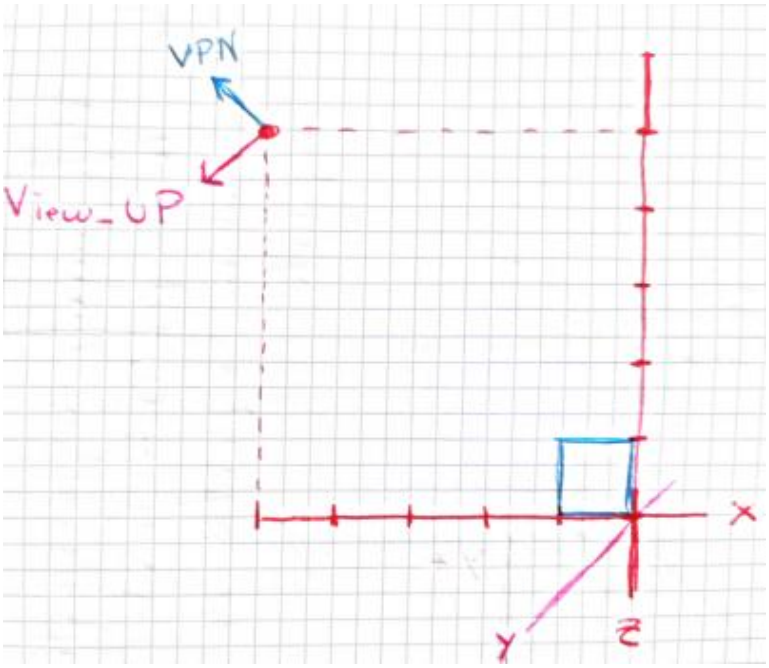


Como se ve desde la cámara:



PRP = (5,0,5), VPN = (1,0,1) y View_UP = (0,1,0)

Donde se encuentra:



Como se ve desde la cámara: