Tema 3: Viendo en 3D

José Ribelles

Departamento de Lenguajes y Sistemas Informáticos, Universitat Jaume I

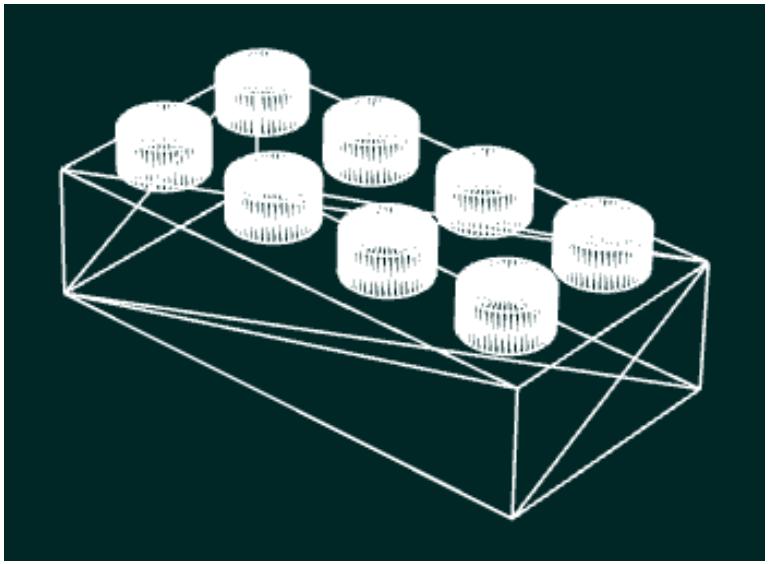
SIU020 - Síntesis de Imagen y Animación

Contenido

Hoy veremos...

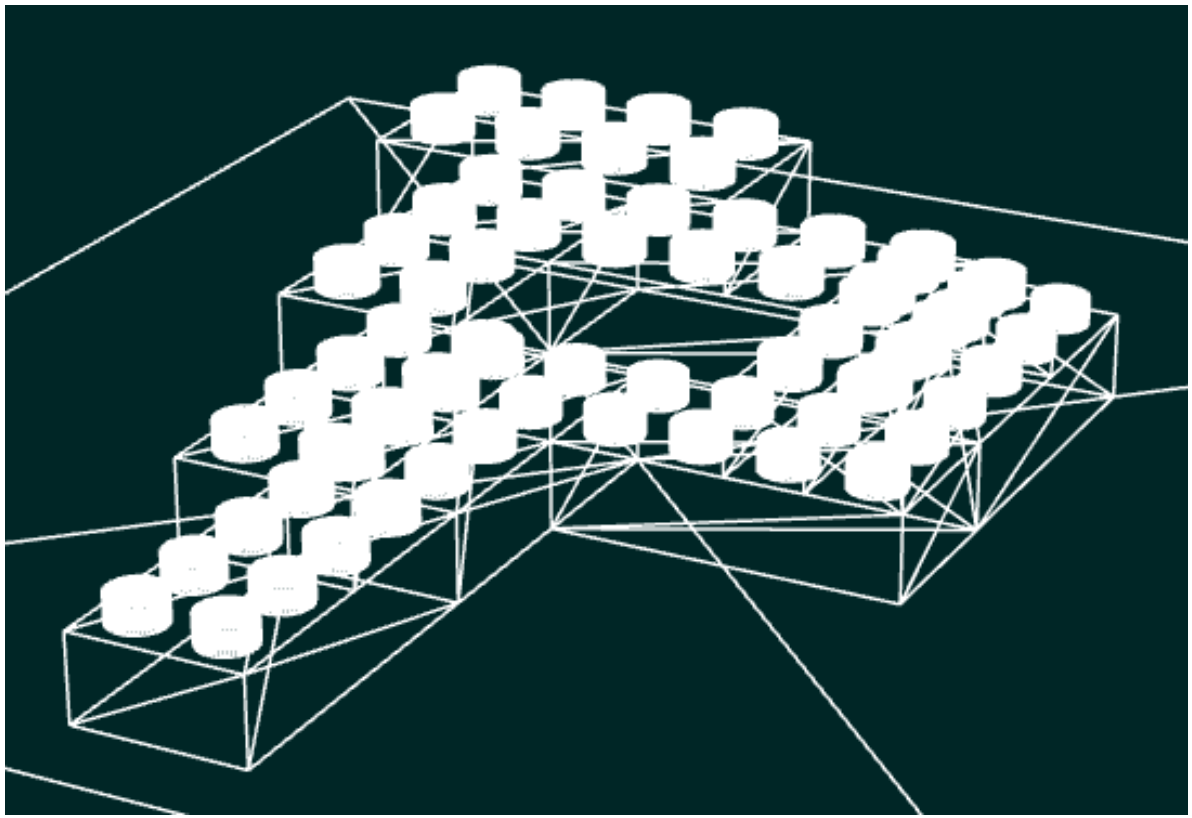
Introducción

Ya sabemos cómo crear y visualizar un modelo como este:



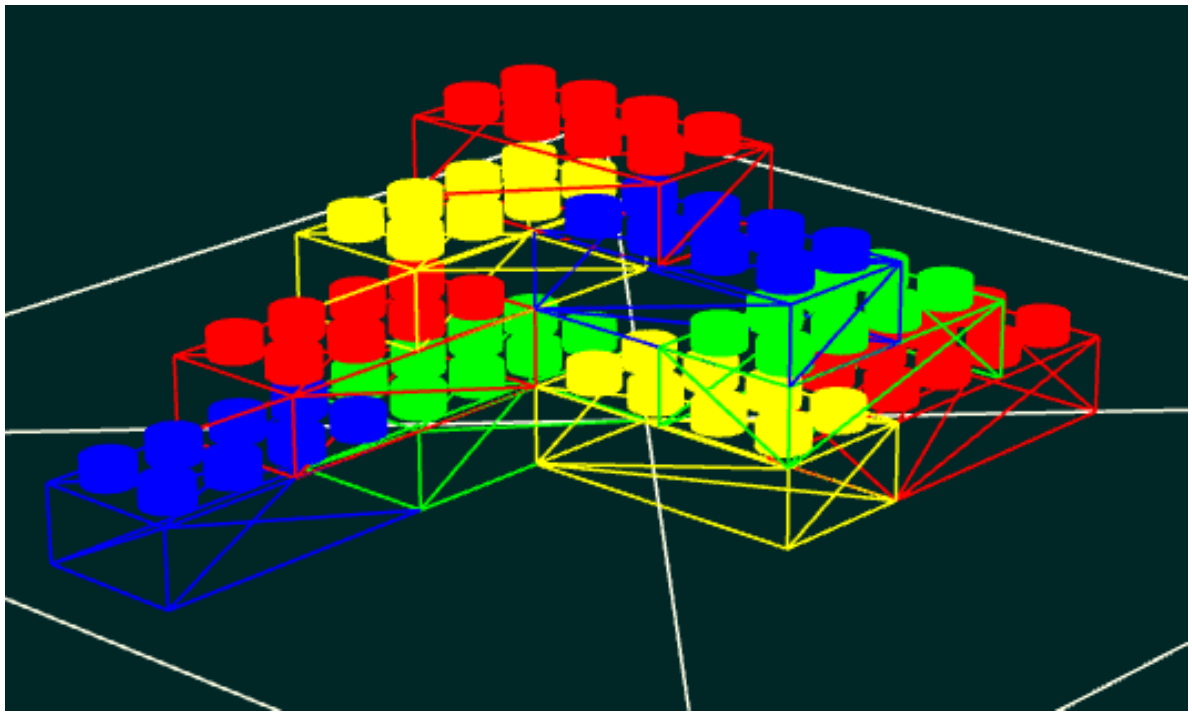
Introducción

Y a partir de dicho modelo ya deberíamos saber cómo crear una escena más compleja:



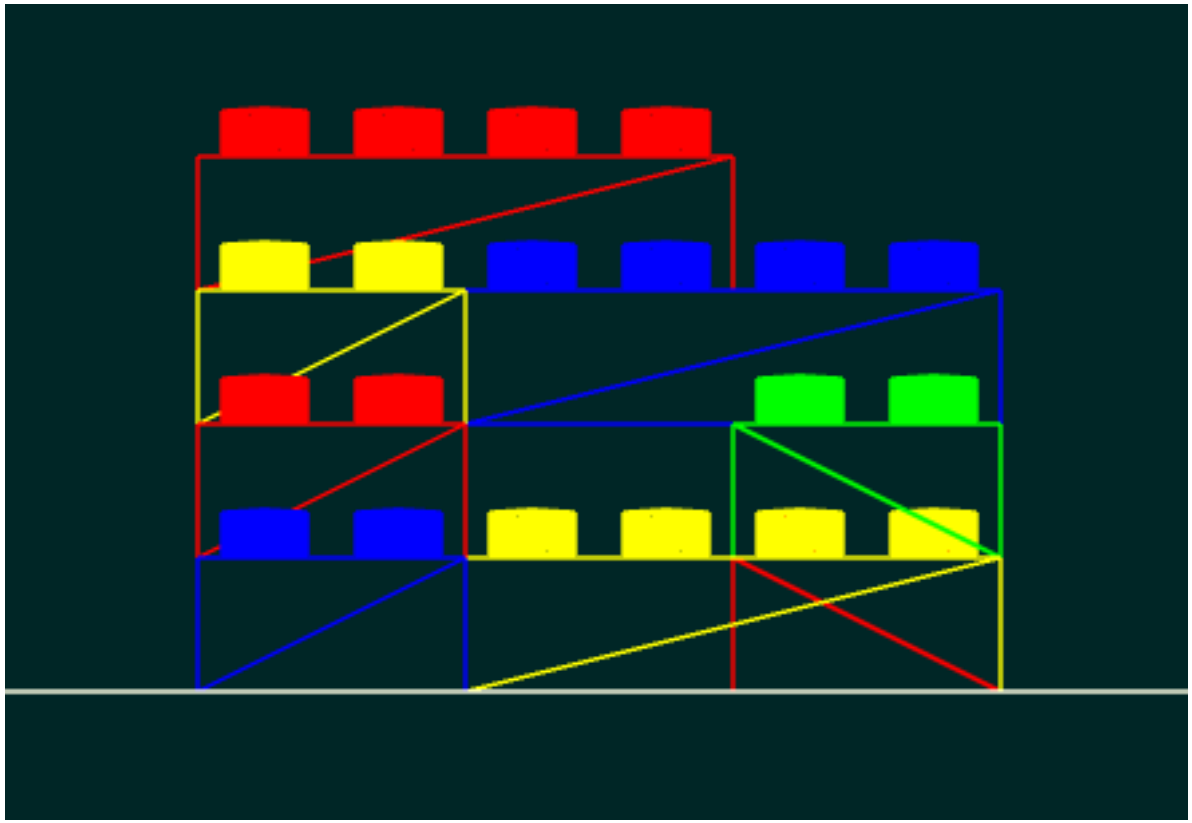
Introducción

E incluso añadirle colores:



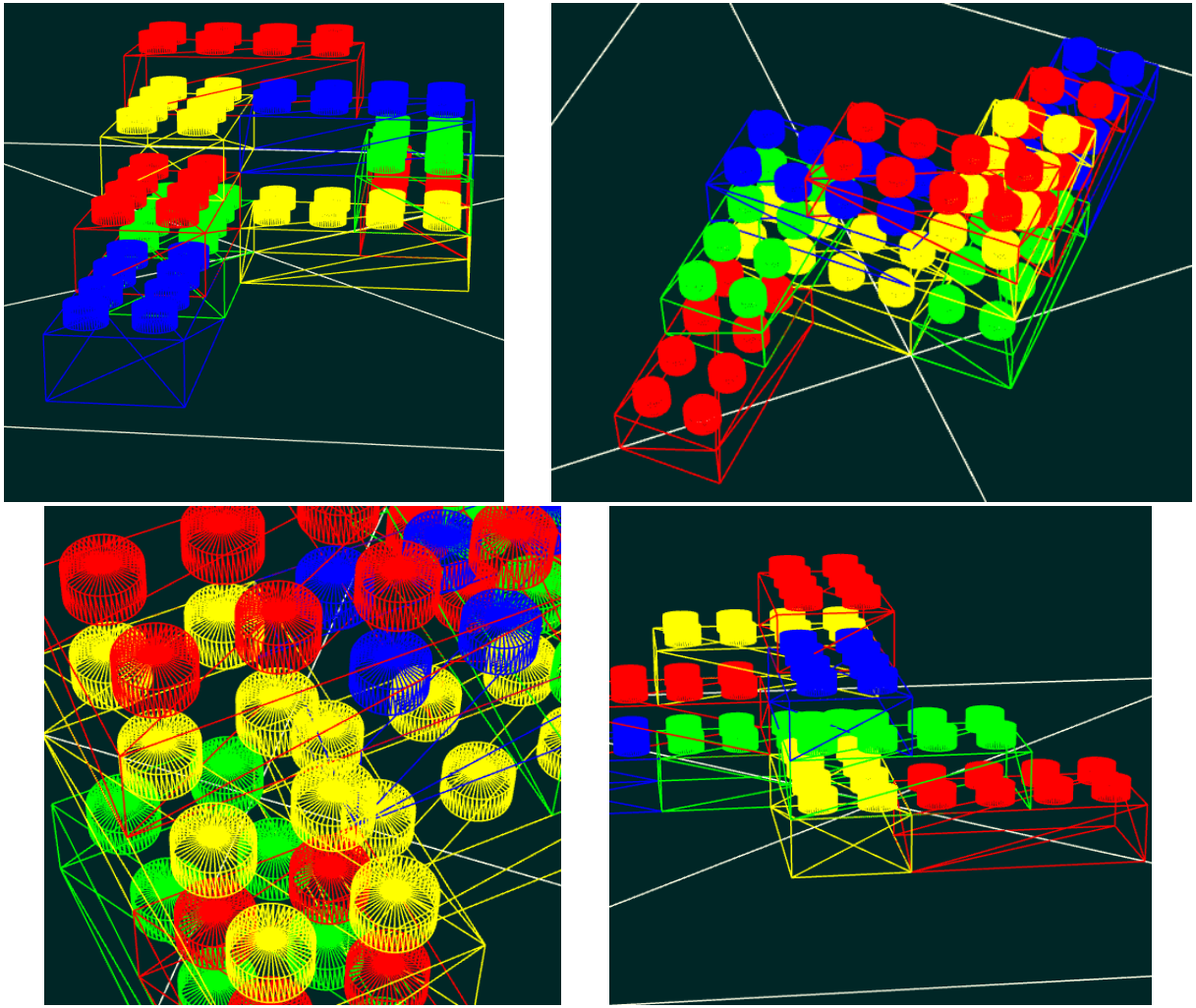
Introducción

Pero lo cierto es que solo obtenemos vistas como esta (¿de verdad esto es 3d?):



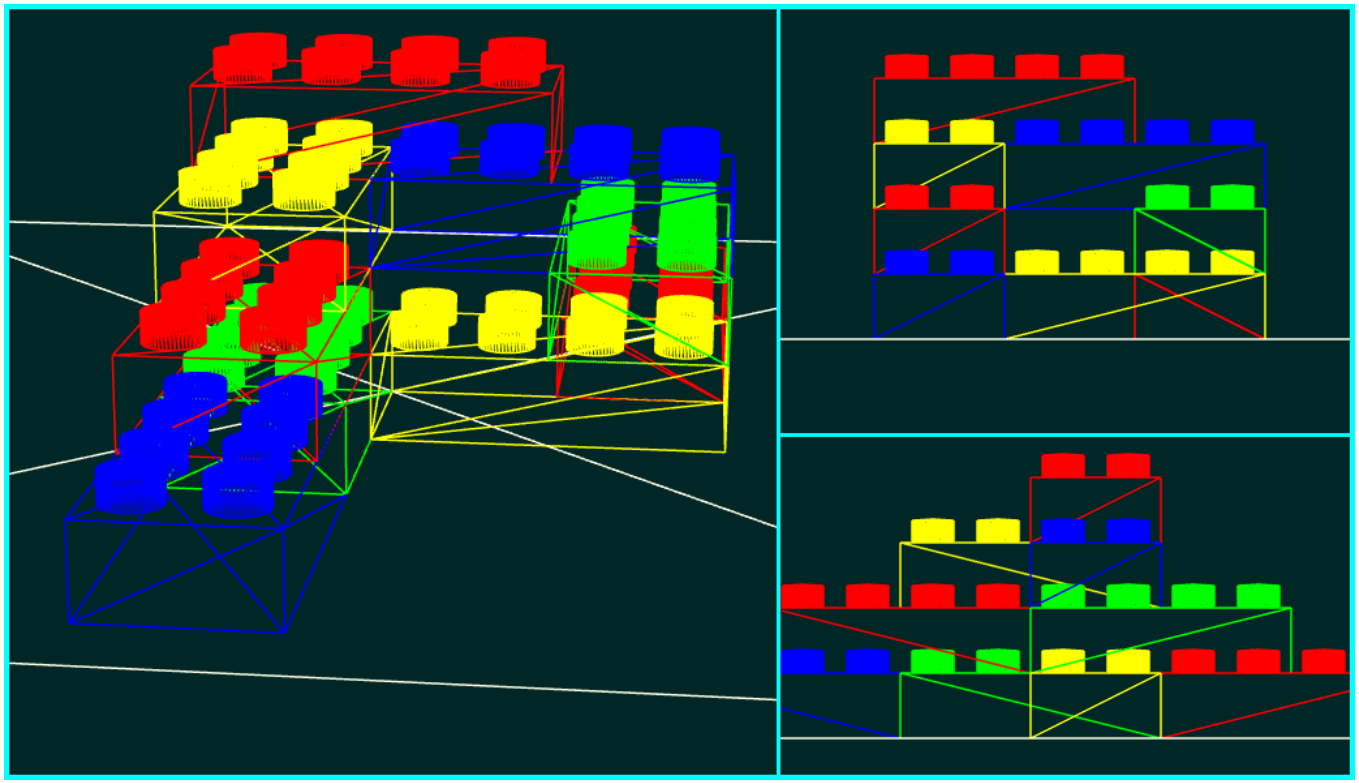
Introducción

¿Cómo podemos obtener diferentes vistas de la escena?



Introducción

¿Y si quiero combinar varias vistas?



¿Cómo conseguirlo?

Obtener una vista se implementa como una secuencia de tres transformaciones:

1. Transformación de la cámara.
2. Transformación de proyección.
3. Transformación al área de dibujo.

¿Sobre qué aplico estas transformaciones?

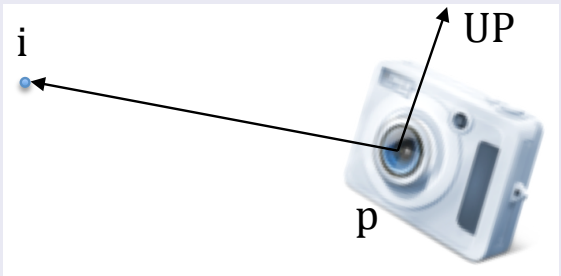
Estas transformaciones son propias de la escena (no de un objeto en concreto) por lo que se aplican a todos y cada uno de los elementos que se dibujen en la escena.

Hoy veremos...

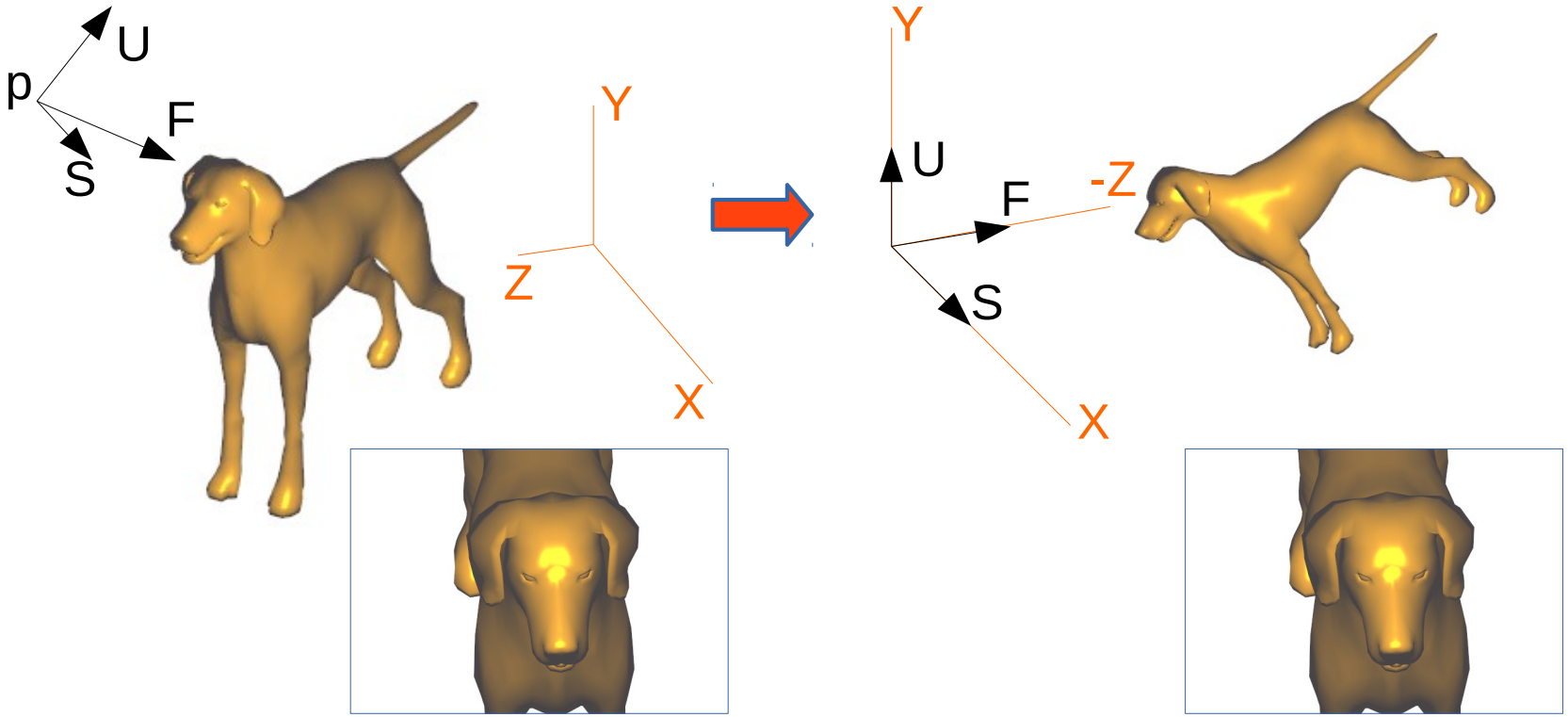
Transformación de la cámara

¿En qué consiste?

- Una vez tengas la escena construida:
  - Ubica, orienta y enfoca la cámara como tu quieras.
- Situación inicial de la cámara:
  - La posición es un punto p del espacio 3D.
  - El objetivo queda apuntando a un punto i.
  - La inclinación se establece mediante el vector UP.



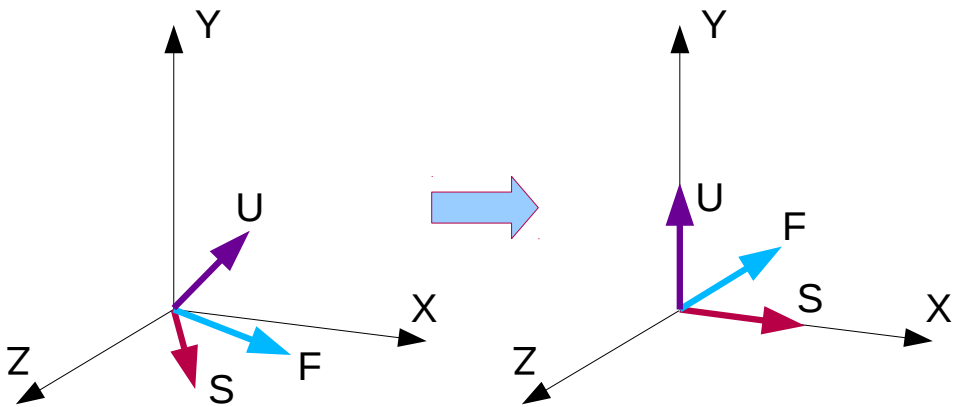
- Y la transformación de la cámara consiste en ... que la cámara quede situada en el origen de coordenadas apuntando en la dirección del eje Z negativo y coincidiendo el vector de inclinación con el eje Y positivo.


p
U
F
S
Y
Z
X
Y
U
F
-Z
S
X

Transformación de la cámara

¿Sabrías hacerlo?

- Necesitas una traslación para llevarla del punto p al origen de coordenadas.
- Para después hacer el cambio de base. Sean:
  - F el vector normalizado que desde la posición de la cámara apunta al punto de interés,
  - UP' el vector de inclinación normalizado,
  - $S = F \times UP'$
  - $U = S \times F$



Matriz de transformación de la cámara

- Entonces M_C es la matriz de transformación de la cámara.

$$M_C = \begin{pmatrix} S_x & S_y & S_z & 0 \\ U_x & U_y & U_z & 0 \\ -F_x & -F_y & -F_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -p_x \\ 0 & 1 & 0 & -p_y \\ 0 & 0 & 1 & -p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (1)$$

La librería glm nos ayuda

- mat4.lookAt (out, p, i, UP)

En definitiva

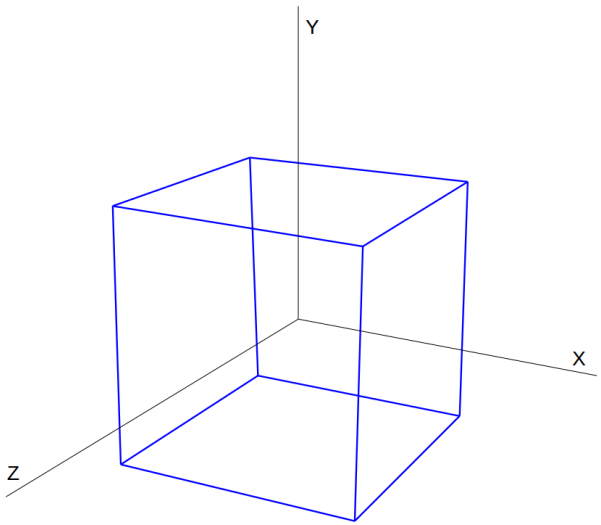
- Es responsabilidad del desarrollador obtener dicha matriz
- Y operarla con todas las primitivas de la escena como una transformación geométrica más, ¿dónde?

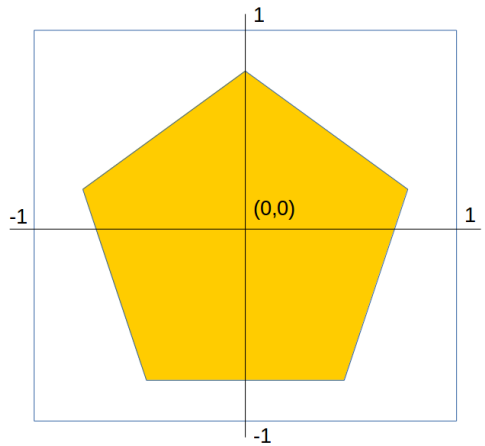
Hoy veremos...

Transformación de proyección

El sistema gráfico establece un volumen finito:

- Conocido con el nombre de: **Volumen canónico de la vista**
- De tamaño, posición y orientación fijos.
- En OpenGL, es un cubo de lado 2 centrado en el origen de coordenadas como el de la siguiente figura.

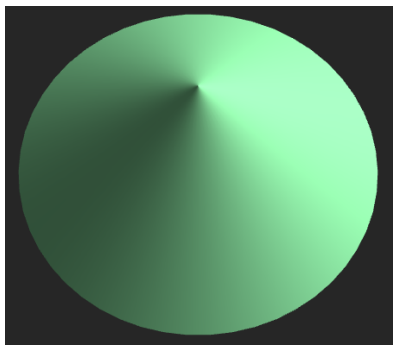


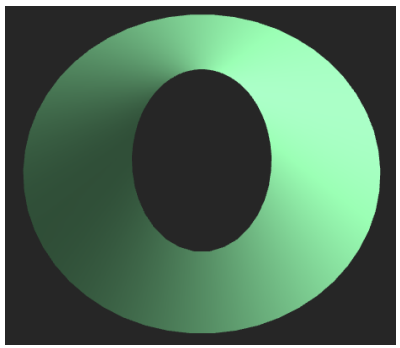


Transformación de proyección

¿De qué manera lo utiliza el sistema gráfico?

- Procesa solo lo que esté dentro del volumen canónico de la vista.
- Se elimina la parte de la escena que quede fuera de dicho volumen.
- ¿Quién comprueba si una primitiva gráfica está dentro o fuera de dicho volumen?
  - Lo hace el sistema gráfico de manera fija (no es una opción)
- ¿Y si parte de la primitiva está dentro y parte fuera?
  - El sistema gráfico recorta la primitiva contra el volumen canónico de la vista y se procesa el trozo que queda en el interior de dicho volumen.

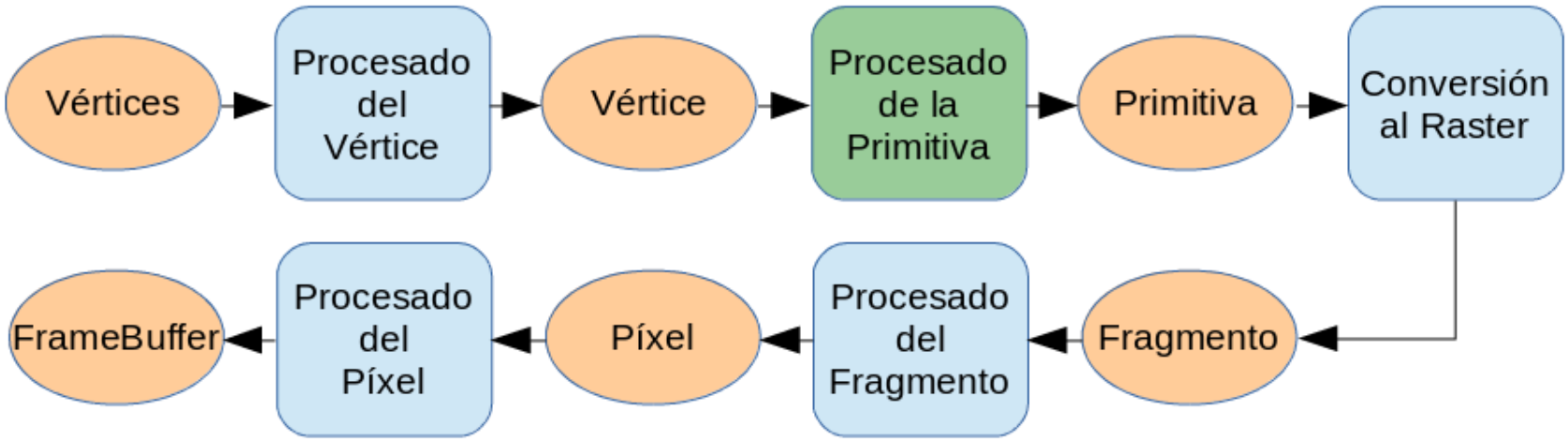




Transformación de proyección

¿Y cuándo lo hace?

- Primero has de saber que a este proceso se le conoce con el nombre de Recortado (Clipping).
- Y se realiza en la etapa de Procesado de la Primitiva.
- ¿Puede intervenir el desarrollador en este proceso?
  - No.



Transformación de proyección

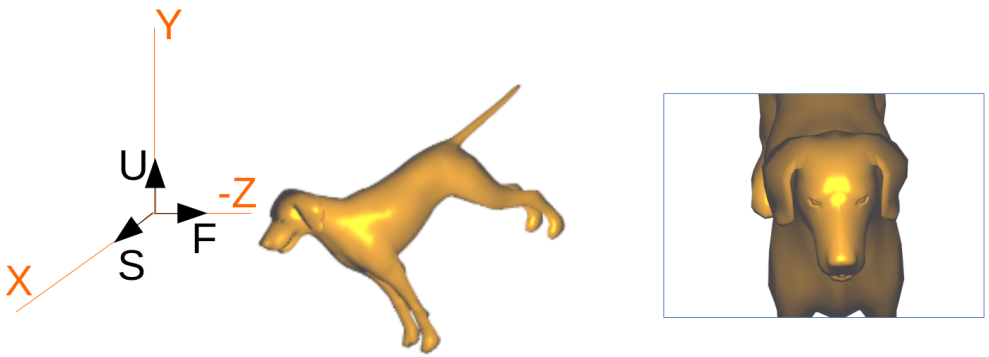
Entonces,

- Si el volumen canónico de la vista es fijo...
- ... y lo que queda fuera del volumen es eliminado del proceso,
- ¿significa esto que es faena del desarrollador hacer que la escena que desea visualizar quede dentro del volumen canónico de la vista?
  - ¡¡¡Si!!!
- ¿Cómo lo hacemos?
  - Mediante la **transformación de proyección**.

Transformación de proyección

Tras aplicar la transformación de la cámara

Nos encontramos en esta situación:



¿Qué hacemos ahora?

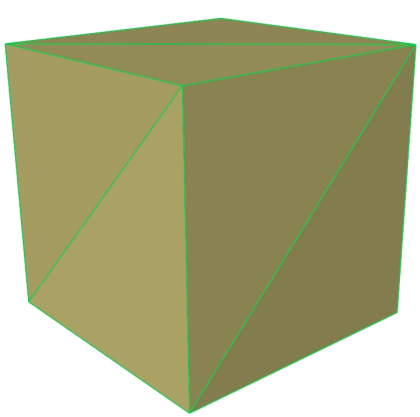
1. Establezcamos nuestro propio volumen de la vista
2. Transformemos nuestro volumen de la vista en el canónico del sistema gráfico

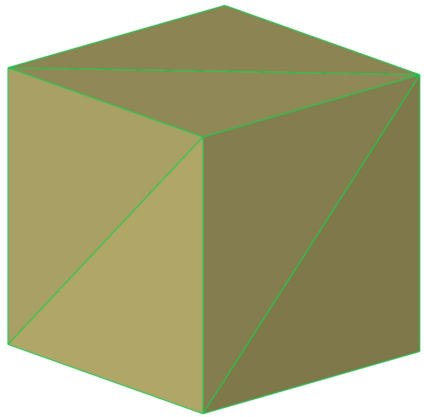
Transformación de proyección

¿Qué forma tendrá nuestro volumen de la vista?

Depende del tipo de proyección.

- Vista perspectiva: se utiliza para generar imágenes más fieles a la realidad en aplicaciones como videojuegos, simulaciones, etc.
- Vista paralela: utilizada principalmente en ingeniería o arquitectura, y se caracteriza por preservar longitudes y ángulos.

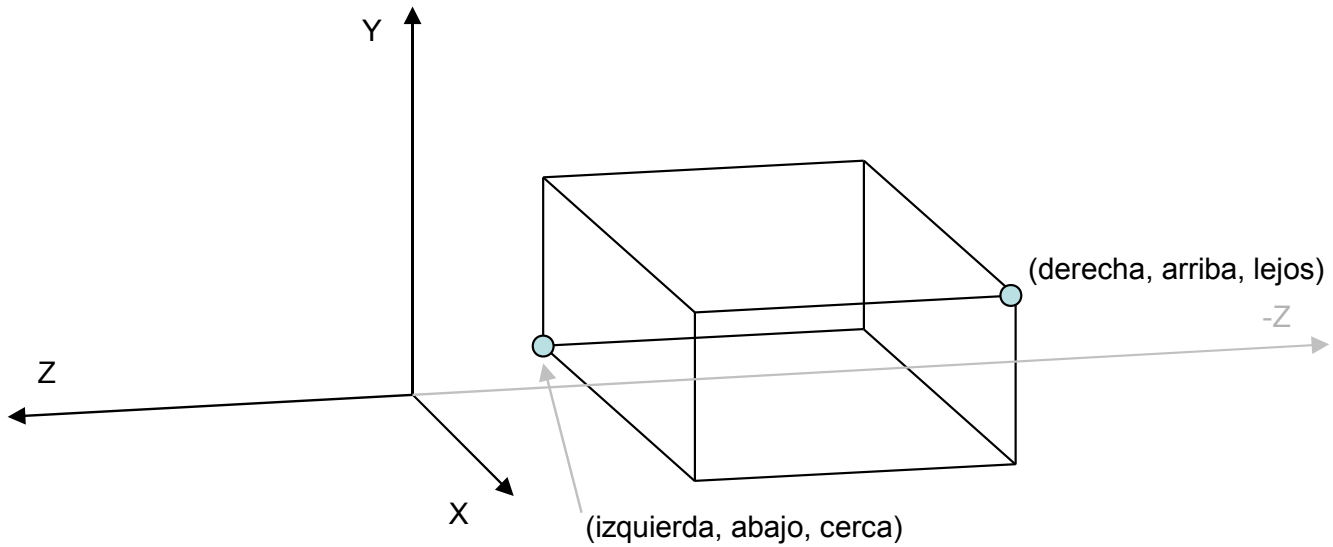




Proyección Paralela

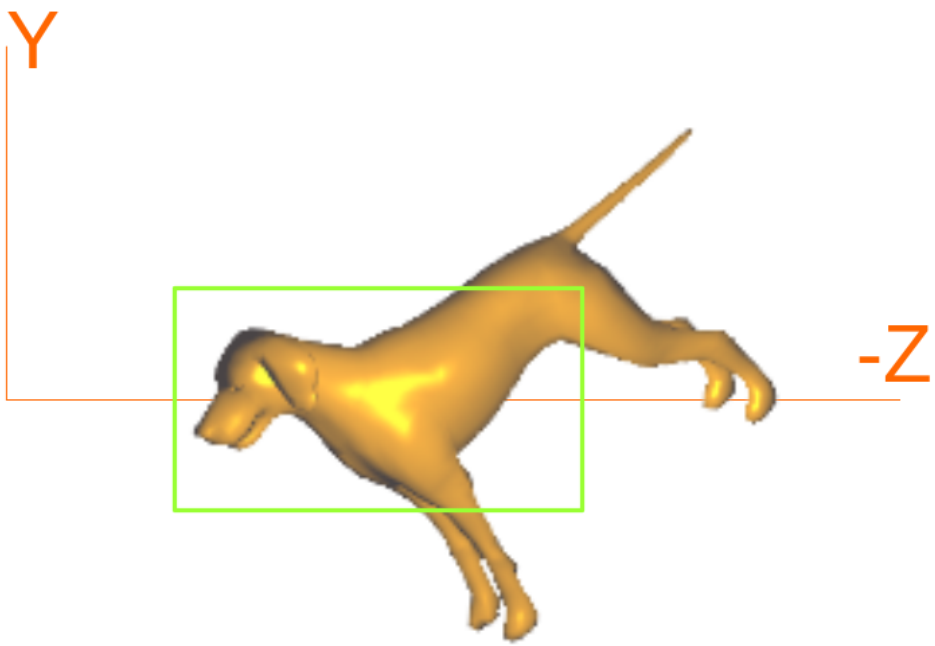
Propiedades

- Los rayos de proyección son paralelos entre sí e intersectan de forma perpendicular con el plano de proyección.
- El volumen de la vista tiene forma de caja que se alinea con los ejes de coordenadas.



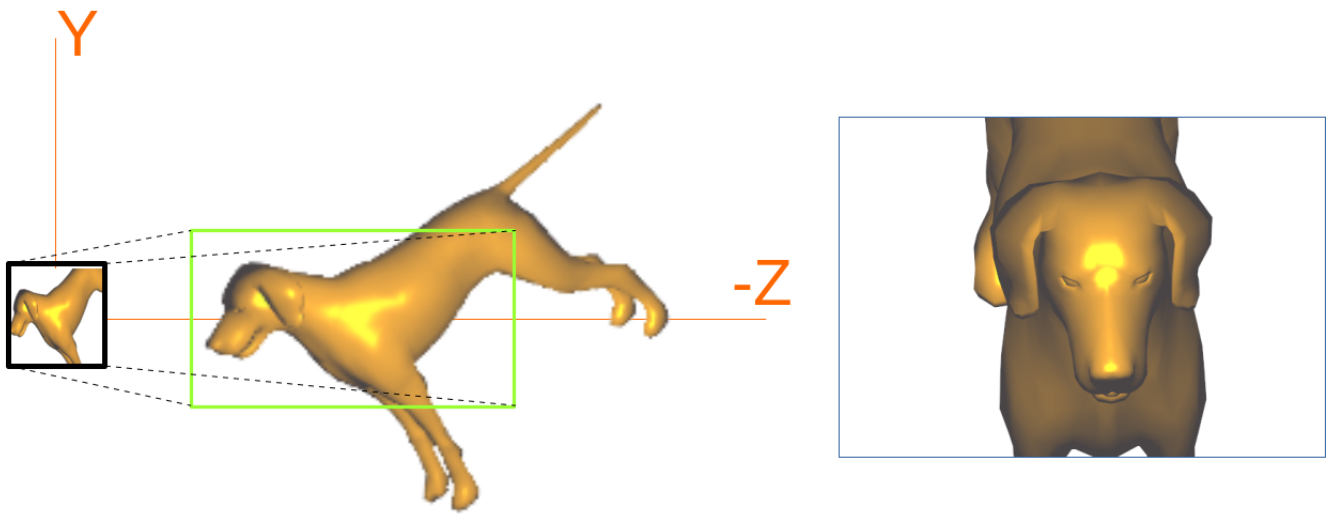
Proyección Paralela

1. Define tu propio volumen de la vista



Proyección Paralela

2. Transforma tu volumen en el canónico



Matriz de transformación paralela

¿Cómo conviertes dicha caja en el volumen canónico de la vista?

- Necesitas transformaciones de traslación y escalado.
- En OpenGL (un cubo de lado dos, etc.):

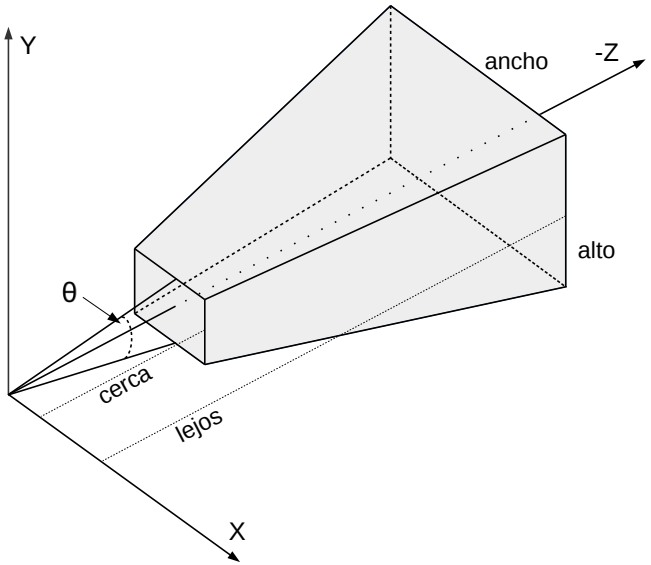
$$M_{par} = \begin{pmatrix} \frac{2}{derecha-izquierda} & 0 & 0 & -\frac{derecha+izquierda}{derecha-izquierda} \\ 0 & \frac{2}{arriba-abajo} & 0 & -\frac{arriba+abajo}{arriba-abajo} \\ 0 & 0 & \frac{2}{lejos-cerca} & -\frac{lejos+cerca}{lejos-cerca} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2)$$

- La librería GLMATRIX nos ayuda:
  - mat4.ortho (out, izquierda, derecha, abajo, arriba, cerca, lejos)

Proyección Perspectiva

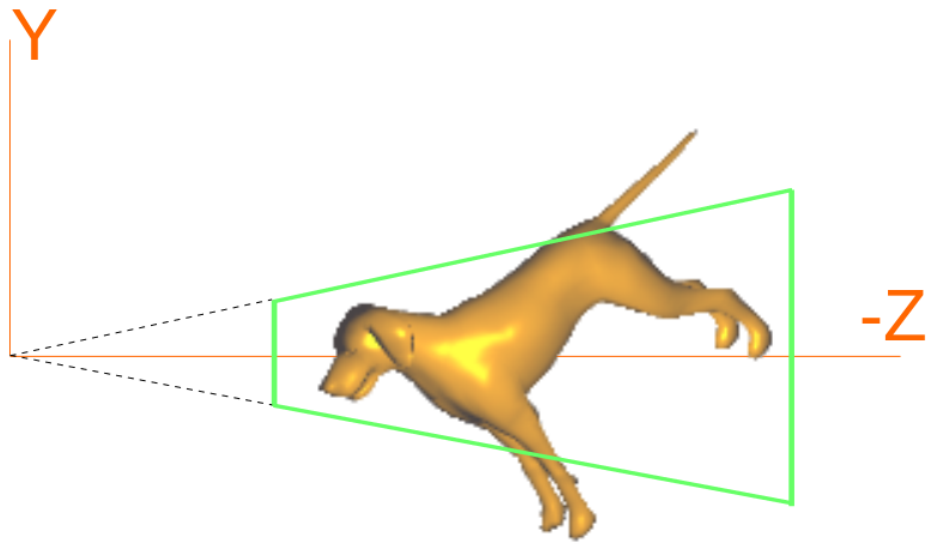
Propiedades

- Los rayos de proyección parten todos ellos desde la posición del observador.
- El volumen de la vista tiene forma de pirámide, definida por cuatro parámetros: los planos cerca y lejos, el ángulo θ en la dirección Y y la relación de aspecto de la base de la pirámide *ancho*/*alto*.



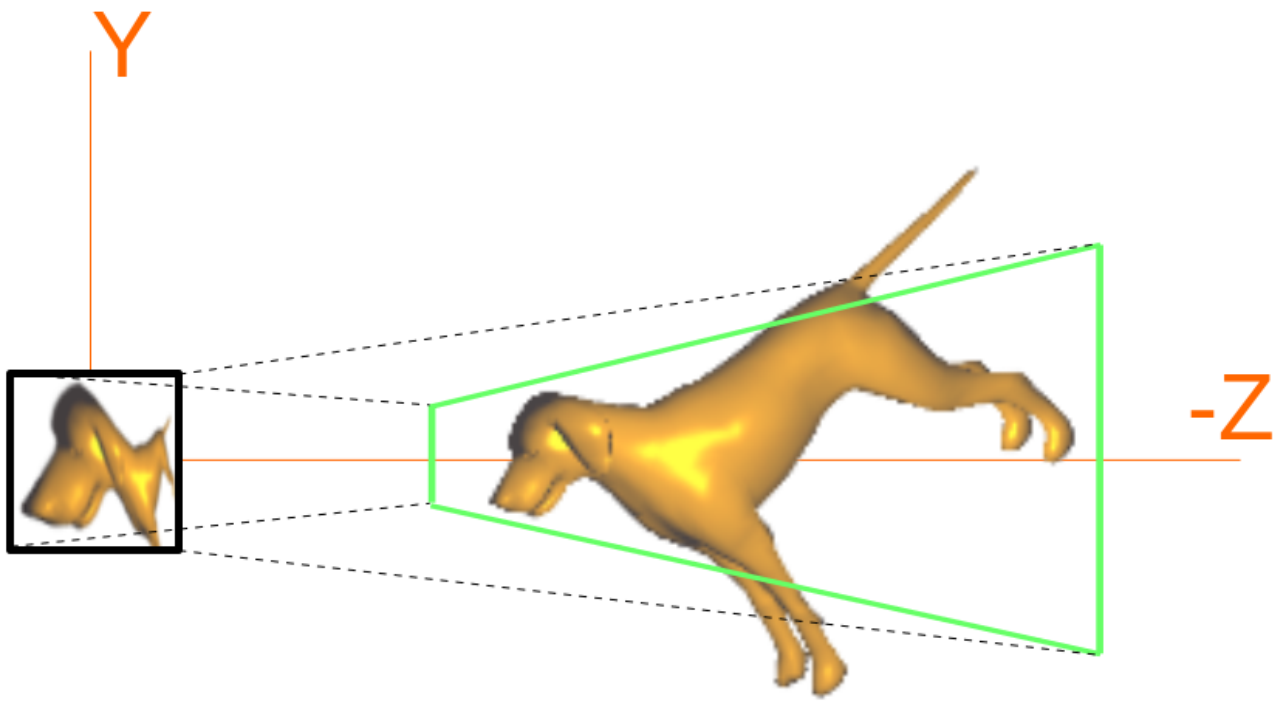
Proyección Perspectiva

1. Define tu propio volumen de la vista



Proyección Perspectiva

2. Transforma tu volumen en el canónico



Matriz de transformación perspectiva

¿Cómo conviertes dicha pirámide truncada en el volumen canónico de la vista?

- En OpenGL (un cubo de lado dos, etc.):

$$M_{per} = \begin{pmatrix} \frac{1}{aspect \cdot tan(\theta/2)} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\theta/2)} & 0 & 0 \\ 0 & 0 & \frac{lejos+cerca}{cerca-lejos} & \frac{2 \cdot lejos \cdot cerca}{cerca-lejos} \\ 0 & 0 & -1 & 0 \end{pmatrix} \quad (3)$$

  - La librería glm nos ayuda:
    - mat4.perspective (out, θ, ancho/alto, cerca, lejos)
- Es una transformación no afín, ¿qué implica esto?
- División perspectiva: cada vértice es dividido por su w.

Hoy veremos...

Transformación al Área de Dibujo

Descripción

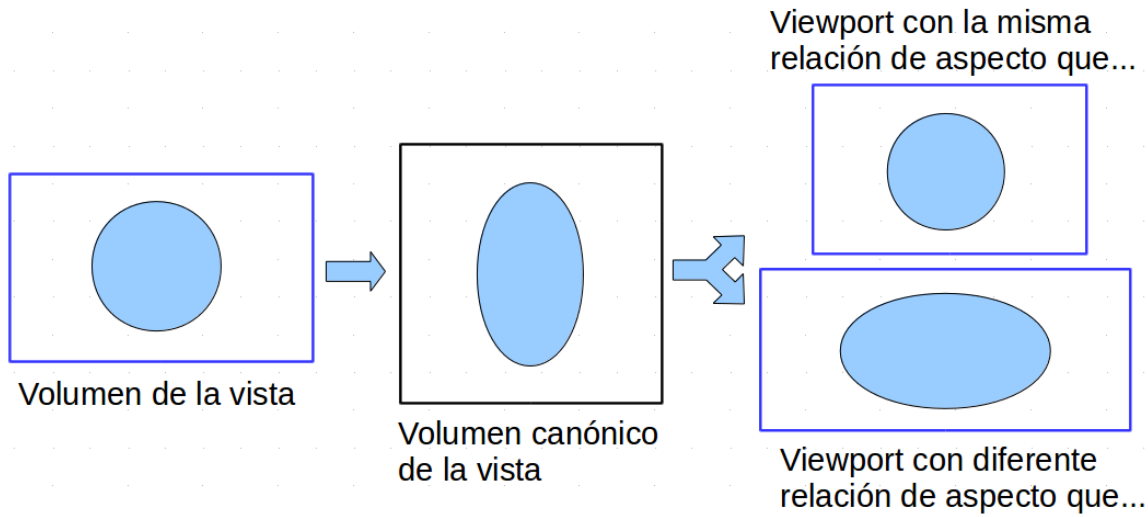
- El área de dibujo, también conocido por *viewport*, es la parte de la ventana de la aplicación donde se muestra la vista 2D.
- Consiste en mover el resultado de la proyección a dicha área.
- La escena a visualizar reside en el volumen canónico de la vista.
- Si n_x y n_y son el ancho y el alto del *viewport* en píxeles, y o_x y o_y son el píxel de la esquina inferior izquierda del *viewport* en coordenadas de ventana, la matriz de transformación es la siguiente:

$$M_{vp} = \begin{pmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} + o_x \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} + o_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4)$$

Transformación al Área de Dibujo

¿Quién realiza esta transfromación?

- En OpenGL, esta transformación la realiza el sistema gráfico.
- Pero hemos de especificar la ubicación del *viewport*:
  - gl.viewport (x, y, *ancho*, *alto*)
- Si cambia el tamaño del canvas....¿qué ocurre?
- Es importante que la relación de aspecto del *viewport* sea igual a la relación de aspecto del volumen de la vista, ¿por qué?



Hoy veremos...

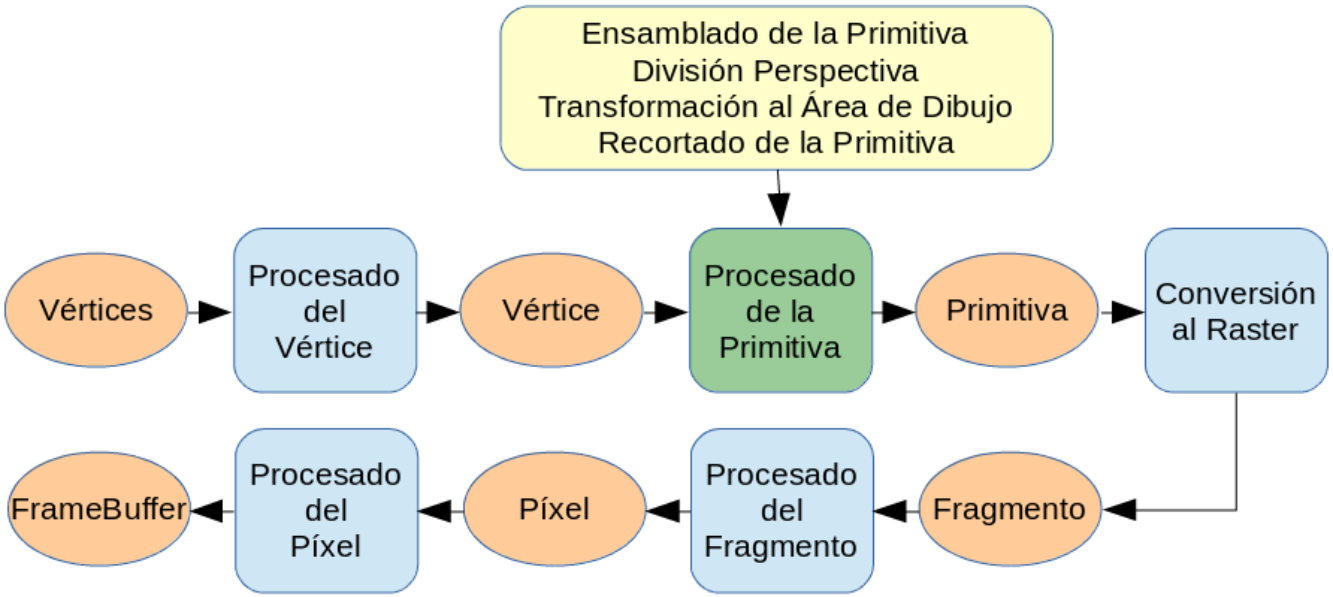
Viendo en OpenGL

Lo que el programador debe hacer:

- Construir la matriz de transformación de la cámara y la matriz de proyección.
- La librería glMatrix proporciona diversas funciones:
  - mat4.lookAt (out, p, i, UP)
  - mat4.ortho (out, izquierda, derecha, abajo, arriba, cerca, lejos)
  - mat4.perspective (out, θ, ancho/alto, cerca, lejos)
- Suministrar ambas matrices al procesador de vértices o ...
- Especificar la ubicación del *viewport*:
  - gl.viewport (x, y, *ancho*, *alto*)

Lo que la funcionalidad fija de la GPU hace

- Entre el procesador de vértices y el de fragmentos:
  - Los elementos, o partes de ellos, que queden fuera del volumen de la vista son eliminados en la etapa de recortado.
  - Cada vértice es dividido por w, proceso denominado *división perspectiva*.
  - La transformación al área de dibujo.



Ejercicios

- Utilizando un volumen de la vista correspondiente al de la proyección paralela, ¿qué le ocurre al tamaño de la proyección de la escena si alejas la cámara de la escena?
- ¿En qué Shader operas con la matriz de transformación al área de dibujo?
- Si P es la matriz de proyección, C la de la cámara y M la del modelo, ¿es la matriz T = M*C*P por la que debes operar cada vértice (v' = T * v) ?
- Piensa qué harías para ofrecer varias vistas de la misma escena cada una de ellas en una parte distinta del canvas.
- Discute sobre la veracidad de la siguiente frase: si tras aplicar la transformación de la cámara no realizas la transformación de proyección, únicamente puedo obtener una proyección de la parte de la escena que resida en el interior del volumen canónico de la vista, pero aún así el resultado de la proyección podrá ser tanto paralela como perspectiva.

Hoy veremos...

Eliminación de partes ocultas

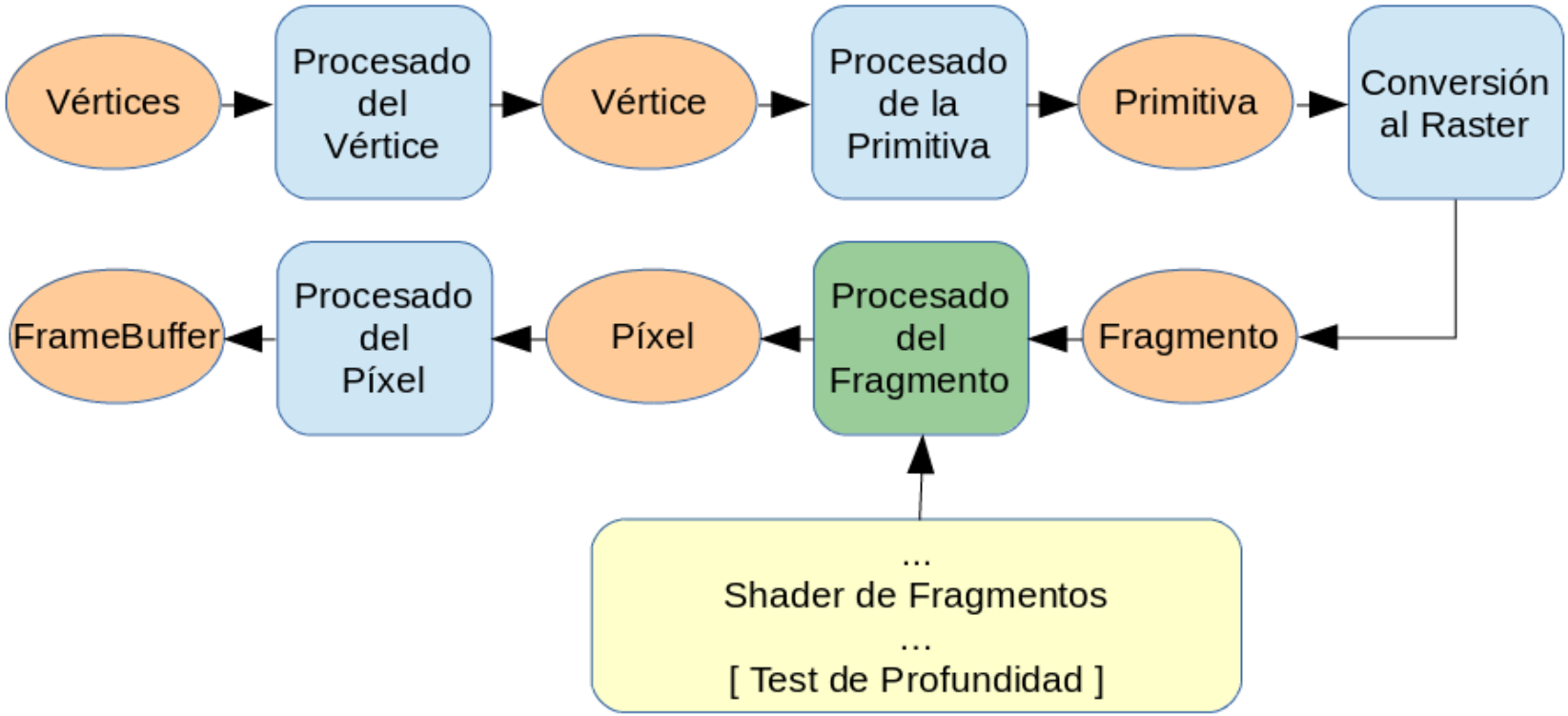
Descripción

- Consiste en determinar qué primitivas de la escena son tapadas por otras primitivas desde el punto de vista del observador.
- El algoritmo más utilizado es el algoritmo conocido como *Z-Buffer*.
  - Requiere de un buffer denominado *buffer de profundidad*.
  - No importa el orden en que se pinten las primitivas.
  - Sí es muy importante que el buffer de profundidad se inicialice siempre al valor de profundidad máxima antes de ...

Listado 1: Algoritmo del Z-Buffer

```
if (pixel.z < bufferProfundidad(x,y).z)
{
  bufferProfundidad(x,y).z = pixel.z;
  bufferColor(x,y).color   = pixel.color;
}
```

¿Dónde ocurre?



El test de profundidad es **opcional**. En el caso de estar habilitado lo realizará la GPU para cada fragmento siempre después de ejecutar el Shader de fragmentos.

Lo que la funcionalidad fija de la GPU hace

- En la etapa de procesado del fragmento:
  - Test de profundidad: la GPU comprueba la profundidad de cada fragmento.
  - Requiere ayuda del programador:
    1. Habilitar la operación del test de profundidad: gl.enable (gl.DEPTH_TEST);
    2. Inicializar el buffer a la profundidad máxima antes de comenzar a dibujar: gl.clear (... | gl.DEPTH_BUFFER_BIT);