Informática Gráfica.
Sesión 6: Transformación de vértices.

Carlos Ureña, Sept 2025.
Dept. Lenguajes y Sistemas Informáticos.
Universidad de Granada.

Índice

Sección **1.**

Espacios de coordenadas y transformaciones.

Introducción.

El término **cauce gráfico** (***graphics pipeline***) se suele usar para referirnos al conjunto de pasos de cálculo que se realizan para visualizar polígonos en el contexto del **algoritmo de Z-buffer**

- El algoritmo de Z-buffer se usa para presentar polígonos incluyendo **eliminación de partes ocultas** (EPO) en 3D (es decir: lograr presentar únicamente las partes visibles de los polígonos que se dibujan).
- OpenGL, DirectX y otras librerías 3D usan Z-buffer.
- Estos pasos **se implementan en hardware** en las tarjetas gráficas modernas (GPUs: *Graphics Processing Units*).
- Estos pasos **no se aplican en otros algoritmos** de visualización y EPO en 3D, como por ejemplo en Ray-tracing.

Pasos del cauce gráfico.

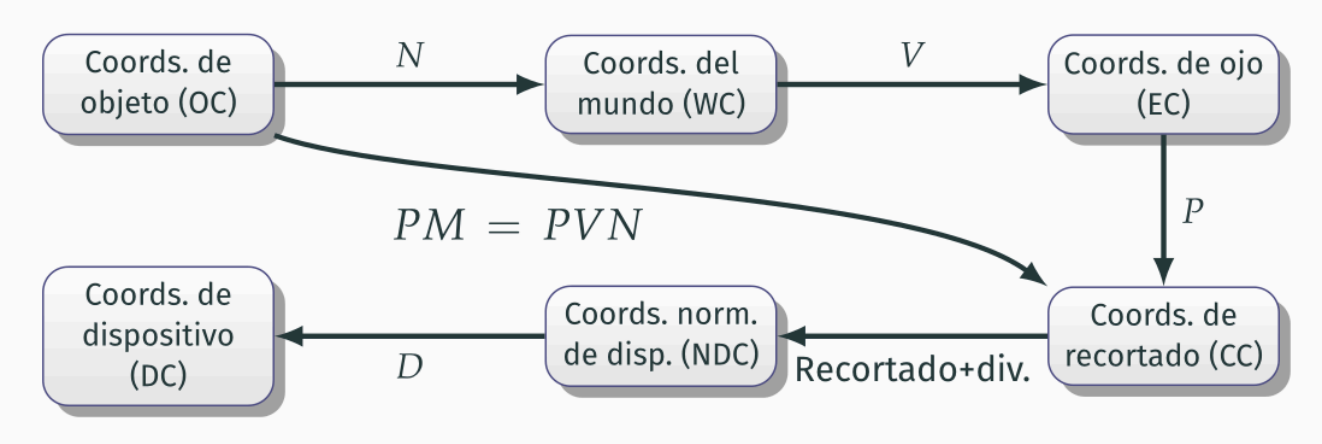
Los pasos del cauce gráfico suelen implementarse en secuencia, cada paso obtiene datos del anterior, los transforma de alguna manera y los entrega al siguiente paso. Los pasos (muy resumidos) son:

1. **Transformación** de coordenadas de vértices: cálculo de donde se proyecta en pantalla cada vértice.
2. **Recortado**: eliminación de partes de polígonos fuera de la zona visible.
3. **Rasterización y EPO**: cálculo de los píxeles donde se proyecta un polígono.
4. **Iluminación y texturización**: cálculo del color de cada pixel donde se proyecta un polígono.

la transformación y el recortado se pueden mezclar de diversas formas, ambos pasos son necesariamente previos a los otros dos, que también se pueden combinar de varias formas entre ellos

Esquema de la transformación y recortado

En estas etapas del cauce gráfico, esencialmente los datos que se transforman son coordenadas de vértices y conectividad entre ellos. El esquema es el siguiente:



este esquema corresponde al recortado en CC (hay otras posibilidades, esta es la mejor).

Sistemas de coordenadas (1/2)

El cauce gráfico implementado en las *engines* y APIs de rasterización contempla una secuencia de 6 sistemas de coordenadas distintos. Las cordenadas de los vértices **son convertidas desde el primero al último**

La transformación de coordenadas entre cada sistema de referencia y el siguiente en la lista se hace mediante una matriz 4x4 específica de esa etapa (y en algún caso algo más).

1. **Coordenadas de objeto o maestras** (*Master coordinates*, OC): las coordenadas son distancias relativas a un sistema de referencia específico o distinto de cada objeto, que se crea en este espacio. En Godot, son las coordenadas de los vértices, relativas al nodo donde se encuentran.
2. **Coordenadas del mundo** (*World coordinates*, WC): son distancias relativas a un sistema de referencia único, común para todos los objetos de una escena.

Sistemas de coordenadas (2/2)

Los otros cuatro sistemas de coordenadas son:

3. **Coordenadas de cámara (o de ojo o de vista)** (*Eye coordinates or view-coordinates*, EC): son distancias relativas a un sistema de referencia posicionado y alineado con la *cámara virtual* en uso.
4. **Coordenadas de recortado** (*Clip coordinates*, CC): son distancias normalizadas (los vértices visibles están en el rango $[-1, +1]$ en los tres ejes), y con $w \neq 1$, relativas a un sistema asociado al rectángulo que forma la imagen en pantalla.
5. **Coordenadas normalizadas de dispositivo** (*Normalized device coordinates*, NDC): son similares a las coordenadas de recortado, pero con $w = 1$, y con todos los vértices en $[-1, +1]$ (los que están fuera se han eliminado).
6. **Coordenadas de dispositivo** (*Device coordinates*, DC): similares a NDC, pero ahora con las componentes X e Y en unidades de pixels.

Matrices de transformación

Las matrices de transformación (4x4) involucradas permiten convertir coordenadas en un sistema de coordenadas a coordenadas en otro:

- **La matriz de modelado y vista** (***modelview***) M, compuesta de:
  - **Matriz de modelado** N: convierte de OC a WC
  - **Matriz de vista** V: convierte de WC a EC
- La **matriz de proyección** P: convierte de EC a CC. (recibe coordenadas con $w = 1$, pero produce coordenadas en general con $w \neq 1$)
- La **matriz del viewport** D: convierte de NDC a DC (depende de la resolución de la imagen en pantalla y de la zona de esta donde se visualiza).

Las coordenadas de dispositivo (con $w = 1$ y en unidades de pixels) se usan como entrada para las siguientes etapas del cauce gráfico (rasterización, EPO, iluminación y texturas)

Sección **2.**

Transformación de vista

1. La matriz de vista 3D
2. Transformación de vista 3D en Godot

La transformación de vista.

La **transformación de vista** es el cálculo que permite convertir **coordenadas de mundo** (*world coordinates*, WCC) en **coordenadas de ojo** (o **de cámara**) (*eye or camera coordinates*, ECC).

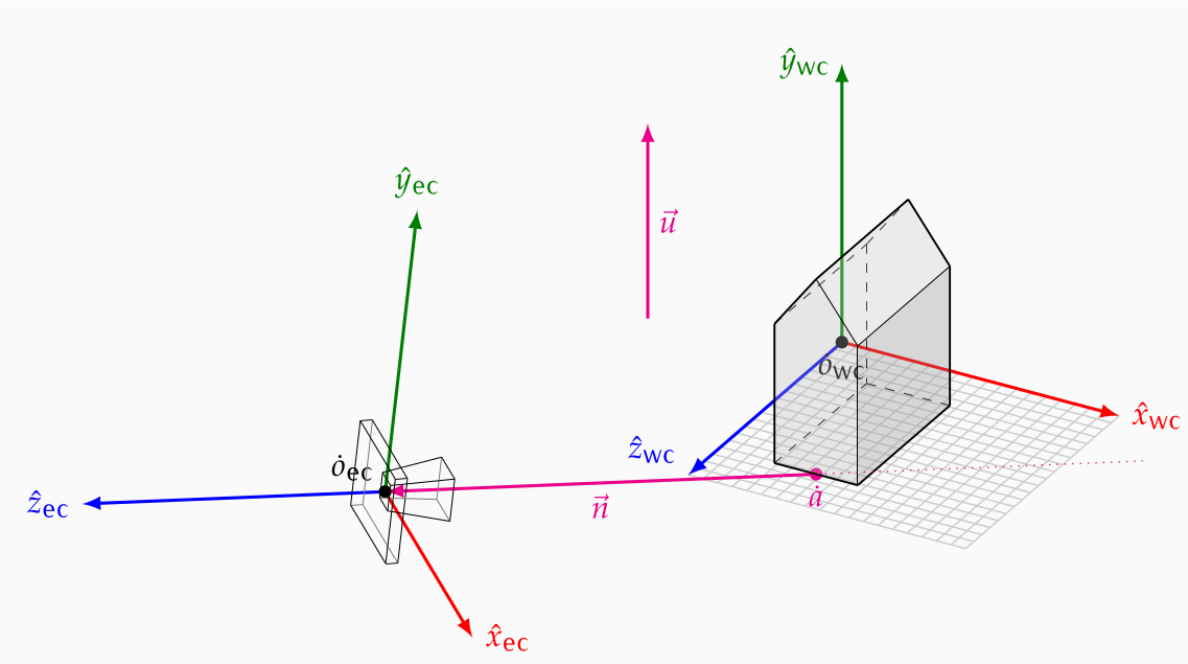
- Se usa un marco de referencia cartesiano $\mathcal{V} = \{\hat{x}_{\text{ec}}, \hat{y}_{\text{ec}}, \hat{z}_{\text{ec}}, \dot{o}_{\text{ec}}\}$, llamado **marco de cámara** (o **de vista**), que está posicionado y alineado con la cámara virtual. Las **coordenadas de cámara** (o **de vista**) de un punto son las coordenadas de ese punto en el marco $\mathcal{V}$.
- Para hacer la conversión de coordenadas se debe usar la **matriz de vista**, la llamamos V.
- Puesto que el marco de coordenadas de mundo $\mathcal{W}$ es cartesiano y $\mathcal{V}$ también, la matriz V puede construirse fácilmente como la composición de una matriz de traslación por $\dot{o}_{\text{wc}} - \dot{o}_{\text{ec}}$ seguida de una matriz (ortonormal) de rotación, que tiene las coordenadas de mundo de $\hat{x}_{\text{ec}}, \hat{y}_{\text{ec}}$ y $\hat{z}_{\text{ec}}$ en **sus filas**.

Subsección **2.1.**

La matriz de vista 3D

El marco de coordenadas de vista o de cámara.

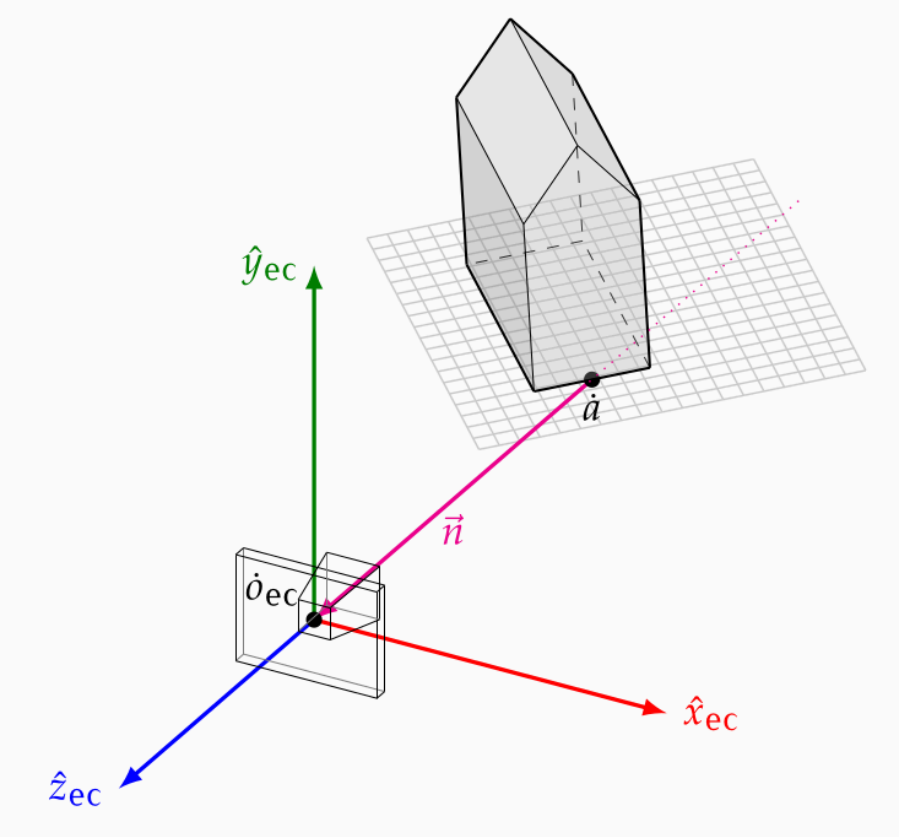
El **marco de coordenadas de vista** se construye usando el punto $\dot{a}$, el punto $\dot{o}_{\text{ec}}$, el vector $\vec{u}$ y el vector $\vec{n}$. El observador está situado en $\dot{o}_{\text{ec}}$, y mira en la dirección de la rama negativa del eje Z ($-\hat{z}_{\text{ec}}$):



Escena posterior a transformación de vista.

Aquí vemos la escena anterior una vez transformada por la matriz de vista V.

Tras aplicar la matriz, todas las coordenadas son relativas al marco de vista $\mathcal{V}$.



Cálculo del marco de vista.

El marco de referencia de vista $\mathcal{V}$, se define a partir de los siguientes parámetros:

- $\dot{o}_{\text{ec}} =$ **posición del observador**: es el punto del espacio foco de la proyección, donde estaría situado el observador ficticio que contempla la escena (*projection reference point*, PRP)
- $\vec{n} =$ **vector normal**: vector libre perpendicular al *plano de visión* (plano ficticio donde se proyecta la imagen perpendicular al *eje óptico* de la camara virtual) (*view plane normal*, VPN).
- $\dot{a} =$ **punto de atención**: punto en el eje óptico, por tanto se va a proyectar en el centro de la imagen (*look-at point*).
- $\vec{u} =$ **vector hacia arriba**: es un vector libre que indica una dirección que el observador ve proyectada en vertical en la imagen (apuntando hacia arriba) (*view-up vector*, VUP)

De los tres parámetros $\dot{o}_{\text{ec}}$, $\vec{n}$ y $\dot{a}$ solo hay que especificar dos, ya que no son independientes, se cumple: $\dot{o}_{\text{ec}} = \dot{a} + \vec{n}$.

Cálculo del marco de vista.

A partir de esos parámetros se obtiene se calculan los **versores del marco de vista**:

$$\hat{z}_{\text{ec}} = \frac{\vec{n}}{\|\vec{n}\|} \qquad (\text{paralelo a } \vec{n} \text{ (VPN), normalizado})$$

$$\hat{x}_{\text{ec}} = \frac{\vec{u} \times \vec{n}}{\|\vec{u} \times \vec{n}\|} \qquad (\text{perpendicular a } \vec{n} \text{ (VPN) y } \vec{u} \text{ (VUP), normalizado})$$

$$\hat{y}_{\text{ec}} = \hat{z}_{\text{ec}} \times \hat{x}_{\text{ec}} \qquad (\text{perpendicular a los otros dos, normalizado})$$

Para que este cálculo pueda hacerse, los vectores $\vec{u}$ y $\vec{n}$ no pueden ser nulos ni paralelos, de forma que siempre $\|\vec{u} \times \vec{n}\| > 0$.

Coordenadas del mundo del marco de vista sC

El marco de referencia de vista se suele representar en memoria usando las coordenadas del mundo de los vectores y el punto (coordenadas relativas a $\mathcal{W}$), es decir:

$$\begin{aligned}
\hat{x}_{\text{ec}} &= \mathcal{W}\big(a_x, a_y, a_z, 0\big)^T &= \mathcal{W}\mathbf{x}_{\text{ec}} \\
\hat{y}_{\text{ec}} &= \mathcal{W}\big(b_x, b_y, b_z, 0\big)^T &= \mathcal{W}\mathbf{y}_{\text{ec}} \\
\hat{z}_{\text{ec}} &= \mathcal{W}\big(c_x, c_y, c_z, 0\big)^T &= \mathcal{W}\mathbf{z}_{\text{ec}} \\
\dot{o}_{\text{ec}} &= \mathcal{W}\big(o_x, o_y, o_z, 1\big)^T &= \mathcal{W}\mathbf{o}_{\text{ec}}
\end{aligned}$$

Estas coordenadas se calculan a partir de las coordenadas de mundo (en el marco $\mathcal{W}$) de los vectores $\vec{u}$,$\vec{n}$ y el punto $\dot{o}$ como hemos visto. Esas coordenadas son $\mathbf{u}$, $\mathbf{n}$ y $\mathbf{o}$, respectivamente.

La matriz V se puede construir directamente a partir de ellas.

Conversión de coordenadas: de vista a mundo

La **matriz de vista** V es la matriz que convierte desde coordenadas de mundo (en $\mathcal{W}$) hacia coordenadas de vista (en $\mathcal{V}$).

Su inversa V^{-1} hace la conversión contraria, **de vista a mundo**, y se puede escribir explícitamente V^{-1}, con las tuplas de coordenadas de mundo $\mathbf{x}_{\text{ec}}, \mathbf{y}_{\text{ec}}, \mathbf{z}_{\text{ec}}$ y $\mathbf{z}_{\text{ec}}$ dispuestas **en sus columnas**:

$$V^{-1} = \begin{pmatrix} a_x & b_x & c_x & o_x \\ a_y & b_y & c_y & o_y \\ a_z & b_z & c_z & o_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \overbrace{\begin{pmatrix} 1 & 0 & 0 & o_x \\ 0 & 1 & 0 & o_y \\ 0 & 0 & 1 & o_z \\ 0 & 0 & 0 & 1 \end{pmatrix}}^{T_{\mathbf{o}_{\text{ec}}}} \overbrace{\begin{pmatrix} a_x & a_y & a_z & 0 \\ b_x & b_y & b_z & 0 \\ c_x & c_y & c_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}^{R}$$

Aquí se ha descompuesto V^{-1} en una **matriz ortonormal** R de alineamiento, seguida de una traslación al origen $\mathbf{o}_{\text{ec}}$ de $\mathcal{V}$:

$$V^{-1} = T_{\mathbf{o}_{\text{ec}}} R$$

Cálculo de la matriz de vista

A partir de lo anterior es fácil obtener una expresión explícita de la **matriz de vista** V. Se puede escribir usando la composición inversa de la anterior:

$$V = \left(T_{\mathbf{o}_{\text{ec}}} R\right)^{-1} = R^{-1} T_{\mathbf{o}_{\text{ec}}}^{-1} = R^T T_{-\mathbf{o}_{\text{ec}}}$$

Donde se usa el hecho de que R es ortonormal. Expandiendo las matrices:

$$V = \overbrace{\begin{pmatrix} a_x & a_y & a_z & 0 \\ b_x & b_y & b_z & 0 \\ c_x & c_y & c_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}^{R^T} \overbrace{\begin{pmatrix} 1 & 0 & 0 & -o_x \\ 0 & 1 & 0 & -o_y \\ 0 & 0 & 1 & -o_z \\ 0 & 0 & 0 & 1 \end{pmatrix}}^{T_{-\mathbf{o}_{\text{ec}}}} = \begin{pmatrix} a_x & a_y & a_z & d_x \\ b_x & b_y & b_z & d_y \\ c_x & c_y & c_z & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

donde:

$$d_x = -\mathbf{x}_{\text{ec}} \cdot \mathbf{o}_{\text{ec}} \qquad d_y = -\mathbf{y}_{\text{ec}} \cdot \mathbf{o}_{\text{ec}} \qquad d_z = -\mathbf{z}_{\text{ec}} \cdot \mathbf{o}_{\text{ec}}$$

Subsección **2.2.**

Transformación de vista 3D en Godot

La clase *Camera3D* de Godot

En las aplicaciones 3D de Godot, la clase `Camera3D` es la encargada de **definir la matriz de vista** para el *viewport* (la zona de pantalla donde se visualiza cada frame).

- Es necesario situar una instancia de `Camera3D` (derivada de `Node3D`) en el árbol de escena.
- La instancia se registra como la cámara activa del primer viewport que se encuentra ascendiendo desde el nodo de la cámara hacía la raíz, si no hay ninguno, se registra en el viewport por defecto.
- Tiene una propiedad `transform` (una matriz, cuyas columnas son los ejes y origen de un marco del nodo). Ese marco de referencia es el **marco de vista** $\mathcal{V}$
- La matriz `transform` permite transformar coordenadas de vista en coordenadas de mundo.
- La **matriz inversa** de `transform`, por tanto, es V: **permite transformar coordenadas de mundo en coordenadas de vista**.

Situar y apuntar la cámara

En Godot se puede especificar las coordenadas de mundo de $\mathbf{a}$ (punto de atención, `p_atencion`), el vector $\mathbf{u}$ (vector hacia arriba `v_arriba`) y el origen $\mathbf{o}$ (punto `p_origen`) para apuntar una cámara (objeto `cam`) hacia dicho punto, para ello modificamos su sistema de referencia, se puede hacer de varias formas:

- Modificando la propiedad `transform` (de tipo `Transform3D`) del nodo cámara:

```
cam.transform.origin = p_origen
cam.transform = cam.transform.looking_at( p_atencion, v_arriba )
```

- Modificando el nodo directamente, con el método `look_at` de `Node3D`:

```
cam.position = p_origen
cam.look_at( p_atencion, v_arriba )
```

- Usando el método `look_at_from_position` de `Node3D`:

```
cam.look_at_from_position( p_origen, p_atencion, v_arriba )
```

Desplazamientos *locales* una cámara

En Godot se puede desplazar la cámara en las direcciones de sus propios ejes, lo cual es muy típico en aplicaciones tipo *first person shooter*.

- En la aplicación el efecto es que el observador se mueve ciertas distancias **en las direcciones de los ejes del marco de cámara.**
- Puede ser hacia delante (eje Z negativo del marco de cámara), hacia atrás (Z+), hacia su izquierda (X-), su derecha (X+), arriba (Y+) y abajo (Y-).
- Para ello se puede usar el método `translate_object_local` de `Node3D`, aplicado al nodo cámara, lo cual desplaza el nodo en su propio sistema de referencia (compone traslaciones por la derecha):

  ```
  cam.translate_object_local( Vector3( delta_x, delta_y, delta_z ) )
  ```

  donde `delta_x`, `delta_y` y `delta_z` son las distancias a desplazar en las direcciones de los ejes X, Y y Z del marco de vista.

Rotaciones *locales* de una cámara

También se puede rotar la cámara entorno a sus propios ejes, (de nuevo es muy típico en aplicaciones tipo *first person shooter*).

- En la aplicación el efecto es que el observador rota su punto de vista entorno a sus propios ejes.
- Puede ser rotar su vista hacia arriba y hacia abajo (entorno al eje X del marco de cámara), girar su vista hacia la izquierda y hacia la derecha (entorno al eje Y), y hacer un giro lateral (entorno al eje Z).
- Para ello se puede usar el método `rotate_object_local` de `Node3D`, aplicado al nodo cámara, lo cual rota el nodo en su propio sistema de referencia (compone rotaciones por la derecha):

  ```
  cam.rotate_object_local( eje_rotacion, angulo_rad )
  ```

  donde `eje_rotacion` es un vector unitario que indica el eje de rotación en el marco de vista, y `angulo_rad` es el ángulo de rotación en radianes.

Cámara orbital

Otra posibilidad es el uso de una **cámara orbital**, la cual *orbita* alrededor de un punto, usando dos ángulos de rotación y una distancia a dicho punto:

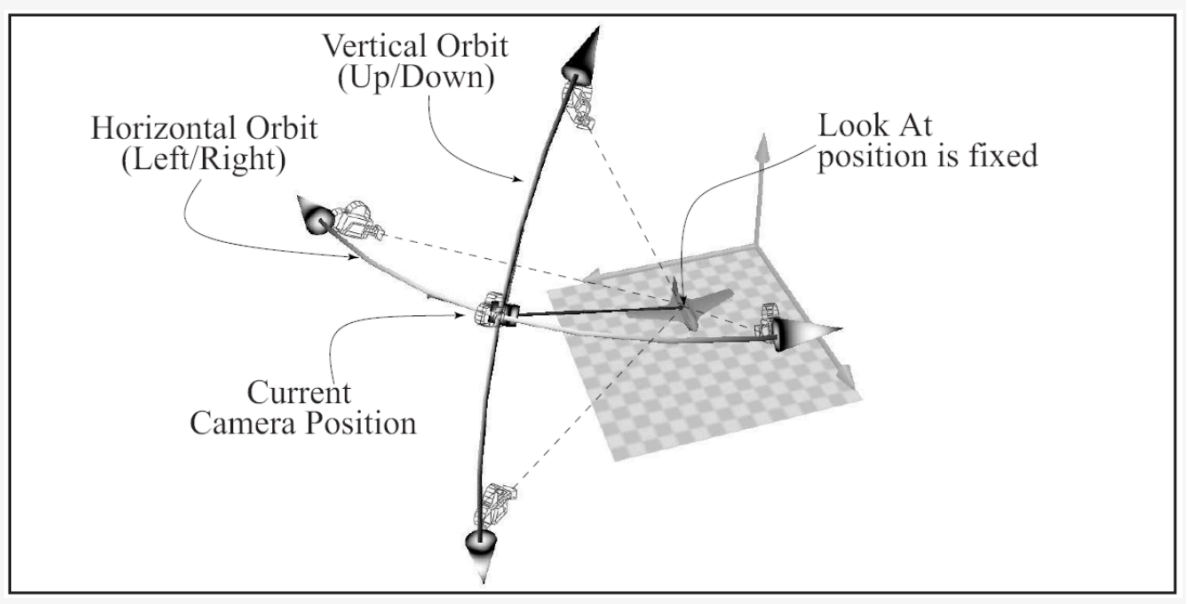
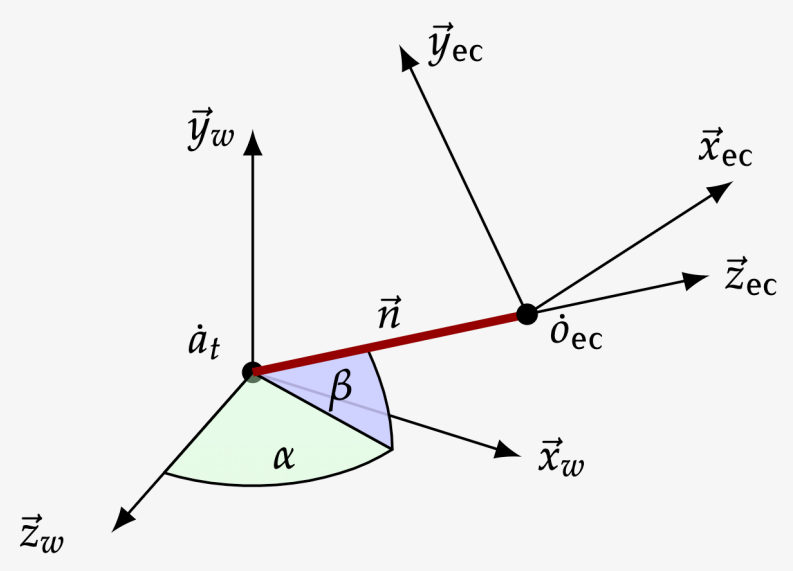


Figura obtenida de: *K.Sung*, *P.Shirley*, *S.Baer* **Essentials of Interactive Computer Graphics: Concepts and Implementation.** Ed. Routledge, 2008. Ver: página del libro.

Marco de vista de la cámara orbital

El marco de vista está desplazado una distancia en el eje Z del mundo (a partir del punto de atención $\dot{a}_t$), y luego rotado usando dos ángulos α y β (entorno a los ejes Y y X, respectivamente) que determinan el vector $\vec{n}$:



Cámara orbital en Godot

En Godot, una posibilidad es usar el siguiente código en el nodo `Camera3D` en uso, que actualiza el marco de vista asignando su propiedad `transform`:

```
var ahr  := ((45.0+float(dxy.x))*2.0*PI)/360.0 ## ang.horiz.radi.
var avr  := ((30.0+float(dxy.y))*2.0*PI)/360.0 ## ang.vert.radi.
var tras := Transform3D().translated( Vector3( 0.0, 0.0, dz))
var rotx := Transform3D().rotated( Vector3.RIGHT, -avr )
var roty := Transform3D().rotated( Vector3.UP, ahr )
transform = roty*rotx*tras   ## actualiza transform del nodo cámara
```

Los datos de entrada son `dxy` y `dz`

- `dxy` son dos enteros (`Vector2i`) con los ángulos de rotación en grados (inicialmente a cero). A partir de ahí se calculan `ahr` (α) y `avr` (β), los ángulos en radianes.
- `dz` es un flotante con la distancia al origen (es decir, es $\|\vec{n}\|$)

Estos tres valores se modifican interactivamente con el ratón y el teclado.

Sección **3.**

Transformación de proyección

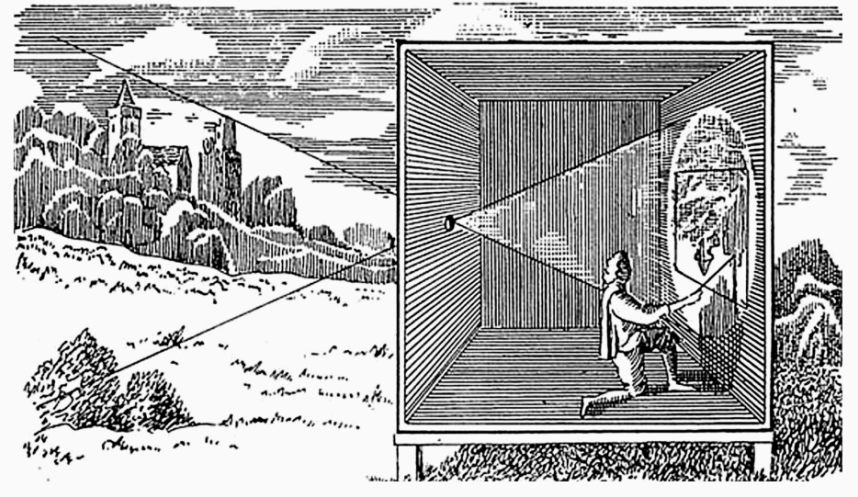
1. Tipos de proyección en 3D.
2. El *view-frustum*. Parámetros de proyección.
3. La matriz de proyección.

Subsección **3.1.**

Tipos de proyección en 3D.

Introducción

La **transformación de proyección 3D** (perspectiva) emula la proyección que ocurre idealmente en una *cámara oscura*, sobre la pared opuesta a la apertura. Es similar a lo que ocurre en una cámara de fotografía, al proyectarse la escena sobre el sensor.



Grabado de una *Camera Obscura*, por Athanasius Kircher en *Ars Magna Lucis et Umbrae* (1645).
www.essentialvermeer.com/camera_obscura/co_one.html

El *plano de visión*. Tipos de proyección

Los vértices se *proyectan* sobre un plano alineado con el sistema de referencia de la cámara:

- Dicho plano se denomina **plano de visión** (***viewplane***), es siempre perpendicular al eje Z del marco de vista.

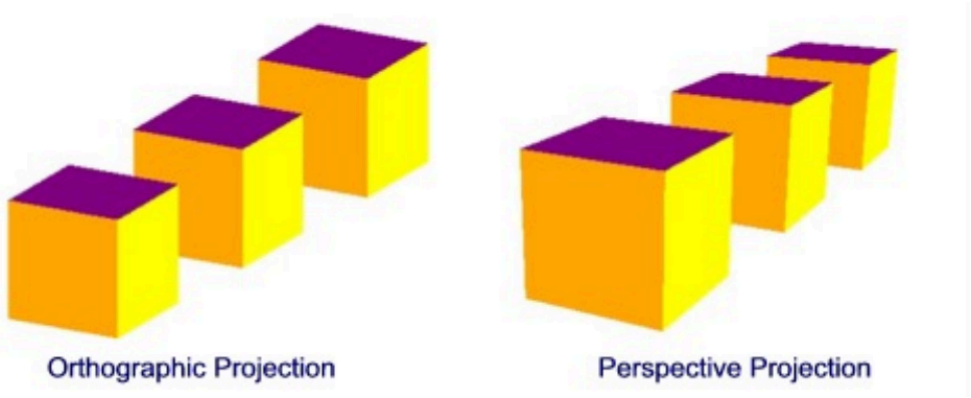
La proyección puede ser de dos tipos

Proyección perspectiva: los vértices se proyectan sobre el plano de visión usando líneas que van desde cada punto al origen del marco de coordenadas de la cámara (a esas líneas se les llama **proyectores**, el origen actua como **foco** de la proyección). La coordenada Z del plano de visión debe ser estrictamente positiva.

Proyección ortográfica: (o **paralela**) los proyectores son todos paralelos al eje Z. El plano de visión puede estar situado en cualquier valor de Z. Es un caso límite de la perspectiva, con el foco infinitamente alejado de la escena.

Comparación de proyecciones

Aunque ninguna de las dos formas de proyección es igual al comportamiento del sistema visual humano, la proyección perspectiva nos parece más natural (la ortográfica es poco realista):

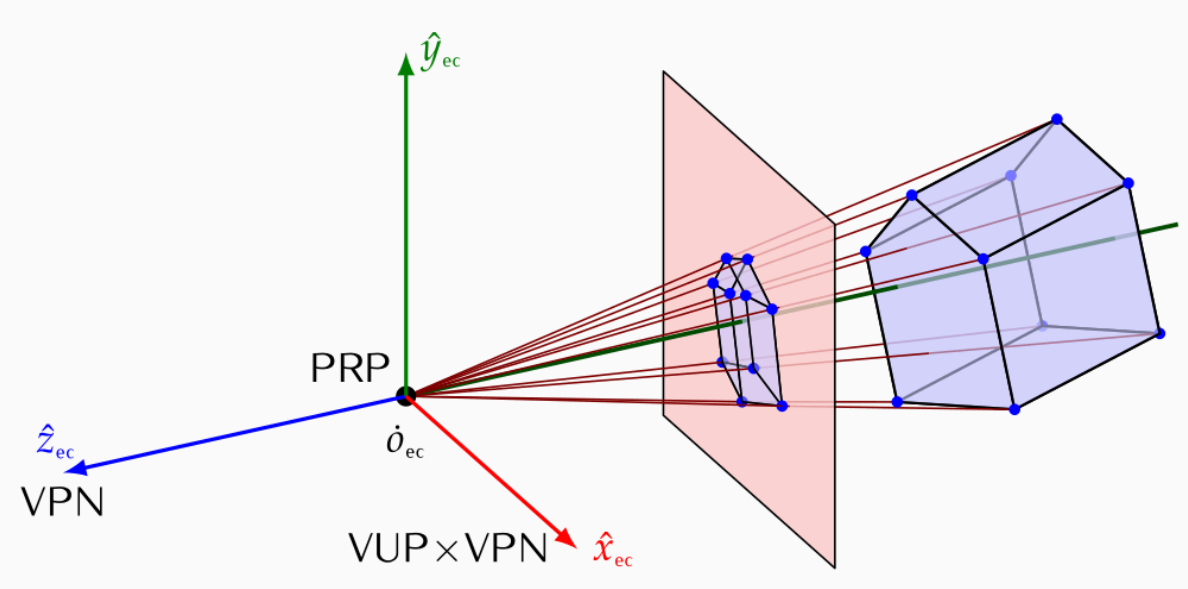


A la izquierda, se interpreta que el cubo más lejano es más grande que los otros, aunque en la imagen son los tres del mismo tamaño.

Imagen de
docs.microsoft.com/es-es/dotnet/framework/wpf/graphics-multimedia/3-d-graphics-overview

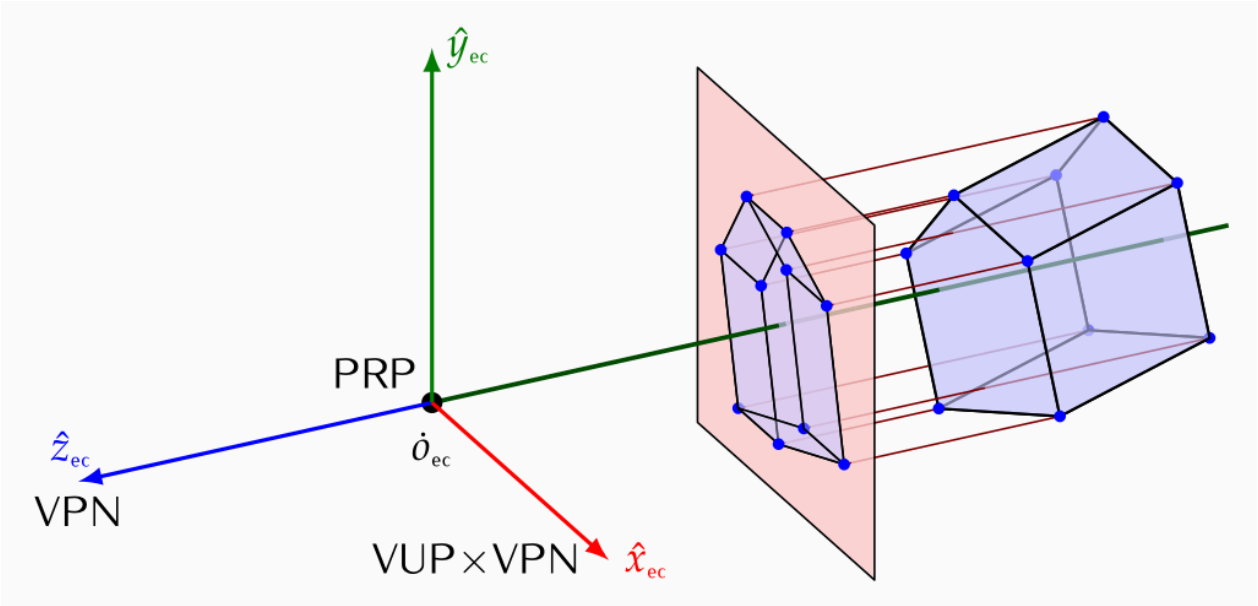
Proyección perspectiva

La **proyección perspectiva** cambia el tamaño de los objetos, usando un factor de escala s que crece de forma inversamente proporcional a la distancia (d_z) en Z desde el objeto al foco (s es de la forma $1/(ad_z + b)$)



Proyección paralela

La **proyección paralela** no cambia la escala, y la proyección se puede ver como una transformación afín:



Subsección **3.2.**

El *view-frustum*. Parámetros de proyección.

El *view-frustum*

El ***view-frustum*** designa la región del espacio de la escena que es visible en el viewport. Su forma depende del tipo de proyección:

- Perspectiva: es un tronco de pirámide rectangular (izq.).
- Ortográfica: es un paralelepípedo ortogonal u ortoedro (der.).

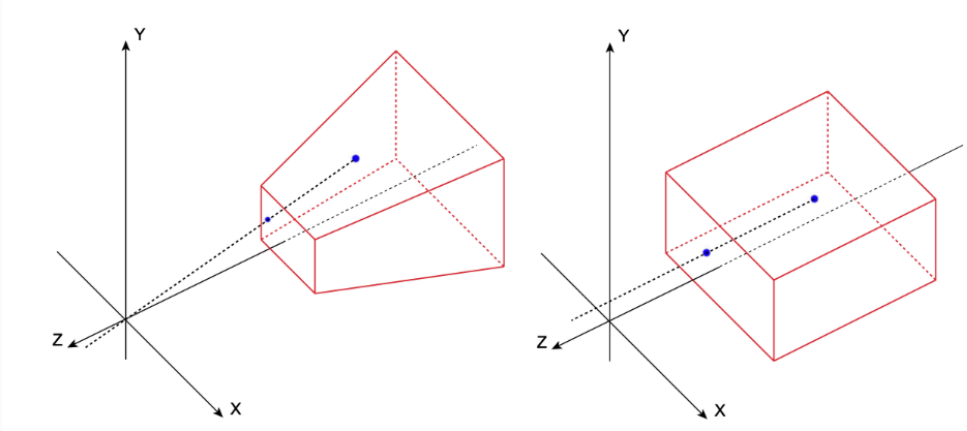


Imagen de: SGI® OpenGL Multipipe™ SDK User's Guide : irix7.com/techpubs/007-4239-004.pdf

Transformación del *view-frustum* en un cubo

El view-frustum está determinado por los 6 planos que contienen a las 6 caras que lo delimitan.

- Estos planos se determinan por sus coordenadas en el marco de coordenadas de vista.
- La transformación de proyección transforma el view-frustum (en coordenadas de vista) en un cubo de lado 2 centrado en el origen, entre −1 y 1 en los tres ejes (en coordenadas de recortado, normalizadas).
- La proyección ortográfica es una transformación afín: una traslación seguida de escalado no necesariamente uniforme.
- La proyección perspectiva no es una transformación afín, aunque se puede expresar usando una transformación afín en coordenadas homogeneas 4D, seguida de una proyección de 4D a 3D.

Parámetros del *view-frustum*. Extensión en Z

Los 6 valores l, r, t, b, n y f (los **parámetros** de frustum) determinan la transformacion de la tupla $(x_{\text{ec}}, y_{\text{ec}}, z_{\text{ec}})$, que está en coordenadas de vista en la tupla $(x_{\text{ndc}}, y_{\text{ndc}}, z_{\text{ndc}})$ en NDCC (*coordenadas normalizadas de dispositivo*, entre −1 y 1):

- Los valores n (**near**) y f (**far**) son los límites en Z del view-frustum, pero cambiados de signo (se cumple $nn = f$).
  - El plano $z_{\text{ec}} = -n$ en EC se transforma en el plano $z_{\text{ndc}} = -1$ en NDC.
  - El plano $z_{\text{ec}} = -f$ en EC se transforma en el plano $z_{\text{ndc}} = +1$ en NDC.
- En la proyección perspectiva, se exige además $0 < n$ y $0 < f$.
- Aunque no se exige así, lo usual es que seleccione $n < f$, es decir, el *view-frustum* se extiende en Z en el intervalo $[-f, -n]$. En adelante supondremos $n < f$, de forma que:
  - El plano $z_{\text{ec}} = -n$ se llama **plano de recorte delantero**.
  - El plano $z_{\text{ec}} = -f$ se llama **plano de recorte trasero**.

Parámetros del *view-frustum*. Extensión en X e Y.

Respecto de los otros cuatro valores (l, r, b y t), determinan la extensión en X y en Y:

- l (**left**) y r (**right**) son los límites en X del view-frustum ($l \neq r$).
- b (**bottom**) y t (**top**) son los límites en Y ($b \neq t$).
- En proy. ortográfica:
  - El plano $x_{\text{ec}} = l$ en EC se transforma en el plano $x_{\text{ndc}} = -1$ en NDC.
  - El plano $x_{\text{ec}} = r$ en EC se transforma en el plano $x_{\text{ndc}} = +1$ en NDC.
  - El plano $y_{\text{ec}} = b$ en EC se transforma en el plano $y_{\text{ndc}} = -1$ en NDC.
  - El plano $y_{\text{ec}} = t$ en EC se transforma en el plano $y_{\text{ndc}} = +1$ en NDC.
- En proy. perspectiva:
  - El plano $-nx_{\text{ec}} = lz_{\text{ec}}$ (EC) se transf. en el plano $x_{\text{ndc}} = -1$ en NDC.
  - El plano $-nx_{\text{ec}} = rz_{\text{ec}}$ (EC) se transf. en el plano $x_{\text{ndc}} = +1$ en NDC.
  - El plano $-ny_{\text{ec}} = bz_{\text{ec}}$ (EC) se transf. en el plano $y_{\text{ndc}} = -1$ en NDC.
  - El plano $-ny_{\text{ec}} = tz_{\text{ec}}$ (EC) se transf. en el plano $y_{\text{ndc}} = +1$ en NDC.

Propiedades de la extensión en X e Y

Hay que tener en cuenta que:

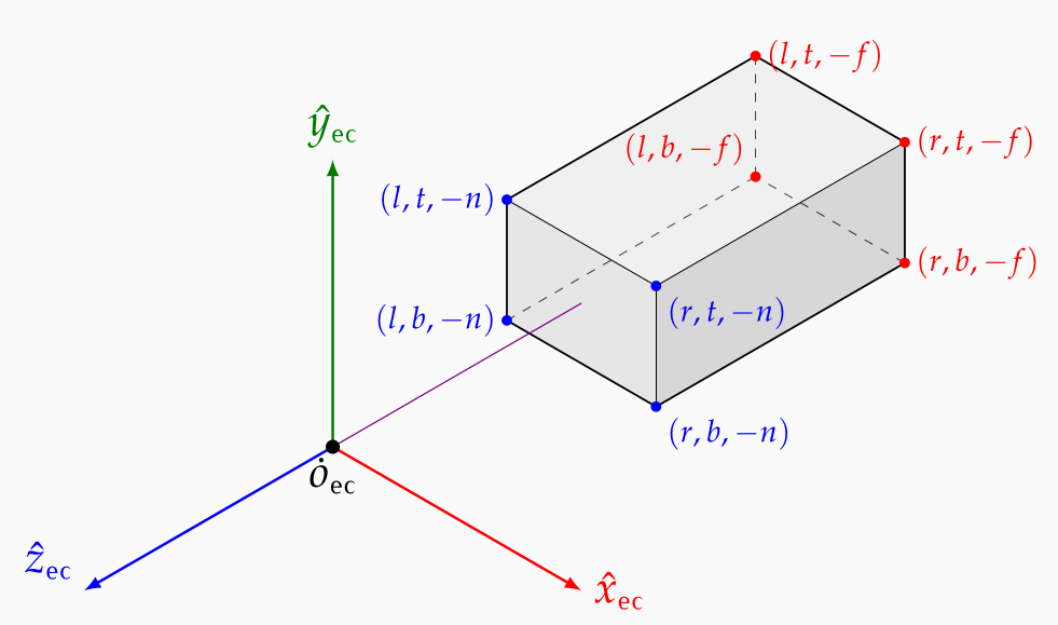
- Aunque esto no es requerido estrictamente, usualmente se seleccionan los parámetros de forma que $l < r$ y $b < t$.
- Cuando se cumple $l = -r$ y $b = -t$, decimos que el **view-frustum está centrado** (el eje Z pasa por el centro de las caras delantera y trasera). Esto es lo más usual, y se corresponde con lo que ocurre en una cámara.
- El valor $(r - l)/(t - b)$ suele coincidir con la relación de aspecto del viewport (ancho/alto, o bien núm.columnas/núm.filas). Si esto no ocurre los objetos aparecerán deformados en la imagen.

En adelante supondremos que siempre seleccionamos $l < r$ y $b < t$.

Parámetros en la proyección ortográfica.

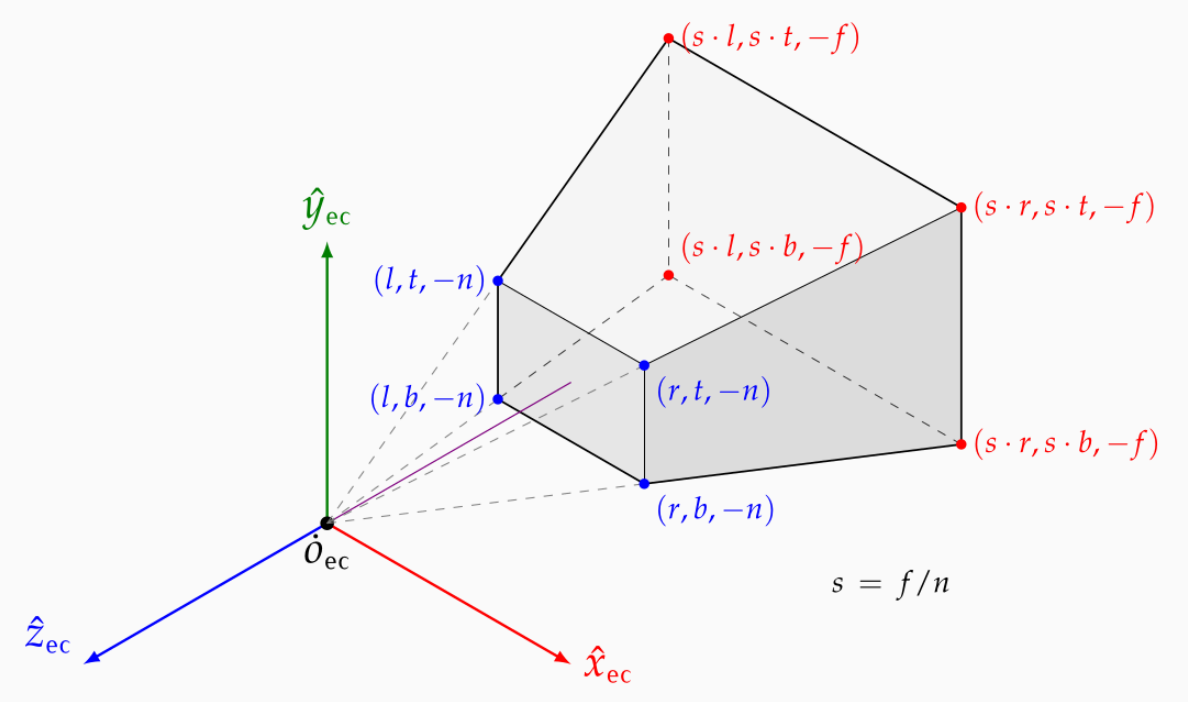
En pr. ortográfica el *view-frustum* es un **ortoedro**. Contiene los puntos cuyas coordenadas de cámara $(x_{\text{ec}}, y_{\text{ec}}, z_{\text{ec}})$ cumplen:

$$l \leq x_{\text{ec}} \leq r \qquad b \leq y_{\text{ec}} \leq t \qquad -f \leq z_{\text{ec}} \leq -n$$



Parámetros en la proyección perspectiva (1/2)

En perspectiva, el view-frustum es una **pirámide rectangular truncada**:



Parámetros en la proyección perspectiva (2/2)

Los puntos dentro del view-frustum son aquellos cuyas coordenadas de cámara $(x_{\text{ec}}, y_{\text{ec}}, z_{\text{ec}})$ cumplen:

En el eje X:

$$l \quad \leq \quad x_{\text{ec}} \left(\frac{n}{-z_{\text{ec}}} \right) \quad \leq \quad r$$

En el eje Y:

$$b \quad \leq \quad y_{\text{ec}} \left(\frac{n}{-z_{\text{ec}}} \right) \quad \leq \quad t$$

En el eje Z:

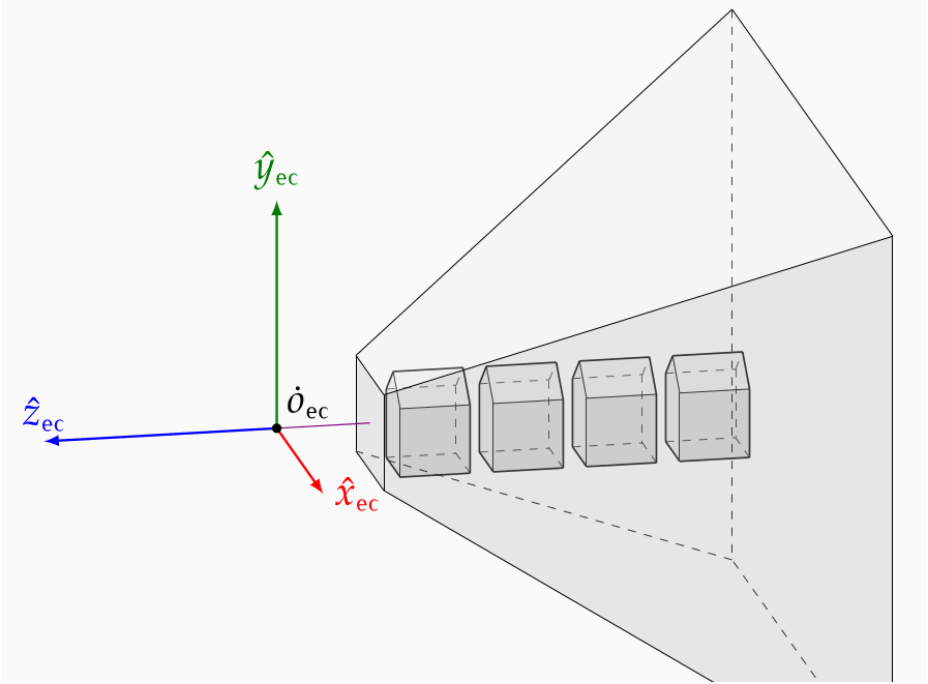
$$-f \quad \leq \quad z_{\text{ec}} \quad \leq \quad -n$$

Subsección **3.3.**

La matriz de proyección.

Escena de ejemplo para transformación perspectiva

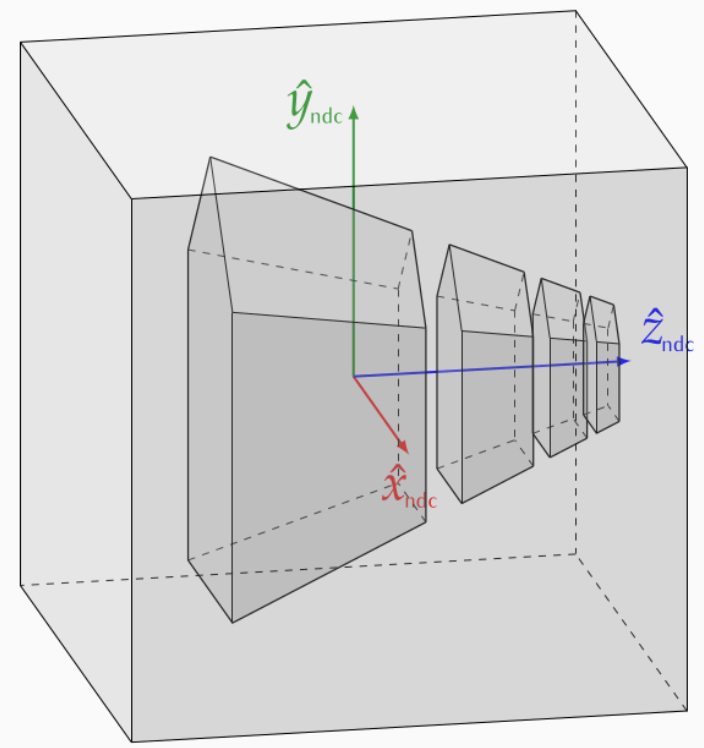
Suponemos que partimos de una escena que vamos a proyectar usando perspectiva. En el espacio de coordenadas de cámara, la escena es esta:



Escena proyectada por la transformación perspectiva

El efecto de la **transformación de proyección perspectiva** será:

- hacer más pequeños los objetos más alejados del observador, y
- situar la escena en un cubo de lado 2 y centrado en el origen del **marco de coordenadas normalizadas de dispositivo** (NDC).



Proyección perspectiva sobre el plano delantero

Podemos suponer que los puntos se proyectan sobre el plano frontal del *view--frustum*(coord. Z igual a $-n$) con foco en $\mathbf{o}_{\text{ec}}$:

- Dado un punto $\dot{p} = \mathcal{V}(x_{\text{ec}}, y_{\text{ec}}, z_{\text{ec}}, w_{\text{ec}})$ queremos calcular las coordenadas de su proyección (x', y', z', w') (en principio, con $w' = w_{\text{ec}} = 1$).
- Si se asume $z_{\text{ec}} < 0$ (el punto está en la rama negativa del eje Z), podemos hacer:

$$x' = \frac{n\, x_{\text{ec}}}{-z_{\text{ec}}} \qquad y' = \frac{n\, y_{\text{ec}}}{-z_{\text{ec}}} \qquad z' = \frac{n\, z_{\text{ec}}}{-z_{\text{ec}}}$$

  esta transformación tiene dos problemas:

  - Las coordenadas resultado no están entre -1 y 1.
  - Colapsa o *aplana* todas las coordenadas Z (siempre $z' = -n$).

Normalización de coordenadas X e Y

Si el punto original está en el *view-frustum*, entonces:

- x' está en el intervalo $[l, r]$
- y' está en el intervalo $[b, t]$.
- queremos dejar ambas coordenadas en el intervalo $[-1, 1]$
- podemos usar un escalado y traslación adicionales en X e Y:

$$x'' = 2\left(\frac{x' - l}{r - l}\right) - 1 = \frac{a_0\, x_{\text{ec}}}{-z_{\text{ec}}} - a_1 = \frac{a_0\, x_{\text{ec}} + a_1 z_{\text{ec}}}{-z_{\text{ec}}}$$

$$y'' = 2\left(\frac{y' - b}{t - b}\right) - 1 = \frac{b_0\, y_{\text{ec}}}{-z_{\text{ec}}} - b_1 = \frac{b_0\, y_{\text{ec}} + b_1 z_{\text{ec}}}{-z_{\text{ec}}}$$

donde hemos definido estas cuatro constantes:

$$a_0 \equiv \frac{2n}{r - l} \qquad a_1 \equiv \frac{r + l}{r - l} \qquad b_0 \equiv \frac{2n}{t - b} \qquad b_1 \equiv \frac{t + b}{t - b}$$

Información de profundidad y normalización en Z

El problema está de hacer $z' = -n$ está en que **se pierde informacion de profundidad en Z**, que es necesaria para EPO). Para evitarlo, se usa una función lineal de z con dos constantes c_0 y c_1:

$$z'' = \frac{c_0\, z_{\text{ec}} + c_1}{-z_{\text{ec}}} \qquad \text{donde:} \qquad c_0 = \frac{n+f}{n-f}, \qquad c_1 = \frac{2fn}{n-f}.$$

- los dos valores c_2 y c_3 se eligen de forma que, para $z_{\text{ec}} = -n$, se hace $z'' = -1$, y para $z_{\text{ec}} = -f$, se hace $z'' = 1$.
- es decir: el rango $[-f, -n]$ se lleva al rango $[-1, 1]$ (invirtiendo el orden).
- esta transformación conserva el orden (invertido) de las coordenadas Z (no *aplana*)
- ahora, **valores menores de Z** implican puntos **más cercanos al observador**, y **valores mayores, más lejanos**.

Coordenadas cartesianas del punto proyectado

En resumen, tenemos estas tres igualdades:

$$x'' = \frac{a_0 x_{\text{ec}} + a_1 z_{\text{ec}}}{-z_{\text{ec}}}$$

$$y'' = \frac{b_0 x_{\text{ec}} + b_1 z_{\text{ec}}}{-z_{\text{ec}}}$$

$$z'' = \frac{c_0 z_{\text{ec}} + c_1}{-z_{\text{ec}}} = \frac{c_0 z_{\text{ec}} + c_1 w_{\text{ec}}}{-z_{\text{ec}}}$$

esta transformación incluye una división, y por tanto

- **no se puede implementar con una matriz** como hacíamos con las anteriores (no es **lineal**)
- aunque sí transforma líneas rectas en líneas rectas

Obtención de las coordenadas de recortado

Para solventar el problema anterior (para poder usar una matriz), se definen las **coordenadas de recortado** (***clip coordinates***), a partir de las coordenadas de cámara del original:

$$\begin{aligned} x_{\text{cc}} &= a_0\, x_{\text{ec}} + a_1\, z_{\text{ec}} \\ y_{\text{cc}} &= b_0\, y_{\text{ec}} + b_1\, z_{\text{ec}} \\ z_{\text{cc}} &= c_0\, z_{\text{ec}} + c_1\, w_{\text{ec}} \\ w_{\text{cc}} &= -z_{\text{ec}} \end{aligned}$$

Esta transformación ya **sí se puede hacer con una matriz 4x4:**

- se ha eliminado la división por $-z_{\text{ec}}$, el resto es igual
- esta división se hace más adelante en el cauce gráfico
- para ello, el denominador de la división ($-z_{\text{ec}}$) queda guardado en w_{cc} (**que ya no es** 1).

La matriz de proyección perspectiva Q

Con todo lo dicho, la proyección perspectiva se puede realizar usando una matriz Q, que se aplica a coordenadas de cámara (con $w_{\text{ec}} = 1$) y produce coordenadas de recortado (con $w_{\text{cc}} \neq 1$):

$$\begin{pmatrix} x_{\text{cc}} \\ y_{\text{cc}} \\ z_{\text{cc}} \\ w_{\text{cc}} \end{pmatrix} = \begin{pmatrix} a_0 & 0 & a_1 & 0 \\ 0 & b_0 & b_1 & 0 \\ 0 & 0 & c_0 & c_1 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_{\text{ec}} \\ y_{\text{ec}} \\ z_{\text{ec}} \\ 1 \end{pmatrix}$$

evidentemente, podemos definir entonces la matriz Q de esta forma:

$$Q = \begin{pmatrix} a_0 & 0 & a_1 & 0 \\ 0 & b_0 & b_1 & 0 \\ 0 & 0 & c_0 & c_1 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

La proyección ortográfica.

En el caso de la **proyección ortográfica** (*orthographic projection*), se hace proyección en una dirección paralela al eje Z:

- Esta transformación **solo requiere la normalización de los rangos de valores en los tres ejes** (se usa traslación más escalado)
- (1) traslación $T_{-\mathbf{c}}$ (lleva el centro del paralepípedo, $\mathbf{c}$, al origen), donde

$$\mathbf{c} \equiv \left(\frac{l+r}{2}, \frac{t+b}{2}, \frac{f+n}{2} \right)$$

- (2) escalado $E_{\mathbf{s}}$ con factores $\mathbf{s} = \left(s_x, s_y, s_z\right)$ (deja los valores en $[-1, 1]$ en los tres ejes), donde:

$$s_x \equiv \frac{2}{r-l} \qquad s_y \equiv \frac{2}{t-b} \qquad s_z \equiv \frac{-2}{f-n}$$

(en Z se cambia de signo para *invertir* el eje Z).

La matriz de proyección ortográfica

La matriz de proyección ortográfica O se obtiene por tanto como composición de $T_{-\mathbf{c}}$ seguido de $E_{\mathbf{s}}$, es decir $O = E_{\mathbf{s}} \cdot T_{-\mathbf{c}}$, o lo que es lo mismo:

$$O = \begin{pmatrix} a'_0 & 0 & 0 & a'_1 \\ 0 & b'_0 & 0 & b'_1 \\ 0 & 0 & c'_0 & c'_1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \text{donde:} \quad \begin{cases} a'_0 \equiv \dfrac{2}{r-l} & a'_1 \equiv -\dfrac{r+l}{r-l} \\ b'_0 \equiv \dfrac{2}{t-b} & b'_1 \equiv -\dfrac{t+b}{t-b} \\ c'_0 \equiv \dfrac{-2}{f-n} & c'_1 \equiv -\dfrac{f+n}{f-n} \end{cases},$$

de forma que ahora hacemos:

$$(x_{\mathrm{cc}}, y_{\mathrm{cc}}, z_{\mathrm{cc}}, w_{\mathrm{cc}})^T = O(x_{\mathrm{ec}}, y_{\mathrm{ec}}, z_{\mathrm{ec}}, w_{\mathrm{ec}})^T,$$

donde w_{cc} ahora sí vale 1 con seguridad.

Construcción de la matriz: ajuste en vertical

Para construir una matriz de proyección (perspectiva u ortográfica) a partir de los seis valores reales l, r, b, t, n y f, usando las constantes definidas antes.

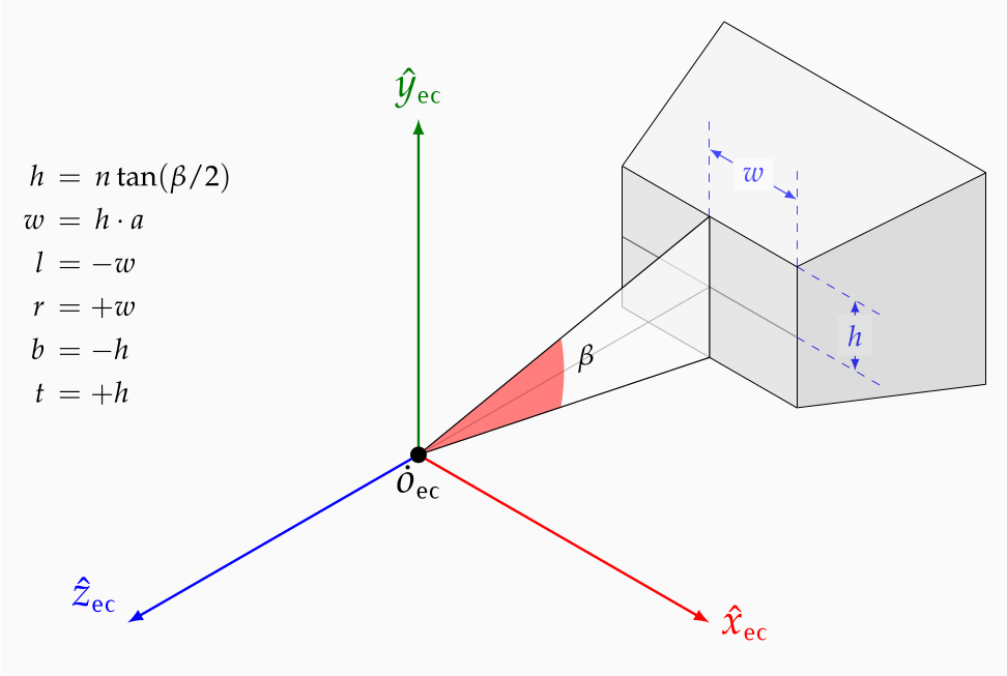
- Otra posibilidad para la matriz perspectiva (más intuitivo) es usar los parámetros β y a (además de n y f):
  - $\beta \equiv$ es la **apertura vertical del campo de visión** (*fovy*), es el ángulo en grados (entre 0 y 180.0) que hay entre la cara superior y la inferior del frustum
  - $a_v \equiv$ es la **relación de aspecto vertical** (*vertical aspect ratio*) de la imagen a producir: su ancho (n. columnas) dividido por el alto (n. filas).

En este caso, los valores l, r, b y t se obtienen como:

$$t \equiv n \tan\left(\frac{\beta}{2}\right) \qquad b \equiv -t \qquad r \equiv a_v\, t \qquad l \equiv -r$$

Parámetros de la matriz de proyección

El significado del valor β (*fovy*) se aprecia en esta figura, donde se han señalado los tamaños del plano delantero en horizontal $w = r$ y en vertical $h = t$:



Esta perspectiva es *centrada*, ya que $r = -l$ y $t = -b$

Construcción de la matriz: ajuste en horizontal

Alternativamente, se puede usar la apertura horizontal de campo (en lugar de la vertical).

Los parámetros son:

- $\alpha \equiv$ es la **apertura horizontal del campo de visión** (*fovx*), es el ángulo en grados (entre 0 y 180.0) que hay entre la cara izquierda y la derecha del frustum
- $a_h \equiv$ es la **relación de aspecto horizontal** (*horizontal aspect ratio*) de la imagen a producir: su alto (n. filas) dividido por el ancho (n. columnas), es la inversa de a_v.

En este caso, los valores l, r, b y t se obtienen como:

$$r \equiv n \tan\left(\frac{\alpha}{2}\right) \qquad l \equiv -r \qquad t \equiv a_h\, r \qquad b \equiv -t$$

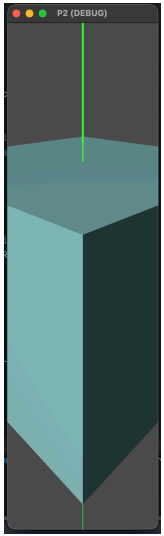
Matriz de proyección 3D en Godot

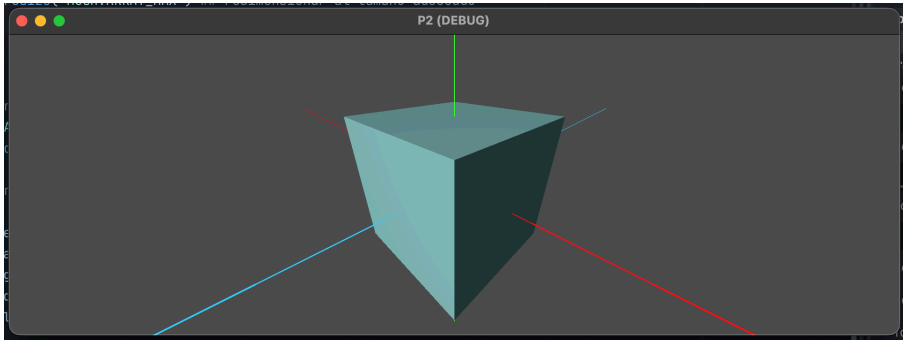
La clase `Camera3D` permite configurar la matriz de proyección mediante estas propiedades:

- `projection`: **tipo de proyección**, de tipo `ProjectionType`, puede valer `PROJECTION_PERSPECTIVE` o `PROJECTION_ORTHOGONAL`.
- `size`: **tamaño vertical u horizontal** del ortoedro visible (solo para `PROJECTION_ORTHOGONAL`), de tipo `float`.
- `fov`: **apertura del campo de visión** (en grados, `float`), vertical u horizontal (solo para `PROJECTION_PERSPECTIVE`), de tipo `float`. Por defecto es 75^{o}.
- `keep_aspect`: **tipo de ajuste (vertical u horizontal)**, de tipo `KeepAspect`, puede valer `KEEP_WIDTH`, entonces las propiedades `fov` y `size` se interpretan en horizontal, o bien `KEEP_HEIGHT` (por defecto) y entonces se interpretan en vertical.
- `near`: **distancia al plano de recorte delantero** (valor n)
- `far`: **distancia al plano de recorte trasero** (valor f)

Ejemplo de ajuste en vertical

Por defecto, **Godot usa ajuste en vertical** (`KEEP_HEIGHT`), de forma que la apertura vertical del campo de visión es la que se fija con `fov`, y la apertura horizontal se ajusta según la relación de aspecto del viewport.

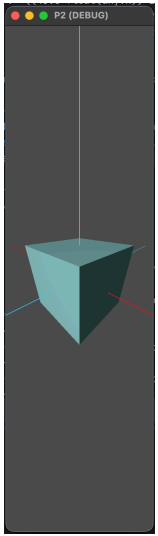


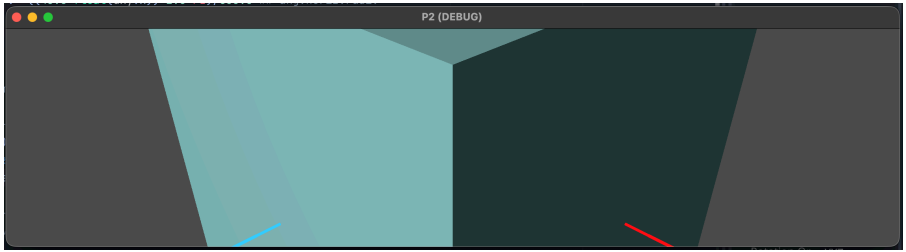


Al redimensionar la ventana, el cubo se ve siempre **completo en vertical**.

Ejemplo de ajuste en horizontal

La cámara se puede configurar para que haga **ajuste en horizontal** (`KEEP_WIDTH`), de forma que la apertura horizontal del campo de visión es la que se fija con `fov`, y la apertura vertical se ajusta según la relación de aspecto del viewport.





Al redimensionar la ventana, el cubo se ve siempre **completo en horizontal**.

Sección **4.**

Recortado y transformación del viewport.

1. Recortado de primitivas y división por w
2. Transformación del viewport

Subsección **4.1.**

Recortado de primitivas y división por w

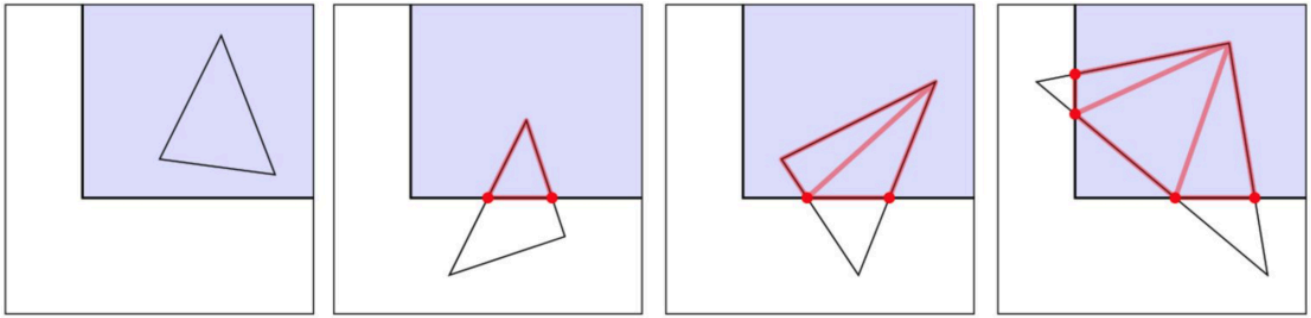
Recortado

Una vez se tienen las coordenadas de recortado de los vértices, se comprueban que primitivas estan dentro o fuera del viewfrustum (que en CC es un cubo de lado dos unidades centrado en el origen):

- Las primitivas completamente dentro de la zona visible **se mantienen**.
- Las primitivas completamente fuera de la zona visible **se descartan**.
- Las primitivas parcialmente dentro **se dividen en partes**: unas completamente dentro (se visualizan) y otras completamente fuera (que se descartan). Esto se hace mediante **la inserción de algunos vértices nuevos** justo en los planos que delimitan el view-frustum.

Inserción de nuevos vértices y triángulos

Varios ejemplos de recortado de triángulos:



- Las coordenadas y otros atributos de los nuevos vértices se interpolan a partir de los vértices en los dos extremos de la arista donde se inserta el nuevo.
- Se hace recortado independiente por cada uno de los 6 planos de recorte.

División por W. Coordenadas normalizadas de dispositivo.

Los vértices (en el view-frustum) resultado del recorte tienen coordenadas de recorte con $w_{\text{cc}} \neq 0$ (si $P = Q$, entonces además $w_{\text{cc}} = 1$).

El siguiente paso es hacer la división por w_{cc} de las tres componentes. Se obtienen las **coordenadas normalizadas de dispositivo** (**NDC**), con componente W de nuevo a 1:

$$(x_{\text{ndc}}, y_{\text{ndc}}, z_{\text{ndc}}, 1) \quad = \quad \frac{1}{w_{\text{cc}}} (x_{\text{cc}}, y_{\text{cc}}, z_{\text{cc}}, w_{\text{cc}})$$

Los valores $x_{\text{ndc}}, y_{\text{ndc}}$ y z_{ndc} están los tres en el intervalo $[-1, 1]$, ya qye los vértices ya han pasado el recortado y están todos dentro del ortoedro visible en NDC.

Subsección **4.2.**

Transformación del viewport

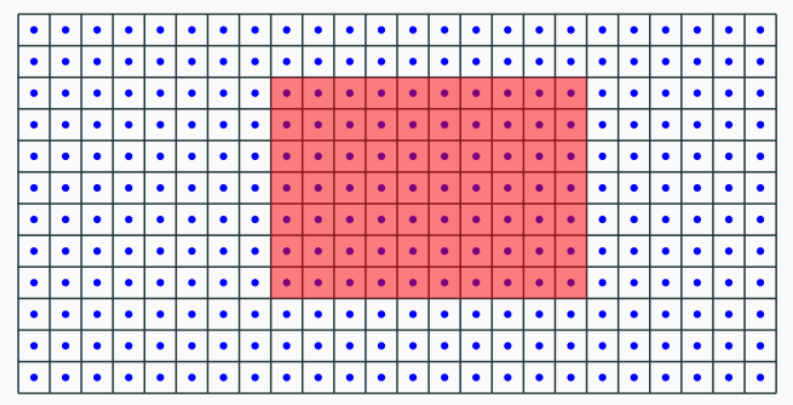
Transformación de Viewport

El siguiente paso consiste en calcular en que posiciones de la imagen se proyecta cada vértice:

- Este paso se puede modelar como una transformación lineal que llamaremos **transformación de viewport**. El término ***viewport*** hace referencia a la zona rectangular de la ventana donde se proyectarán los polígonos que están en el cubo visible (un bloque rectangular de pixels)
- Esta transformación produce **coordenadas de dispositivo** o **de ventana** (DC: *device coordinates*, o también llamadas *screen coordinates*, o *window coordinates*). Las coordenadas X e Y en DC se expresan en unidades de pixels.
- La transformación de viewport es lineal y consta simplemente de escalados y traslaciones.
- La coordenada Z se transforma y se conserva para poder hacer después *eliminación de partes ocultas*.

Coordenadas de dispositivo y pixels del viewport

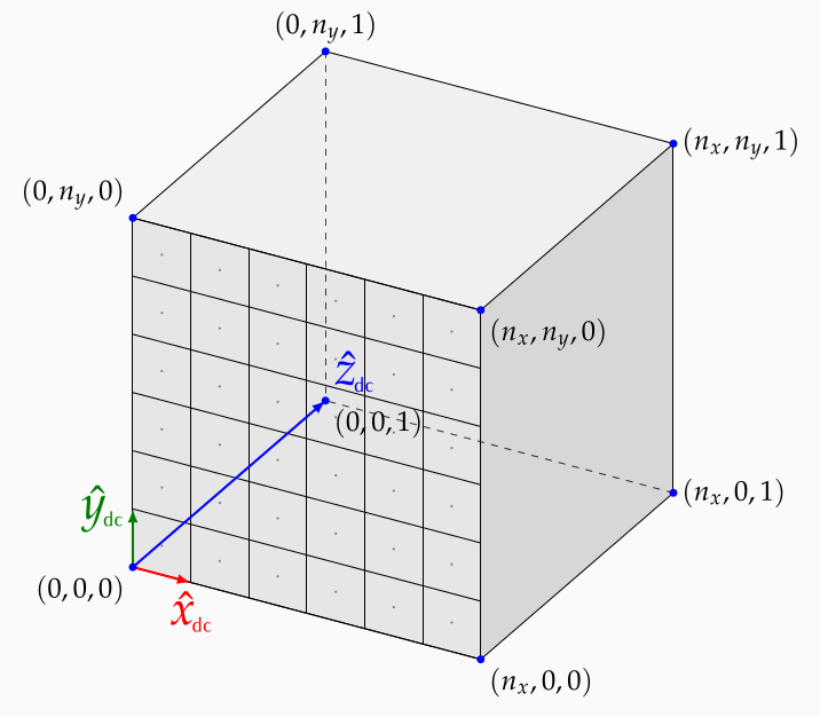
En coordenadas de dispositivo, podemos asociar una región cuadrada (de lado unidad) a cada pixel en el plano de la ventana. El viewport (en rojo) es un bloque rectangular de pixels, contenido en el bloque rectangular correspondiente a la ventana o imagen completa:



los centros de los pixels (puntos azules) tienen coordenadas de dispositivo con parte fraccionaria igual a $1/2$. Los bordes entre pixels tienen coordenadas sin parte fraccionaria (enteras).

El espacio de coordenadas de dispositivo

En 3D el espacio de coordenadas de dispositivo es un ortoedro. Se puede visualizar como aparece aquí, incluyendo el marco de coordenadas de dispositivo:



Matriz del viewport. Parámetros.

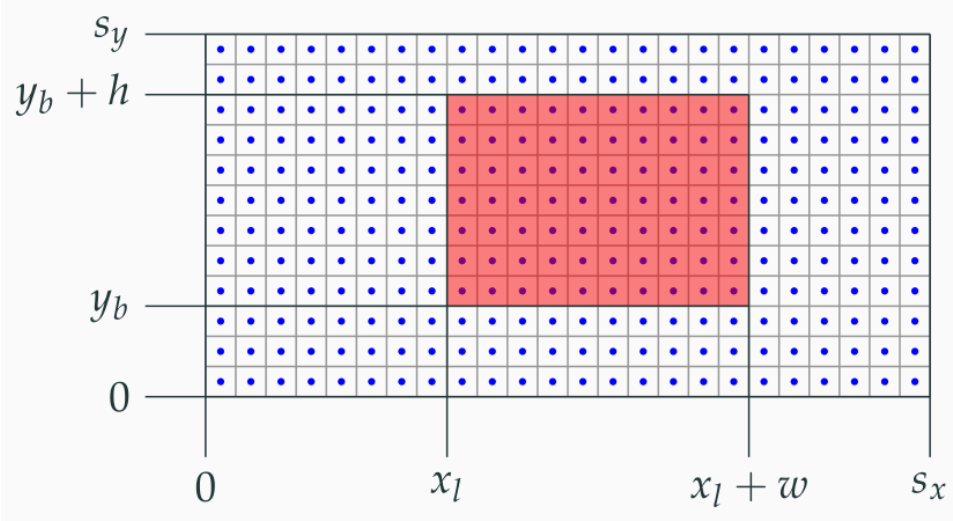
Para convertir a coordenadas de dispositivo se usa una matriz D, que depende de estos parámetros (ver figura):

- x_l, y_b número de columna y fila (enteros no negativos) del pixel que ocupa, en la ventana, la esquina inferior izquierda del viewport.
- w, h (*width* y *height*) número total (enteros no negativo) de columnas y de filas de pixels (respectivamente) que ocupa el viewport.
- n_d, f_d rango de valores de salida en Z en DC. El valor n_d es la profundidad más cercana posible al observador, y f_d la más lejana. En OpenGL suele ser $n_d = 0$ y $f_d = 1$, en otras APIs puede ser $n_d = -1$.

Aunque los cuatro parámetros relevantes (x_l,y_b,w y h) son enteros, las coordenadas de dispositivos son valores reales, ya que las posiciones de los vértices en DC son en general no enteras (no coinciden necesariamente con los centros o bordes de los pixels).

Parámetros del viewport

Suponemos que la ventana tiene s_x columnas y s_y filas, y que el gestor de ventanas acepta coordenadas de pixels enteras no negativas:



Se cumplen las desigualdades: $\begin{cases} 0 \leq x_l < x_l + w \leq s_x \\ 0 \leq y_b < y_b + h \leq s_y \end{cases}$

La transformación de viewport

En NDC las coordenadas están en $[-1, 1]$, luego hay que hacer:

1. traslación de la esquina $\mathbf{p} = (-1, -1, -1)$ al origen (matriz $T_{-\mathbf{p}}$)
2. escalado uniforme (por $1/2$) y por $(w, h, f_d - n_d)$ (matriz $E_{\mathbf{s}}$)
3. traslación del origen a $\mathbf{q} = (x_l, y_b, n_d)$ (matriz $T_{\mathbf{q}}$)

Con lo cual la **transformación del viewport** D queda como el producto de estas tres matrices:

$$D \;=\; T_{\mathbf{q}}\, E_{\mathbf{s}}\, T_{-\mathbf{p}}$$

Por tanto, las **coordenadas de dispositivo** $(x_{\text{dc}}, y_{\text{dc}}, z_{\text{dc}}, 1)$ se definen a partir de las normalizadas $(x_{\text{ndc}}, y_{\text{ndc}}, z_{\text{ndc}}, 1)$ de esta forma:

$$\begin{aligned} x_{\text{dc}} &= (x_{\text{ndc}} + 1)w/2 + x_l \\ y_{\text{dc}} &= (y_{\text{ndc}} + 1)h/2 + y_b \\ z_{\text{dc}} &= (z_{\text{ndc}} + 1)(f_d - n_d)/2 + n_d \end{aligned}$$

La matriz de viewport D

Como consecuencia de todo lo anterior, la **matriz de viewport** D debe definirse así:

$$D = \begin{pmatrix} \frac{w}{2} & 0 & 0 & x_l + \frac{w}{2} \\ 0 & \frac{h}{2} & 0 & y_b + \frac{h}{2} \\ 0 & 0 & \frac{f_d - n_d}{2} & n_d + \frac{f_d - n_d}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

de forma que:

$$(x_{\text{dc}}, y_{\text{dc}}, z_{\text{dc}}, 1)^T = D(x_{\text{ndc}}, y_{\text{ndc}}, z_{\text{ndc}}, 1)^T$$

La clase *Viewport* de Godot

La clase `Viewport` de Godot encapsula la información de un array de pixels en memoria sobre los que se pueden dibujar escenas 2D o 3D. Tiene dos sub-clases:

- Clase `Window` : el viewport ocupa **una ventana visible completa**, todo proyecto Godot tiene asociado un viewport de este tipo por defecto.
- Clase `SubViewport`: un viewport que no ocupa una ventana visible completa, hay de dos dos tipos:
  - El viewport **ocupa una parte de una ventana visible**, debe estar contenido en `SubViewportContainer`, que es una subclase de `CanvasItem`, es decir, es un nodo 2D.
  - El viewport es **una zona de memoria en la GPU** sobre la que se pueden visualizar imágenes, que no son visibles inmediatamente en pantalla, debe estar asignado a un objeto de la clase `ViewportTexture`, que es una subclase de `Texture2D`, es decir, una imagen de textura para 2D y 3D.

Consulta del *viewport* y sus propiedades

En algunos casos será necesario consultar (y modificar) las propiedades del viewport donde se está visualizando un nodo.

- Se puede recuperar el objeto *viewport* donde se está visualizando un nodo, para ello usamos el método `get_viewport` de la clase `Node`

```
var vp := nodo.get_viewport() ## 'vp' es el viewport de 'nodo'
```

- Una vez se ha obtenido el objeto *viewport* se pueden consultar sus propiedades y usar sus métodos, por ejemplo:
  - Se puede acceder a su tamaño accediendo a la propiedad `size` de tipo `Vector2i`, con dos enteros que son el número de filas y de columnas de pixels del viewport.

Sección **5.**

Problemas

1. Transformación de vista.
2. Transformación de proyección

Subsección **5.1.**

Transformación de vista.

Cámara que sigue un objetivo

Problema 6.1:

Escribe el código GDScript para adjuntar a un nodo de tipo `Camera3D`, de forma que en cada frame la cámara apunte a un objeto móvil objetivo (por ejemplo un coche), con estos requerimientos:

- La posición y el vector de velocidad del objetivo (en coordenadas de mundo) se pueden obtener con dos funciones globales, llamadas `objetivo.posicion()` y `objetivo.velocidad()`, ambas devuelven un objeto de tipo `Vector3`.
- La cámara debe situarse *detrás* del objetivo, de forma que el punto devuelto por `objetivo.posicion()` se proyecte en el centro del viewport, y además la camara esté situada 3 unidades en horizontal por detrás del objetivo, y 2 unidades por encima (en el eje Y).

Parámetros para una vista concreta (1/3)

Problema 6.2:

Supongamos una escena que contiene una representación visible del marco de coordenadas del mundo como tres flechas (roja X, verde Y y azul Z), como ocurre en las prácticas. Queremos visualizar esa escena en pantalla, de forma que:

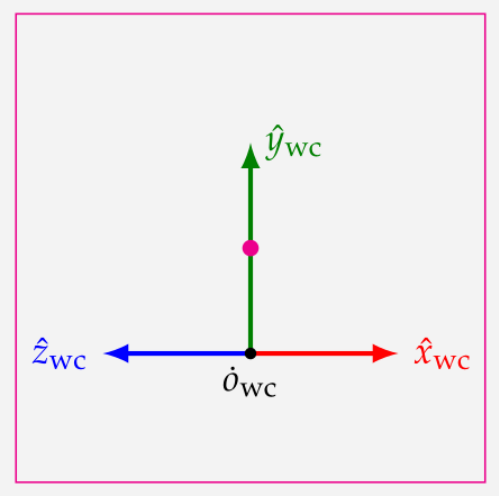
1. El eje Y aparezca vertical, hacia arriba, el eje X horizontal, hacia la derecha, el eje Z horizontal, hacia la izquierda (los ejes X y Z se visualizan con la misma longitud aparente).
2. El punto de coordenadas $(0, 0.5, 0)$ (aparece como un disco de color morado en la figura) debe aparecer en el centro del viewport
3. El observador (foco de la proyeccion) estará a 3 unidades de distancia del punto $(0, 0.5, 0)$

(continua en la siguiente transparencia).

Parámetros para una vista concreta (2/3)

Problema 6.2 (continuación):

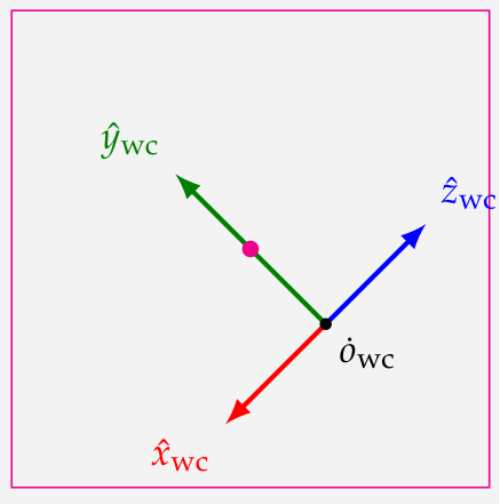
Escribe unos valores que podríamos usar para $\mathbf{a}$, $\mathbf{u}$ y $\mathbf{n}$ de forma que se cumplan estos requisitos. En la figura se observa una vista esquemática de como quedaría la figura en un viewport cuadrado, no necesariamente a escala.



Parámetros para una vista concreta (3/3)

Problema 6.3:

Repite el problema anterior 6.2, pero ahora para esta vista:



Usa una rotación del marco de vista entorno a uno de sus propios ejes.

Construcción de la matriz de vista

Problema 6.4:

Escribe el código GDScript para calcular los vectores de coordenadas $\mathbf{x}_{\text{ec}}$, $\mathbf{y}_{\text{ec}}$, $\mathbf{z}_{\text{ec}}$ y $\mathbf{o}_{\text{ec}}$ que definen el marco de vista a partir de los vectores de coordenadas $\mathbf{a}$, $\mathbf{u}$ y $\mathbf{n}$ (todos estos vectores de coordenadas de mundo, en objetos de tipo `Vector3`).

Problema 6.5:

Partiendo de los vectores de coordenadas $\mathbf{x}_{\text{ec}}$, $\mathbf{y}_{\text{ec}}$, $\mathbf{z}_{\text{ec}}$ y $\mathbf{o}_{\text{ec}}$ que se calculan en el problema anterior, escribe el código que calcula explicitamente la matriz de vista, es una variable de tipo `Transform3D`.

Subsección **5.2.**

Transformación de proyección

Ajuste dinámico

Problema 6.6:

En una copia independiente del código de prácticas, modifica el nodo de la cámara orbital simple para conseguir que el *fov* mínimo (vertical u horizontal) sea siempre de 75^{o}, serviría, por ejemplo, para ver el cubo de las prácticas siempre completo independientemente del ancho y alto de la ventana (ver transparencia 58 y siguiente). Para ello:

1. Añadir al script del nodo de cámara una función que se ejecute siempre que se redimensione la ventana (y al inicio), en esa función:
2. obtener el tamaño (alto y ancho) del viewport,
3. calcular la relación de aspecto (ancho/alto)
4. usar ajuste de la proyección en vertical si el viewport es más ancho que alto, y ajuste en horizontal en caso contrario.

Parámetros de matriz de proyección (1)

Problema 6.7:

Queremos visualizar una escena con mallas indexadas de la cual sabemos que tiene todos los vértices dentro de un cubo de lado s unidades cuyo centro es el punto de coordenadas del mundo $\mathbf{c} = \left(c_x, c_y, c_z\right)$.

Para construir la matriz de vista, se situa el observador en el punto $\mathbf{o}_{\text{ec}} = \left(c_x, c_y, c_z + s + 2\right)$, el punto de atención $\mathbf{a}$ se hace igual a $\mathbf{c}$ (el centro del cubo se ve en el centro de la imagen), y el vector $\mathbf{u}$ es $(0, 1, 0)$. Se visualizará en un viewport cuadrado.

(continua en la siguiente página)

Parámetros de matriz de proyección (2)

Problema 6.7 (continuación):

Queremos construir la matriz de proyeccion perspectiva Q de forma que se cumplan estos requerimientos:

1. No se recorta ningún triángulo.
2. El tamaño aparente de los objetos (proyectados en pantalla) es el mayor posible.
3. El valor del parámetro n es el mayor posible.
4. El valor del parámetro f es el menor posible.
5. Los objetos no aparecen deformados.

Con estos requerimientos, indica como calcular los valores l, r, t, b, n y f (para obtener la matriz Q de proyección), en función de s y $\mathbf{c} = \left(c_x, c_y, c_z\right)$.

Parámetros de matriz de proyección (3)

Problema 6.8:

Repite el problema anterior 6.7, con exactamente los mismos requerimientos y suposiciones, excepto que ahora la escena, en lugar de estar contenida en un cubo de lado s unidades, está contenida en una esfera de radio r unidades (con centro igualmente en $\mathbf{c}$).

Problema 6.9:

Repite el problema anterior 6.7, con exactamente los mismos requerimientos y suposiciones, excepto que ahora, en lugar de suponer que el *viewport* es cuadrado, sabemos que tiene w columnas de pixels y h filas de pixels, y no podemos suponer que $w = h$.

Parámetros de matriz de proyección (4)

Problema 6.10:

Repite el problema anterior 6.7, con exactamente los mismos requerimientos y suposiciones, excepto que ahora se nos da un ángulo β en grados que debe ser la apertura de campo vertical de la proyección perspectiva. Para ello, ahora tenemos libertad para situar al observador en la línea paralela al eje Z que pasa por $\mathbf{c}$, de forma que la apertura de campo vertical sea exactamente β.

Indica como calcular la coordenada Z que debemos usar ahora para $\mathbf{o}_{\text{ec}}$ (la X y la Y son las mismas que antes), de forma que se cumpla lo dicho, también indica como debemos de calcular ahora los valores de l, r, t, b, n y f (todo ello en función de β, s y $\mathbf{c} = \left(c_x, c_y, c_z\right)$).

Fin de transparencias.