Informática Gráfica
Visualización

Juan Carlos Torres
Grupos C y D

Dpt. Lenguajes y Sistemas Informáticos
ETSI Informática y de Telecomunicación
Universidad de Granada

—

Curso 2024-25

Visualización

Para generar la imagen de una escena necesitamos:

- Transformación de visualización: Transformar coordenadas 3D de los objetos a coordenadas de pantalla.
- Recortado: Eliminación de partes de polígonos fuera de la zona visible.
- Rasterización y eliminación de partes ocultas: Determinar los pixeles cubiertos por cada primitiva.
- Iluminación y texturización: Calcular el color de cada pixel.

Transformación de visualización 3D

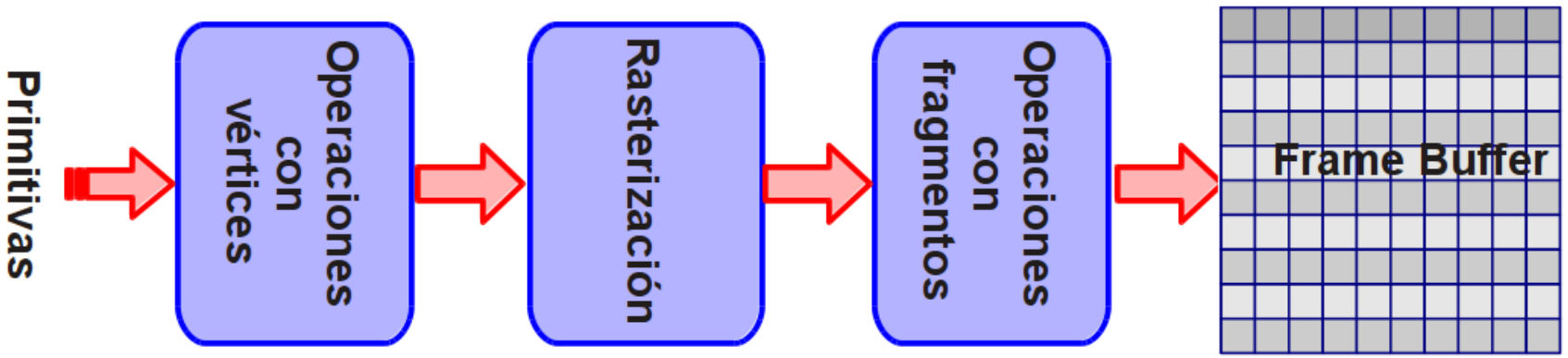
Los dispositivos de visualización generan imágenes 2D. Es necesario transformar las posiciones 3D del modelo en posiciones 2D. Para generar una imagen de un modelo 3D se debe especificar la posición, orientación y focal de la **cámara**.

Cámara

Conceptualmente la imagen se genera como si tuviésemos una cámara virtual en el mundo 3D. La imagen que se genera es la proyección del mundo en un rectángulo del plano de proyección de la cámara virtual (ventana). Esta proyección se transforma al viewport del dispositivo de salida.

Cauce

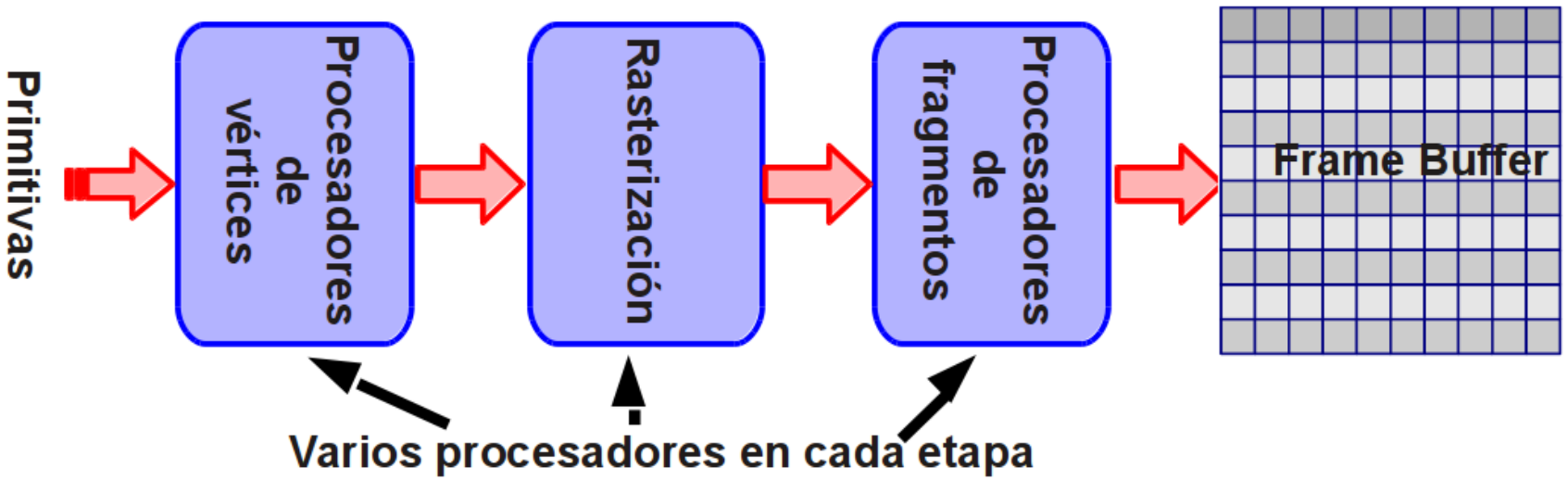
OpenGL recibe primitivas y genera una imagen en el frame buffer. Con las primitivas se realizan sucesivamente operaciones a nivel de vértices, se rasterizan y operaciones a nivel de fragmento.



GPU

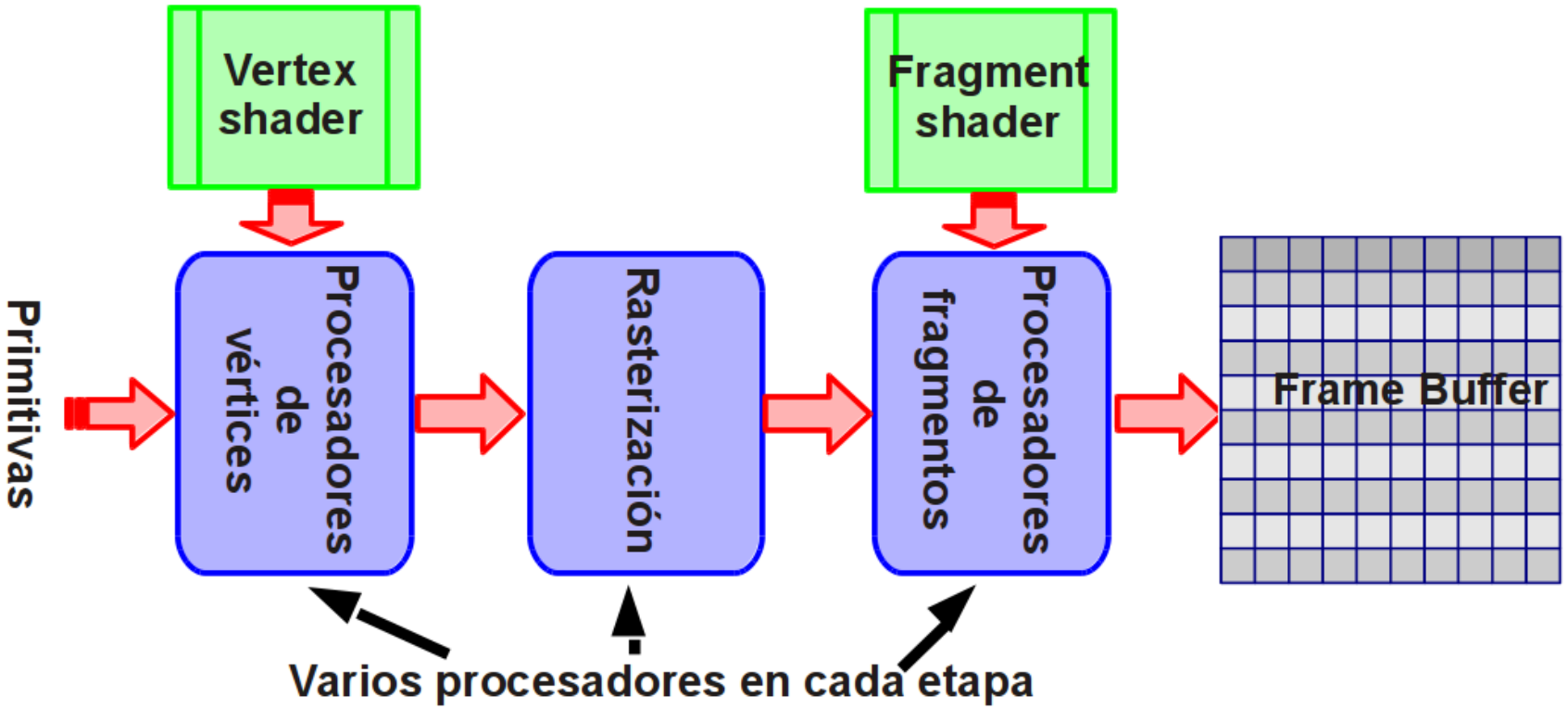
Las GPU son sistemas multiprocesador (SIMD) de propósito especial.

Si OpenGl está implementado a nivel hardware cada una de estas etapas se ejecuta en un conjunto de procesadores especializados.

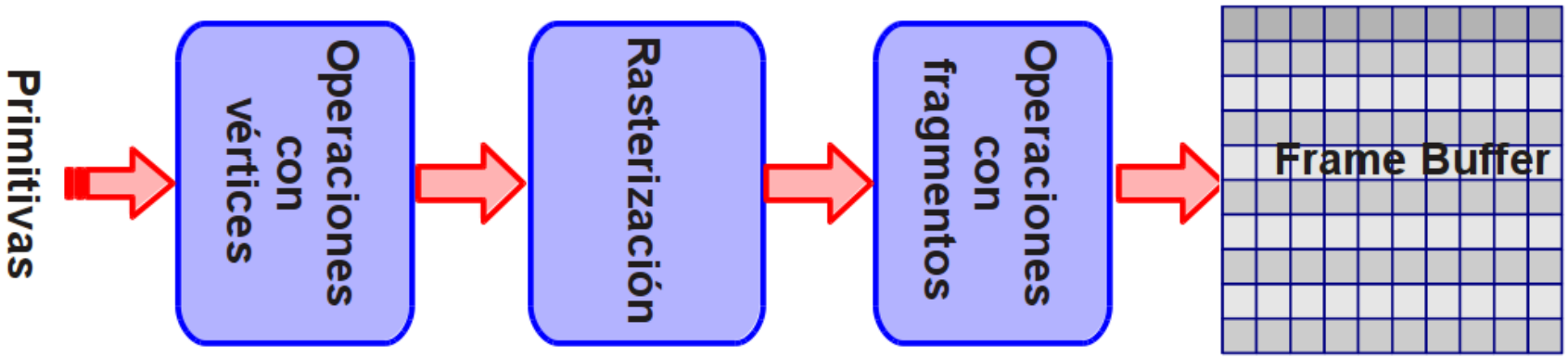


Cauce programable

En las tarjetas modernas es posible modificar el código que ejecutan estos procesadores, cargando programas especiales para ellos (shader).



Operaciones en cada etapa



En las fases programables (operaciones con vértices y con fragmentos) se realizan las transformaciones, la iluminación y la texturización.

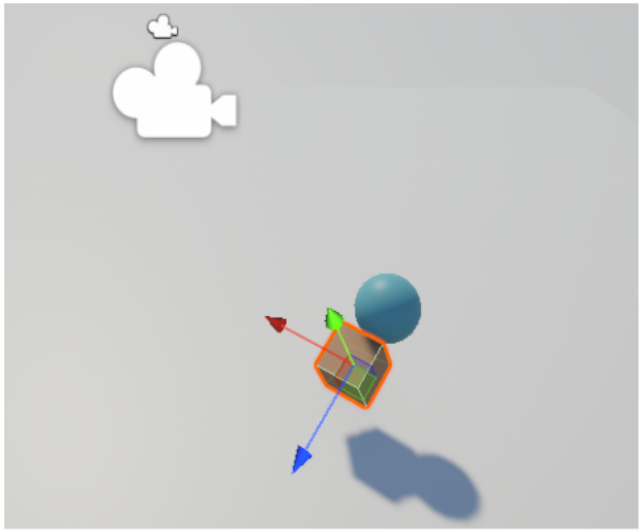
Iluminación: Por vértice o polígono en los procesadores de vértices. Si se calcula por pixel en los procesadores de fragmentos.

Texturas: En los procesadores de fragmentos.

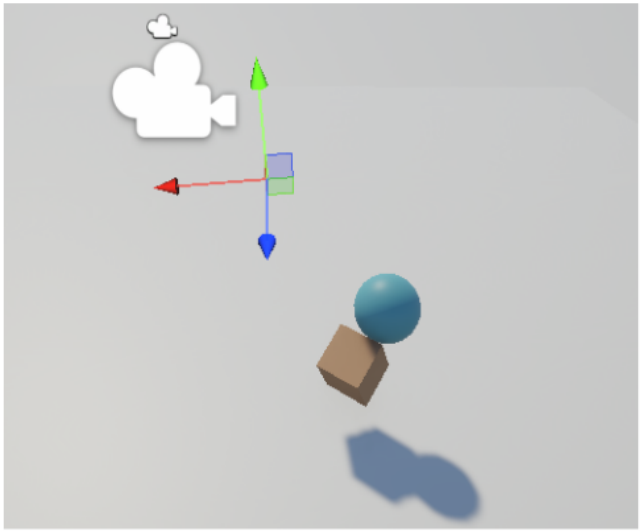
Transformación de visualización: En los procesadores de vértices.

Recortado: En la etapa de geometría (antes de la rasterización).

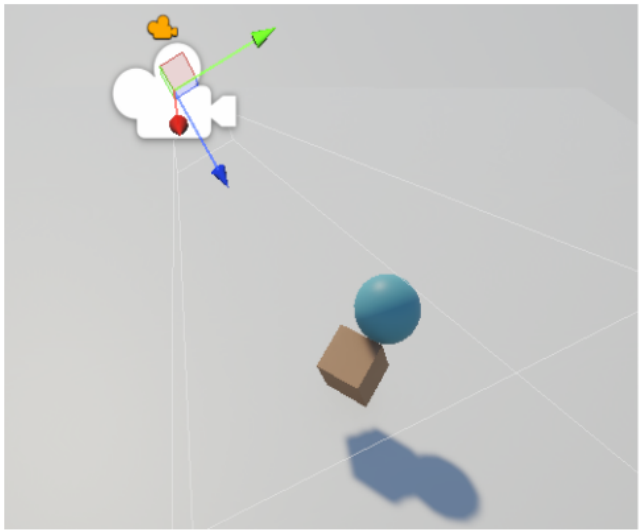
Sistemas de coordenadas



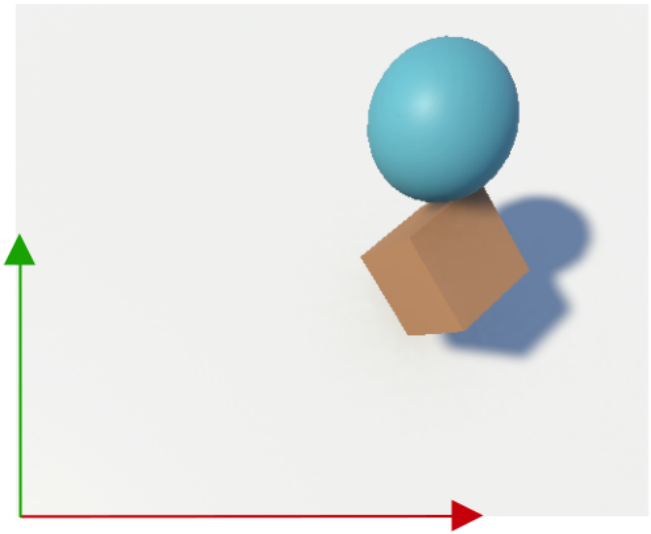
Coordenadas de objeto



Coordenadas del mundo

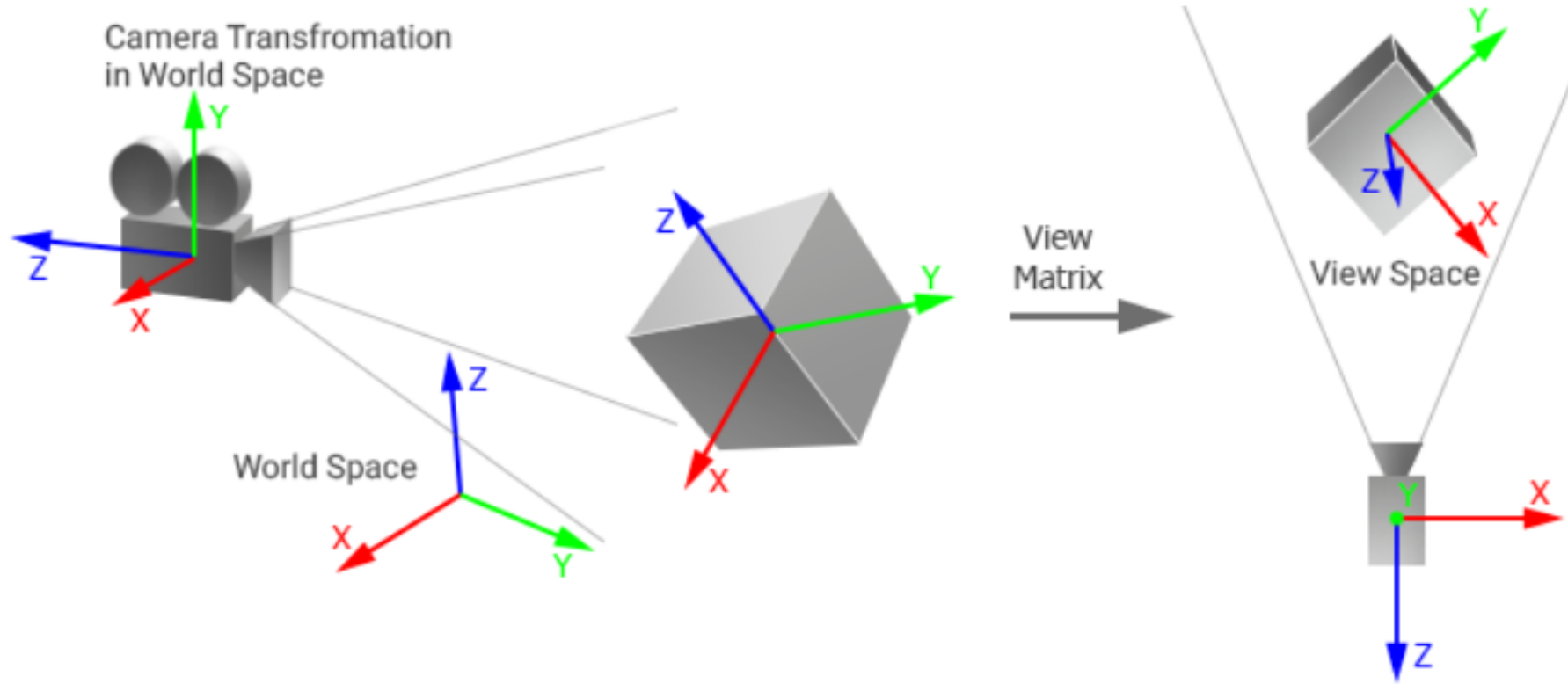


Coordenadas de cámara



Coordenadas de dispositivo

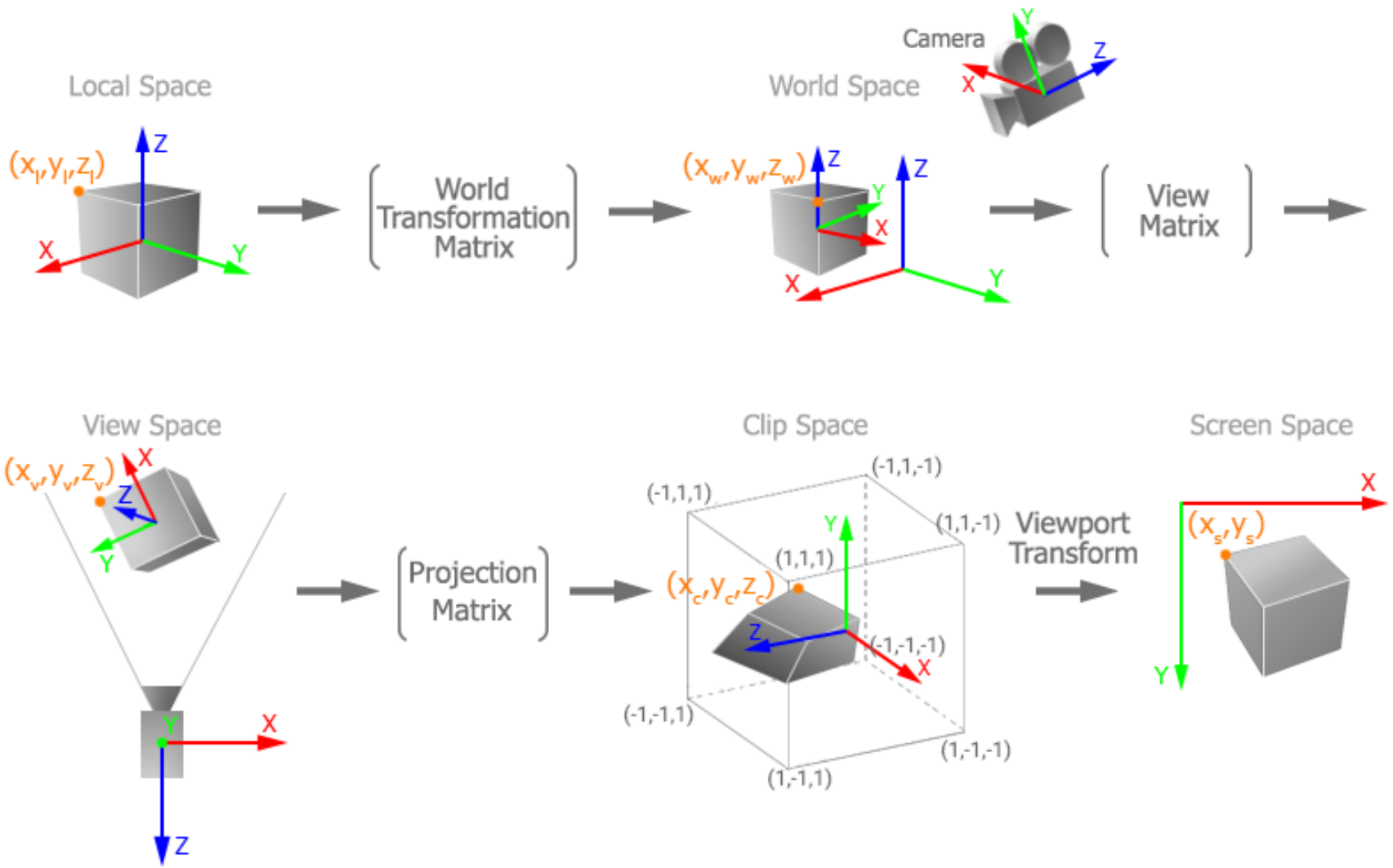
Sistemas de coordenadas de cámara



Object and camera in world space

Object in view space

Coordenadas normalizadas de dispositivo



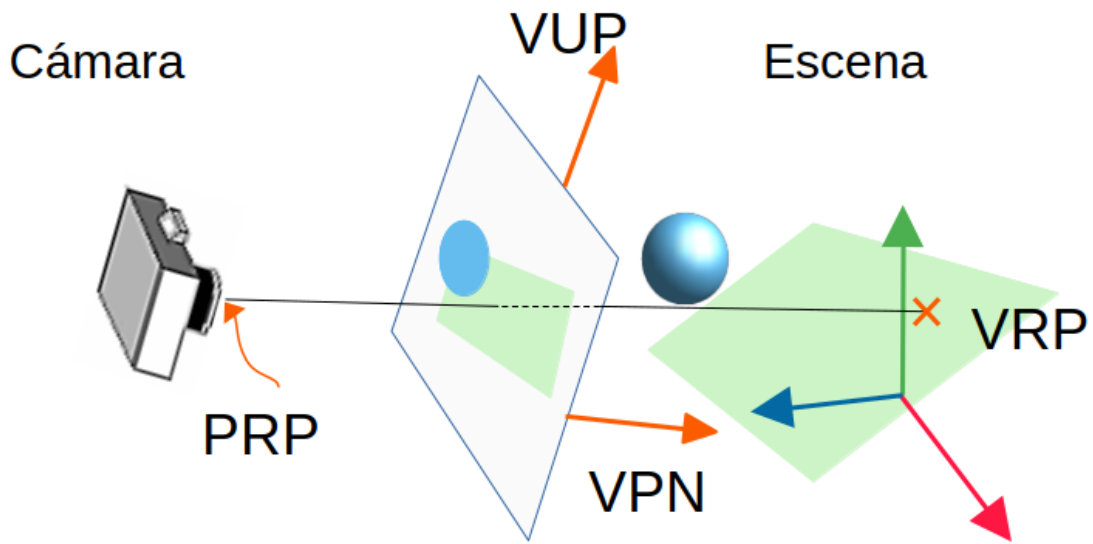
Parámetros del sistema de coordenadas de cámara

PRP (Projection reference point) Punto del espacio foco de la proyección, donde esta situada la cámara.

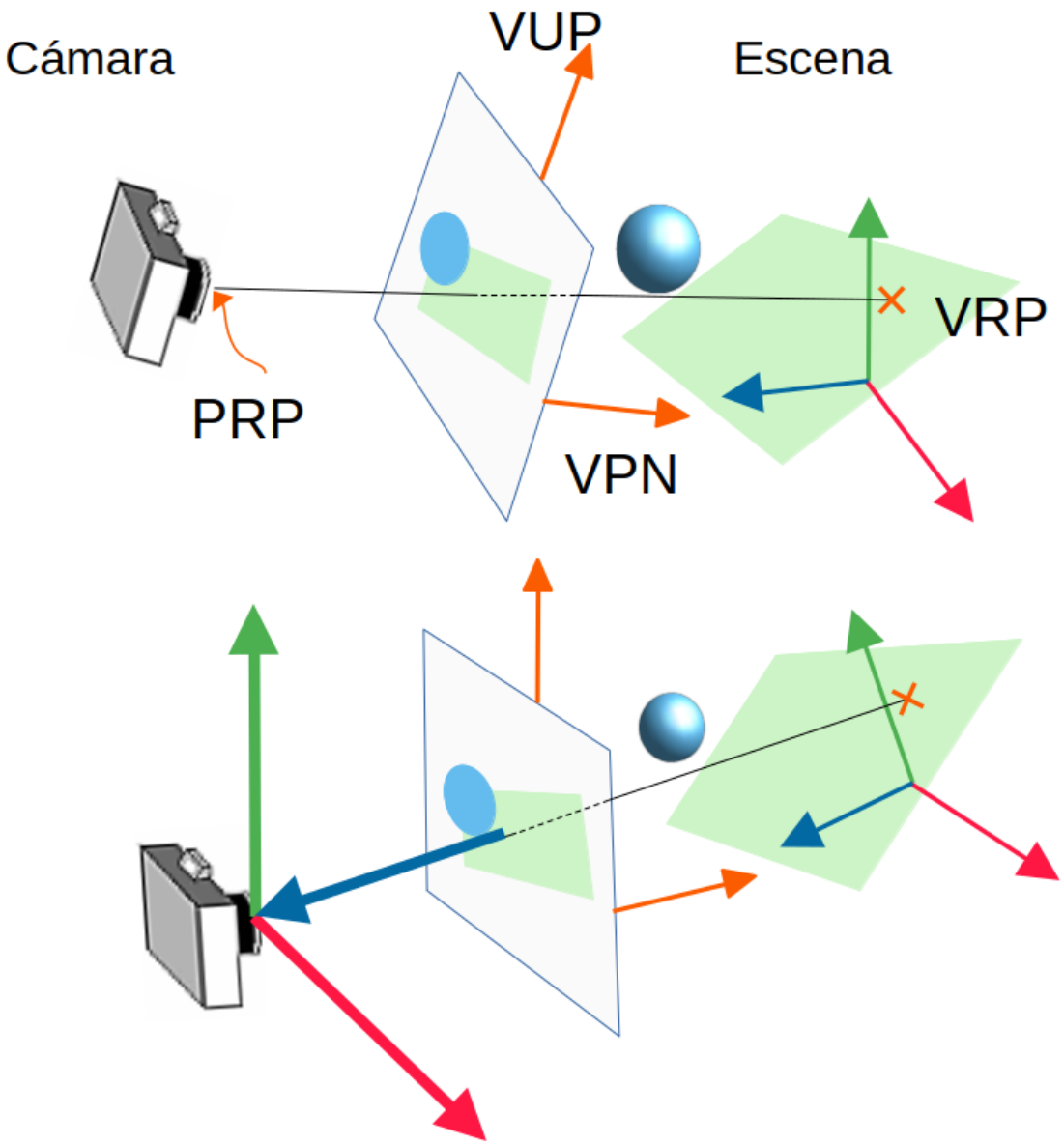
VPN (View plane normal) Vector perpendicular al plano de proyección.

VRP (View reference point) Indica el punto de interés, o target, que es el punto que se proyecta en el centro de la imagen. Podemos pensar en él como el punto al que se mira ("look at point").

VUP (Vector vertical) Dirección vertical de la cámara.



Parámetros del sistema de coordenadas de cámara

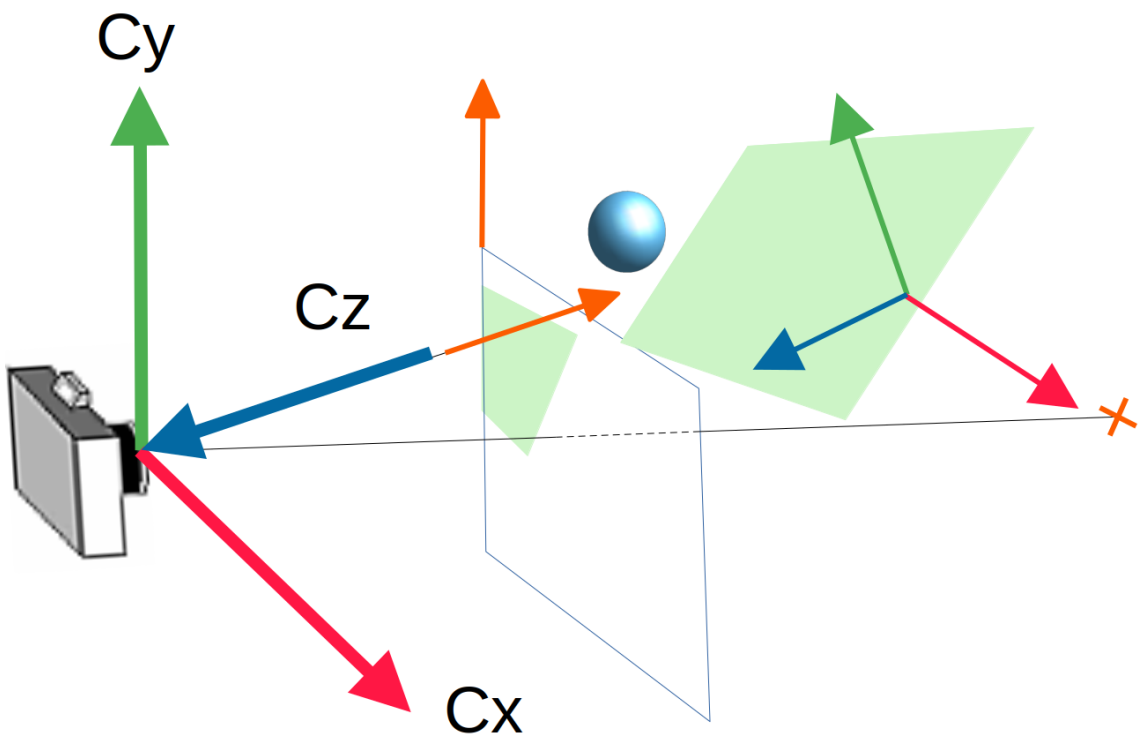


Sistema de referencia de la cámara

$$\mathbf{c_z} = \frac{\mathbf{VPN}}{\|\mathbf{VPN}\|} \qquad \text{(eje Z paralelo a VPN)}$$

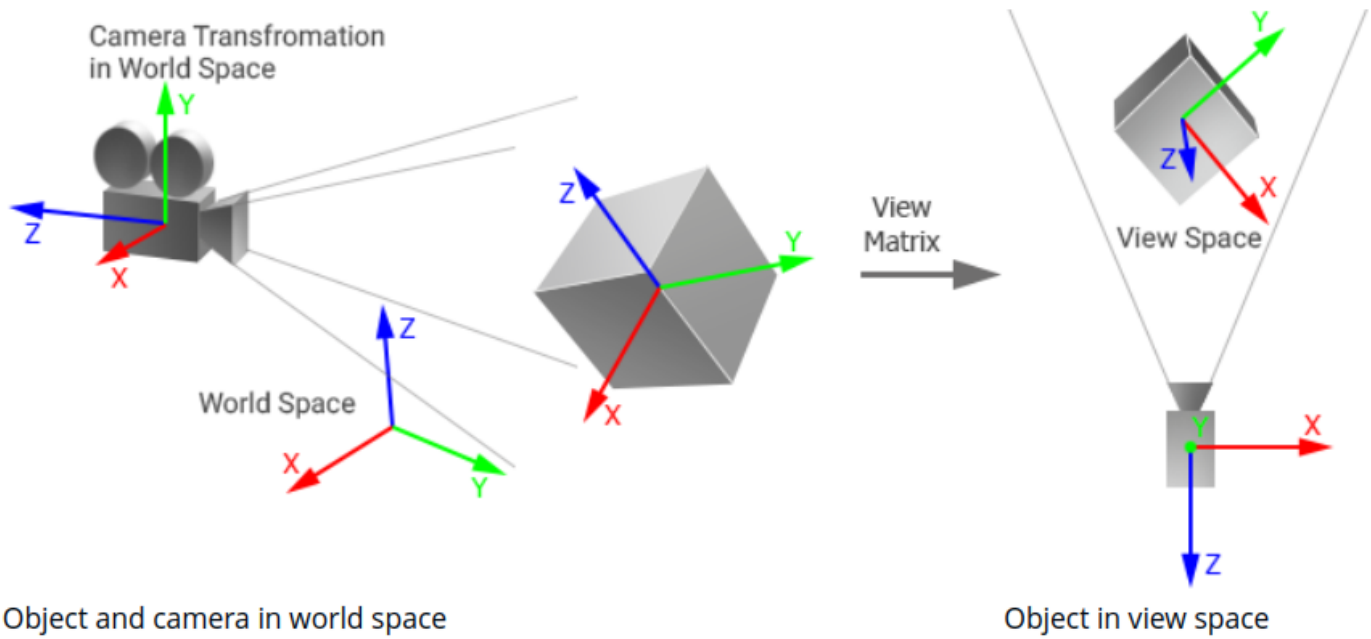
$$\mathbf{c_x} = \frac{\mathbf{VPN} \times \mathbf{VUP}}{\|\mathbf{VPN} \times \mathbf{VUP}\|} \qquad \text{(eje X perpendicular a VPN y VUP)}$$

$$\mathbf{c_y} = \mathbf{c_z} \times \mathbf{c_x} \qquad \text{(eje Y perpendicular a los otros dos)}$$



Transformación a coordenadas de cámara

- Traslación de $-PRP$, haciendo que el origen de coordenadas este en la cámara.
- Rotar respecto al eje X para llevar el vector VPN al plano XZ.
- Rotar respecto al eje Y para llevar VPN al eje $-Z$.
- Rotación respecto al eje Z para llevar a c_y al eje Y.

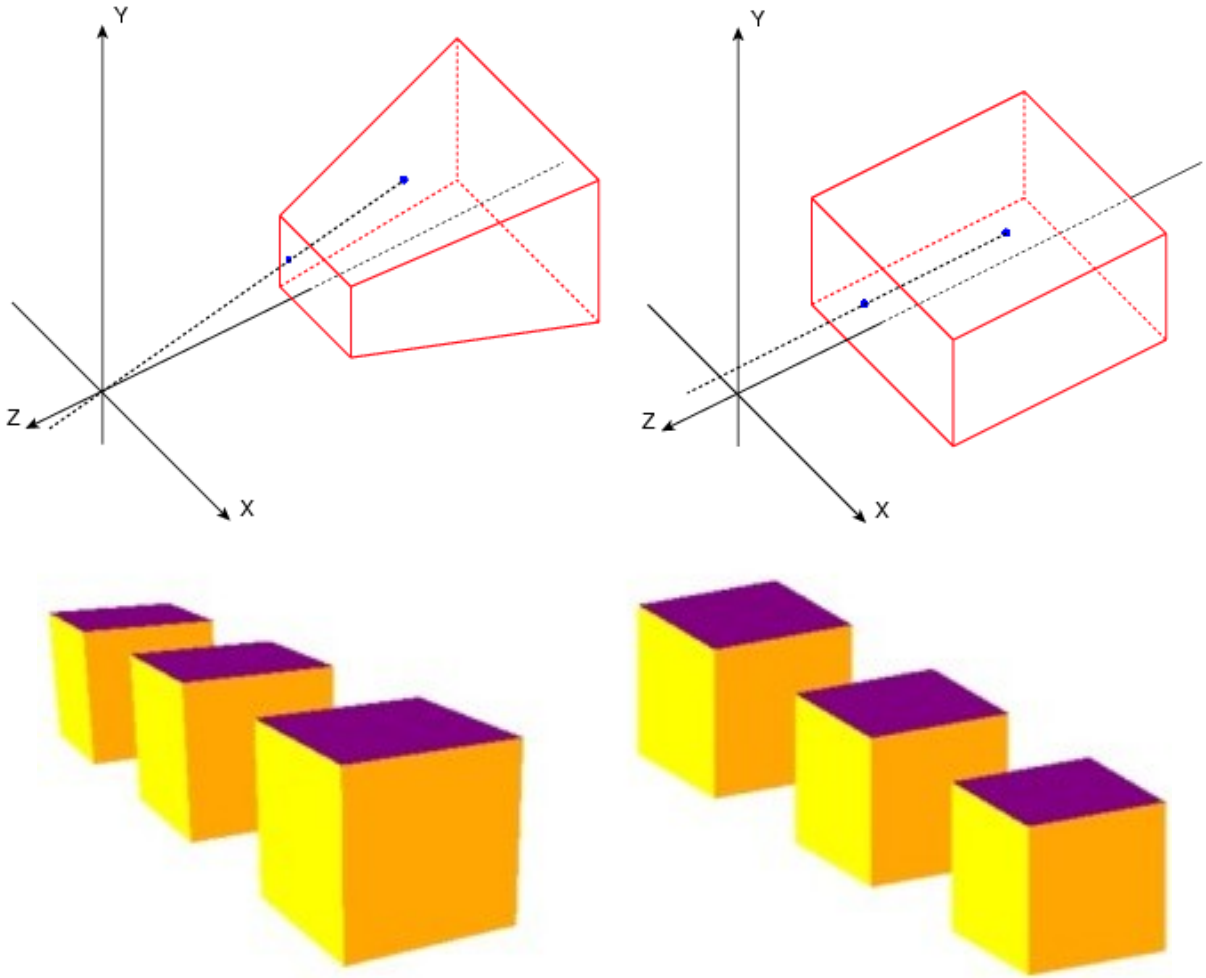


Object and camera in world space

Object in view space

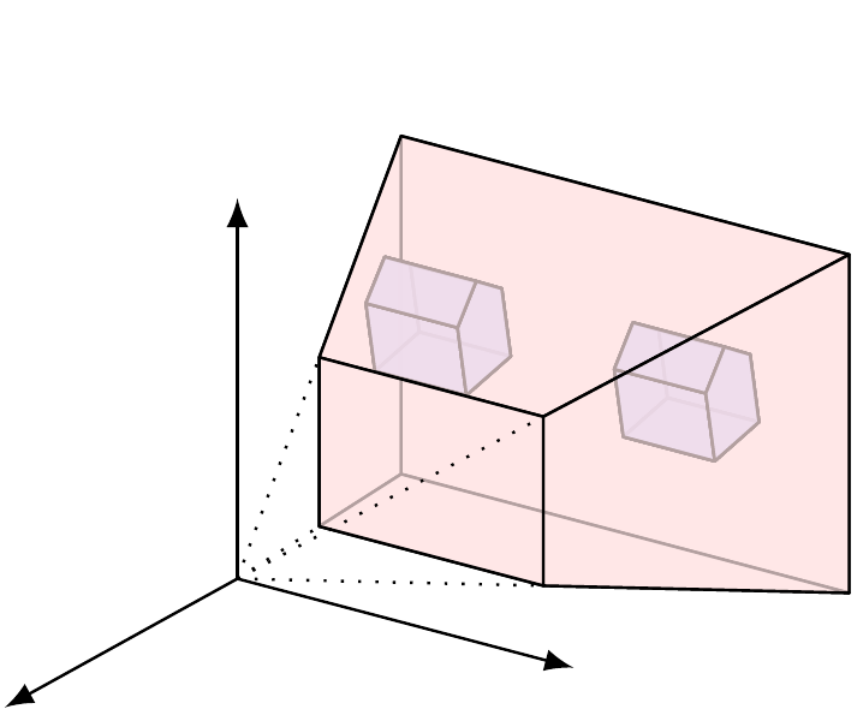
Transformación de proyección

Se emula la proyección que ocurre en las cámaras fotográficas sobre el **plano de visión**. Se modela como una transformación lineal en coordenadas homogéneas, puede ser perspectiva (izq.) u ortográfica (der.):



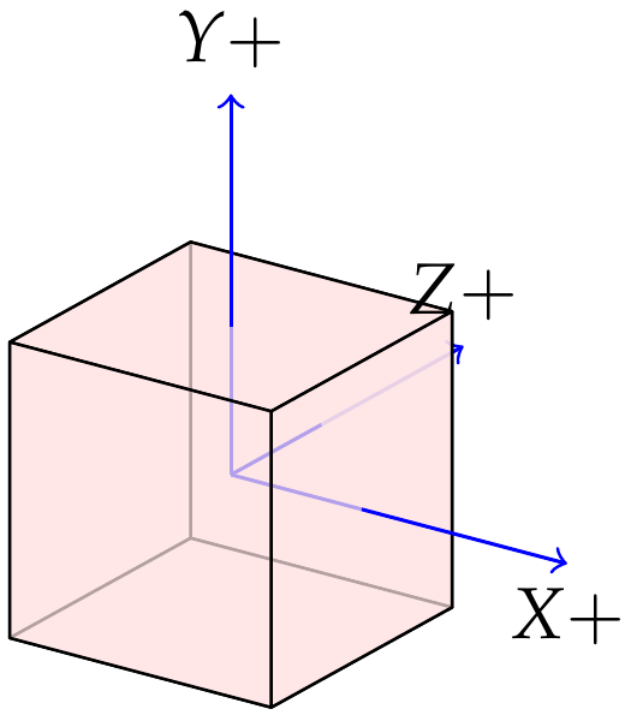
Efecto de la transformación de proyección

El efecto de la **transformación de proyección** (matriz P) es convertir la región visible del espacio en un cubo de lado 2 unidades y con centro en el origen.



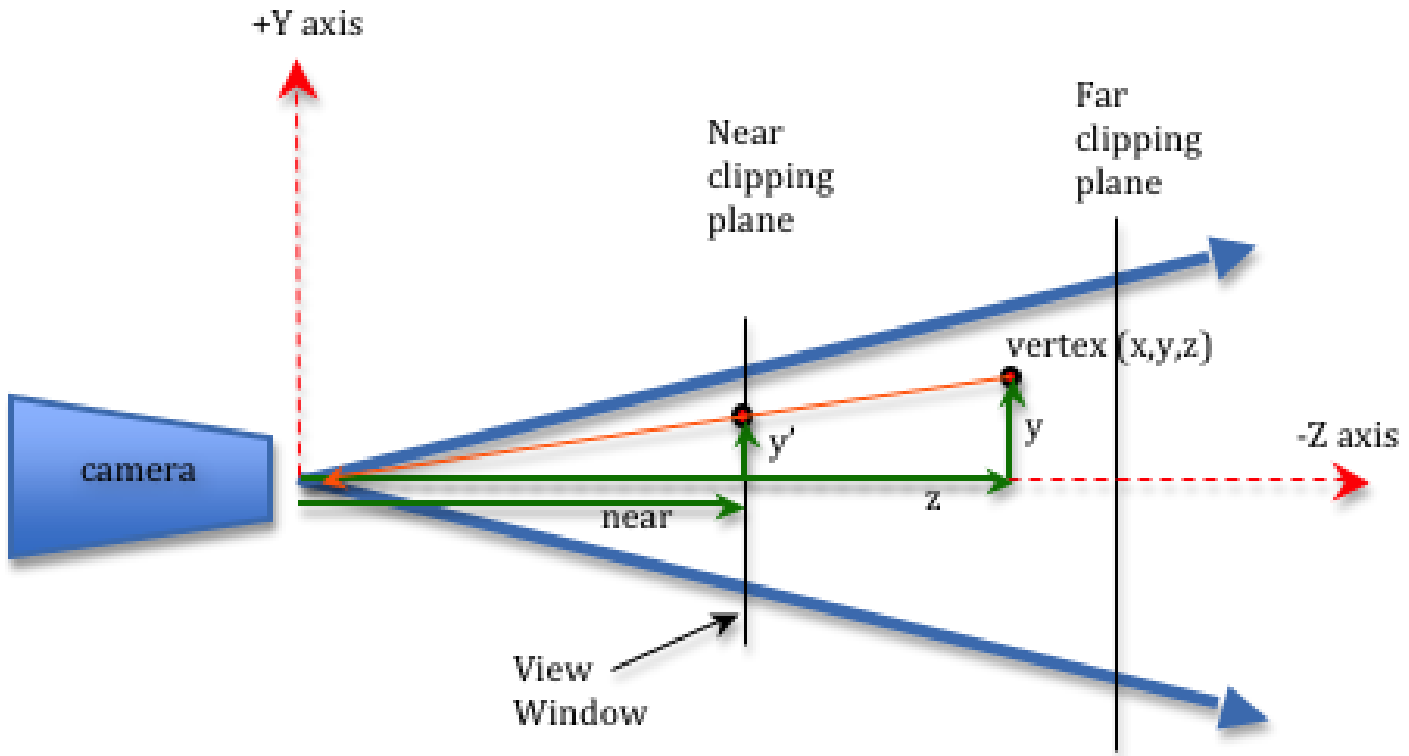
coordenadas
de ojo
(*eye coordinates*)

P $\longrightarrow$



coordenadas
de recortado
(*clip coordinates*)

Ecuación de la transformación de perspectiva



Por semejanza de triángulos que

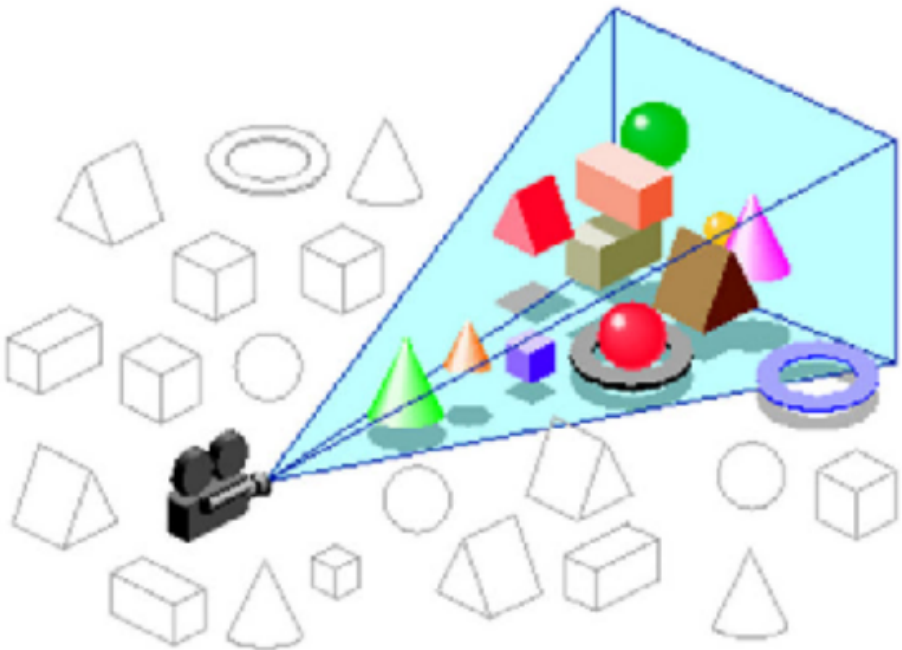
$$y'/near = y/z \qquad => \qquad y' = y \cdot near/z$$

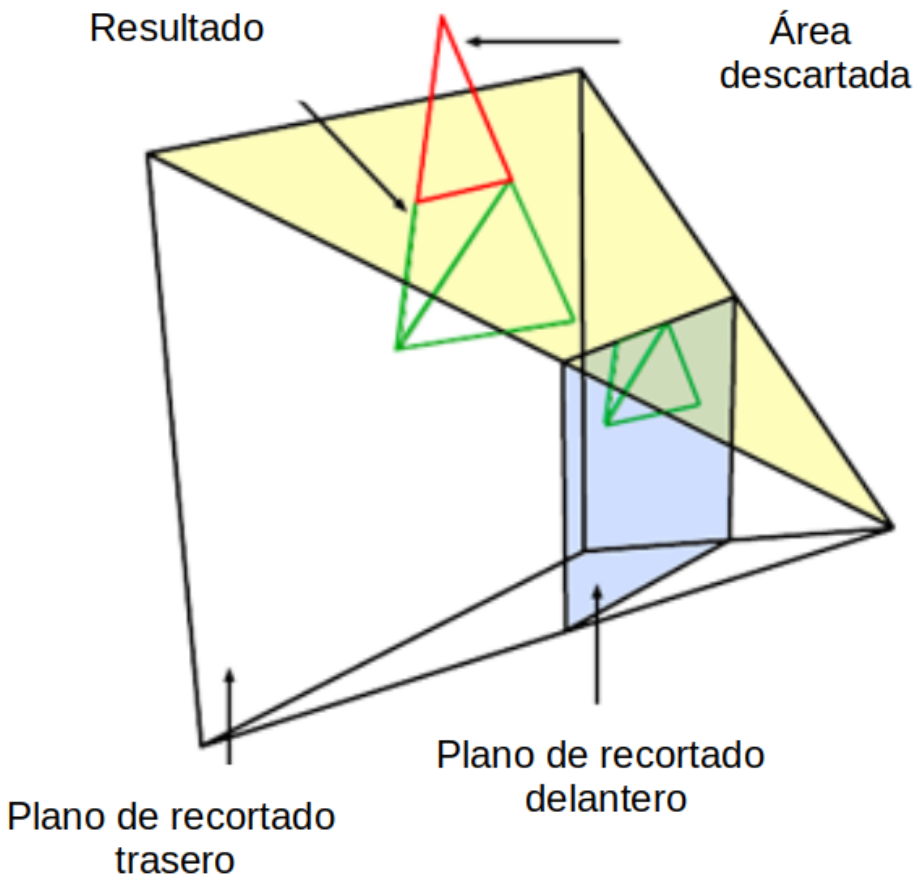
La transformación se representa como una matriz 4x4 en coordenadas homogéneas con valores no nulos en la última fila.

Recortado

View frustrum culling: Descarta los polígonos que está totalmente fuera del volumen de visión

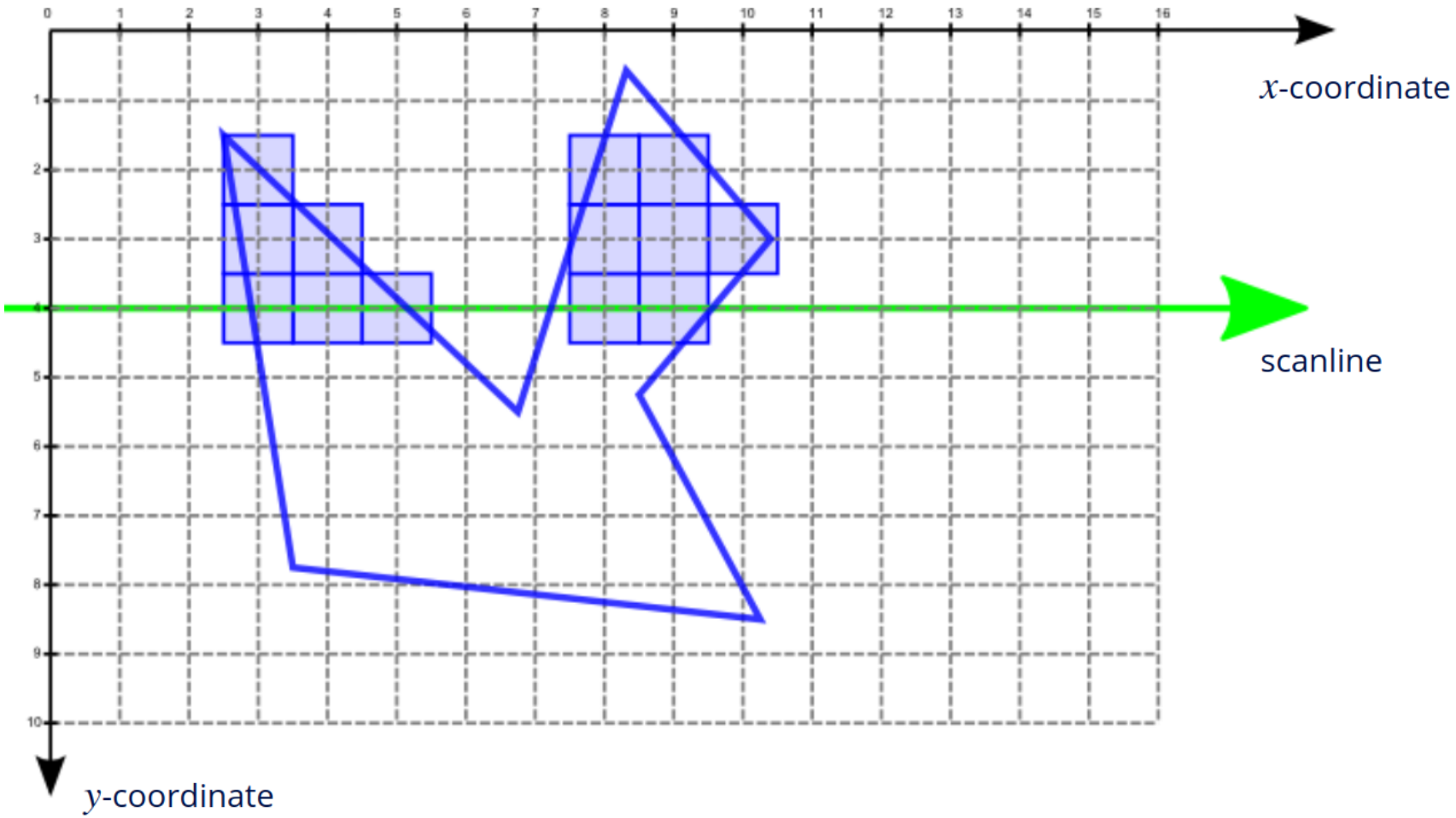
Clipping: Se calcula y triangula la parte visible, descartando el resto





Rasterización

Las primitivas se *ensamblan* a la salida de los procesadores de vértices y se descomponen en pixeles.



Transformaciones en OpenGL

OpenGL almacena en su estado dos pilas de matrices de transformación. En una almacena la concatenación de las matrices de modelado y vista, en la otra la transformación de proyección. Se puede seleccionar a cual de las dos matrices se le concatenan las transformaciones que se den con la orden *glMatrixModel*:

```
glMatrixMode(GL_MODELVIEW);

glMatrixMode(GL_PROJECTION);
```

Las transformaciones que se creen se componen con la actual.

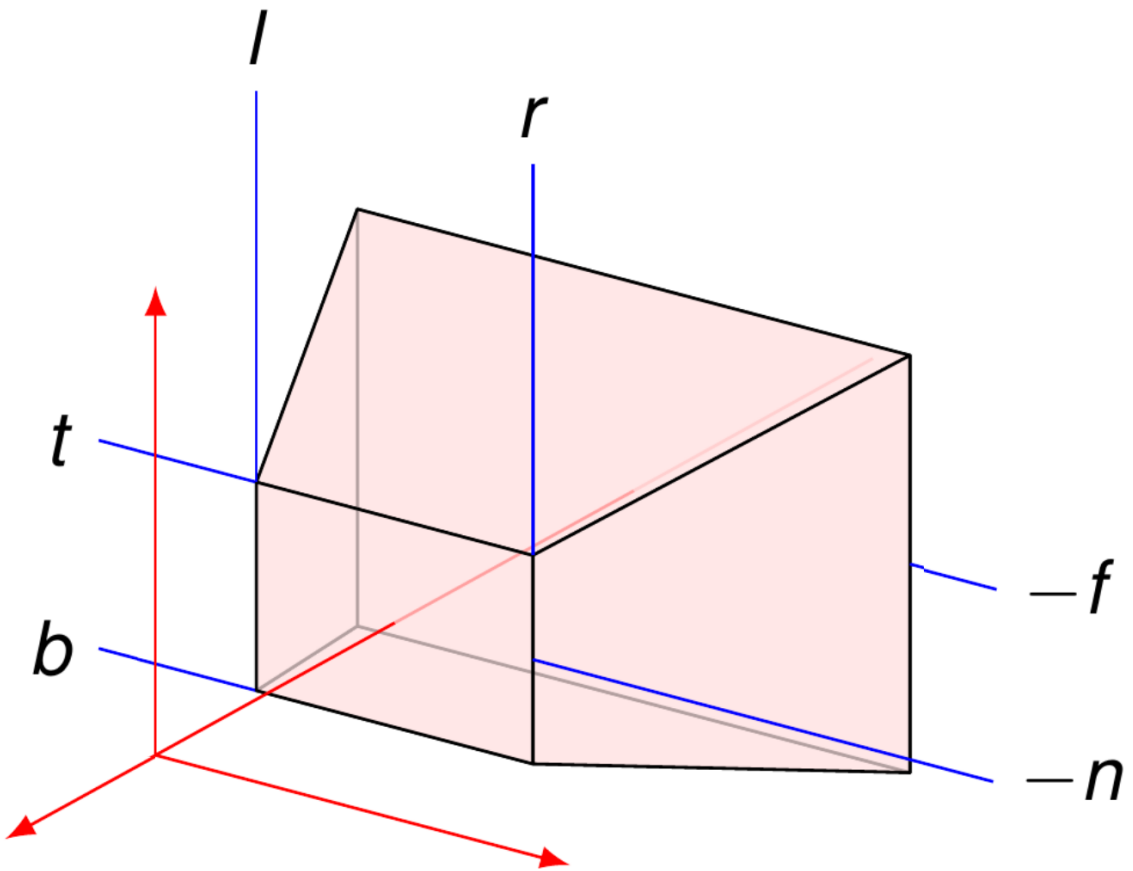
Para inicializar la transformación se usa *glLoadIdentity*:

```
glLoadIdentity()
```

Proyección perspectiva con OpenGL

Para usar una transformación de perspectiva se utiliza la función *glFrustum*:

```
glFrustum( GLdouble l, GLdouble r, GLdouble b, GLdouble t,
           GLdouble n, GLdouble f ) ;
```

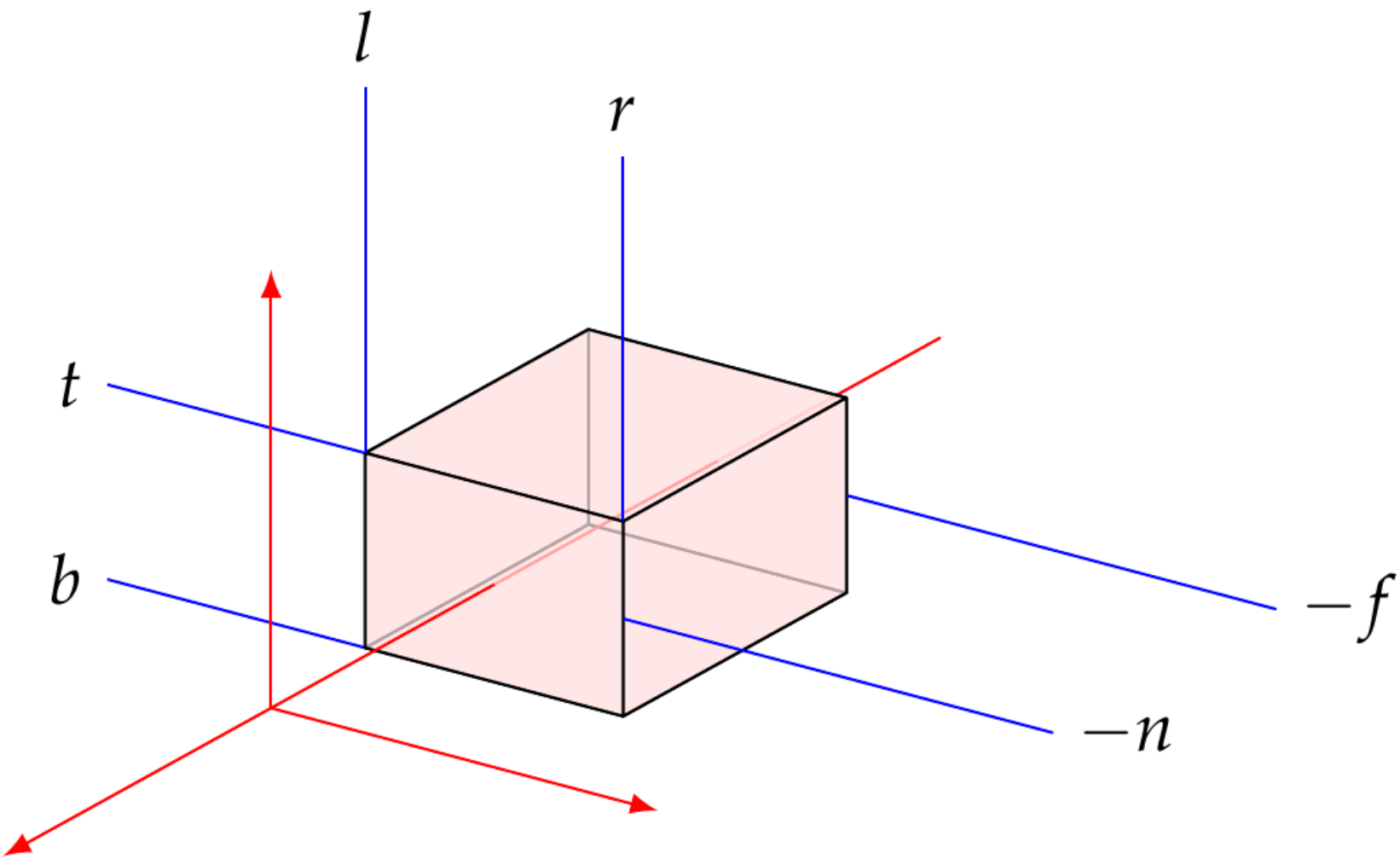


que genera la matriz de transformación y la componen con la matriz de proyección existente.

Proyección paralela con OpenGL

Para usar proyección ortográfica debemos usar *glOrtho*:

```
glOrtho( GLdouble l, GLdouble r, GLdouble b, GLdouble t,
         GLdouble n, GLdouble f ) ;
```



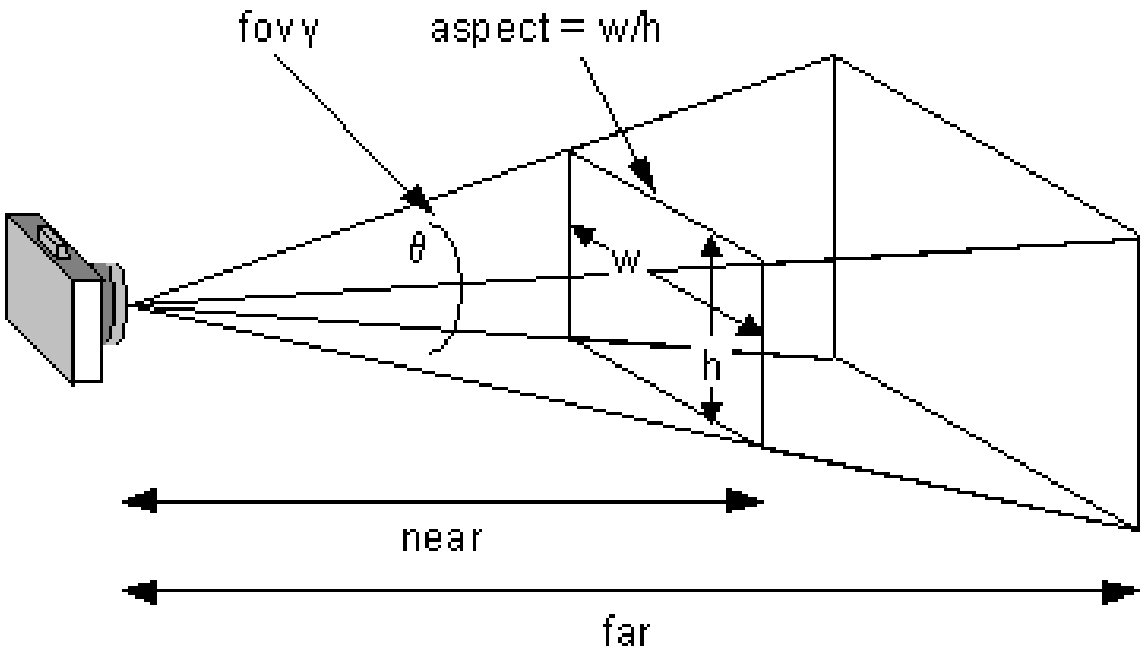
Perspectiva con gluPerspective

La función *gluPerspective* se puede usar también para definir proyecciones en perspectiva

```
gluPerspective(GLdouble alfa, GLdouble a, GLdouble near, GLdouble far);
```

esta función equivale a un *glFrustum* con $r = -l$ y $t = -b$, con:

- $alfa$ es la apertura vertical, en grados, (0 a 180.0).
- a es la relación de aspecto de la imagen (ancho dividido por alto).



Fijar la matriz de viewport en OpenGL

En cualquier momento es posible cambiar la definición del *viewport* usando la función **glViewport**:

```
glViewport( GLint xleft, GLint ybottom, GLsizei w, GLsizei h ) ;
```

Ejemplo mínimo:

```
void inicializaVentana (GLsizei ancho, GLsizei alto)
{
  glViewport (0, 0, ancho, alto); anchura y altura de la ventana X
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  if (ancho > 0) alto = alto / ancho;
  glFrustum (-1, 1, -calto, calto, 1.5, 1500);
  glMatrixMode (GL_MODELVIEW);
  glLoadIdentity ();
}
...
  glutReshapeFunc (inicializaVentana);
```

Especificación de la matriz de modelado y visualización en OpenGL

La matriz *modelview* almacena la concatenación de las transformaciones de modelado y vista se puede especificar en OpenGL mediante estos pasos:

1. Hacer la llamada **`glMatrixMode`**(**`GL_MODELVIEW`**), para indicar que las siguientes operaciones actuan sobre la matriz *modelview* M.
2. usar **`glLoadIdentity`**() para hacer M igual a la matriz identidad.
3. usar rotaciones y traslaciones o **`gluLookAt`** para componer una matriz de vista con M.
4. usar una (o varias) llamadas para componer una matriz de modelado con M (**`glScale`**, **`glRotate`**, o **`glTranslate`**).

Los pasos 1 y 2 se pueden hacer en el callback de redimensionado de ventana o cuando se cambie la transformación de proyección (en este caso el callback de dibujo se debe encerrar entre un **`glPushMatrix`** y un **`glPopMatrix`**).

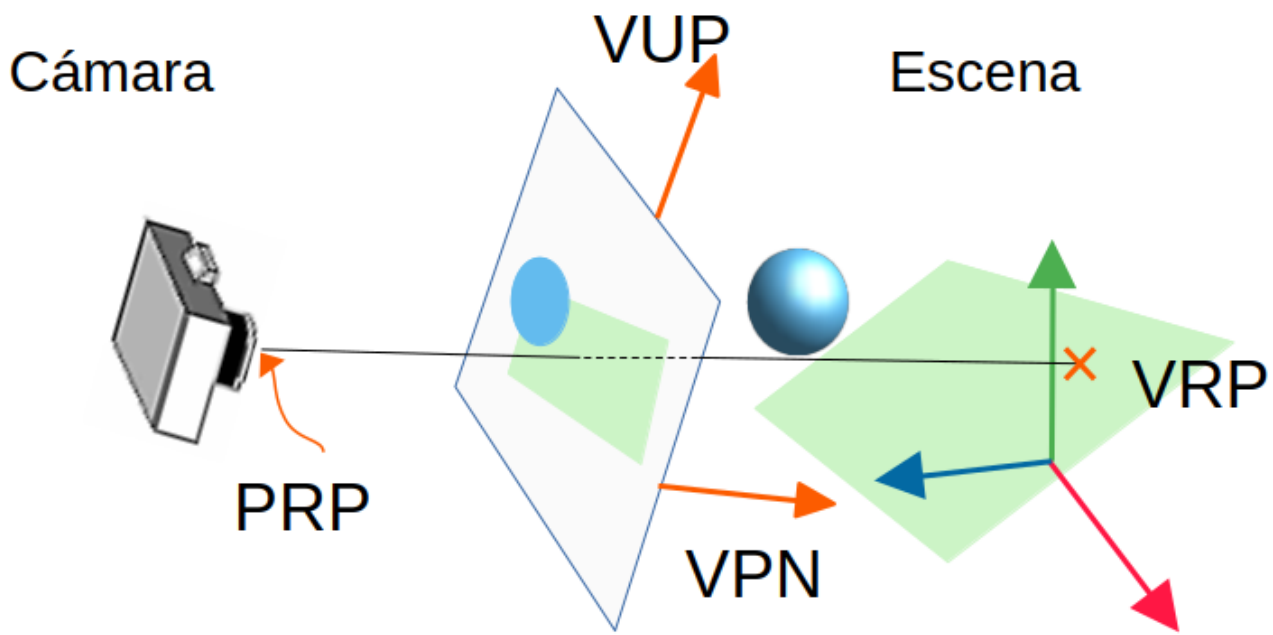
Transformación de visualización

```
// Paso 1: Trasladar el PRP al origen (negativo de PRP)
glTranslatef(-PRP.x, -PRP.y, -PRP.z);

// Paso 2: Rotar alrededor del eje X para alinear VPN al plano XZ
// Calcular ángulo en grados
GLfloat thetaX = atan2(VPN.y, VPN.z) * 180.0 / M_PI;
// Rotación en X para alinear VPN al plano XZ
glRotatef(-thetaX, 1.0f, 0.0f, 0.0f);

// Paso 3: Rotar alrededor del eje Y para llevar VPN al eje -Z
// Calcular ángulo en grados
GLfloat VPNxz = sqrt(VPN.x * VPN.x + VPN.z * VPN.z);
GLfloat thetaY = atan2(VPN.x, VPN.z) * 180.0 / M_PI;
// Rotación en Y para alinear VPN al eje -Z
glRotatef(-thetaY, 0.0f, 1.0f, 0.0f);

// Paso 4: Rotar alrededor del eje Z para alinear VUP al eje Y
// Calcular ángulo en grados
GLfloat thetaZ = atan2(VUP.x, VUP.y) * 180.0 / M_PI;
// Rotación en Z para alinear VUP al eje
glRotatef(thetaZ, 0.0f, 0.0f, 1.0f);
```

Transformación de visualización con gluLookAt

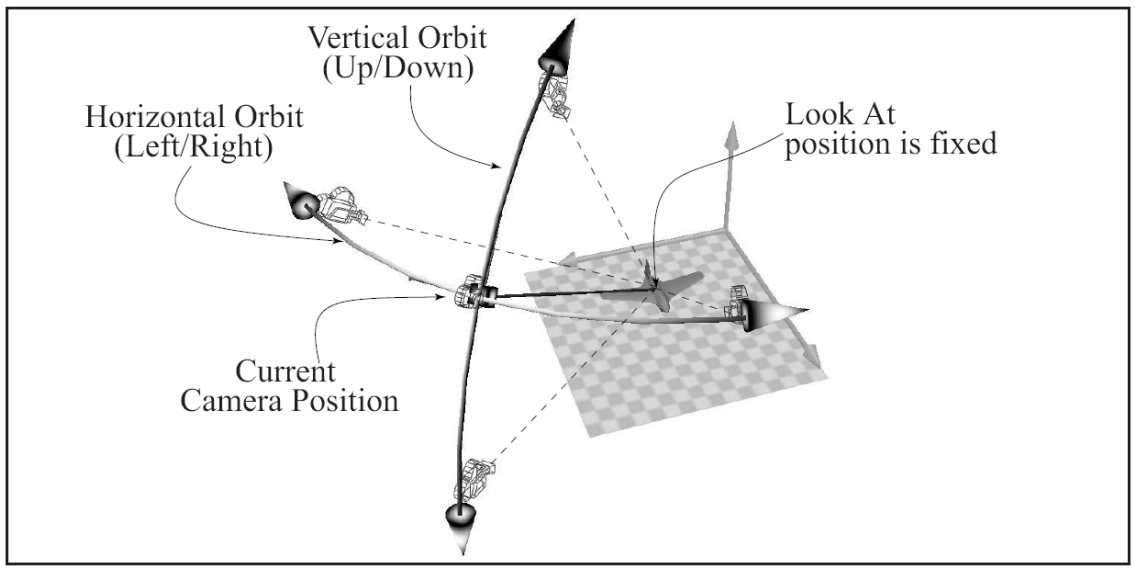
La función **gluLookAt** (de la librería GLU) permite componer una matriz de vista de forma muy cómoda, ya que acepta directamente las componentes de **o**,**a** y **u** como parámetros. Está declarada como sigue:

```
void gluLookAt( GLdouble PRP_x, GLdouble PRP_y, GLdouble PRP_z,
                GLdouble VRP_x, GLdouble VRP_y, GLdouble VRP_z,
                GLdouble VUP_x, GLdouble VUP_y, GLdouble VUP_z ) ;
```



Cámara orbital

La cámara mira al objeto que se está visulizando, moviendose en torno a él sobre una esfera.



(K. Sung, P. Shirley, S. Baer: Essentials of Interactive Computer Graphics: Concepts and Implementation)

Cámara orbital

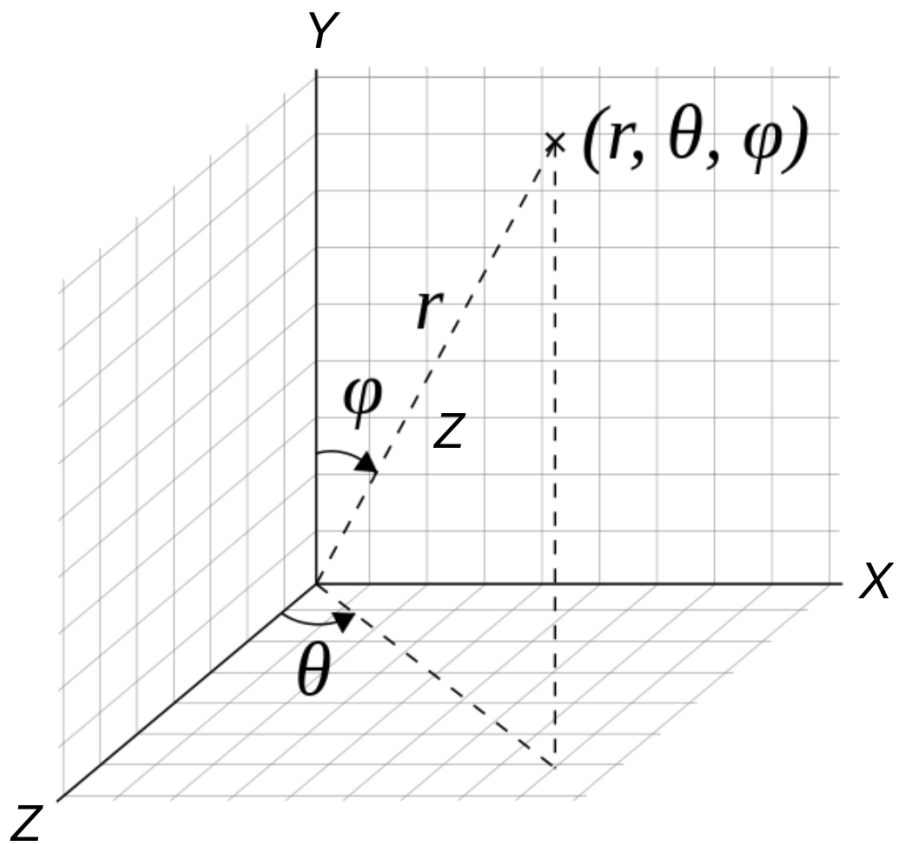
La posición de la cámara puede especificarse en coordenadas esféricas.

$$X = r\ sin(\phi)\ sin(\theta)$$

$$Y = r\ cos(\phi)$$

$$Z = r\ sin(\phi)\ cos(\theta)$$

V_{up} se debe calcular de forma que no sea perpendicular al plano proyección.

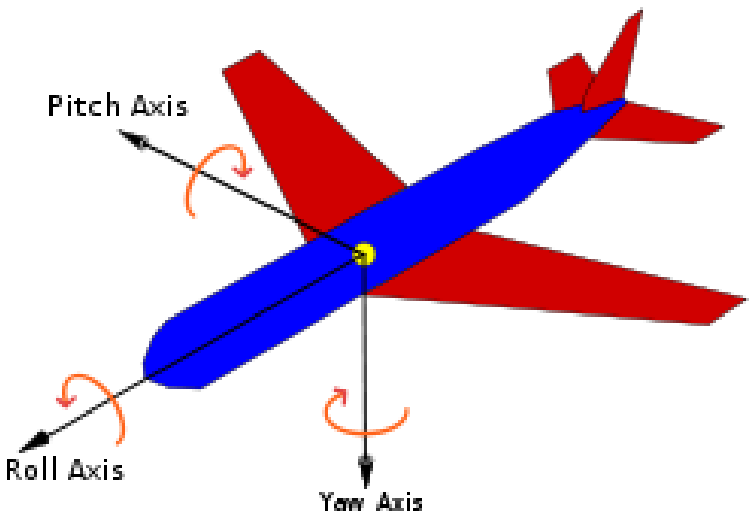


Ángulos de Euler

Alternativamente, la orientación (o la posición de una cámara orbital) se puede expresar como tres giros (uno respecto a cada eje).

Esta representación de la orientación se conoce como ángulos de Euler.

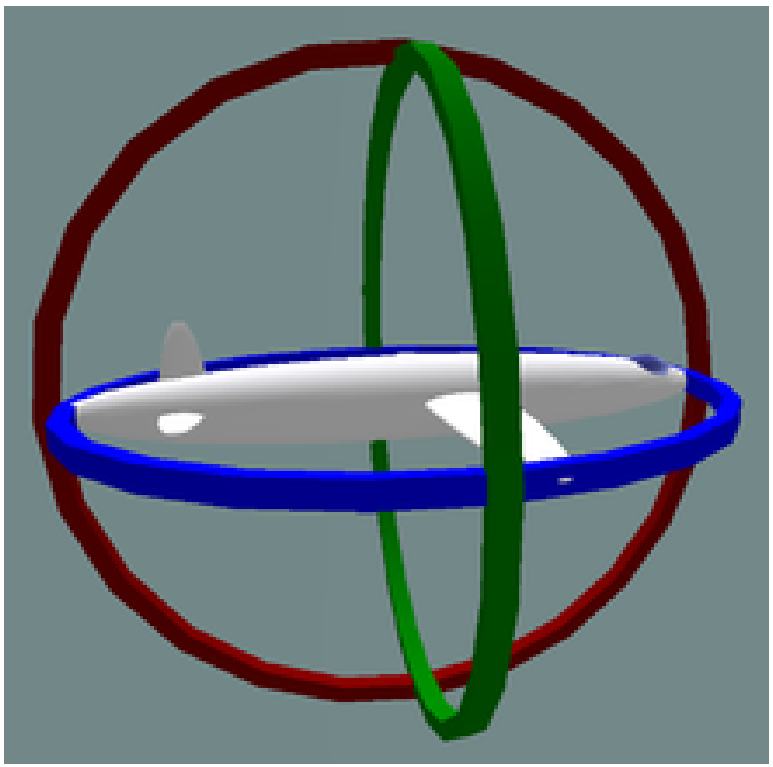
Permite controlar la orientación con tres grados de libertad (incluyendo el control sobre V_{up}.

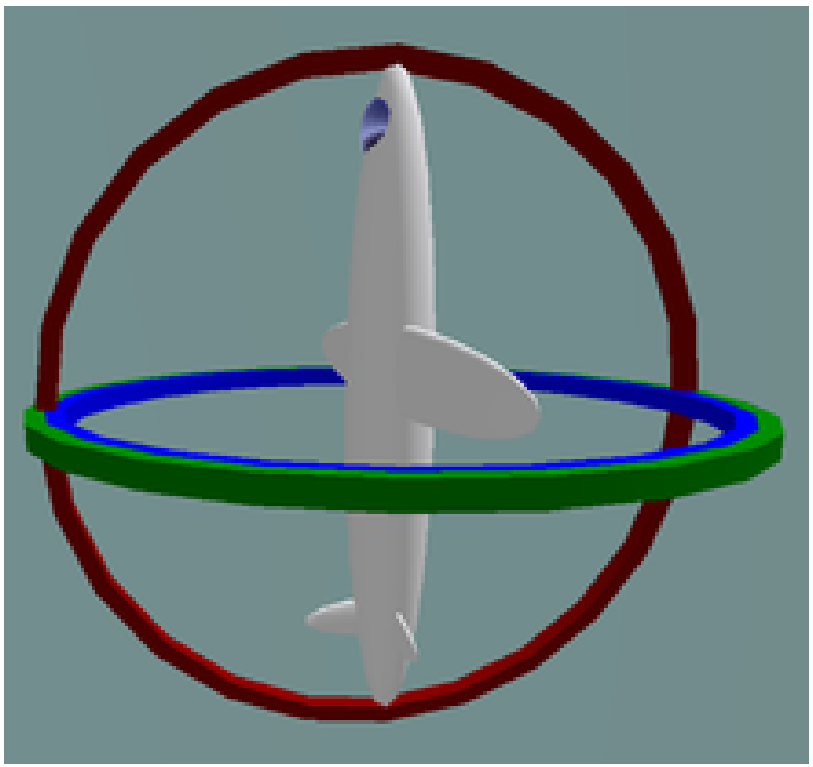


Gimbal lock

La utilización de ángulos de Euler tiene el problema de generar configuraciones en las que dos ejes se alinean provocando la pérdida de uno de los grados de libertad.

Este fenómeno se conoco como gimbal lock.





Cámara orbital usando ángulos de Euler:

```
void Dibuja( void ){
  ...
  glPushMatrix();
  glClearColor(1,1,1,1);
  glClear( GL_COLOR_BUFFER_BIT  | GL_DEPTH_BUFFER_BIT );
  glTranslatef(0, 0,-D); // distancia camara VRP-PRP
  glRotatef( view_rotz, 0.0, 0.0, 1.0 ); // spin
  glRotatef( view_roty, 1.0, 0.0, 0.0 ); // Posicion angular camara
  glRotatef( view_rotx, 0.0, 1.0, 0.0 );
  glTranslatef(-VRPx,-VRPy,-VRPz); // Centro de atencion al origen
  ...
  glPopMatrix();
  glutSwapBuffers();
}
```

Cámara orbital en OpenGL con gluLookAt

```
void Dibuja() {
  ....
  glPushMatrix();
  glClearColor(1,1,1,1);
  glClear( GL_COLOR_BUFFER_BIT  | GL_DEPTH_BUFFER_BIT );  ....
  glClear( GL_COLOR_BUFFER_BIT  | GL_DEPTH_BUFFER_BIT );
  ...
  // Convertir los ángulos de rotación a radianes
  GLfloat rotx_rad = view_rotx * M_PI / 180.0;
  GLfloat roty_rad = view_roty * M_PI / 180.0;
  // Calcular la posición de la cámara en coordenadas esféricas
  GLfloat camX = Ax + D * cos(roty_rad) * sin(rotx_rad);
  GLfloat camY = Ay + D * sin(roty_rad);
  GLfloat camZ = Az + D * cos(roty_rad) * cos(rotx_rad);

  // Definir la cámara usando gluLookAt
  gluLookAt(
        camX, camY, camZ,  // Posición de la cámara
        Ax, Ay, Az,        // Punto de atención (centro de la órbita)
        0.0f, 1.0f, 0.0f );  // Vector "hacia arriba"(Y positivo)
  ...
  glPopMatrix();
  glutSwapBuffers();
}
```

Cámara orbital en OpenGL: interacción

Con teclado en glut

```
static void especial(int k, int x, int y) {
  switch (k) {
    case GLUT_KEY_UP: // Pulsación de tecla cursor arriba
      view_rotx += 5.0; // Invrementar rotacion x
      break;
    case GLUT_KEY_DOWN:
      view_rotx -= 5.0;
      break;
    case GLUT_KEY_LEFT:
      view_roty += 5.0;
      break;
    case GLUT_KEY_RIGHT:
      view_roty -= 5.0;
      break;
    default:
      return;
    }
  glutPostRedisplay();
}
```

Cámara orbital en OpenGL: interacción

Con ratón en glut

```
void clickRaton( int boton, int estado, int x, int y )
{
  if(boton== GLUT_MIDDLE_BUTTON && estado == GLUT_DOWN) {
    Xref=x;          // Almacena posicion (en coor. de pantalla)
    Yref=y;
    RatonPulsado=GL_TRUE;
    }
  else RatonPulsado=GL_FALSE;
        ..
}

void RatonMovido( int x, int y )
{
  float sensibilidad =100;
  if(RatonPulsado)  {       // Si el raton esta pulsado
    view_roty -= (Yref -y) / sensibilidad;
    view_rotx += (Xref -x) / sensibilidad;
    Xref=x;                 // Actualiza ultima posicion de raton
    Yref=y;
    glutPostRedisplay();
    }
  ...
}
```

Cámara primera persona

La cámara está en la posición de un avatar que se mueve en el escenario. El usuario puede controlar la dirección en la que mira la cámara.

La dirección se puede especificar como dos giros en direcciones ortogonales a la dirección de avance del personaje (con las mismas formulas de la posición del caso anterior pero ahora usadas para calcular $\vec{V}_{pn}$).

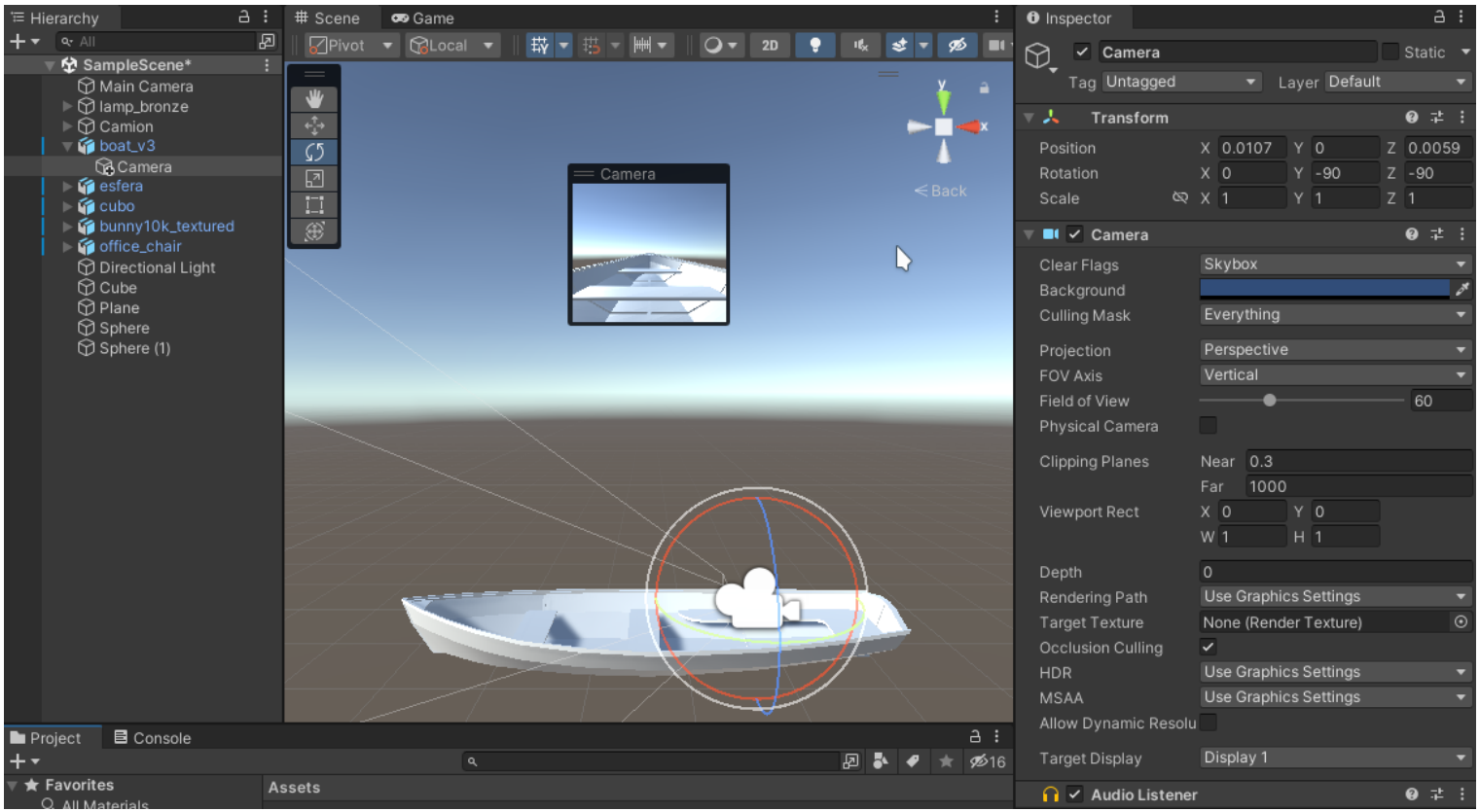
$$\vec{V}_{pn} = (sin(\phi)\ sin(\theta), cos(\phi), sin(\phi)\ cos(\theta))$$

V_{up} se debe calcular de forma que no sea perpendicular al plano proyección.

Además es conveniente hacer que la orientación de la cámara se ajuste a la dirección de movimiento del personaje. Para ello debemos componer dos giros de la cámara: alinear con la dirección de movimiento del personaje y el indicado por el usuario.

Cámara primera persona en Unity

Para que la cámara siga a un componente de la escena basta con incluirla en el grafo de escena dependiedo de ese componente.



Cámara tercera persona (seguimiento)

La cámara sigue la posición de un avatar que se mueve en el escenario a un distancia fija de él. El usuario puede controlar la dirección en la que mira la cámara.

Ahora la posición de la cámara ($\vec{P}_c$) estará detrás del personaje ($\vec{P}_p$) en la dirección $-\vec{D}_p$, siendo $\vec{D}_p$ su dirección de movimiento

$$\vec{P}_c = \vec{P}_p - d\ \vec{D}_p$$

d indica la distancia de la cámara al personaje.