

# Gràfics i Visualització de Dades

## Tema 4: Càmera

Anna Puig

**NOTA:** Aquestes transparències es corresponen a les explicacions del tema 1.5 i 4 del llibre de referència bàsica

[Angel2011] Edward Angel, Dave Shreiner, **Interactive Computer Graphics: A Top-Down Approach with Shader-Based OpenGL, 6/E**, ISBN-10: 0132545233. ISBN-13: 9780132545235, Addison-Wesley, 2011

# Tema 4: Càmera

## 4.1. Atributs de la càmera

## 4.2. Pipeline de visualització:

4.2.1. Matriu ModelView

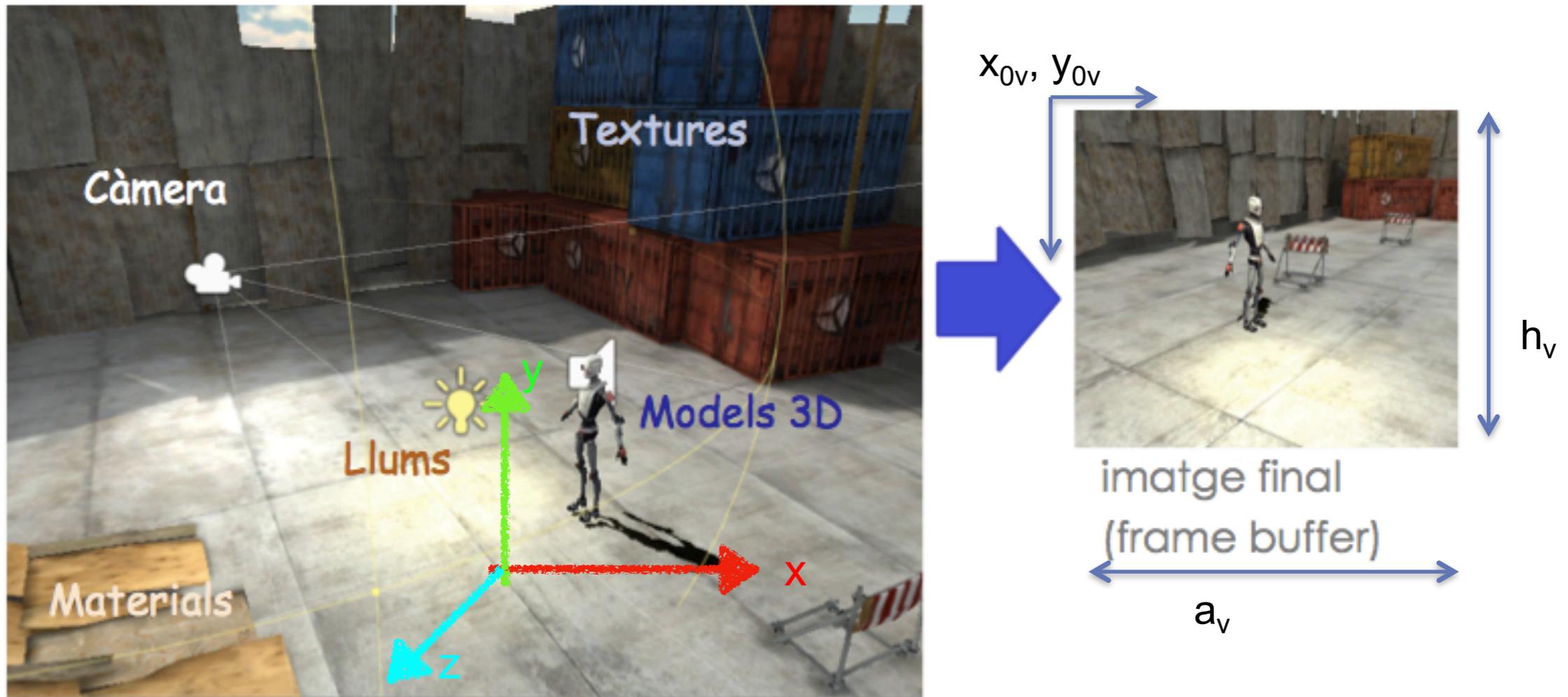
4.2.2. Matriu Projection

4.2.3. Transformació Window-Viewport

# 4.1. Càmera: atributs

Com es troba la visualització final dels objectes definits?

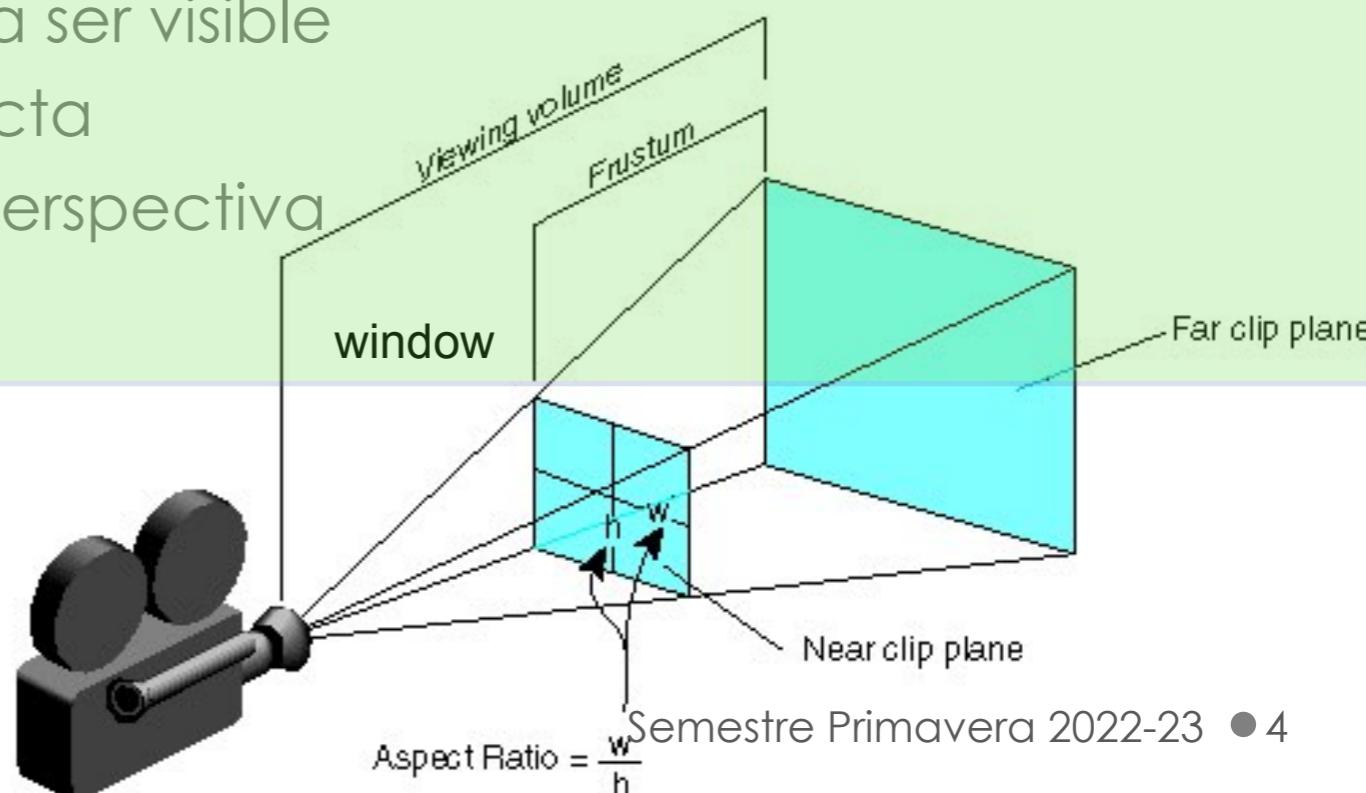
Cal definir la **càmera** on es situa l'**observador**, el **tres** de l'escena que veu i la **projecció** i amb aquesta informació transformar els punts de món a píxels (passant-los pel pipeline de visualització)



# 4.1. Càmera: atributs

- Components de la càmera:

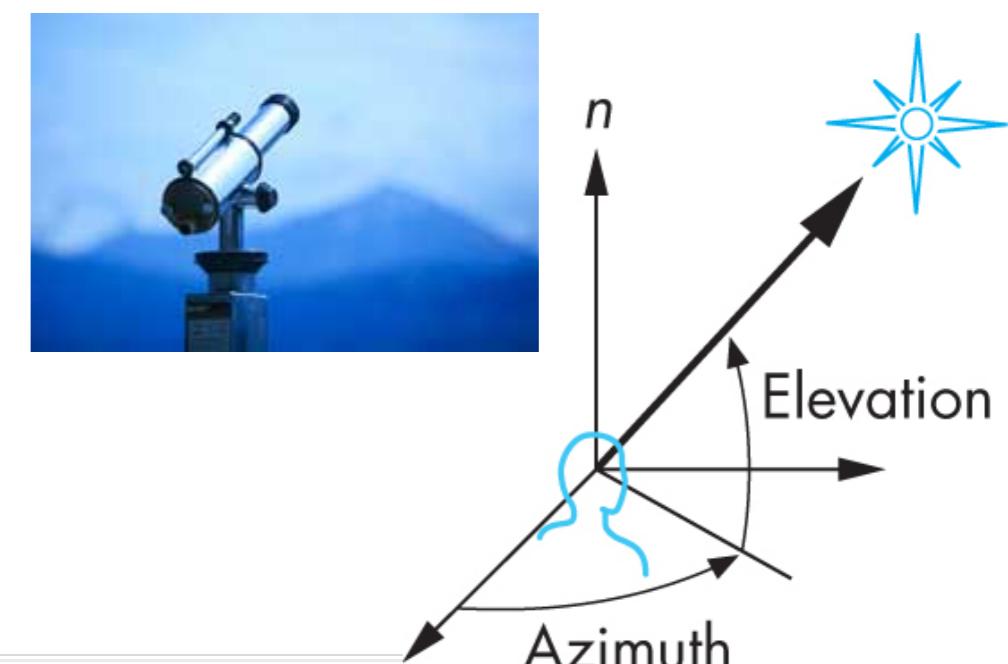
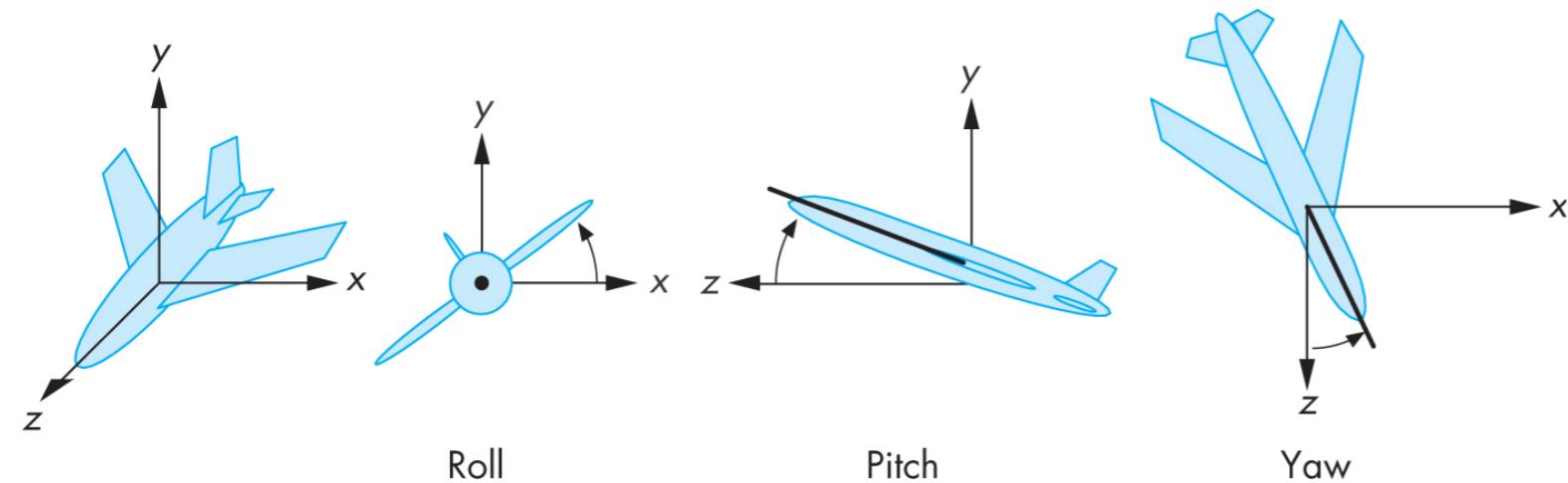
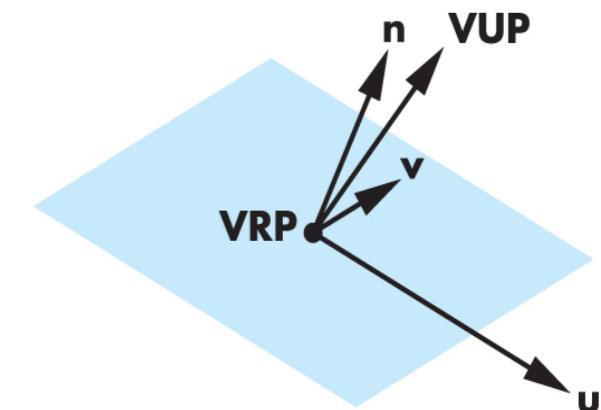
- **Posició de l'observador**: punt 3D en coordenades de món on es situa l'observador o el *viewpoint*. (*lookFrom*)
  - Pot ser donada per tres angles i una distància
- **VRP**: View Reference Point: punt 3D en coordenades de món on mira l'observador (*lookAt*)
- **VUP**: vector de verticalitat: vector 3D que defineix la direcció vertical de l'observador
- **D**: distància de l'observador al VRP
- **Frustum**: volum de l'escena visible realment (**window**,  $z_{near}$ ,  $z_{far}$ )
- **Volum de visió**: volum candidat a ser visible
- **Pla de projecció**: pla on es projecta
- **Tipus de projecció**: paral·lela o perspectiva
- **Viewport**: imatge de píxels



# 4.1. Càmera: atributs

**Posició i orientació:** Existeixen moltes formes per definir la posició i orientació de la càmera:

- VRP, posició observador i el vector vup (PHIGS, GKS-3D)
- usant yaw, pitch, roll
- elevació, azimuth, twist
- angles de visió, posició de l'observador i VRP



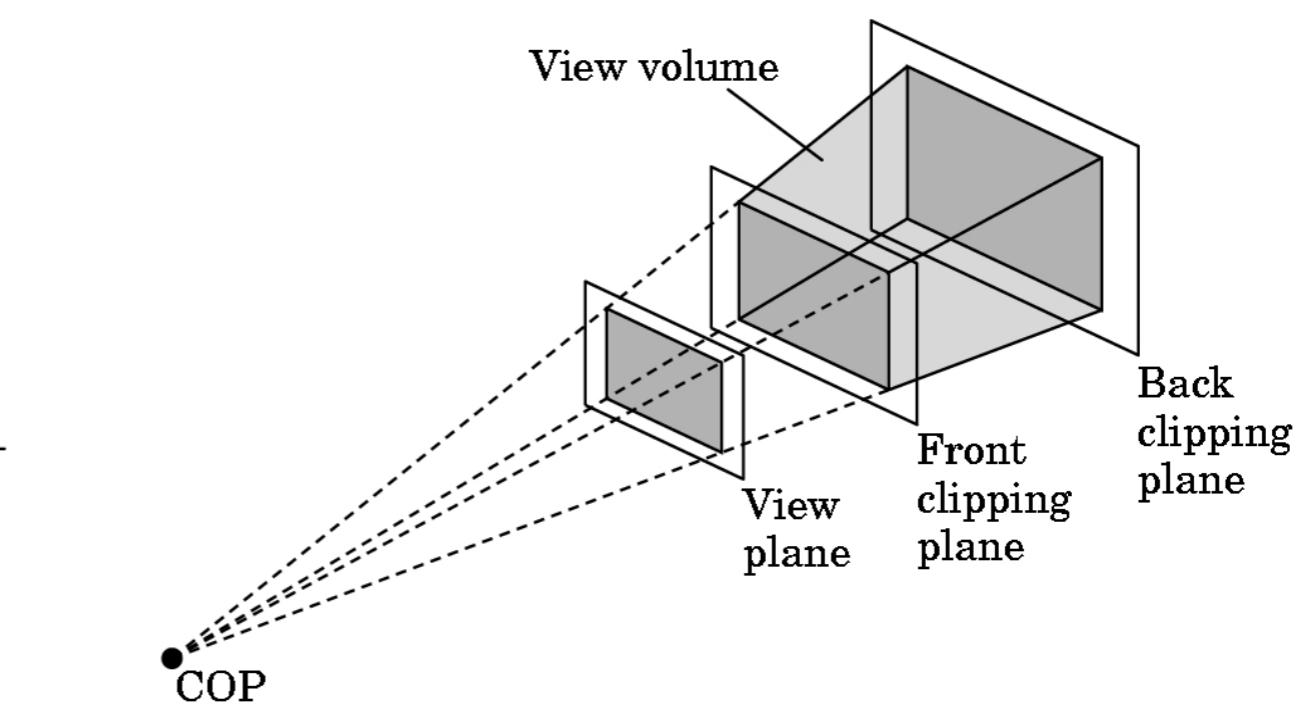
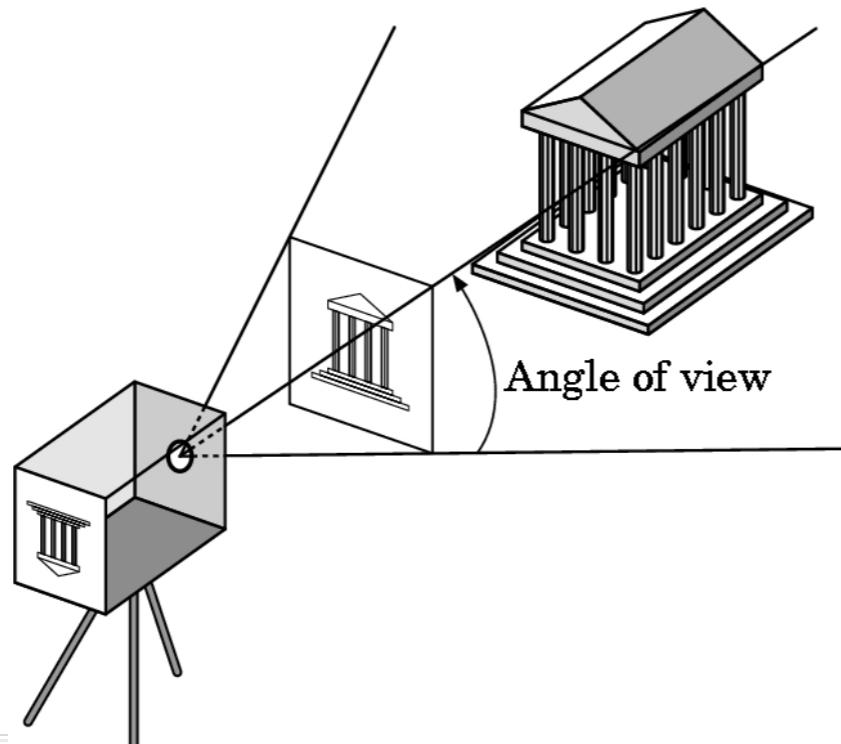
## 4.1. Càmera: atributs

**Clipping** (o retallat): La càmera només pot veure una part del món o de l'escena:

Els objectes que no estan dins d'aquest volum (o *frustum*) es diu que son retallats (*clipped out*) de l'escena.

Plans de clipping lateral i antero-posteriors ( $Z_{near}$  i  $Z_{far}$ ).

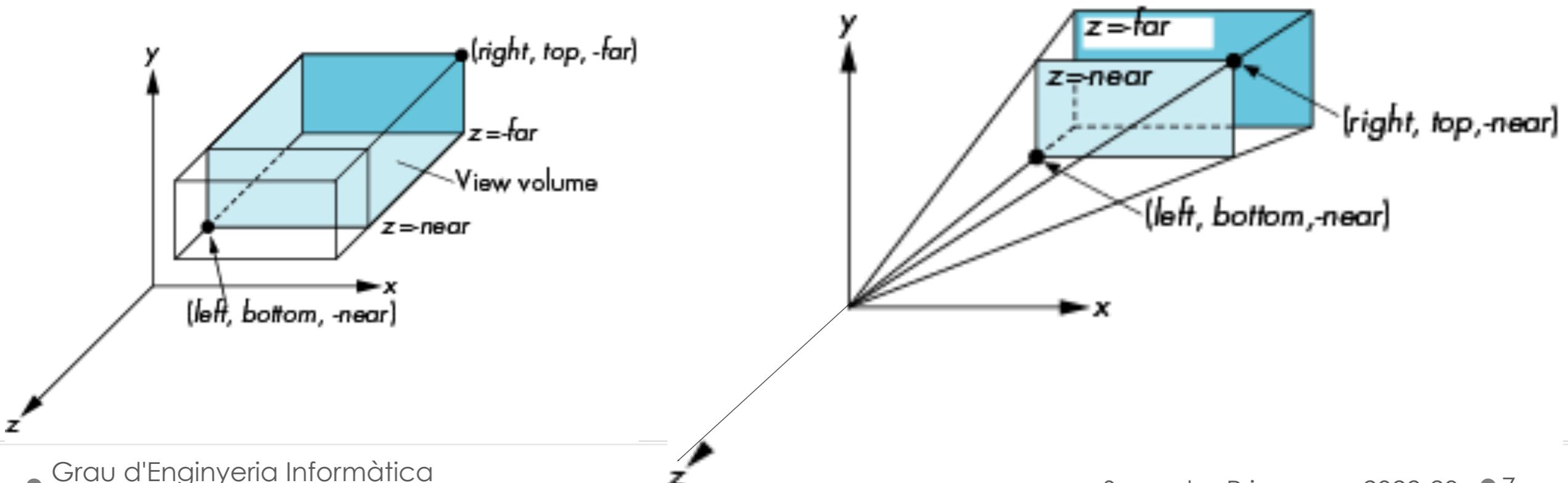
**Pla de projecció** (view plane): és el pla on es projecta l'escena (**window**) donant el que es veurà en un troç o tot el *frame buffer* final (o **viewport**)



## 4.1. Càmera: atributs

Els **plans de retallat** o de *clipping* defineixen el volum visible de l'escena a visualitzar:

- el volum visible en projeccions **paral·leles** és un paral·lelepípede
- el volum visible en projeccions **perspectives** és una piràmide truncada.



## 4.1. Càmera: atributs

Es defineixen dos plans addicionals paral·lels al pla de projecció:

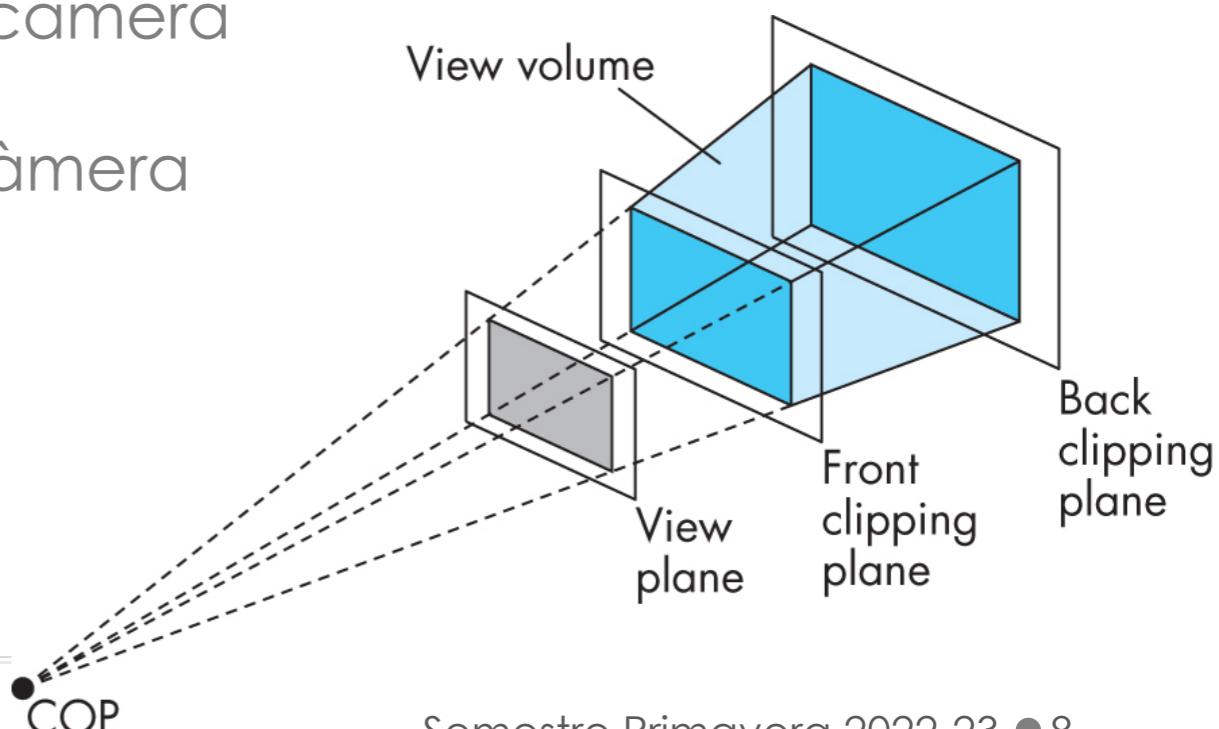
- **el pla de retallat anterior ( $z_{near}$ )**
- **el pla de retallat posterior ( $z_{far}$ )**

$z_{near}$  i  $z_{far}$  es mesuren com la distància (amb signe) dels plans al pla de la càmera.

Exemple:

$z_{near} = 1$  pla anterior està davant de la càmera

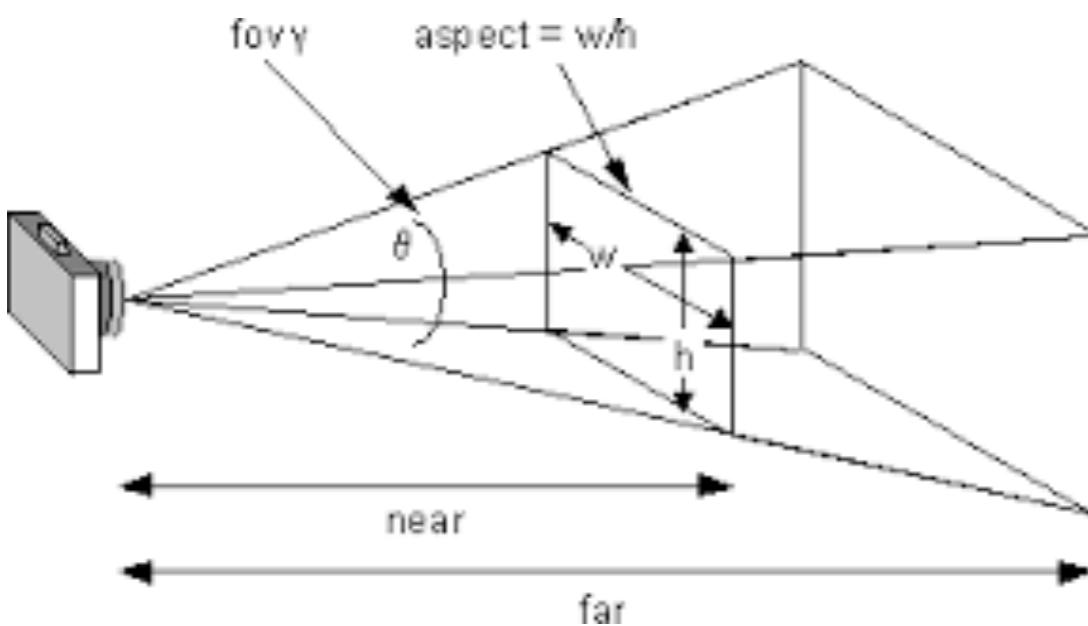
$z_{far} = 2$  pla posterior està davant de la càmera



# 4.1. Càmera: atributs

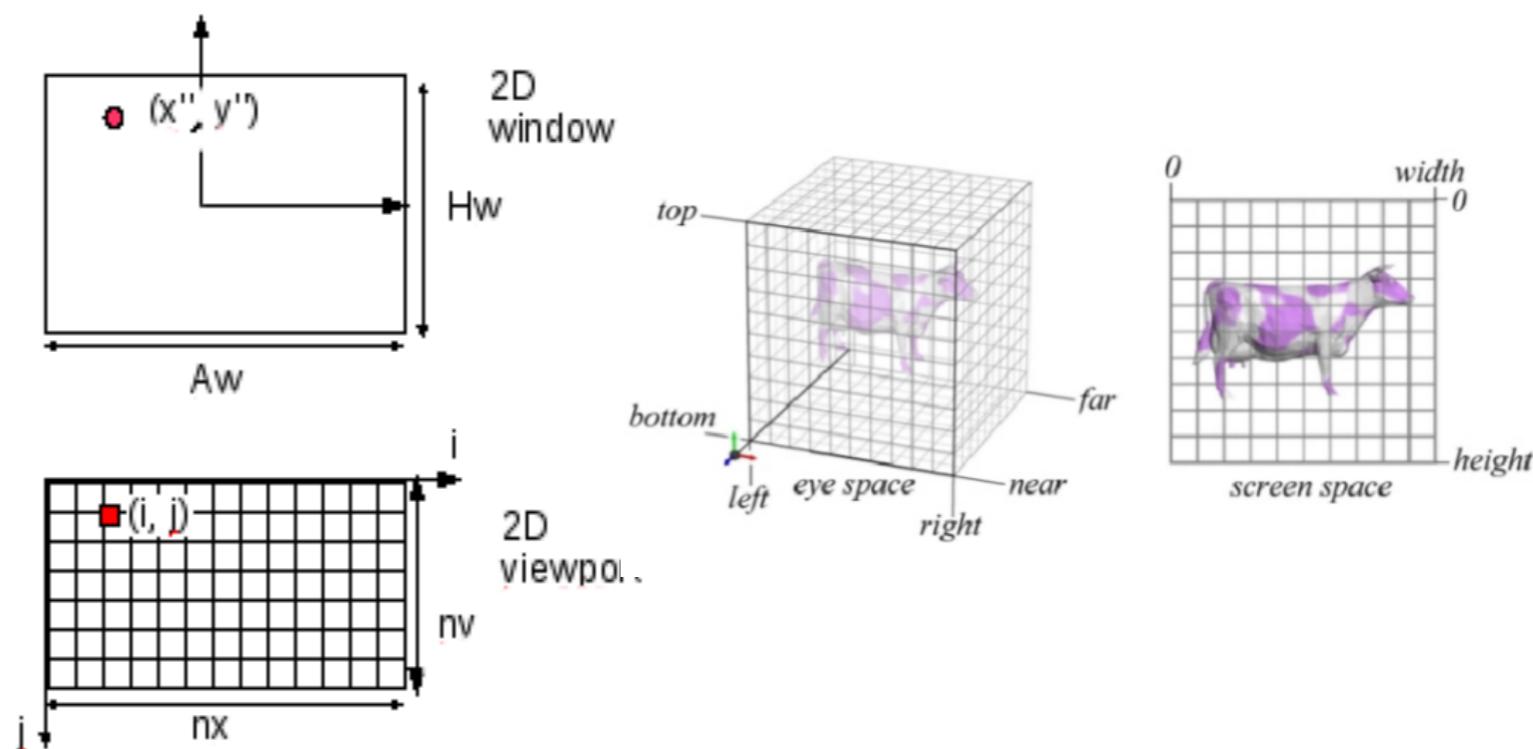
Com es defineixen els límits laterals del volum de visió?

**window**: àrea 2D en el pla de projecció de la zona visible centrada en el 0,0.



Es pot definir com:

- Una **capsa 2D**
- Un camp de visió (amb dos angles d'obertura de la càmera – en x i en y –, o amb un angle d'obertura vertical, **fov**, i l'**aspect ratio** = amplada/alçada)



# Tema 4: Càmera

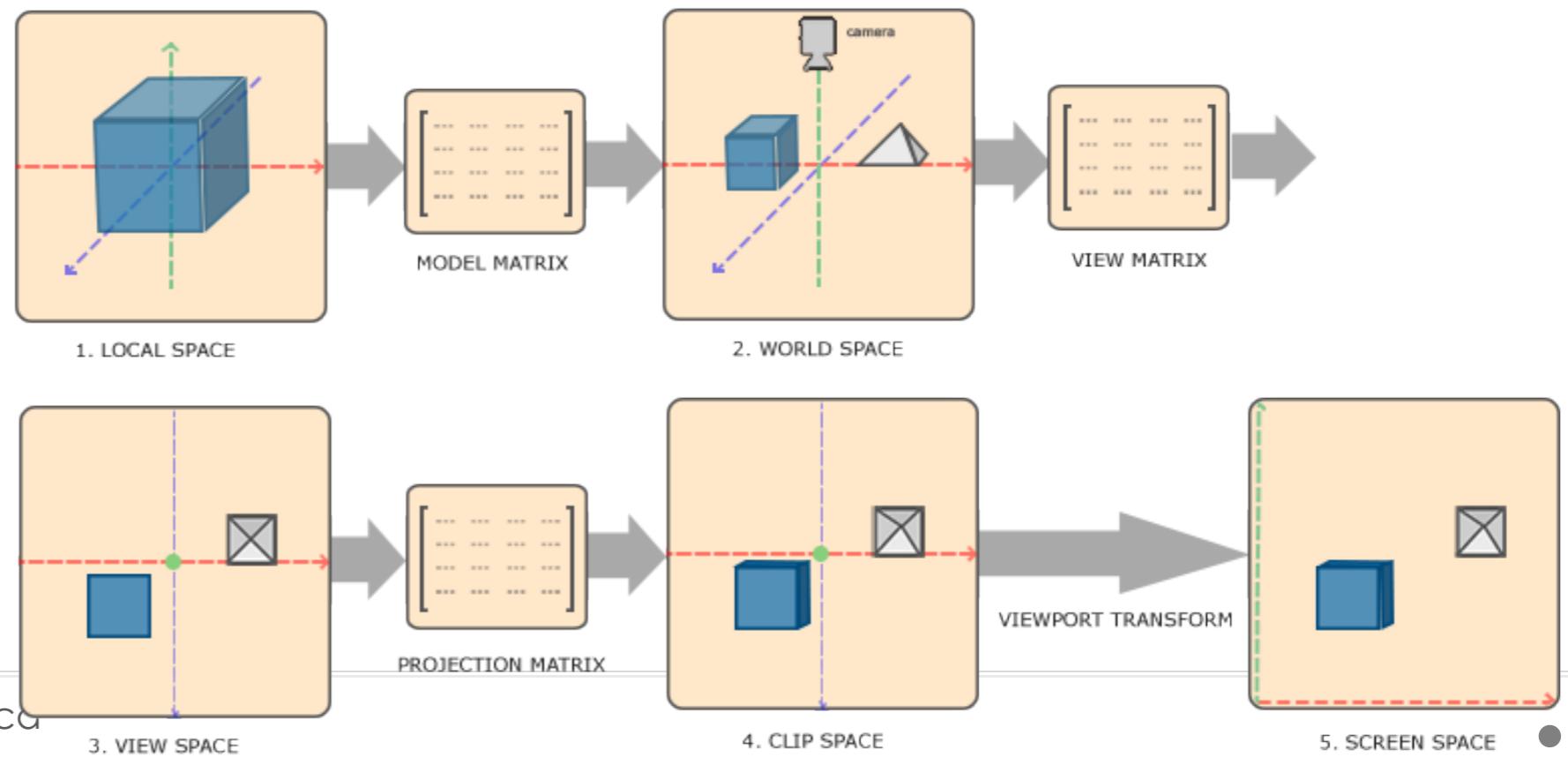
## 4.1. Atributs de la càmera

## 4.2. Pipeline de visualització:

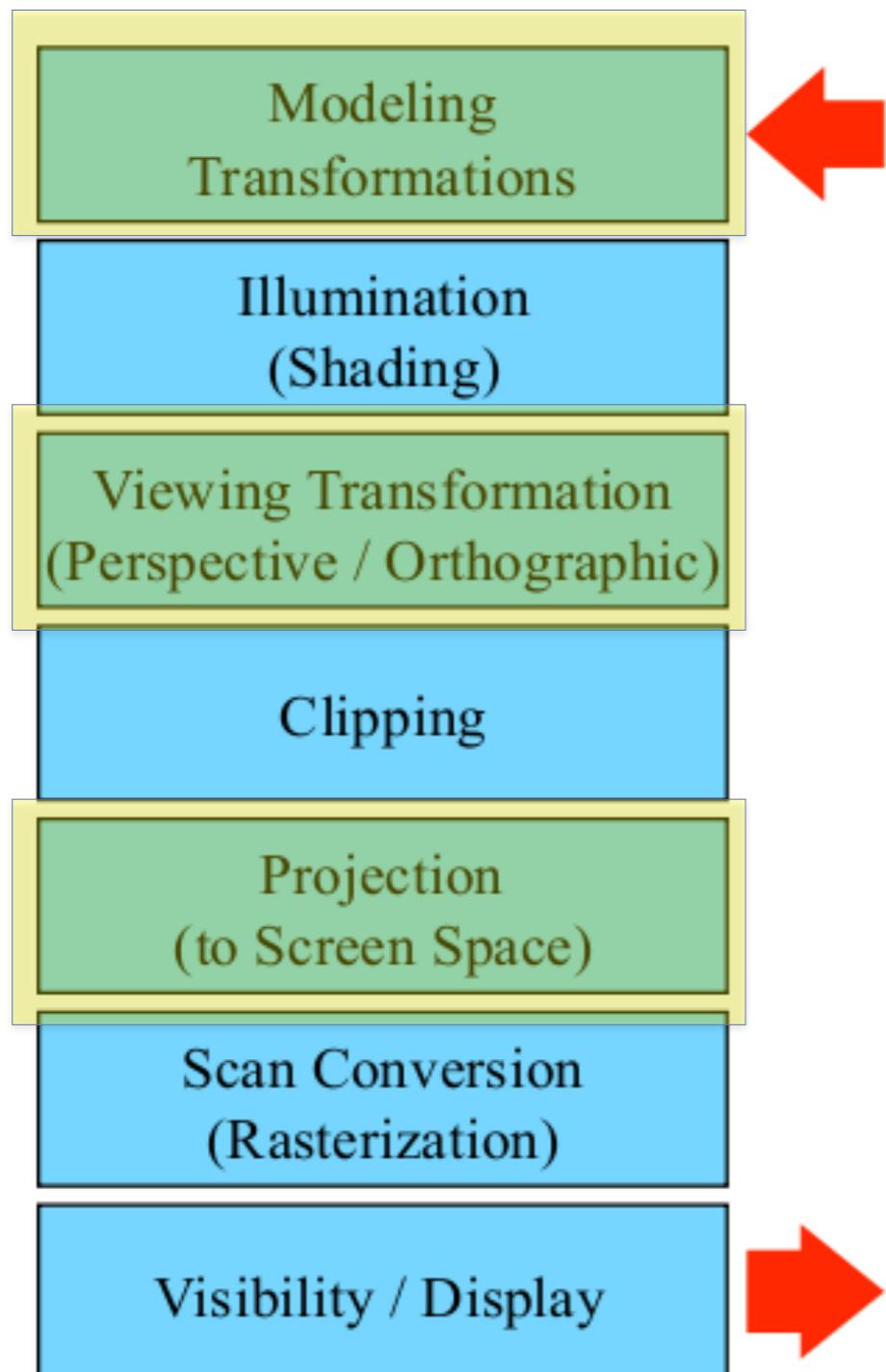
4.2.1. Matriu ModelView

4.2.2. Matriu Projection

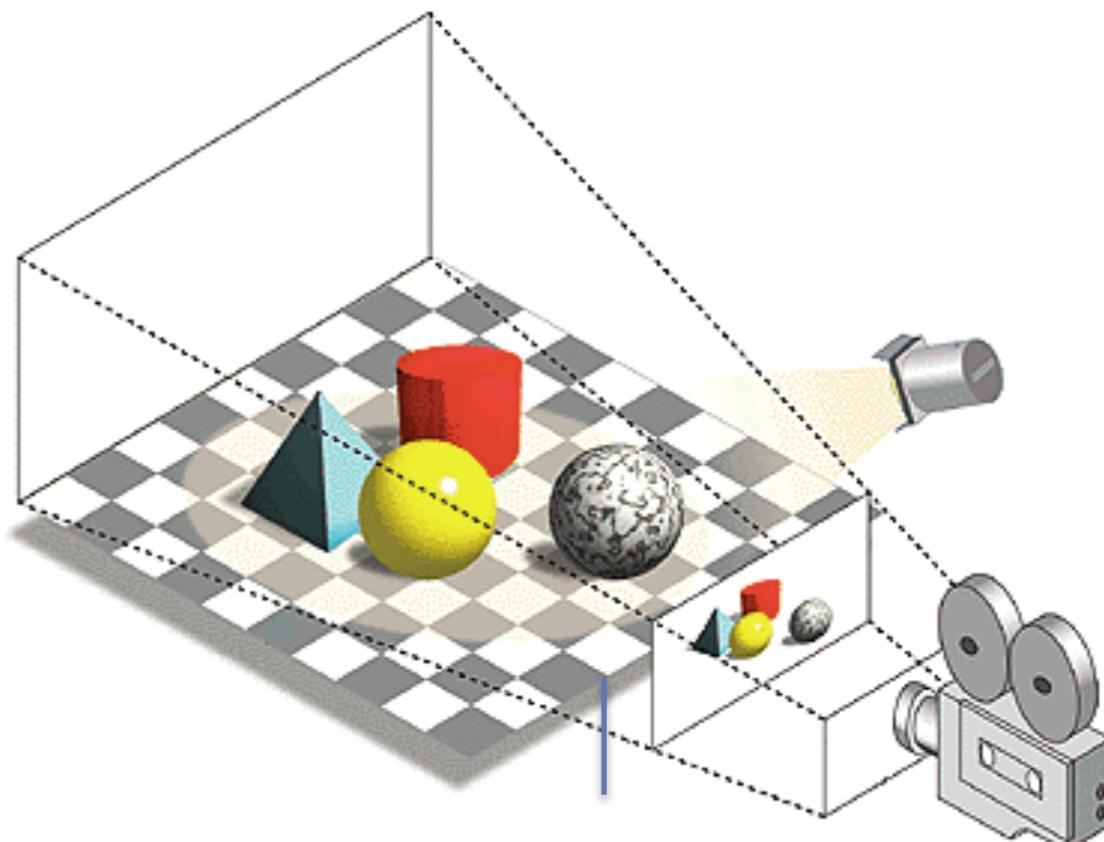
4.2.3. Transformació Window-Viewport



## 4.2. Càmera: pipeline de visualització



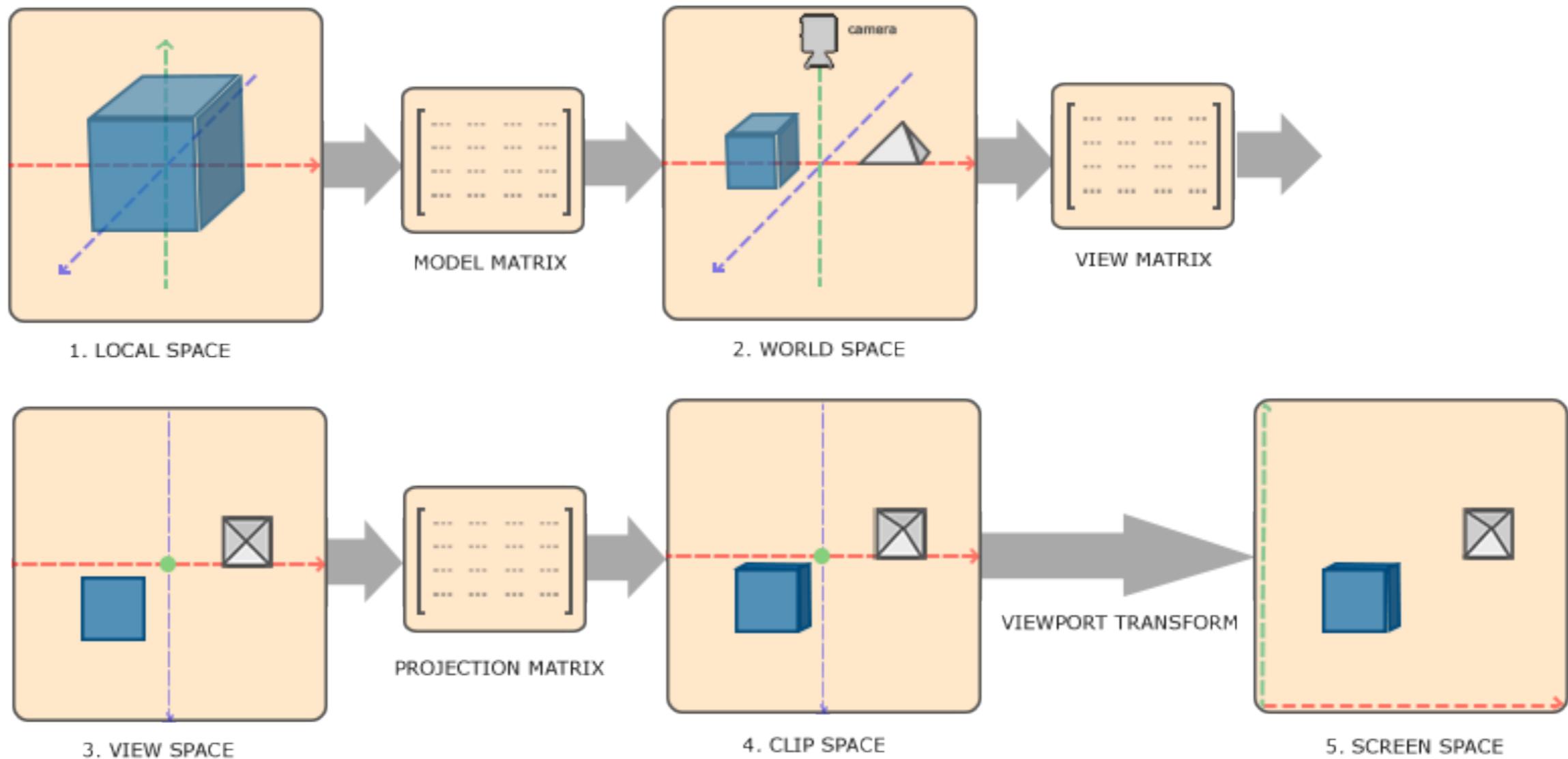
- **Input:** Objectes, Llum, Càmera i viewport



- **Output:** Frame buffer  
(Colors/Intensitats (ex. 24-bit RGB a cada píxel))

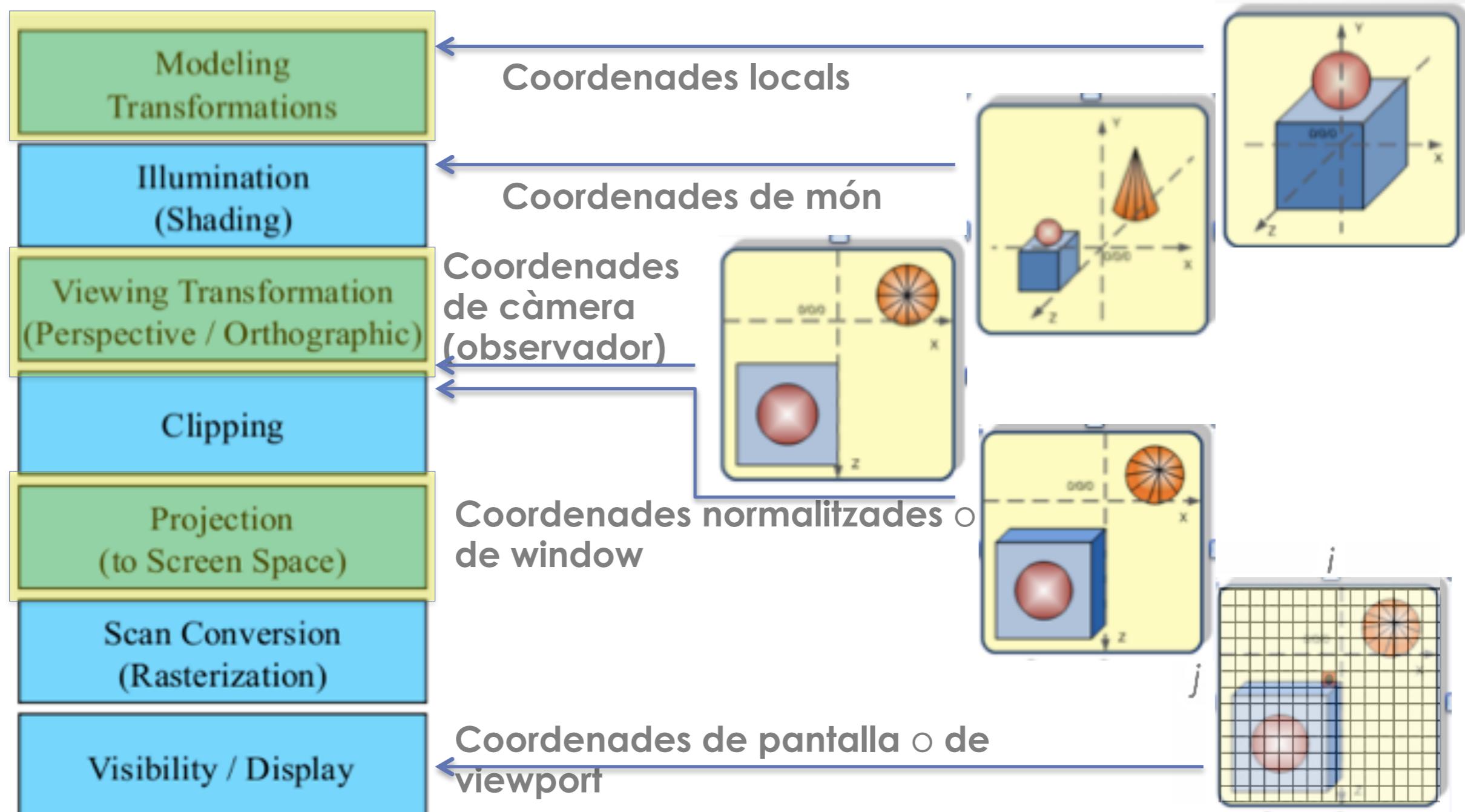
## 4.2. Càmera: pipeline de visualització

Quines transformacions cal fer per passar de món virtual a pantalla?



## 4.2. Càmera: pipeline de visualització

Diferents sistemes de coordenades a diferents etapes:

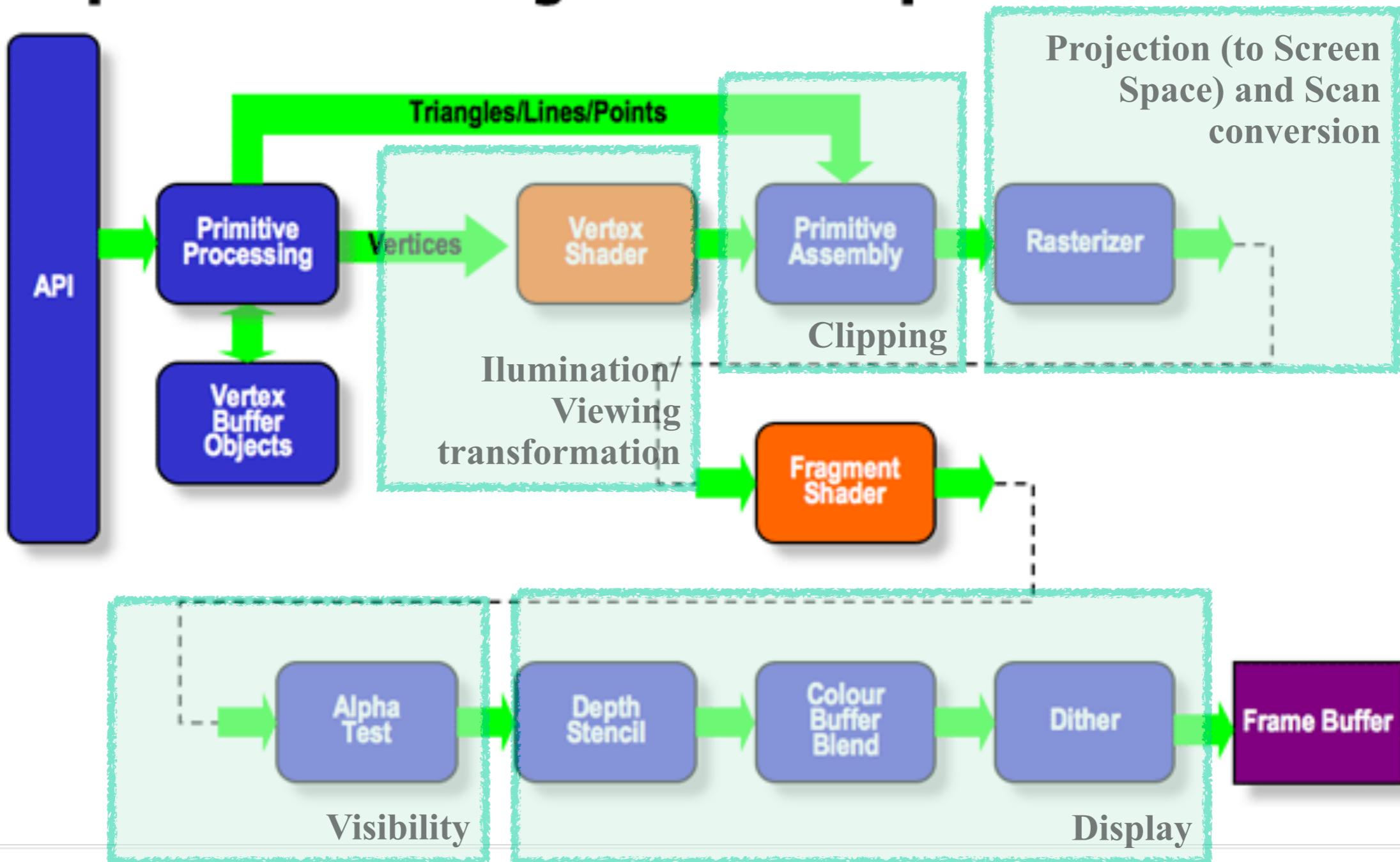


Secció 3.4 del llibre de referència

## 4.2. Càmera: pipeline de visualització

- Com es mapeja el pipeline amb OpenGL?

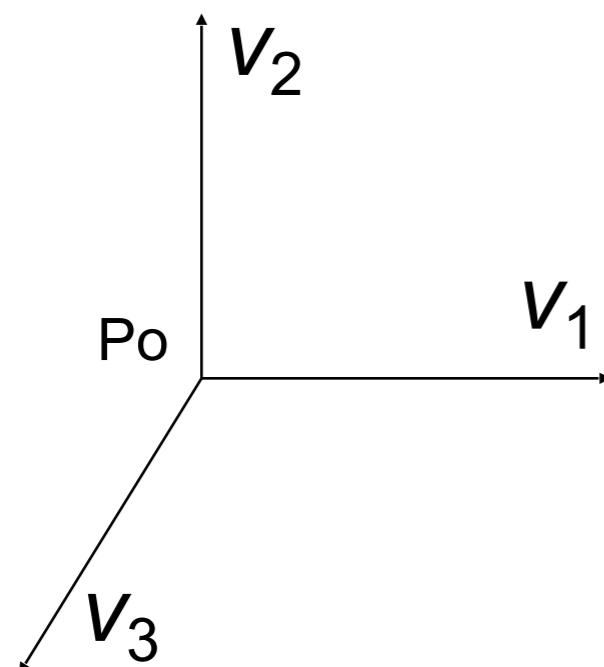
### OpenGL ES 2.0 Programmable Pipeline



## 4.2. Càmera: sistemes de coordenades afins

**Sistema de coordenades afí (frame):** està format pel punt d'origen,  $P_0$ , i els vectors de la base  $(v_1, v_2, v_3)$ .

S'utilitzen bases ortonormals: Els vectors de la base són linealment independents, perpendiculars entre sí i són unitaris



Un **vector** s'escriu com:

$$v = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3$$

Un **punt** s'escriu com:

$$p = p_0 + \beta_1 v_1 + \beta_2 v_2 + \beta_3 v_3$$

**Coordenades homogènies 3D en un sistema afí: punts i vectors**

$$v = a_1 v_1 + a_2 v_2 + a_3 v_3 = [a_1 \ a_2 \ a_3 \ 0] [v_1 \ v_2 \ v_3 \ P_0]^T$$

$$p = p_0 + b_1 v_1 + b_2 v_2 + b_3 v_3 = [b_1 \ b_2 \ b_3 \ 1] [v_1 \ v_2 \ v_3 \ P_0]^T$$

[1] Veure secció 3.3 del llibre [Angel2011]

## 4.2. Càmera: sistemes de coordenades afins

En general, la forma en **coordenades homogènies d'un punt 3D**  $[x \ y \ z]$  és:

$$\mathbf{p} = [x' \ y' \ z' \ w]^T$$

Es pot trobar el punt 3D (per què  $w$  és diferent de 0), dividint cada component per  $w$  (**homogenització**)

$$x = x'/w$$

$$y = y'/w$$

$$z = z'/w$$

## 4.2. Càmera: sistemes de coordenades afins

**Coordenades locals (o d'objectes):** és el sistema de coordenades propi de l'objecte

$(Q_0, q_1, q_2, q_3)$  on       $Q_0$  és el punt origen del sistema local  
 $q_1, q_2, q_3$  són els vectors base

**Coordenades globals (o de món):** és el sistema de coordenades general de l'escena on estan posicionats els objectes

$(P_0, e_1, e_2, e_3)$  on       $P_0 = (0,0,0,1)$ ,  $e_1 = (1,0,0,0)$ ,  $e_2 = (0,1,0,0)$ ,  $e_3 = (0,0,1,0)$

**Coordenades de càmera (o eye coordinates):** és el sistema de coordenades de la càmera o de l'observador

$(P_{\text{obs}}, x_{\text{obs}}, y_{\text{obs}}, z_{\text{obs}})$  on       $P_{\text{obs}} = (P_{\text{obs}_x}, P_{\text{obs}_y}, P_{\text{obs}_z})$  és l'observador  
 $x_{\text{obs}}, y_{\text{obs}}, z_{\text{obs}}$  són els vectors base del sistema de càmera

# Tema 4: Càmera

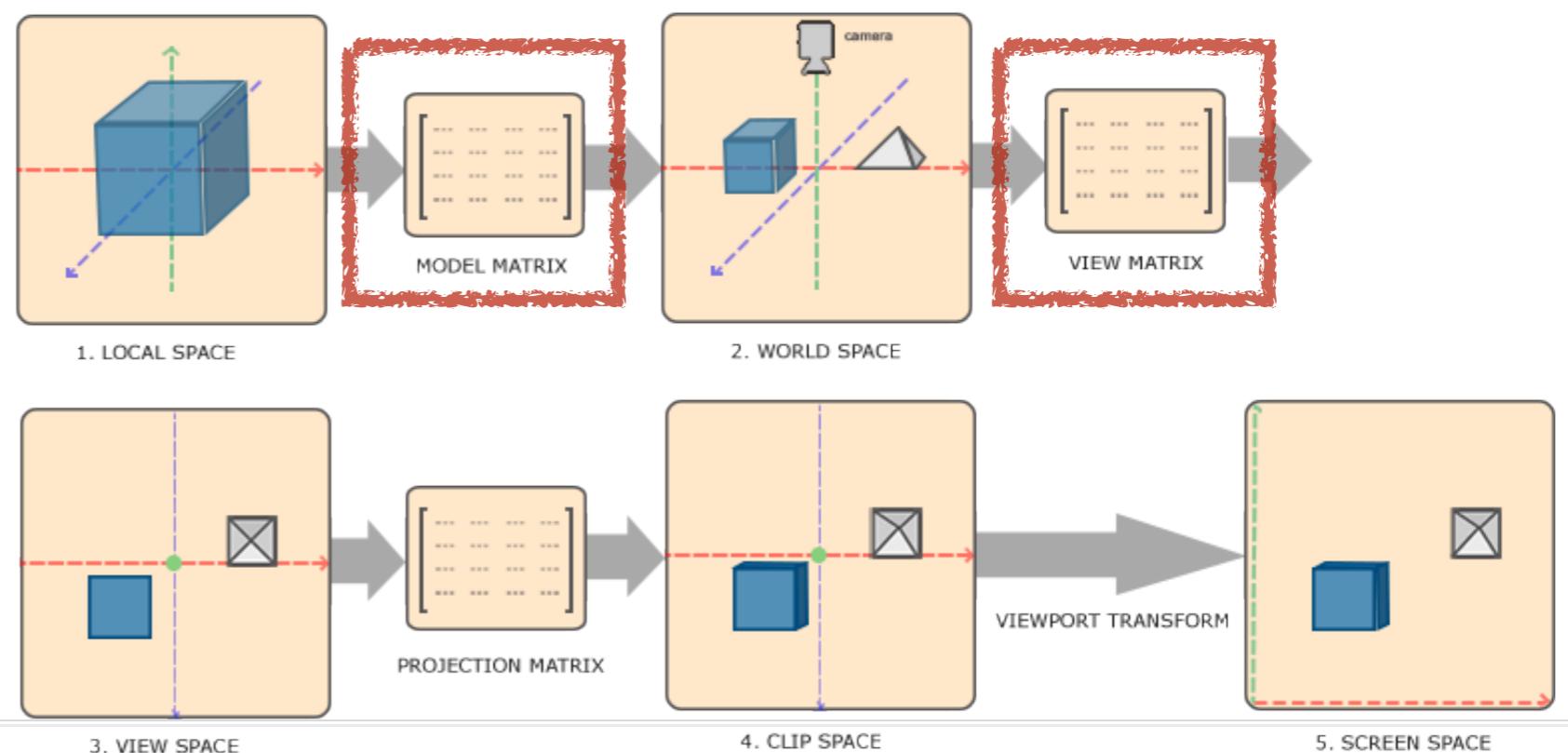
## 4.1. Atributs de la càmera

## 4.2. Pipeline de visualització:

### 4.2.1. Matriu ModelView

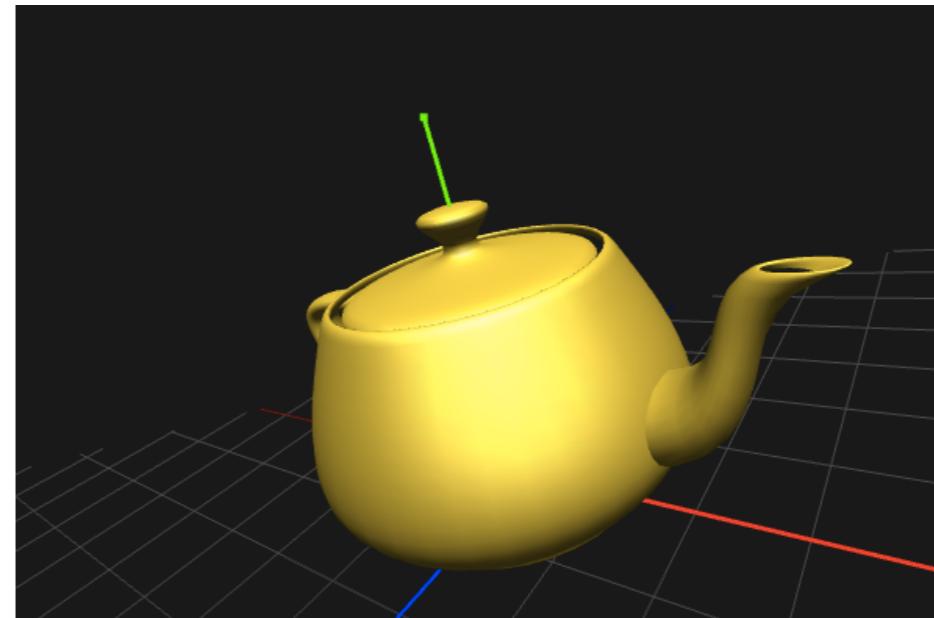
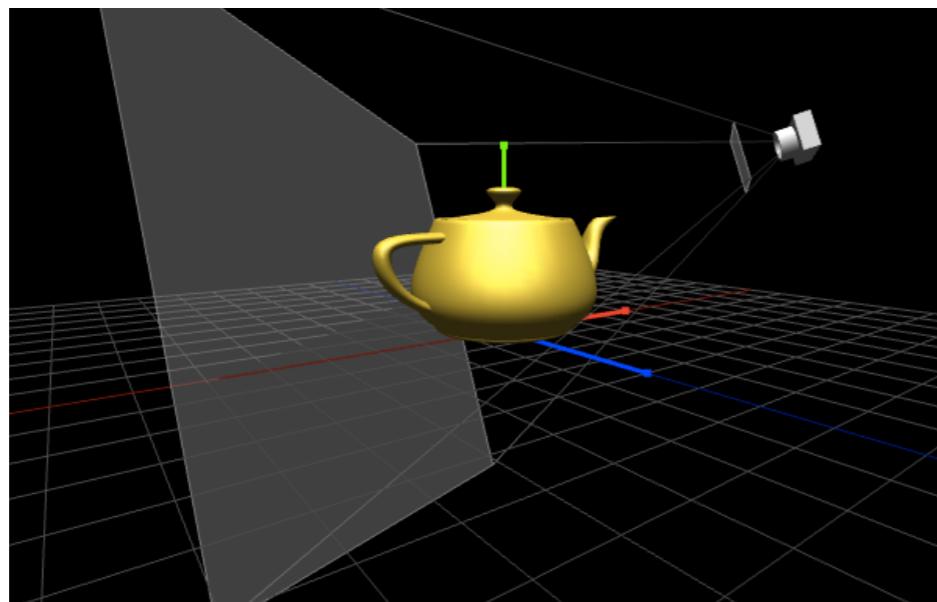
#### 4.2.2. Matriu Projection

#### 4.2.3. Transformació Window-Viewport



## 4.2.1. Matriu model-view

**MODEL-VIEW:** És la matriu que permet passar de coordenades de **món\*** a coordenades de **càmera**



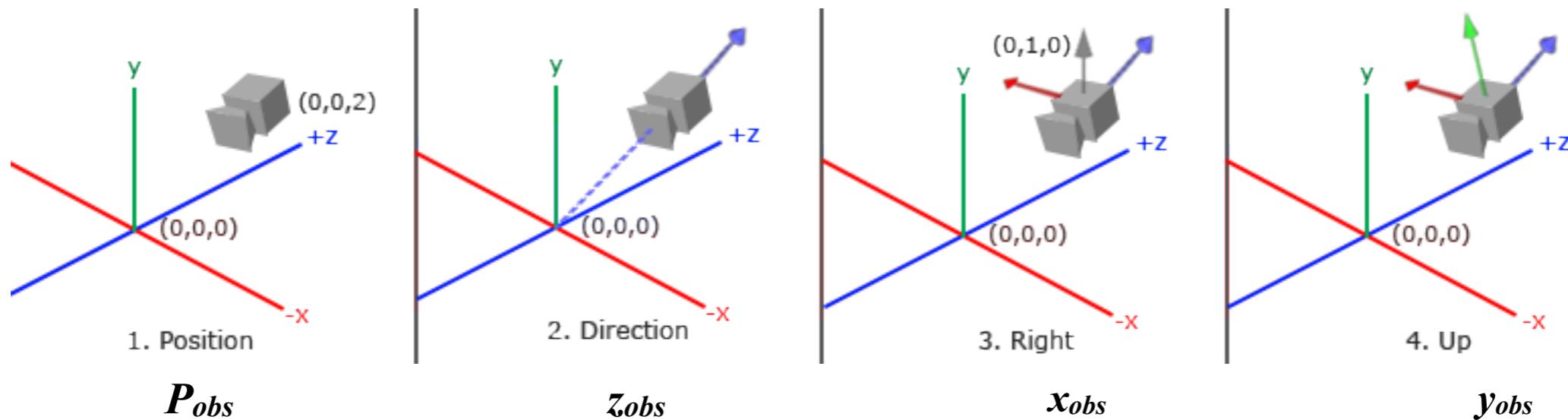
Es pot veure com (els dos mètodes són equivalents)

1. **Un canvi de base** (del sistema de coordenades de món al sistema de coordenades de càmera)
2. A base de **transformacions geomètriques** (1 translació+3 rotacions + 1 translació)

\*en la matriu model-view també s'acumula la transformació de coordenades locals a globals, tot i que a la pràctica generalment només treballem en coordenades globals

## 4.2.1. Matriu model-view

Com calcular el sistema de coordenades de càmera?



$$Look = VRP - P_{obs}$$

$$z_{obs} = \frac{-Look}{\|Look\|}$$

$$x_{obs} = VUP \times z_{obs}$$

$$y_{obs} = z_{obs} \times x_{obs}$$

$$x_{obs} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}, \quad y_{obs} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}, \quad z_{obs} = \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix}$$

## 4.2.1. Matriu model-view

### 1. Canvi de base: MV

$(e_1, e_2, e_3)$  coordenades globals

$P_{obs} = (P_{obs_x}, P_{obs_y}, P_{obs_z})$  coordenades observador

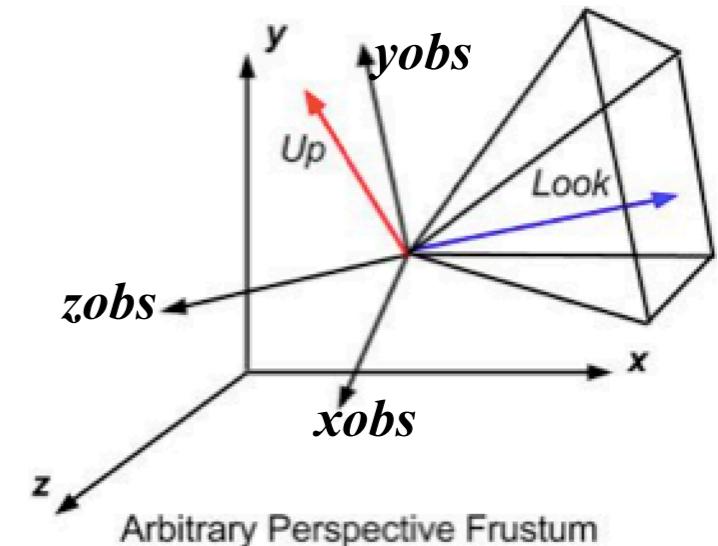
$x_{obs}, y_{obs}, z_{obs}$  vectors base de l'observador

$$x = \mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad y = \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad z = \mathbf{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$



$$x_{obs} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}, \quad y_{obs} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}, \quad z_{obs} = \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix}$$

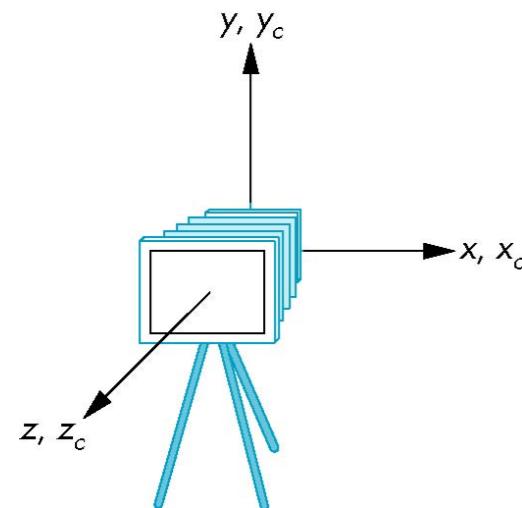
$$MV = \begin{bmatrix} u_x & v_x & w_x & -P_{obs_x} \\ u_y & v_y & w_y & -P_{obs_y} \\ u_z & v_z & w_z & -P_{obs_z} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



## 4.2.1. Matriu model-view

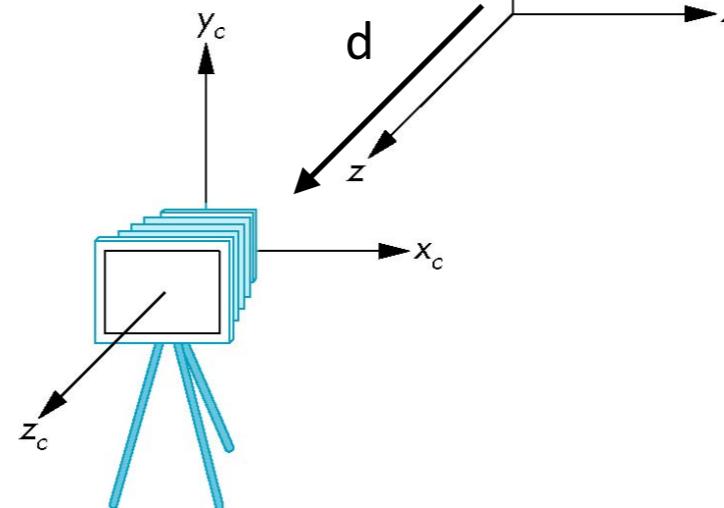
2. Transformacions geomètriques: Càlcul de les matrius de rotació de visió:

Situació de la càmera a l'espai



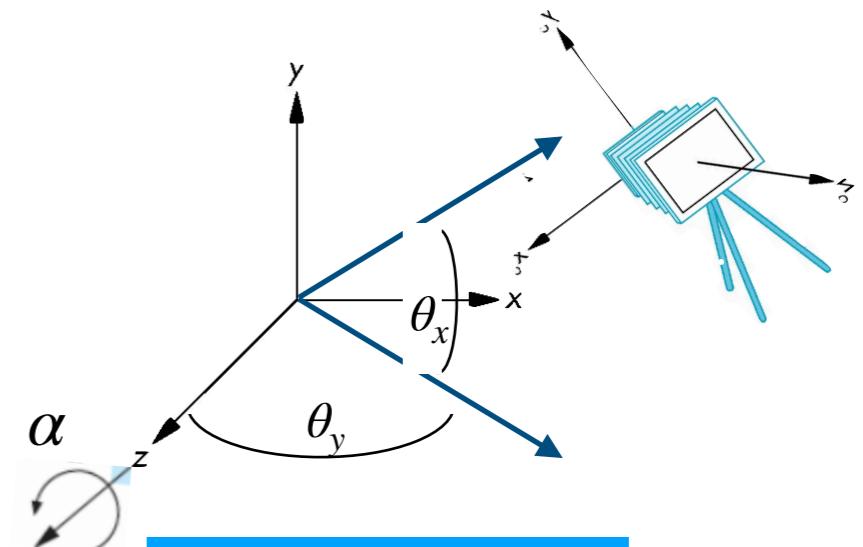
1. Posició de partida

Càmera en el  $(0,0,0)$



2. Translació en  
Z un valor  $d$

$d$  = distància ( $P_{\text{obs}}$  i VRP)



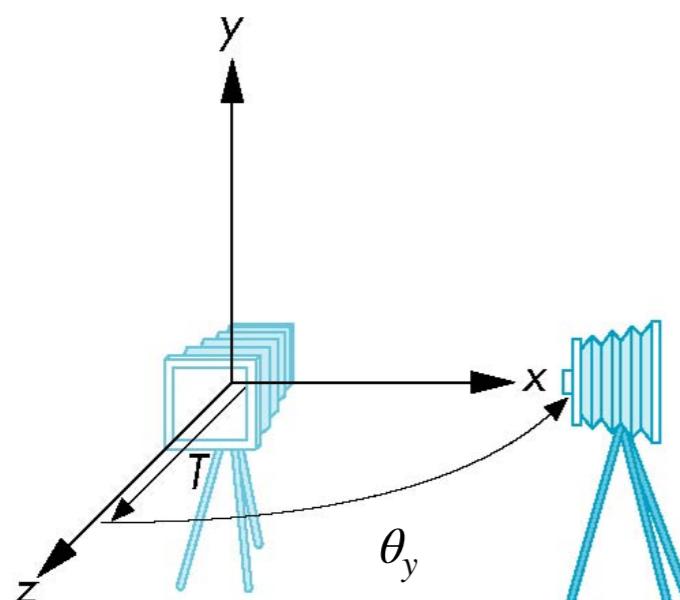
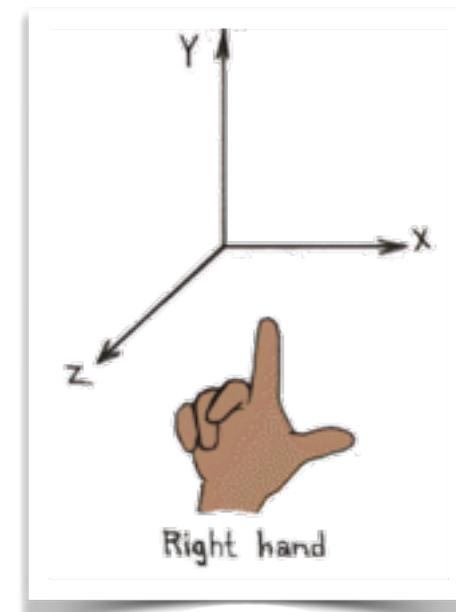
3. Angles d'Euler  
per situar la  
càmera

$\alpha, \theta_y, \theta_x$

## 4.2.1. Matriu model-view

### 2. Transformacions geomètriques: Càlcul de les matrius de rotació de visió:

Exemple: Es vol situar la càmera a l'eix x+ a una distància T



1. Posició de partida

Càmera en el (0,0,0)

2. Translació en Z un valor T

TranslateZ (T)

3. Angles d'Euler per situar la càmera

RotateY ( $\theta_y$ ) amb  $\theta_y=90^\circ$

La matriu **model view** és la transformació inversa ja que s'aplica als objectes.

// Usant les utilitats de mat.h

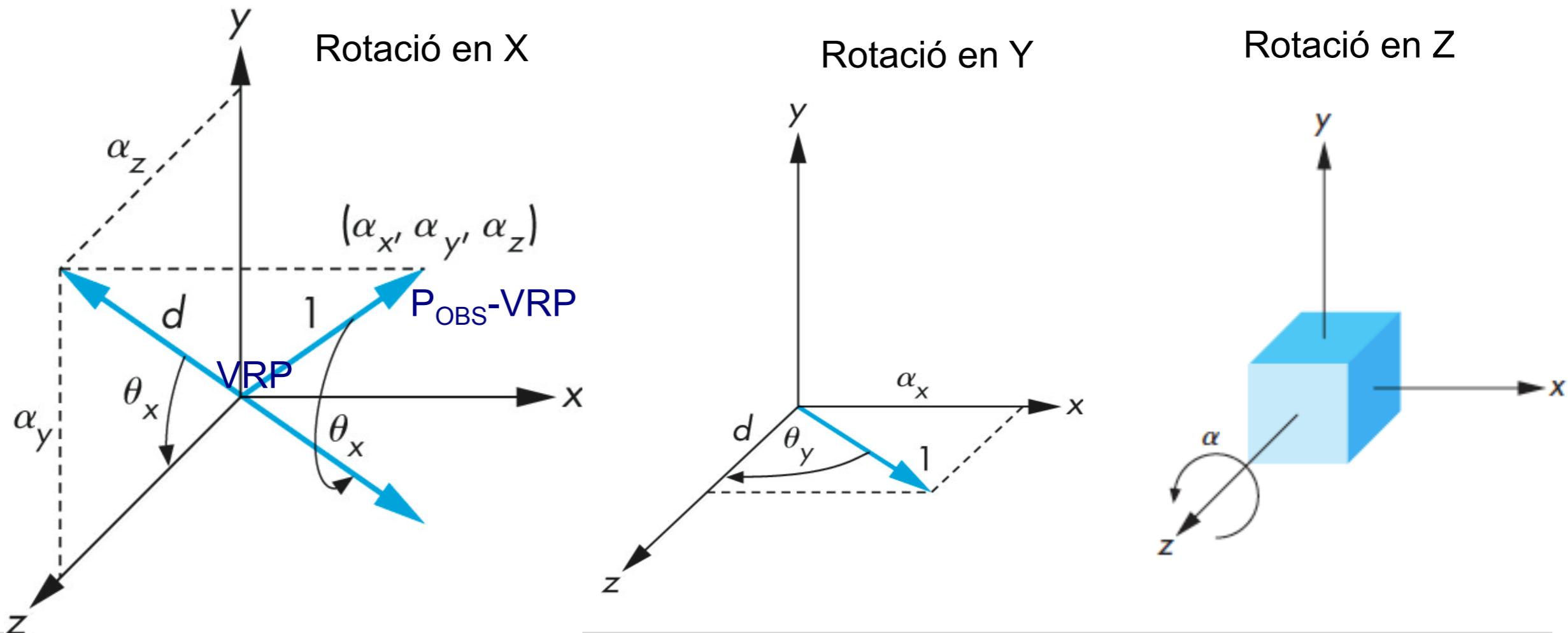
```
mat4 t = Translate (0.0, 0.0, -T);  
mat4 ry = RotateY(-90.0);  
mat4 mv = t*ry;
```

## 4.2.1. Matriu model-view

### 2. Transformacions geomètriques: Càlcul de les matrius de rotació de visió:

A partir del vector de visió ( $P_{\text{obs}}\text{-VRP}$ ) es calculen els angles de rotació per a deixar-lo en l'eix de la Z:

<https://www.youtube.com/watch?v=75o5pmeXUMo>



## 4.2.1. Matriu model-view

### 2. Transformacions geomètriques:

Passos a seguir per calcular la MV:

1. Començar amb la matriu identitat:  $I$
2. Moure el VRP a l'origen:  $T(-vRP)$
3. Rotar segons els tres angles contraris a la definició de la càmera

**Resultat:**  $MV = R(-angz) \cdot R(-angy) \cdot R(-angx) \cdot T(-vRP)$

Si es segueixen les convencions OpenGL cal una translació addicional  $T(-d)$  per colocar l'observador al punt  $(0,0,0)$  de les coordenades de càmera.

Per tant, si  $v_{mon}$  són els vèrtexs en coordenades de món, obtindrem els vèrtexs en coordenades de càmera  $v_{obs}$  fent l'operació:

$$v_{obs} = MV \cdot v_{mon}$$

## 4.2.1. Matriu model-view

Els passos a seguir per a crear la matriu **model-view** i aplicar-los des del vertex shader són:

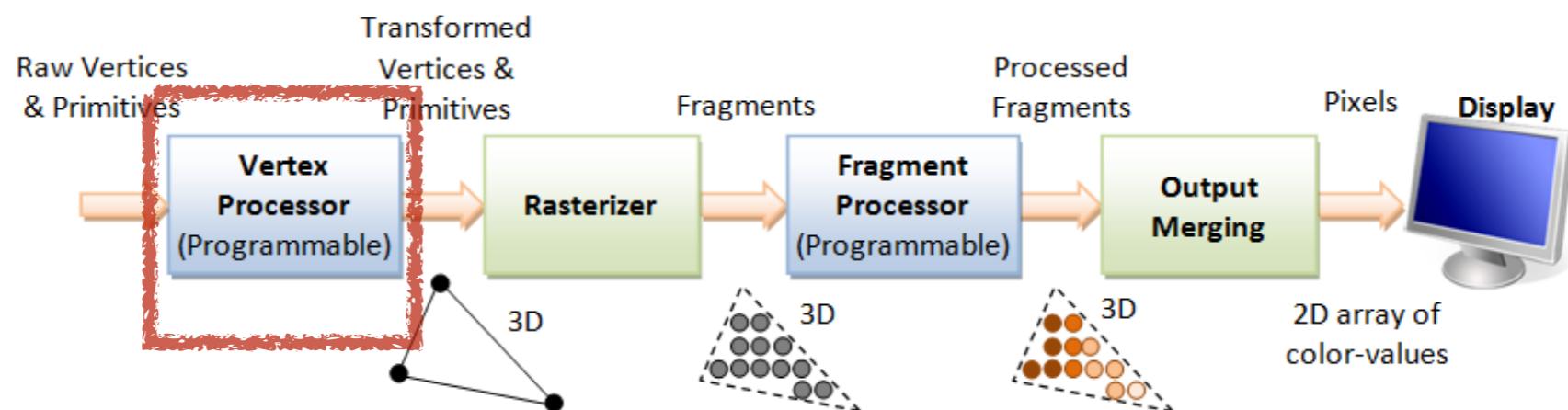
1. Des de l'**aplicació**, es crea la matriu de transformació que correspon a la matriu model-view



2. Es passa al **vertex shader** aquesta matriu com **UNIFORM**



3. En el **vèrtex shader** es multiplica el vèrtex per a la matriu

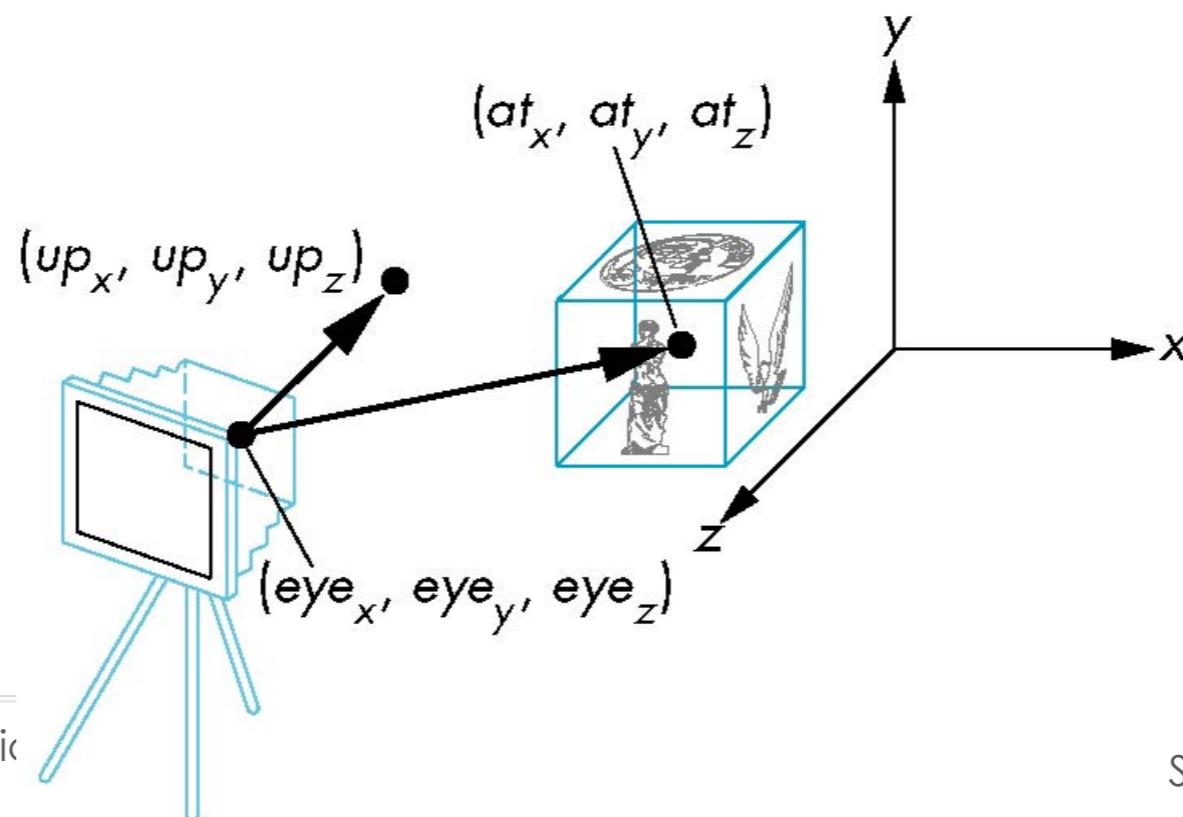


## 4.2.1. Matriu model-view

### Pas 1: Definició de la matriu model view: (Aplicació)

- A la llibreria **glu** (addicional a OpenGL) existeix la funció **gluLookAt** que dóna una interfície fàcil per a definir la matriu model-view.
- S'ha substituït pel mètode **LookAt** (en el fitxer mat.h)

```
mat4 mv = LookAt(vec4 eye, vec4 at, vec4 up);
```



## 4.2.1. Matriu model-view

### Pas 2: Es passa al *vertex shader* aquesta matriu.

Com és una dada que serà constant per tot vèrtex, serà del tipus **uniform**. Des de l'aplicació, un cop calculada la matriu model-view

1. S'obté el lloc de memòria que correspon a la GPU, fent la comanda:

```
GLuint model_view;  
model_view = glGetUniformLocation( program, "model_view" );
```

2. Es fa la correspondència entre les dues variables:

```
glUniformMatrix4fv( model_view, 1, GL_TRUE, mv );
```

## 4.2.1. Matriu model-view

Pas 3: En el *vertex shader* es multiplica el punt per a la matriu.

```
#version 330

layout (location = 0) in vec4 vPosition;

uniform mat4 model_view;

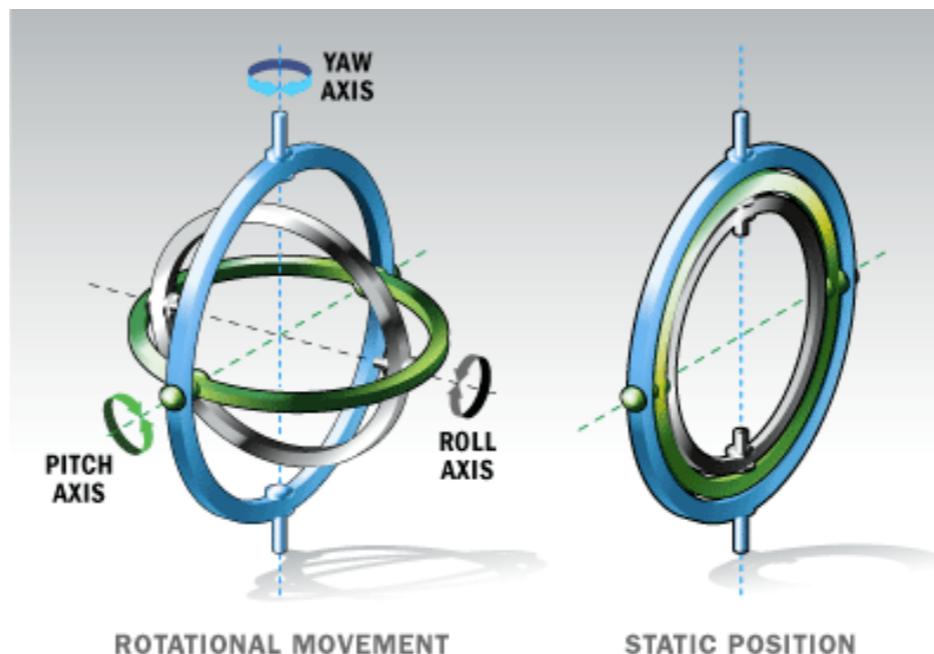
void main()
{
    gl_Position = model_view*vPosition;

}
```

## 4.2.1. Matriu model-view

Problemes dels Angles d'Euler:

Gimball lock: [https://en.wikipedia.org/wiki/Gimbal\\_lock](https://en.wikipedia.org/wiki/Gimbal_lock)



Solució: ús de quaternions

Un quaternió és un número complex (4D)

$$a + bi + cj + dk$$

on  $i, j, k$  són les unitats fonamentals dels quaternions

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$$

<https://www.youtube.com/watch?v=zjMulxRvygQ>

# Tema 4: Càmera

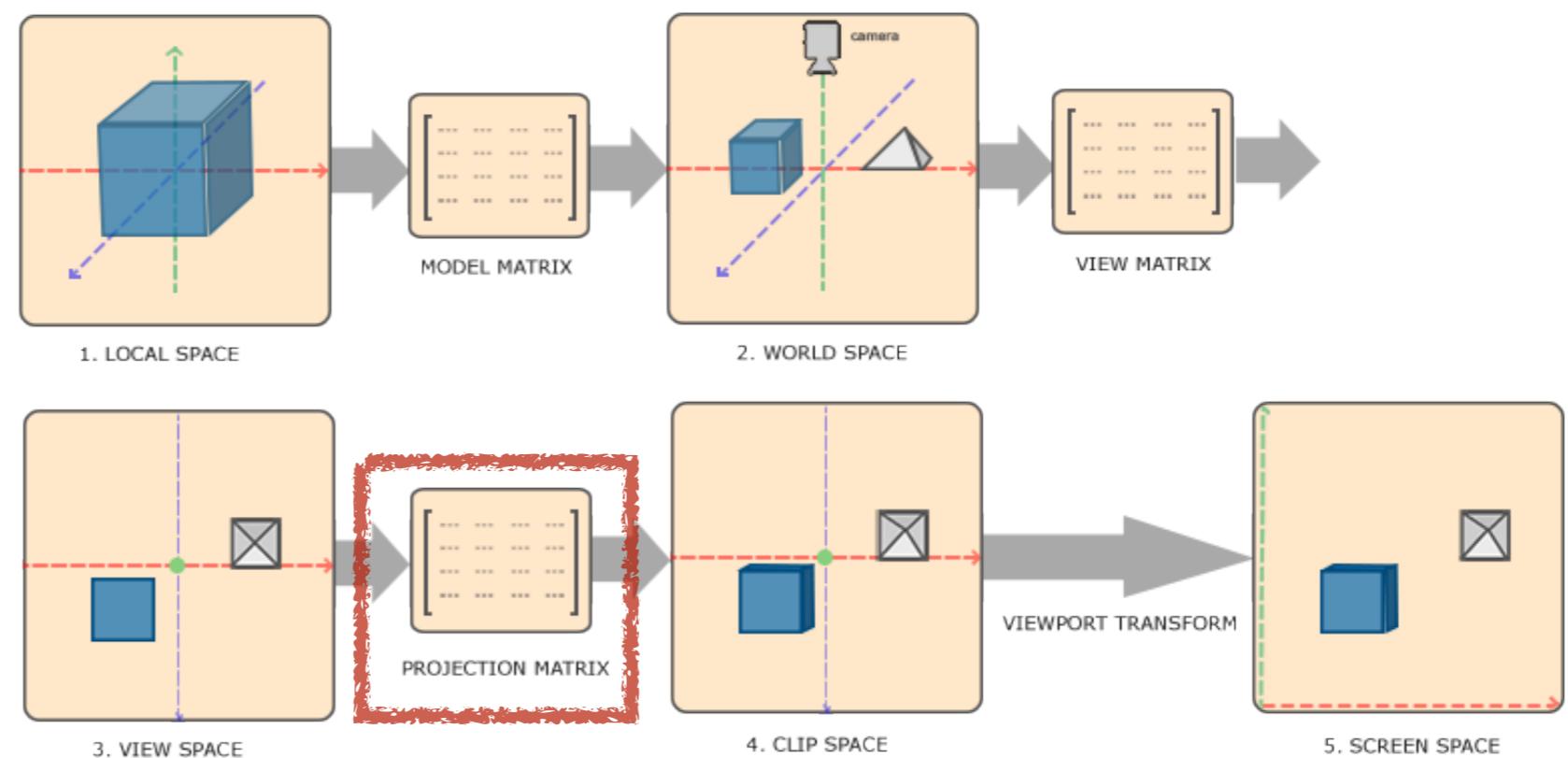
## 4.1. Atributs de la càmera

## 4.2. Pipeline de visualització:

### 4.2.1. Matriu ModelView

### 4.2.2. Matriu Projection

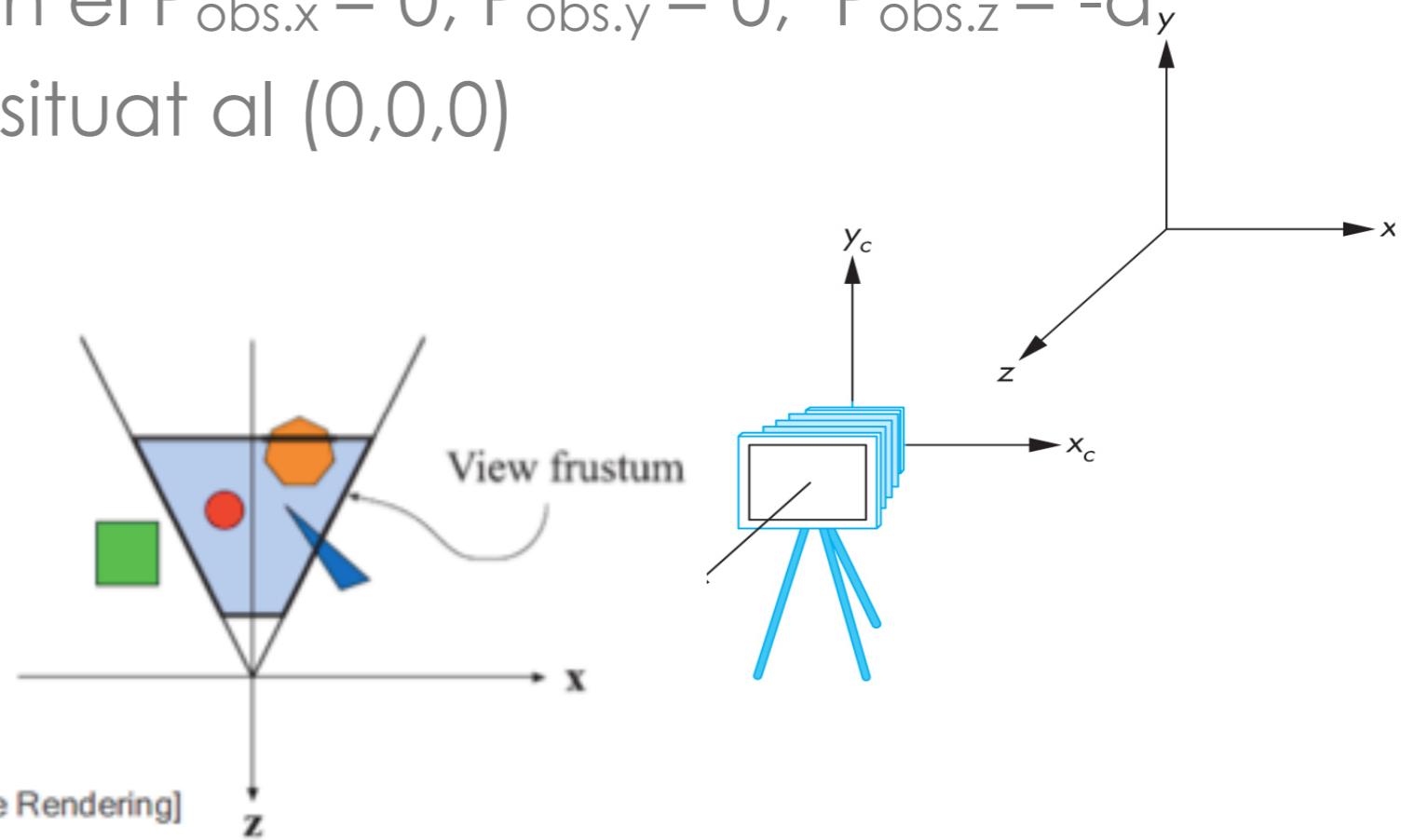
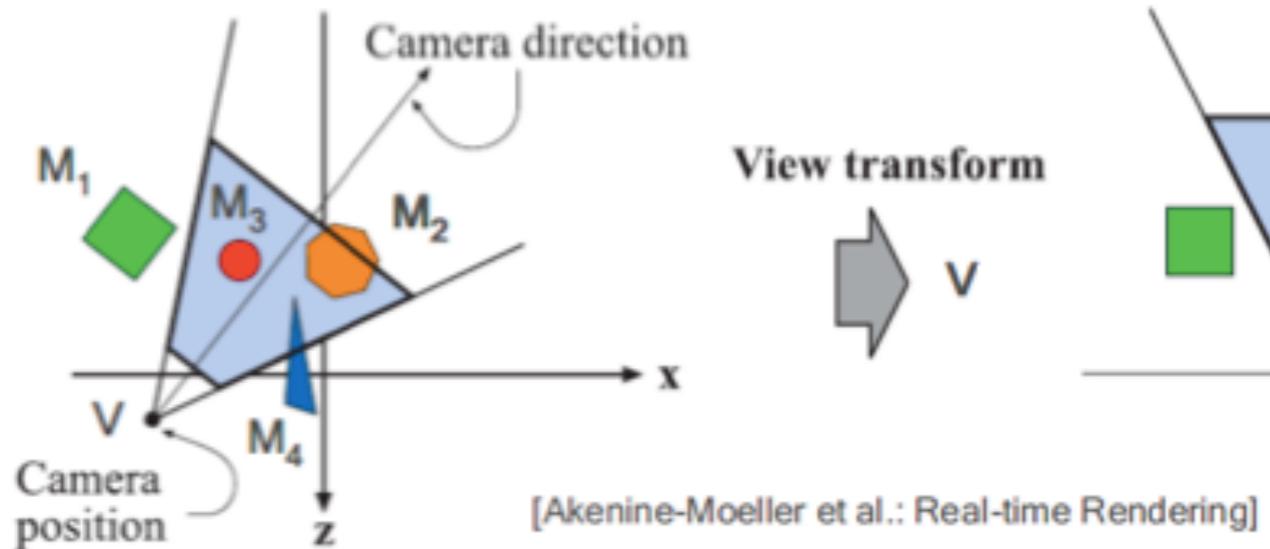
### 4.2.3. Transformació Window-Viewport



## 4.2.2. Matriu Projection

Una vegada s'han definit els paràmetres de la càmera i s'ha realitzat la transformació amb la matriu **model-view**, s'ha de complir que:

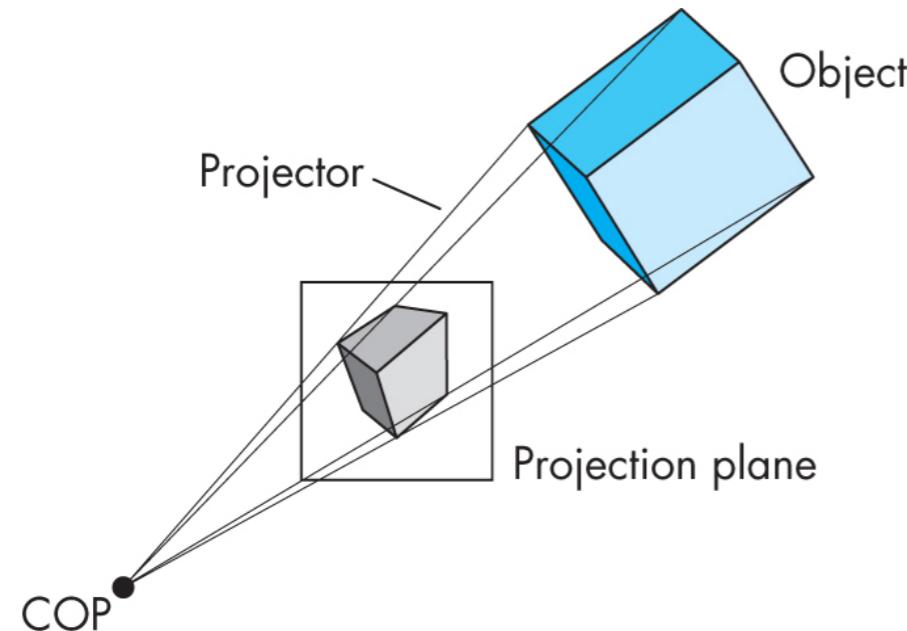
- La càmera mira cap a les  $z_{\text{obs}}$ 's negatives
- El VRP està situat en el  $P_{\text{obs},x} = 0$ ,  $P_{\text{obs},y} = 0$ ,  $P_{\text{obs},z} = -d_y$
- L'observador està situat al  $(0,0,0)$



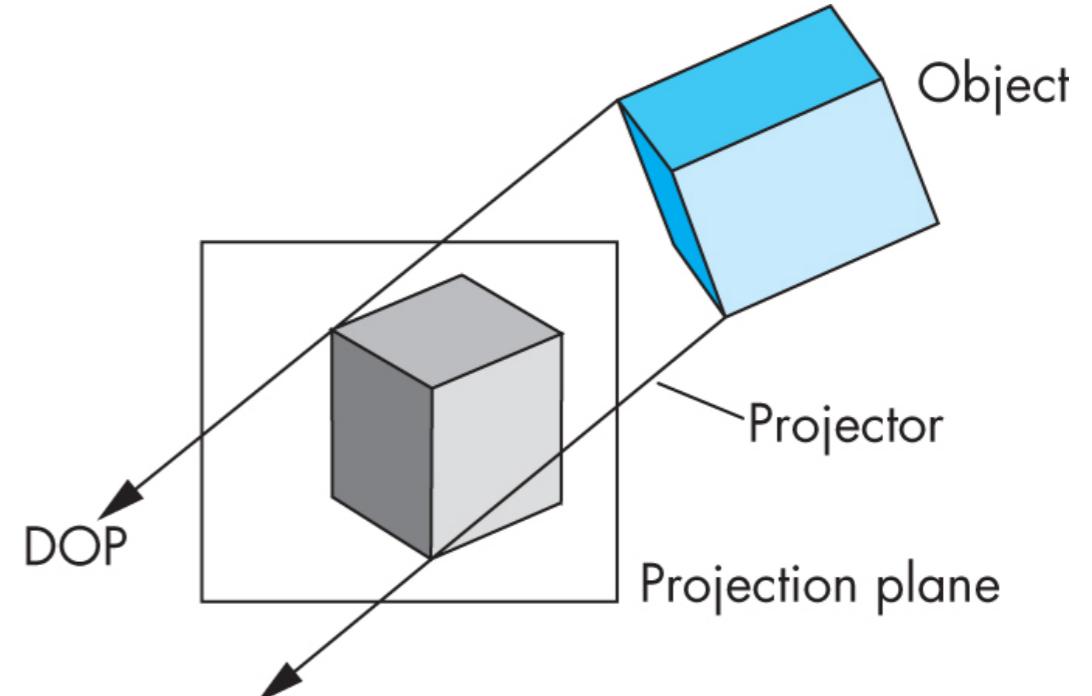
## 4.2.2. Matriu Projection

Les projeccions estàndard projecten en un pla: són les **projeccions geomètriques planars**.

- Els projectors són **rectes**
- Són projeccions que preserven les rectes però no necessàriament els angles



COP: Centre de projecció

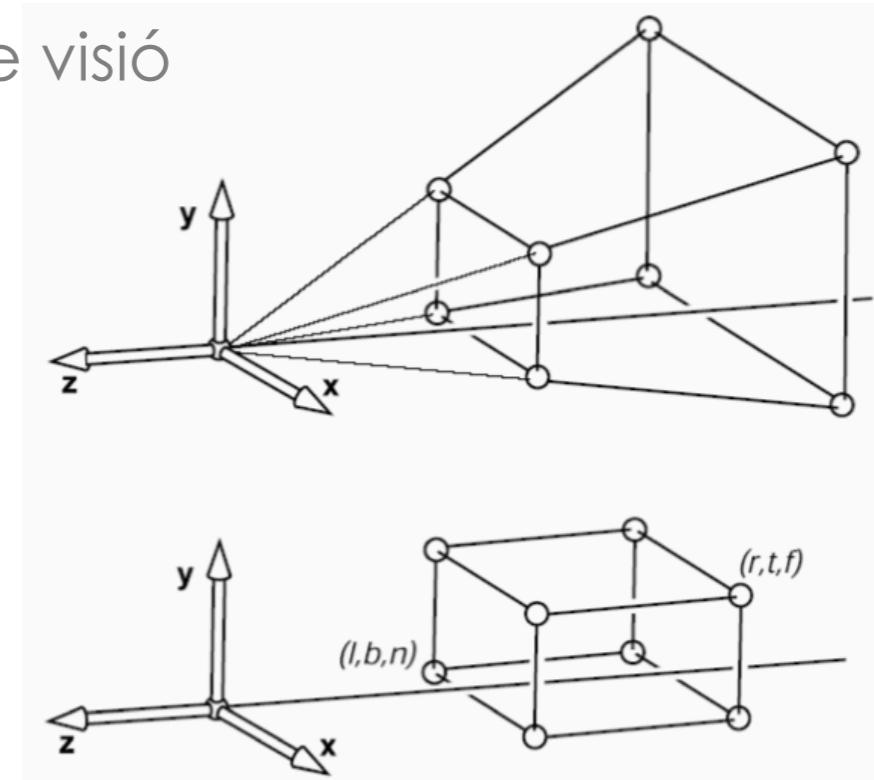


DOP: Direcció de projecció

## 4.2.2. Matriu Projection

El càlcul de la TG per a realitzar la projecció de les coordenades 3D a coordenades 2D (**matriu de projecció**) es caracteritza per:

- Aquest càlcul es fa a partir de la piràmide de visió
- La piràmide de visió té definida per:
  - El **tipus de projecció** (paral·lela o perspectiva)
  - El **pla** de projecció
  - El centre de projecció o la direcció de projecció
  - El **volum de visió** (*clipping volume*)
- Aquesta matriu es concatenarà per l'esquerra a la matriu **model view**
- La matriu **projecció** acabarà donant els punts en la **window** normalizada entre -1 i 1.

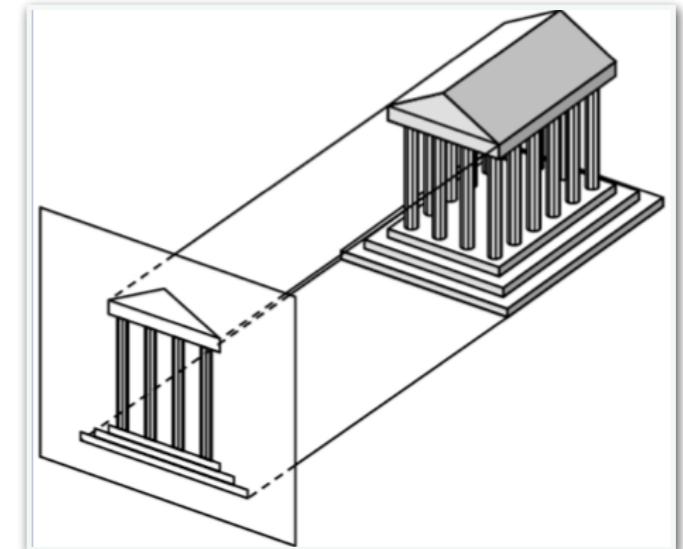


## 4.2.2. Matriu Projection

### Exemple: Projecció ortogràfica o paral·lela: matriu projecció

- Es projecta al pla  $z = 0$ , simplement posant  $z=0$ .
- Equivalent a la transformació en coordenades homogènies següent

$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



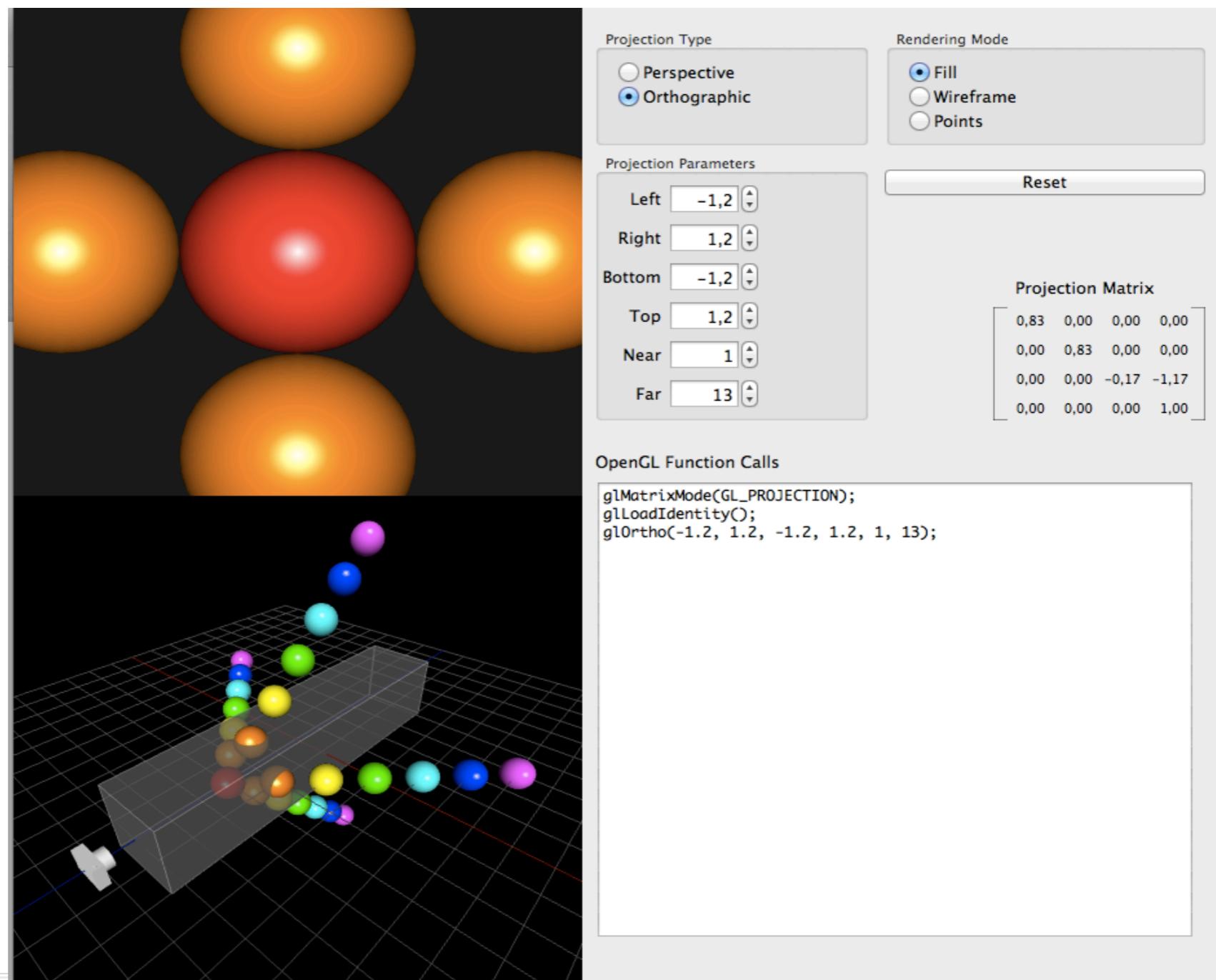
- La projecció ortogràfica directa és:

$$\mathbf{P} = \mathbf{M}_{\text{orth}}$$

## 4.2.2. Matriu Projection

Projeccions paral·leles o ortogonals:

### Software de simulació de la matriu projection i els seus paràmetres



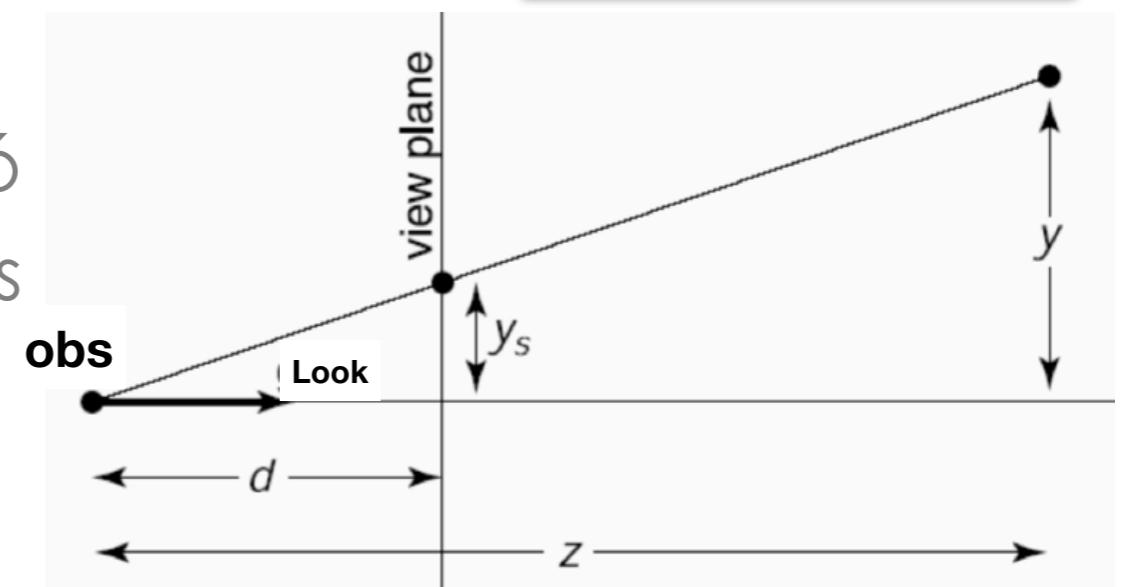
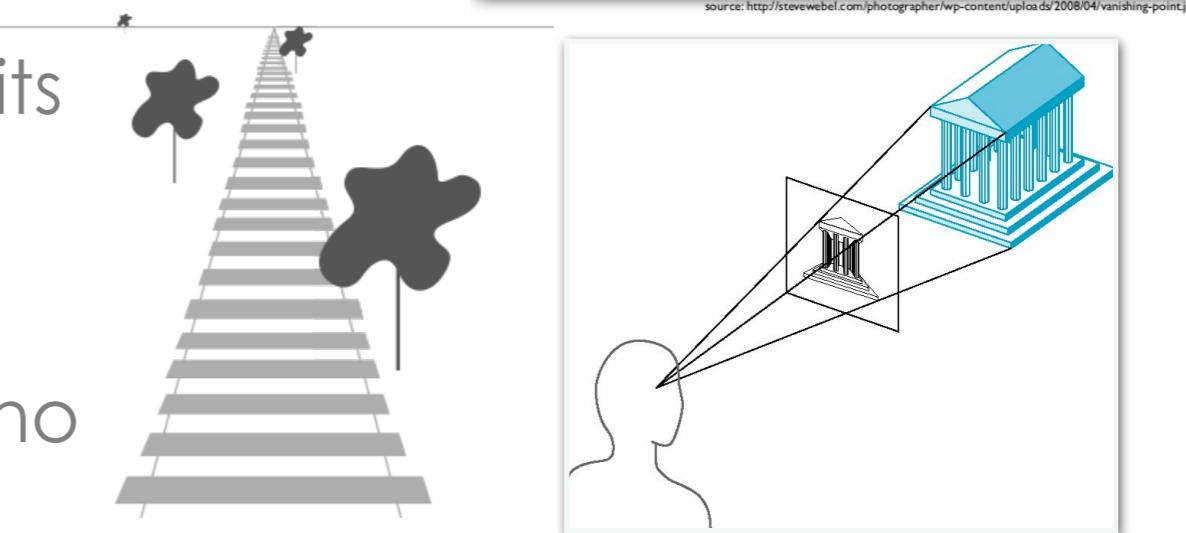
## 4.2.2. Matriu Projection

**Projeccions perspectives:** els projectors convergeixen al centre de projecció

- Els objectes més llunyans a l'observador es projecten més petits

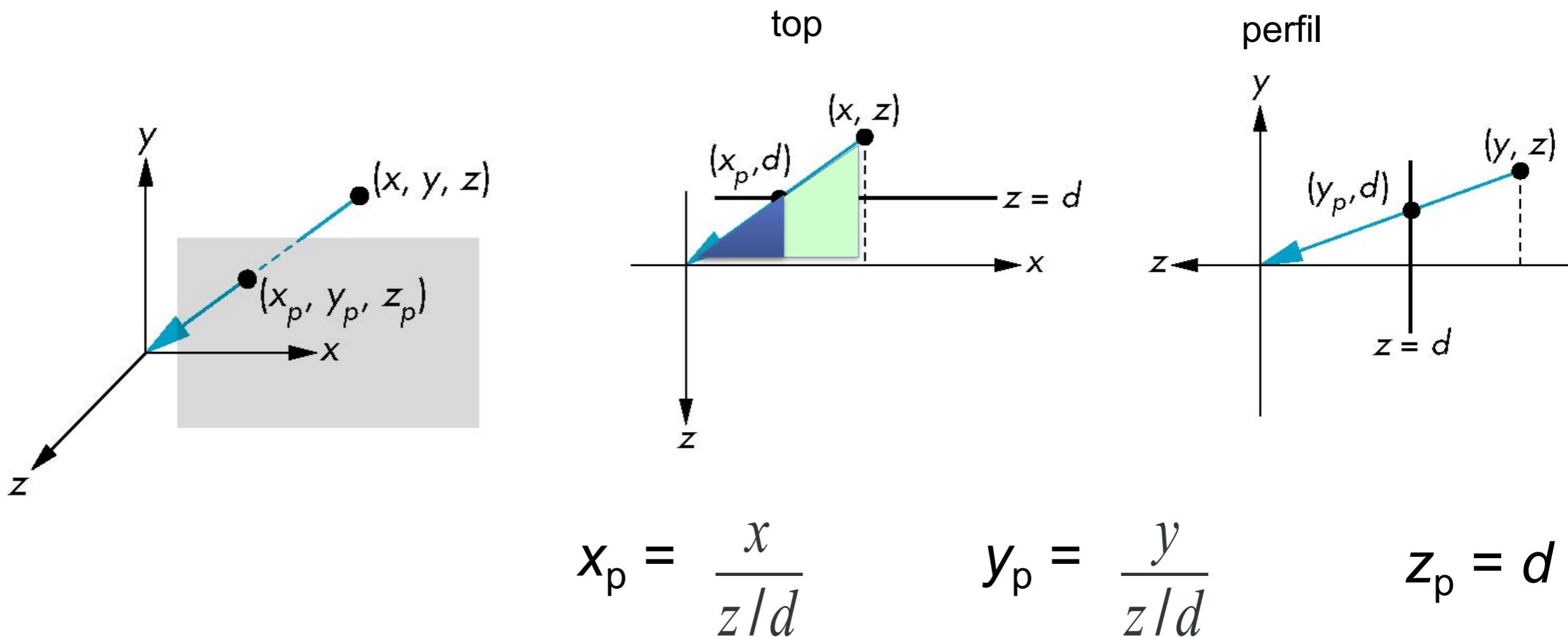
$$y_s = d^* \frac{y}{z}$$

- Realisme
- Distàncies iguals entre els objectes no es projecten en distàncies iguals
- Només es preserven els angles de cares paral·leles al pla de projecció
- Són més costoses de construir que les projeccions paral·leles



## 4.2.2. Matriu Projectió

**Definició de la matriu projecció:** Per una projecció perspectiva, si tenim el centre de projecció a l'origen i tenim el pla de projecció a distància  $d$  de la càmera ( $z=d$ ,  $d<0$ ), la projecció perspectiva d'un punt es calcularia com:



## 4.2.2. Matriu Projection

Aquest càcul, posat en forma de matrius, és:

$$\text{Si } \mathbf{p} = \mathbf{M}\mathbf{q}, \text{ on } \mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \quad \mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \Rightarrow \mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

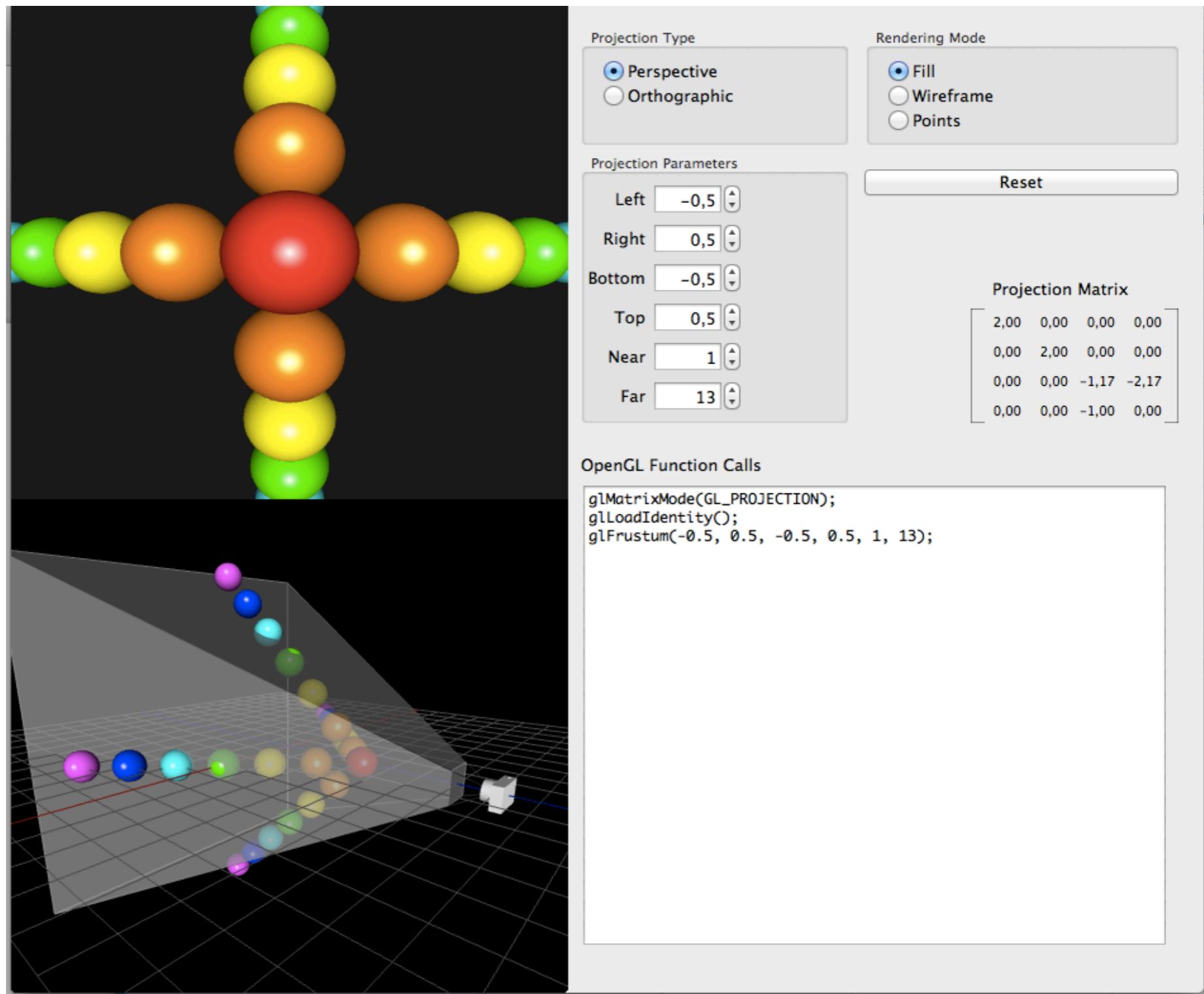
Per obtenir **coordenades homogènies** de la forma  $(x,y,z,1)$  cal que fem la divisió de cada un dels components del punt per  $z/d$

$$x_p = \frac{x}{z/d} \quad y_p = \frac{y}{z/d} \quad z_p = d$$

Aquesta divisió es fa de forma automàtica després d'executar-se el vertex shader

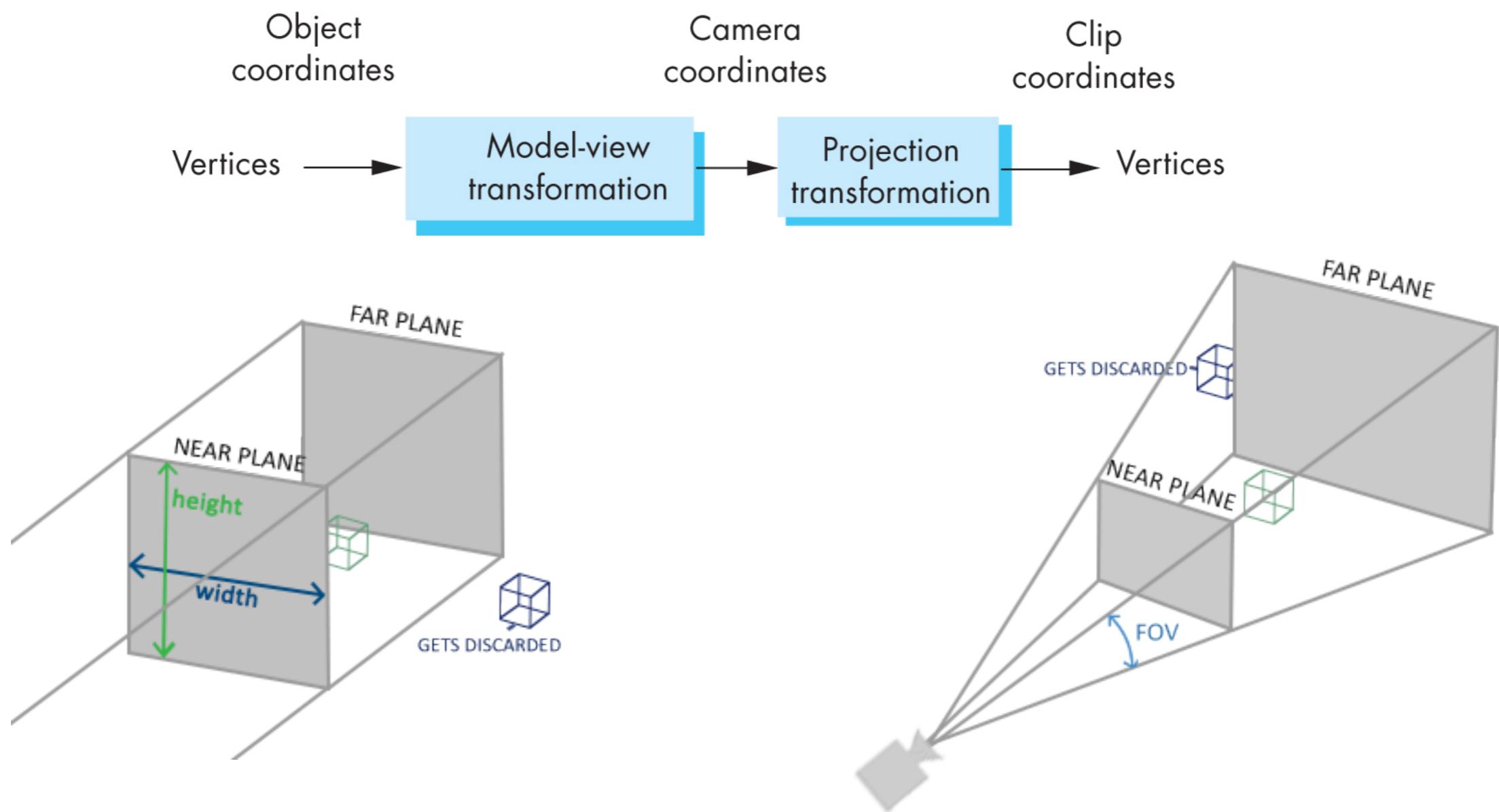
## 4.2.2. Matriu Projector

Projeccions  
perspectives:



## 4.2.2. Matriu Projection

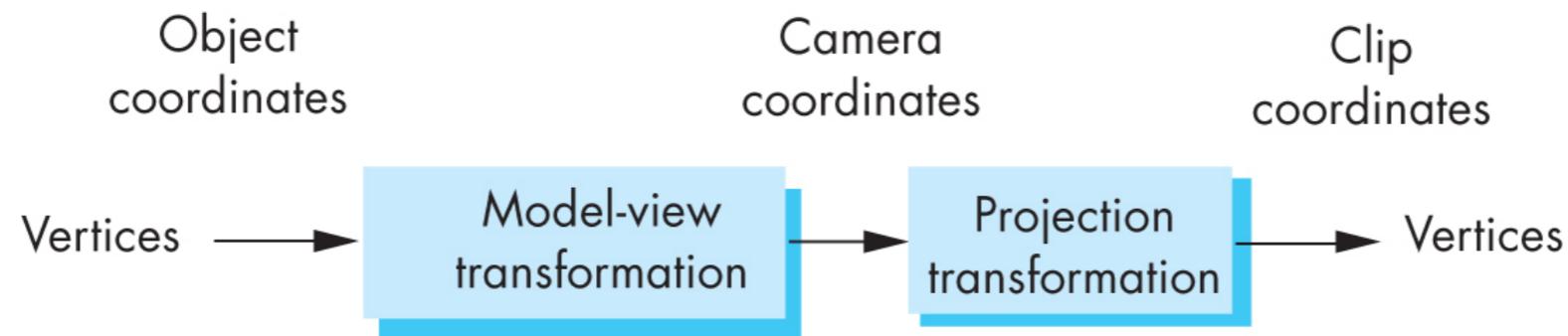
Matriu projection: acumula també una normalització que facilita el clipping.



## 4.2.2. Matriu Projection

Els vèrtexs es defineixen en coordenades homogènies i es transformen amb les matrius **model view** i **projection**.

Per defecte són les matrius identitat (que definirien una vista ortogonal)



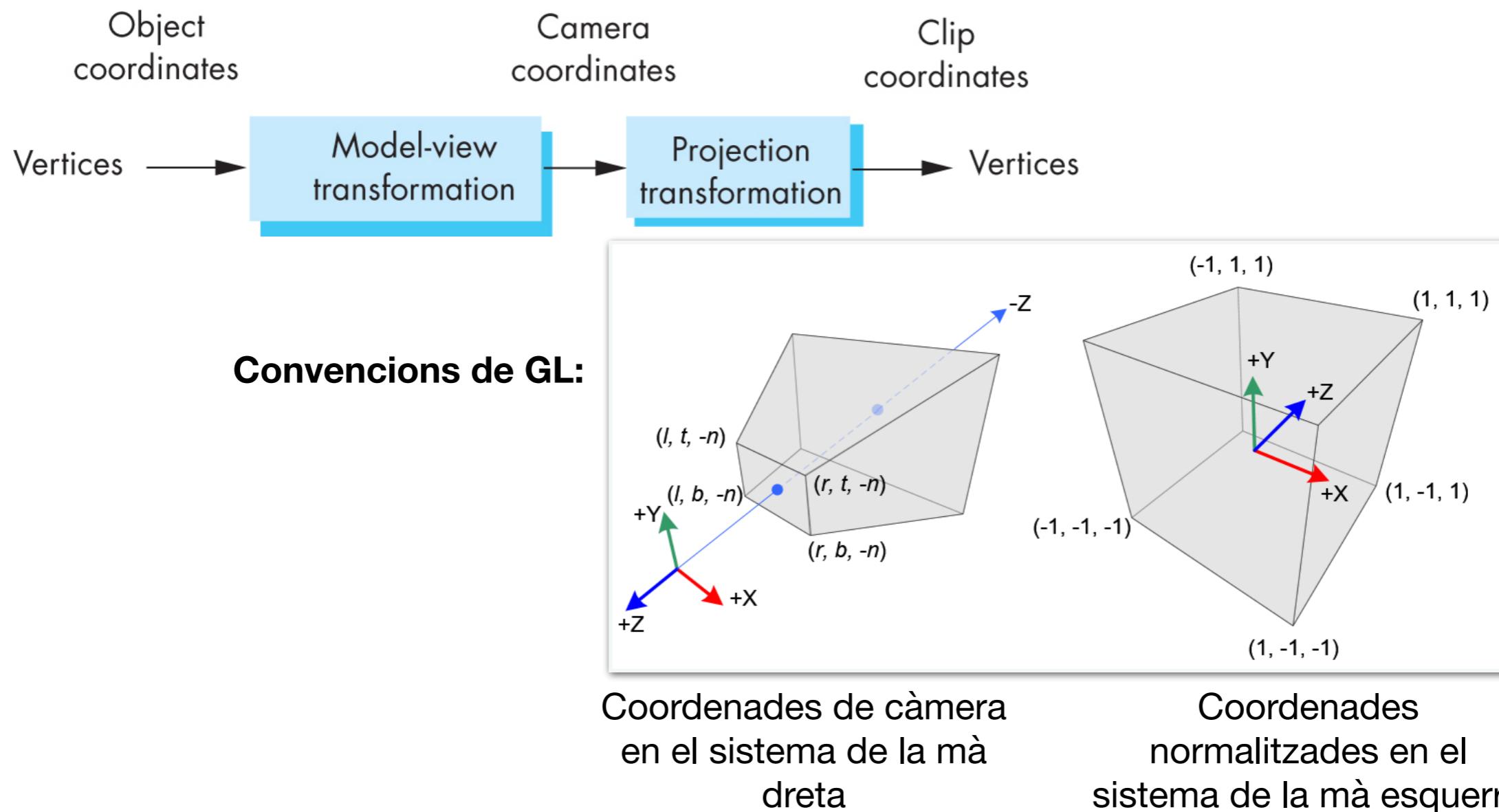
Per a calcular totes els tipus de projeccions dins del mateix *pipeline*, es fa un procés de **normalització** que:

- Permet fer el *clipping* contra un cub simple, independentment del tipus de projecció
- permet realitzar la projecció final a coordenades 2D al final del *pipeline*

Aquest fet és important ja que per l'eliminació de parts amagades és necessari conservar la informació de profunditat tant com es pugui.

## 4.2.2. Matriu Projection

La **normalització** és una transformació que permet convertir el volum de visió en un cub centrat a l'origen i d'aresta 2. Es codifica dins de la matriu **projection**



## 4.2.2. Matriu Projection

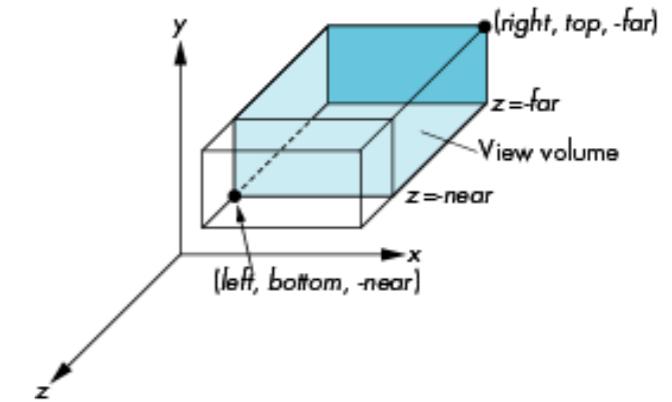
Exemple: Projecció paral·lela:

1. - Es mou el centre a l'origen:

Translate(-(left+right)/2, -(bottom+top)/2, **(near+far)/2**)

2. - S'escala a un cub d'aresta 2

Scale(2/(right-left), 2/(top-bottom), -2/**(far-near)**)



$$\mathbf{ST} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{P} = \mathbf{M}_{\text{orth}} \mathbf{ST}$$

## 4.2.2. Matriu Projection

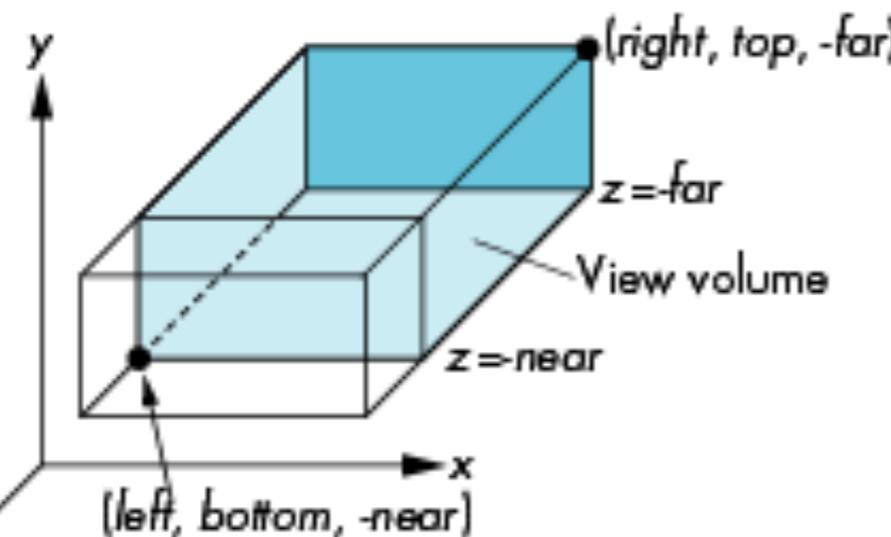
**Definició de la matriu projection:** Per una projecció **ortogonal** qualsevol, s'usa la crida:

- En versions d'OpenGL on el pipeline és fixe:

**void glOrtho(left,right,bottom,top,near,far);**

- En el codi de la pràctica:

**glm::ortho(left, right, bottom, top, near, far)**



**Near i far** són positius estan mesurats des de la càmera

## 4.2.2. Matriu Projection

**Projecció perspectiva:** Es redefineix la projecció de la transparència 34:  $z_{\text{near}}$  i  $z_{\text{far}}$

$$\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Aplicada al punt  $(x, y, z, 1)$ :

$$\begin{aligned} x' &= x/z \\ y' &= y/z \\ z' &= -(a+b/z) \end{aligned}$$

$$\alpha = \frac{\text{near} + \text{far}}{\text{far} - \text{near}}$$

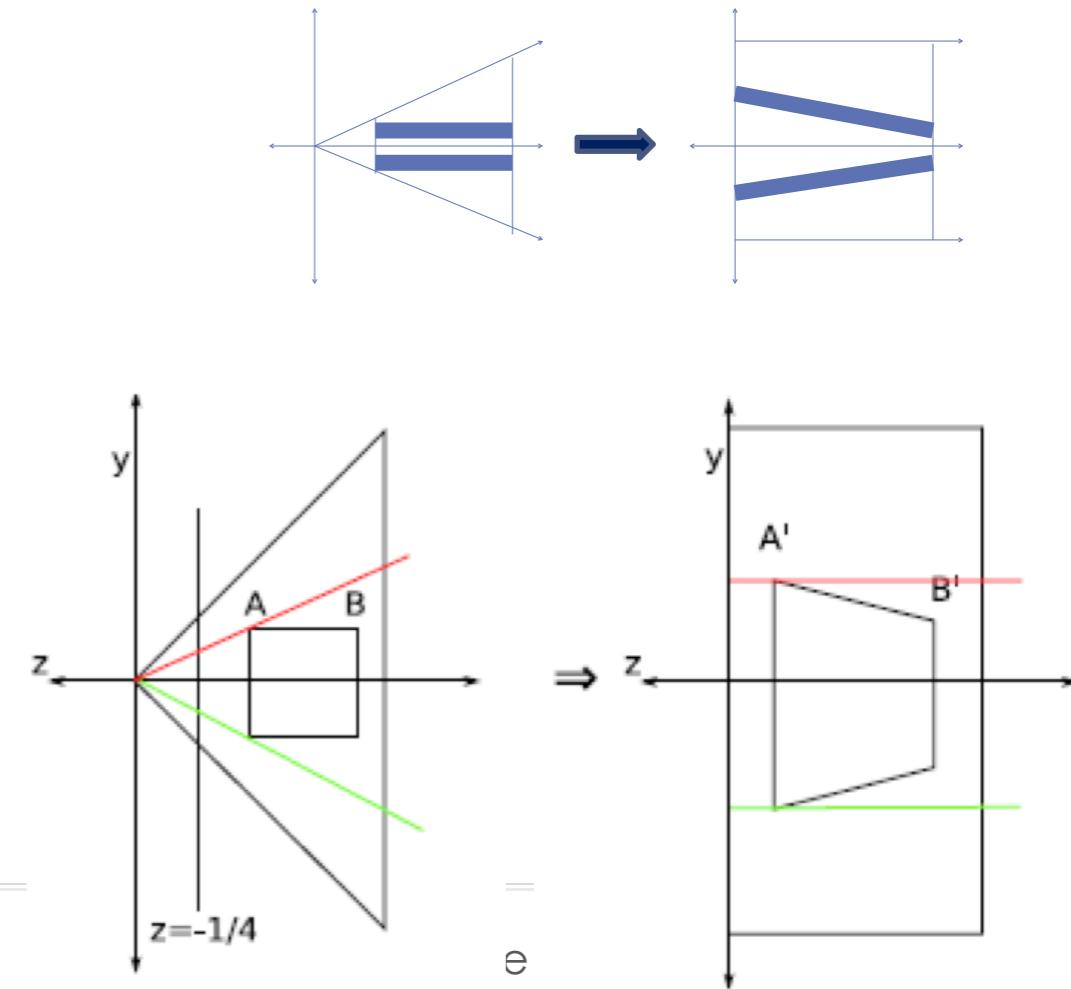
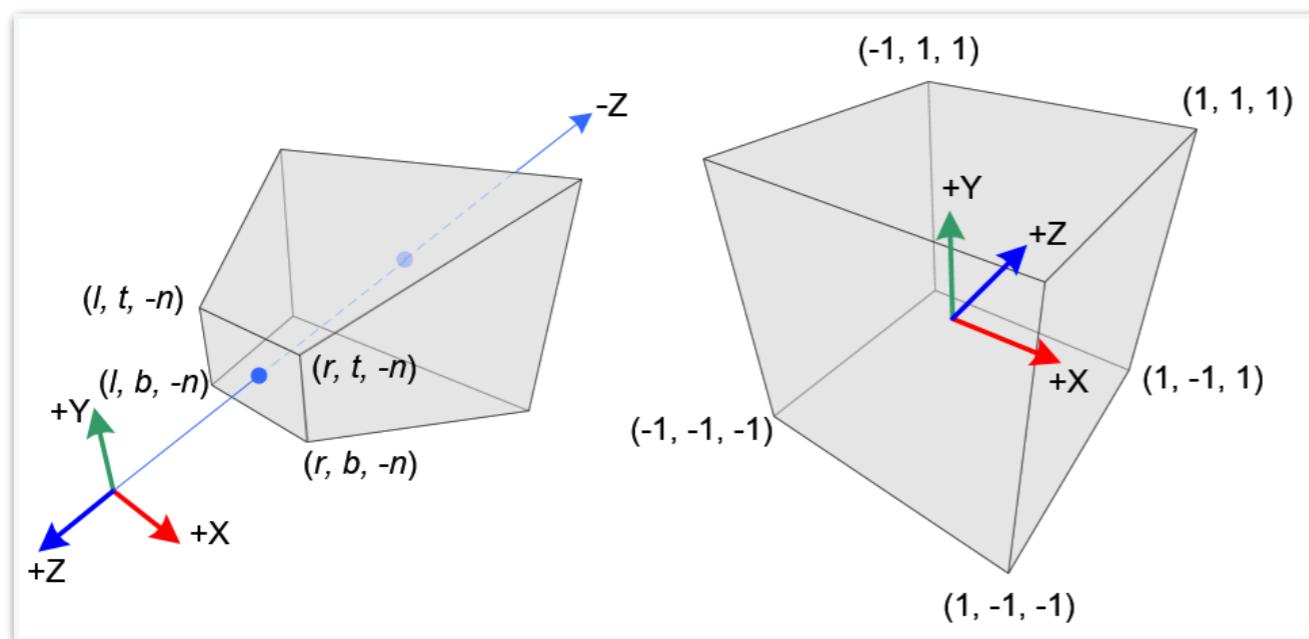
$$\beta = \frac{2\text{near} * \text{far}}{\text{near} - \text{far}}$$

- El pla anterior ( $z=\text{near}$ ) serà  $z' = -1$
- El pla posterior ( $z=\text{far}$ ) serà  $z' = 1$
- Els plans laterals,  $x' = \pm 1, y' = \pm 1$
- El pla de projecció és el  $z = \text{near}$
- Es preserven les relacions entre  $z$ 's

## 4.2.2. Matriu Projection

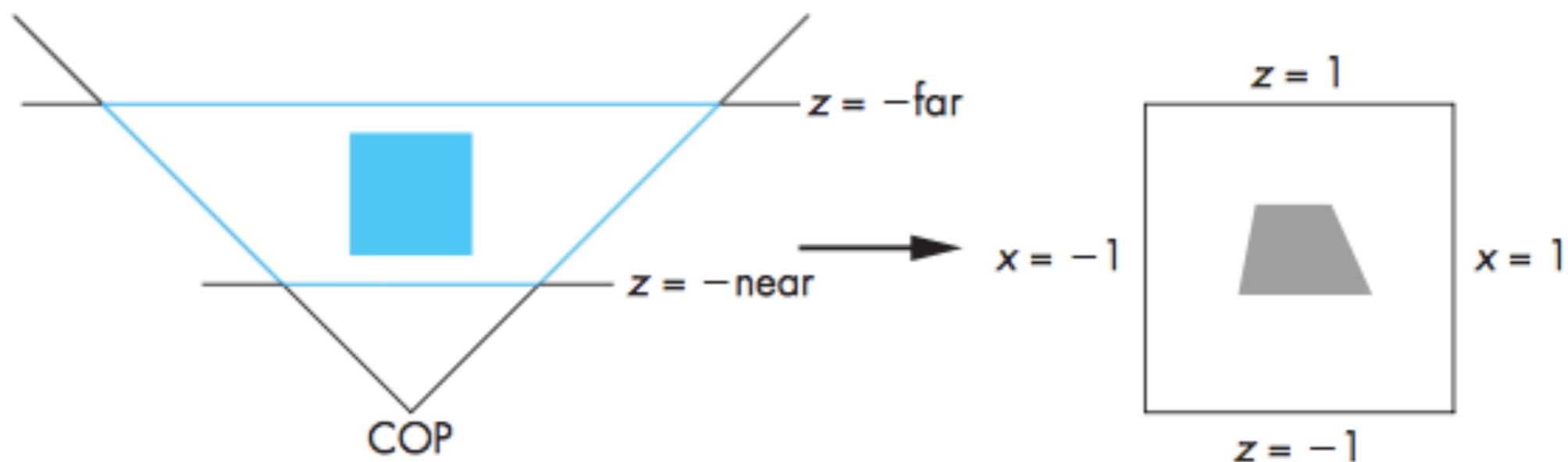
**Per saber-ne més...** En el cas general, on els angles d'obertura no són simètrics ni de  $90^\circ$ :

1. la conversió a un volum simètric i de  $90^\circ$  (amb un shear,  $H$ )  
Shear:  $((\text{left} + \text{right})/2, (\text{top} + \text{bottom})/2, -\text{near})$  a  $(0, 0, -\text{near})$
2. un escalat  $(S(-2 * \text{near}/(\text{right} - \text{left}), -2 * \text{near}/(\text{top} - \text{bottom}), 1))$
3. normalització(N)



## 4.2.2. Matriu Projection

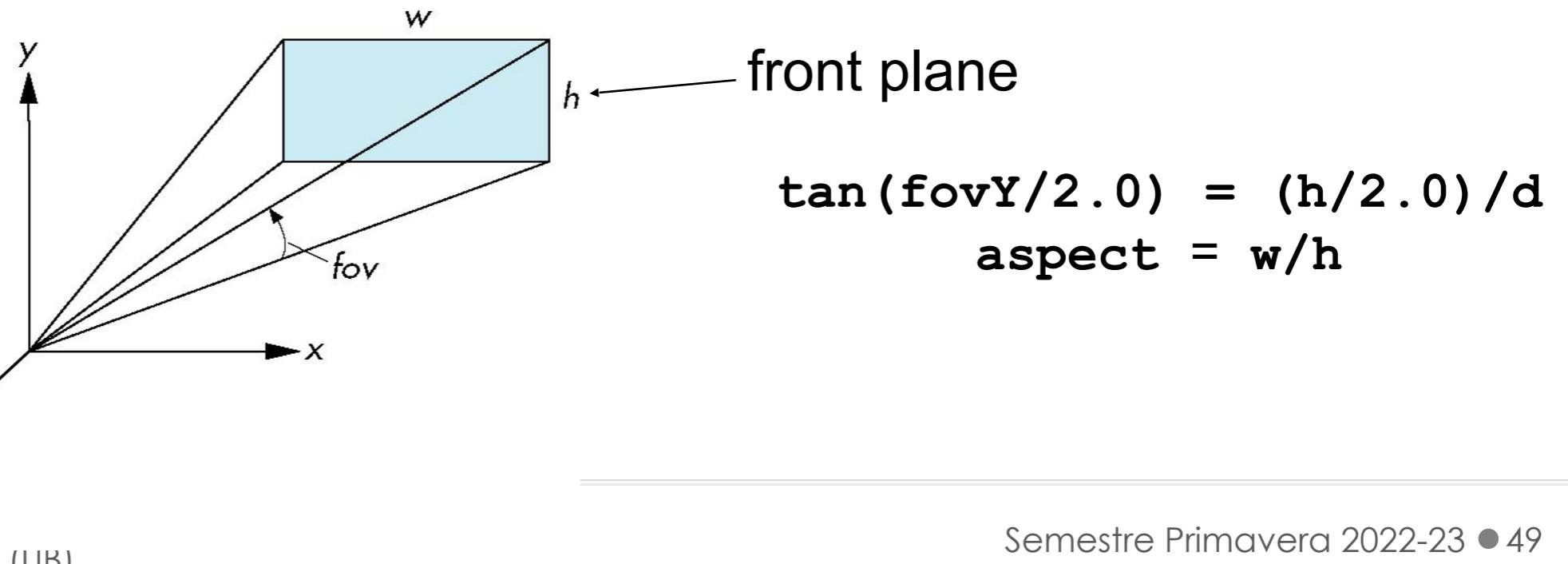
Un cop aplicada la matriu **projection**, s'ha passat de coordenades de càmera a coordenades de window normalitzada, d'aresta 2 i centrada al (0,0,0).



## 4.2.2. Matriu Projection

**Definició de la matriu projection:** Per una projecció **perspectiva simètriques**, s'usa la crida:

- En versions d'OpenGL on el pipeline és fix, s'usa la crida:  
**void gluPerspective(fovy, aspect, zNear, zFar);**
- En la pràctica, usarem:  
**glm::mat4 Perspective (fovy, aspect, near, far)**



## 4.2.2. Matriu Projection

### Z-Buffer

Els valors de Z sempre són positius i van des del pla de clipping anterior ( $z_{near}$ ) al posterior ( $Z_{far}$ ).

El z-buffer enmagatzema enters positius

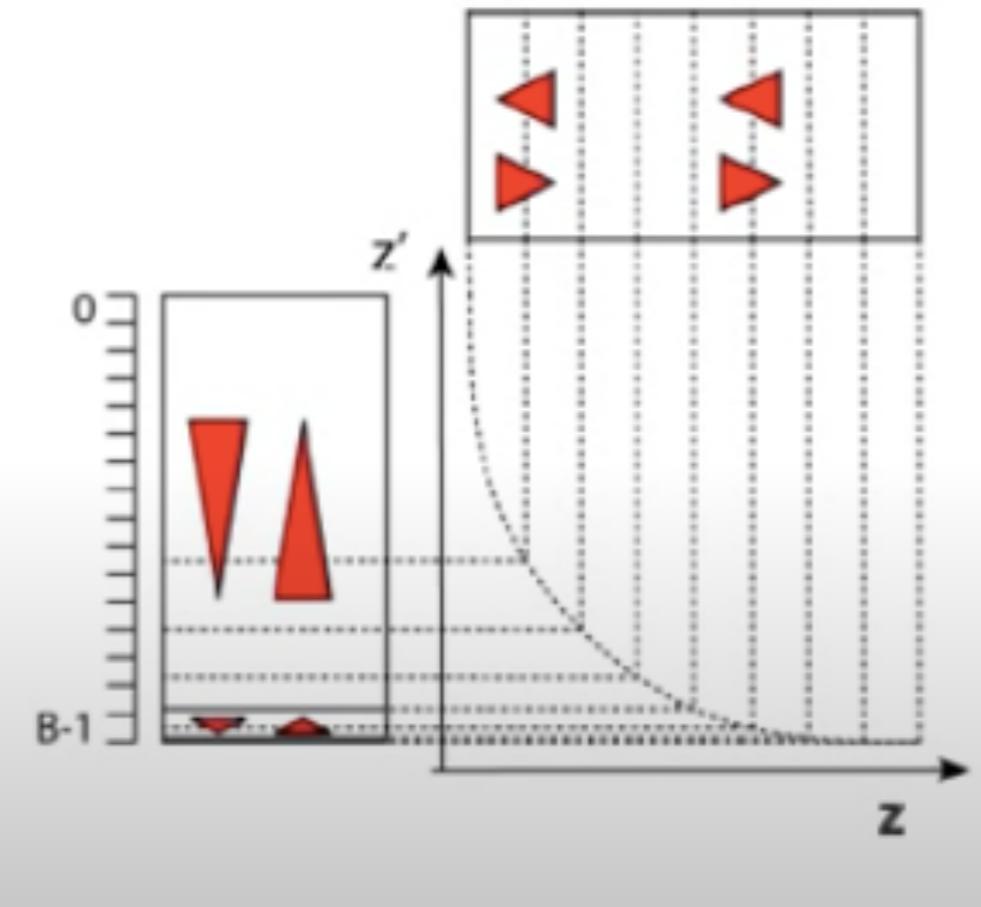
$$b = \{0, 1, \dots, B-1\}$$

Els valors de z's en coordenades de càmera es mapegen a aquests valors, en intervals regulars de mida

$$\Delta z = \frac{(Z_{far} - z_{near})}{B}$$

I un z d'un punt es calcula com:

$$\frac{z - z_{near}}{Z_{far} - z_{near}} * (B - 1)$$



En projeccions perspectives,  
es té més precisió a prop de  
l'observador i menys precisió  
quan més lluny

# Índex

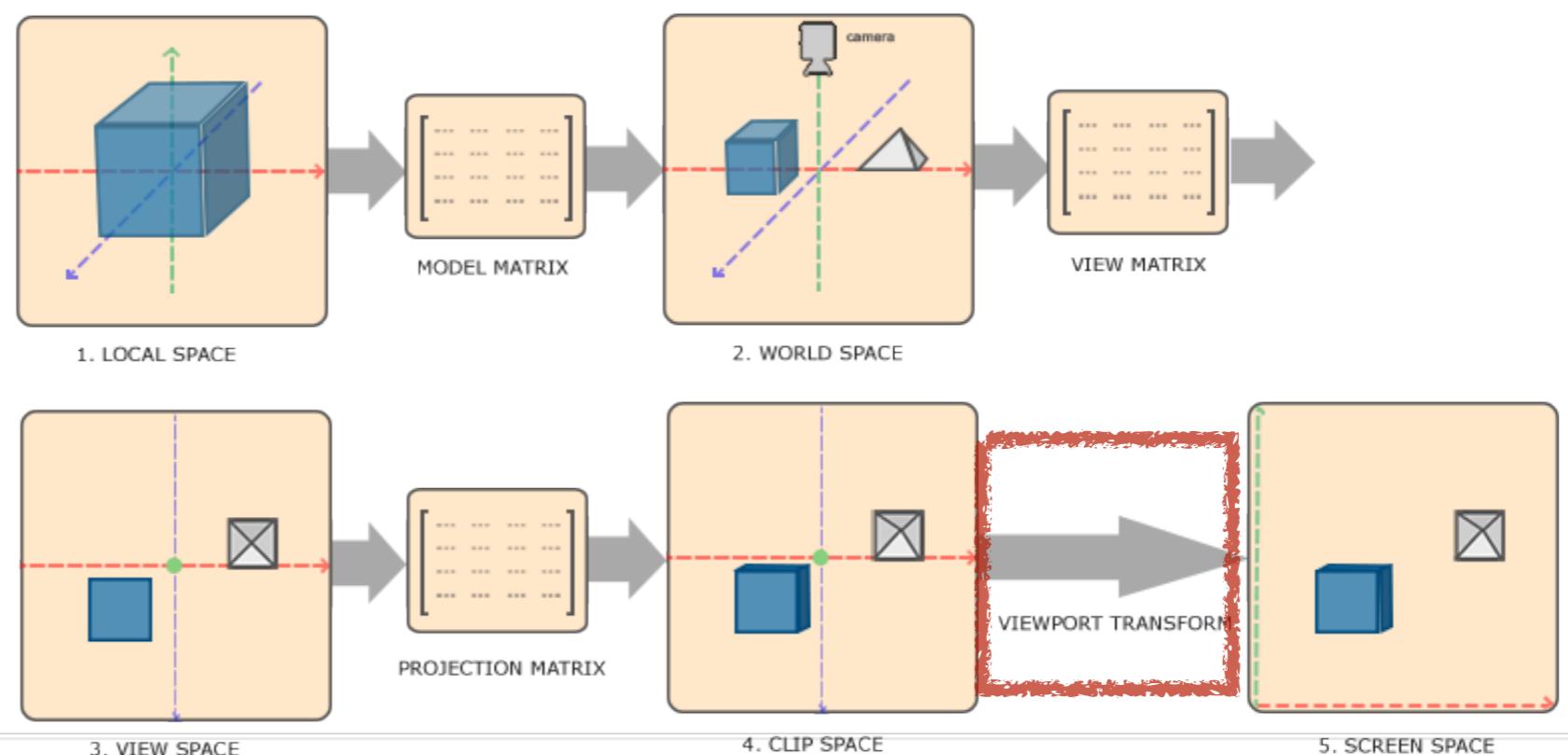
4.1. Atributs de la càmera

4.2. Pipeline de visualització:

4.2.1. Matriu ModelView

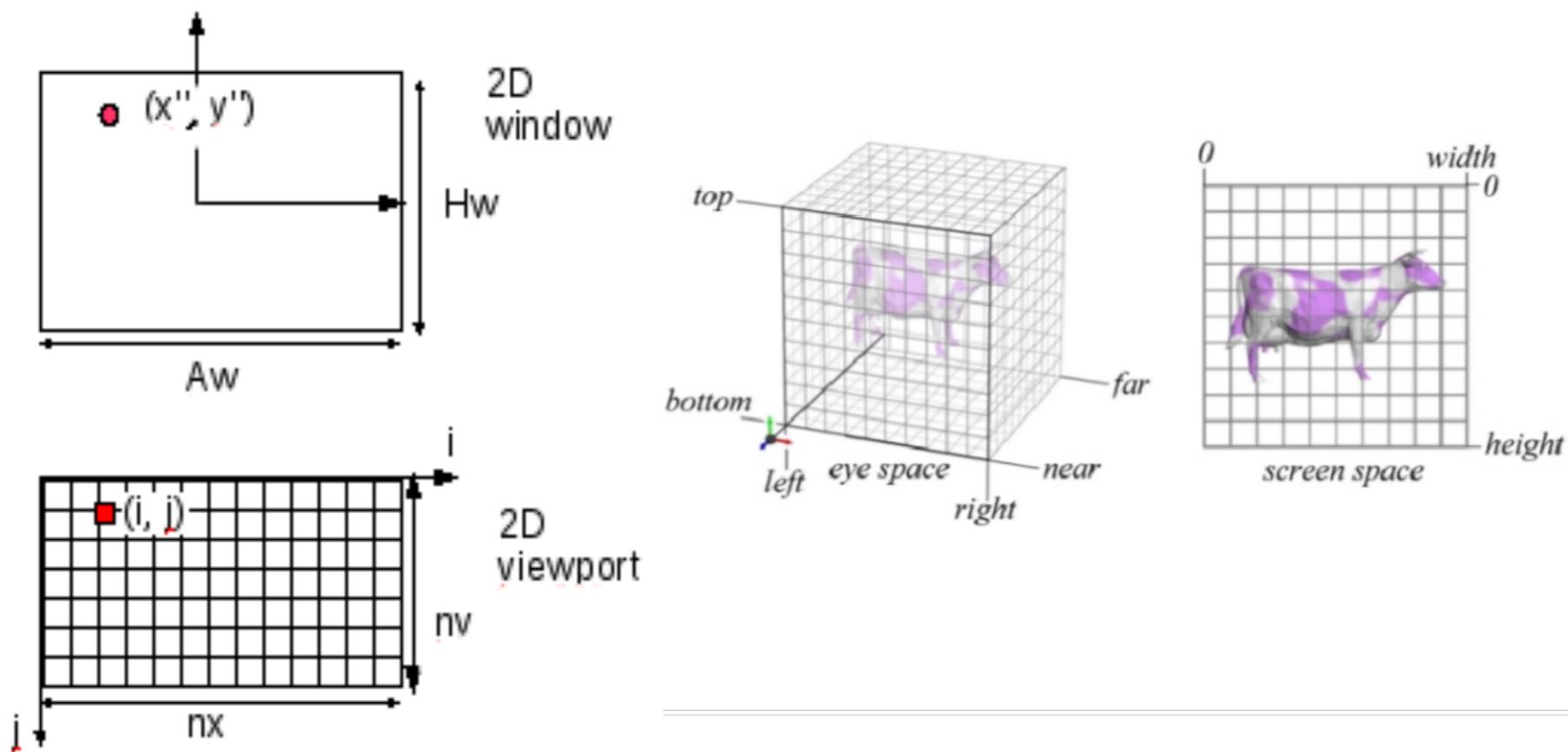
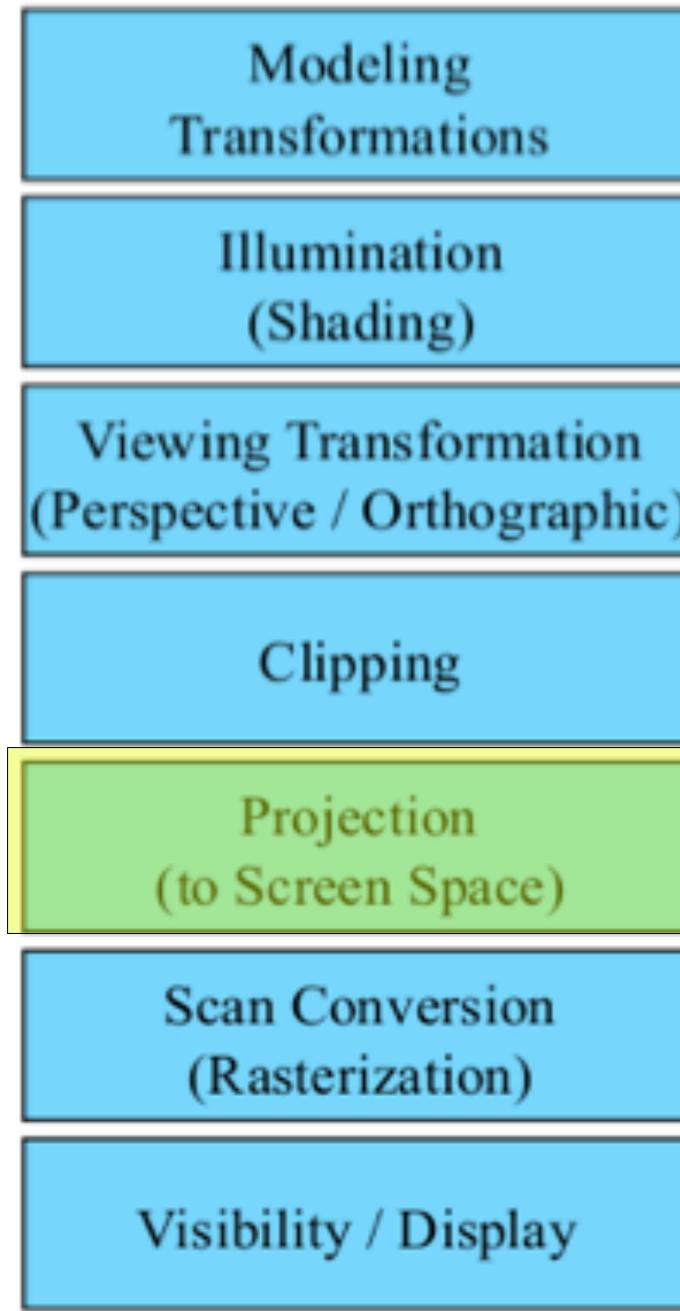
4.2.2. Matriu Projection

**4.2.3. Transformació Window-Viewport**



## 4.2.3. Transformaciones window-viewport

- Es projecta cada vèrtex de l'espai 2D continu (**window**) al frame buffer (o **viewport**)



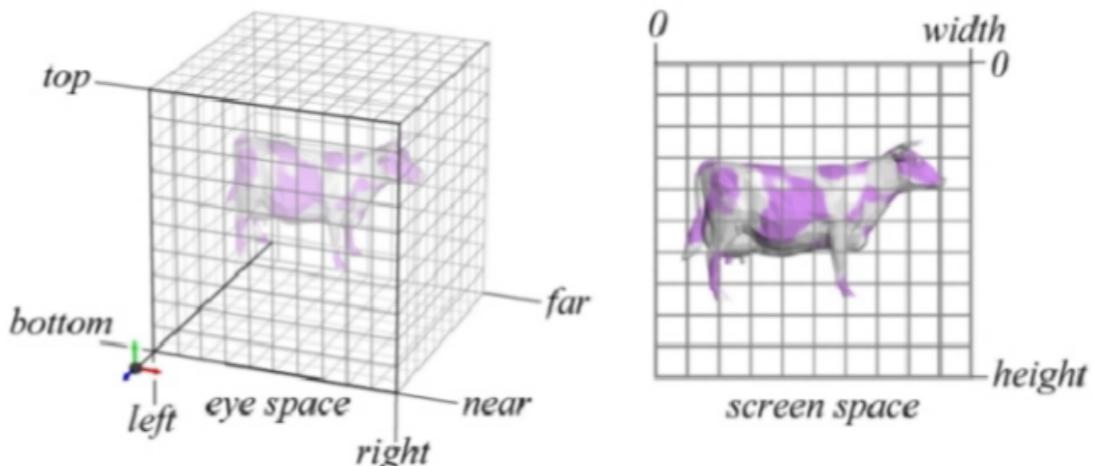
## 4.2.3. Transformacions window-viewport

Les projeccions obtenen coordenades normalitzades pertanyents a  $\mathbb{R}^4$ :

- la capsà mínima contenidora 2D d'aquests punts transformats en el pla de projecció s'anomena **window normalitzada**.

Aquestes coordenades es corresponen a coordenades de pantalla discretes:

- el **viewport** és l'àrea de la pantalla que ocuparà la visualització final.



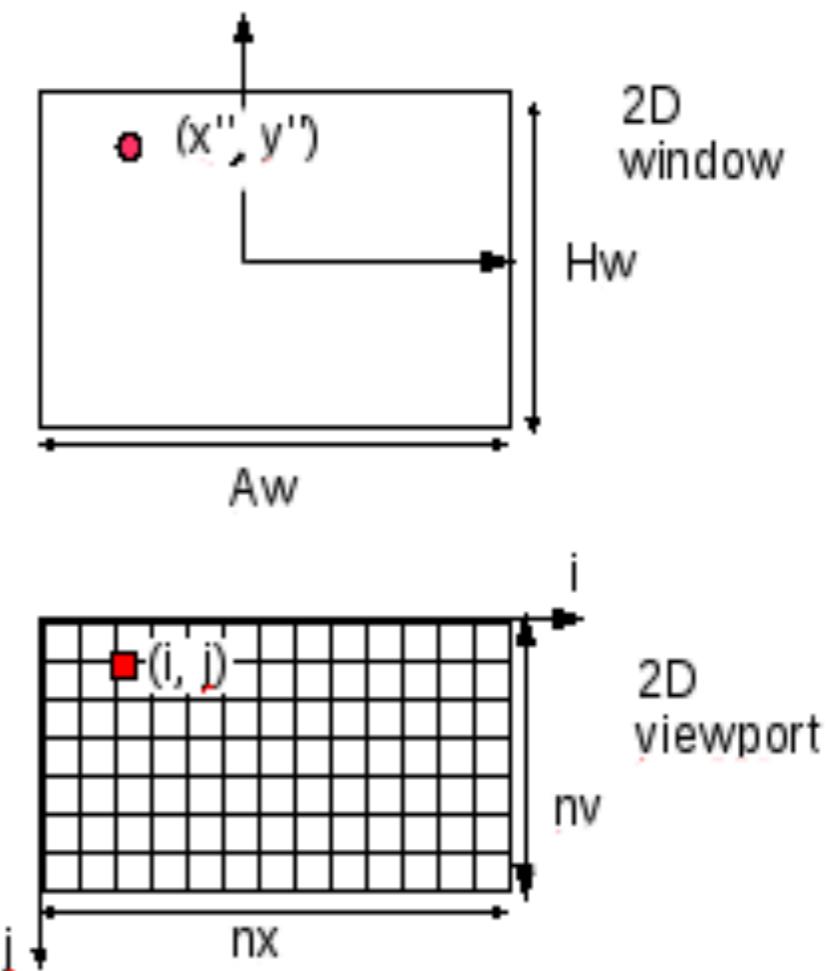
## 4.2.3. Transformacions window-viewport

Per a realizar la conversió entre coordenades de window ( $x'', y'', 1$ ) i coordenades de viewport ( $i, j$ ), podem trobar la relació següent:

$$\frac{i}{nx} = \frac{0.5 Aw + x''}{Aw}$$

$$\frac{j}{ny} = \frac{0.5 Hw - y''}{Hw}$$

Suposem que l'origen del viewport  
és el punt (0,0)



## 4.2.3. Transformacions window-viewport

**Pas de  $(x'', y'')$  de window normalizada a  $(i, j)$  de viewport:**

Aillant les variables  $i, j$ , obtenim les següents expressions que es poden posar en forma matricial, amb un escalat i una translació (transformació window-viewport):

$$i = \frac{nx}{Aw} * x'' + nx * 0.5$$

$$j = \frac{-ny}{Hw} * y'' + ny * 0.5$$

$$\mathbf{T}_{\text{window-viewport}} = \mathbf{T}_{xy} \cdot \mathbf{S}_{xy} = \begin{bmatrix} 1 & 0 & nx * 0.5 \\ 0 & 1 & ny * 0.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{nx}{Aw} & 0 & 0 \\ 0 & -\frac{ny}{Hw} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$