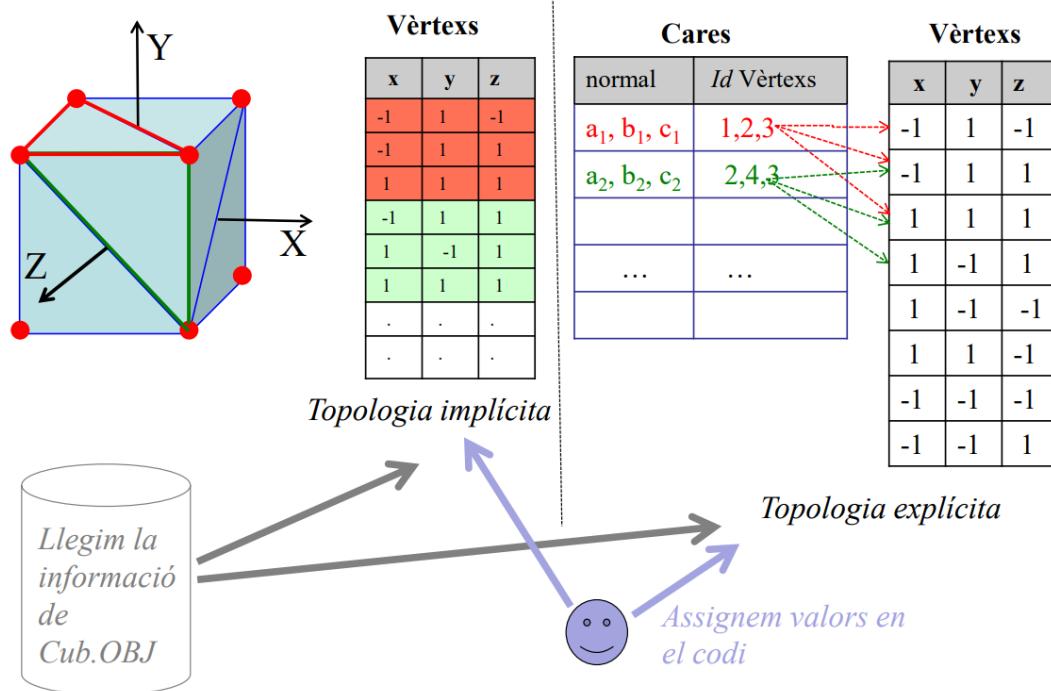


# Model Fronteres: conjunt de triangles

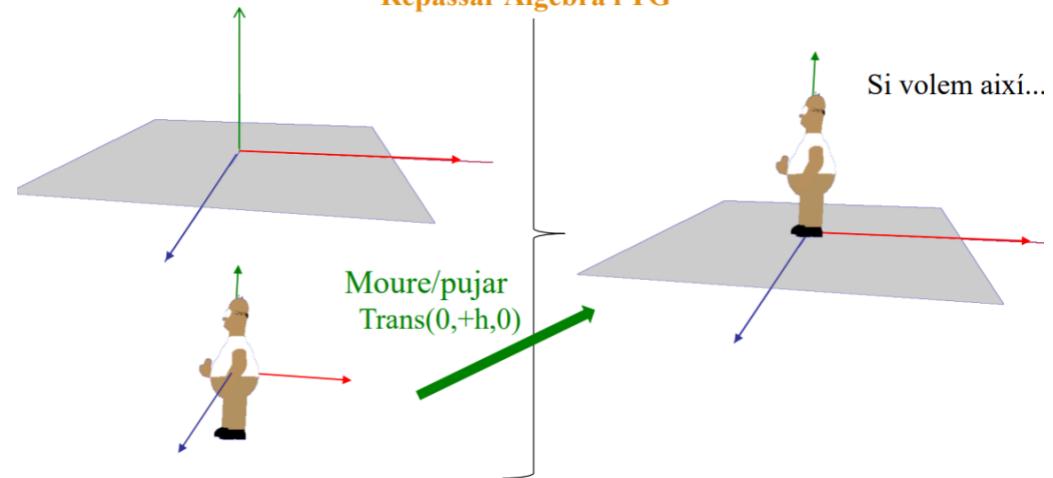


## Pintar en OpenGL 3.3: “core” mode

1. Crear en GPU/OpenGL un *VAO* que encapsularà dades del model. Crear *VBO* que guardarà les coordenades dels vèrtexs (i si cal, normal, color,...)
2. Guardar la llista de vèrtexs en el *VBO* (i si cal, color i normal en els seus *VBO*)
3. Cada cop que es requereix pintar, indicar el *VAO* a pintar i dir que es pinta: `glDrawArrays(...)`. Acció **pinta\_model()** a teoria.



## Repassar Àlgebra i TG



- Cal aplicar TG que modifica coordenades vèrtexs
- TG queda definida per matriu  $4 \times 4$ :

$$\mathbf{V}_A = \text{TG } \mathbf{V}_m = T(0, +h, 0) \mathbf{V}_m$$

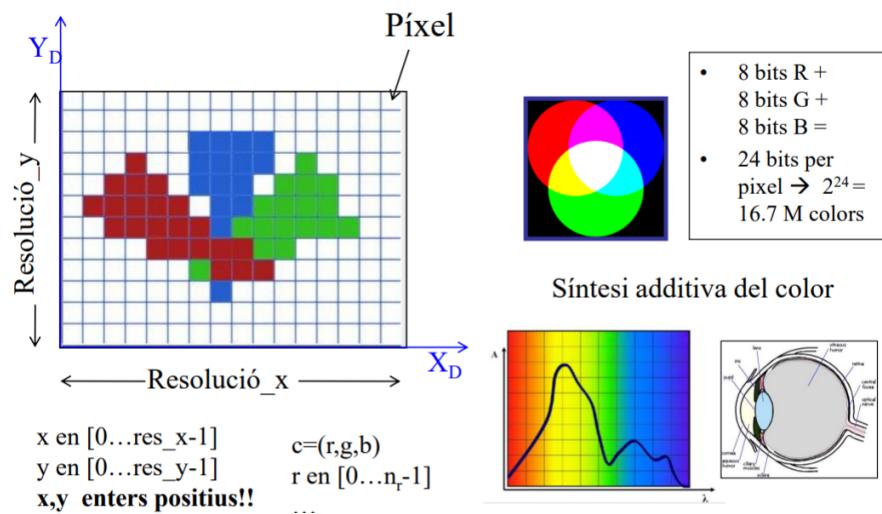
$$T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Sobre processament de vèrtexs:

- Important diferenciar SCA (sistema coordenades aplicació / món / escena) de SCO (sistema de coordenades de l'observador).<sup>1</sup>
- La posició de l'OBS no depèn del tipus d'òptica (obvi).
- Visualització: posició + orientació, òptica, fer la foto, emmarcar.
- El viewport és tota la finestra d'OpenGL.

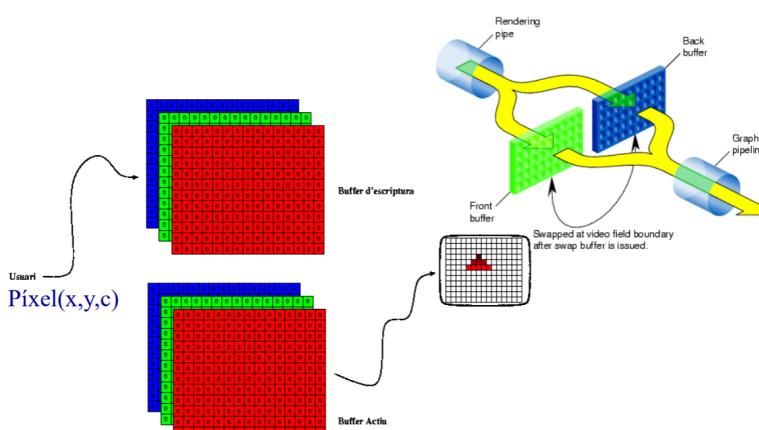
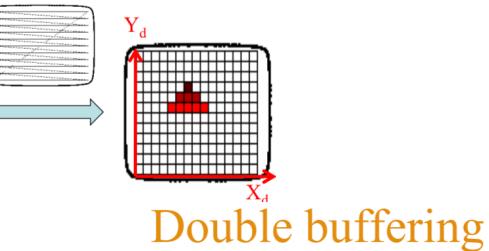
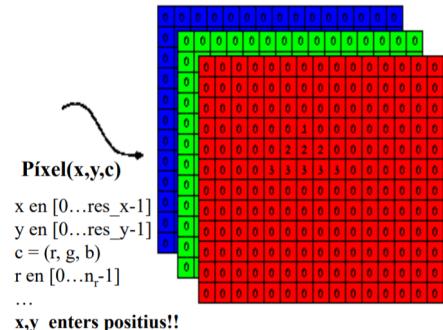
<sup>1</sup> Un Patricio centrat a l'escena està a la posició (0, 0, 0) en SCA (perquè *està* centrat en l'escena); però, si mirem el Patricio des de davant, segurament el centre del Patricio serà a la coordenada (0, 0, -2R), perquè, des de la càmera (observador), ens hem de desplaçar 2 vegades el radi de l'escena per aplegar al punt mitjà del Patricio.

# Pantalles d'escombrat/raster

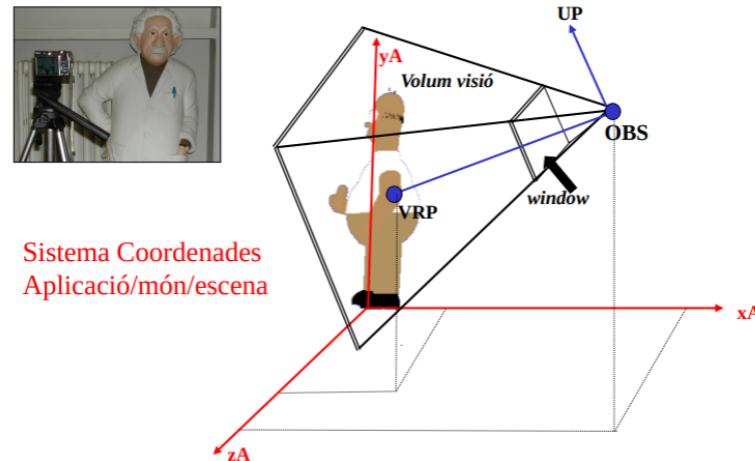


## Frame buffer

fb és taula [res\_x][res\_y] de color  
 $fb[x][y] = c$

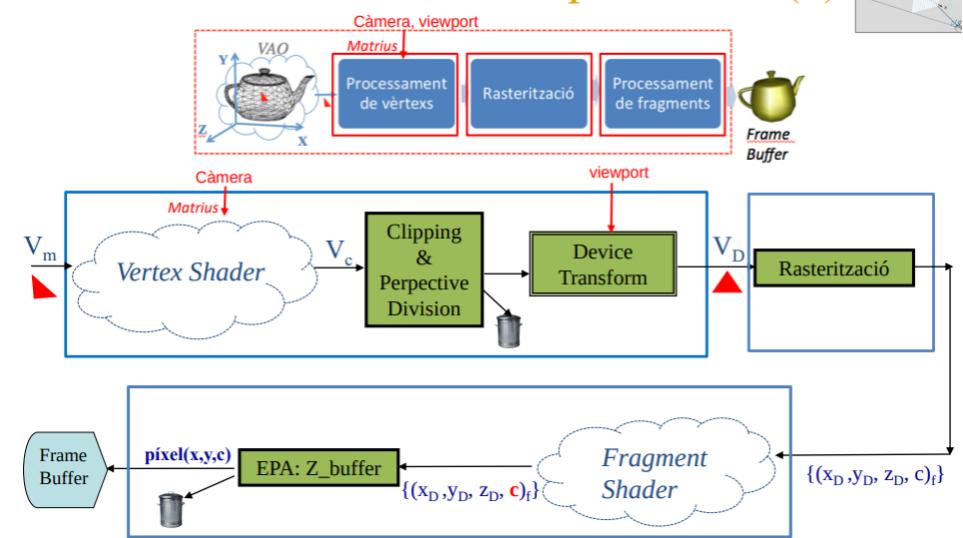


# Com indicar la càmera?



1. Ubicació respecte SCA: obs, vrp, up
  2. Definir el Volum de Visió: òptica (window, zNear, zFar)
- Per visualitzar:
    - Fem un cop: per cada model, crear i omplir el seu VAO.
    - Cada cop que refresquem finestra: per cada model m, calculem la seva TG (transformacions geomètriques), indiquem a OpenGL que la volem usar (modelMatrix(TG)) i pintar\_model().
    - Cura, perquè apliquem la TG abans de tot!! Per tant, donat un vèrtex de model V, el passem a coordenades de clipping així<sup>2</sup>: VC = PM VM TG V.

## Pintar/visualitzar en OpenGL 3.3 (2)

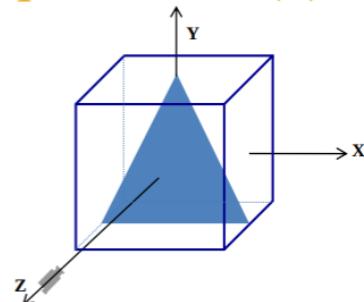


## Pintar/visualitzar en OpenGL 3.3 (3)

### Vertex Shader

```
#version 330 core
in vec3 vertex;

void main() {
    gl_Position = vec4 (vertex, 1.0);
}
```

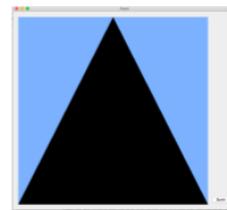


Volum de Visió cub de (-1,-1,-1) a (1,1,1)

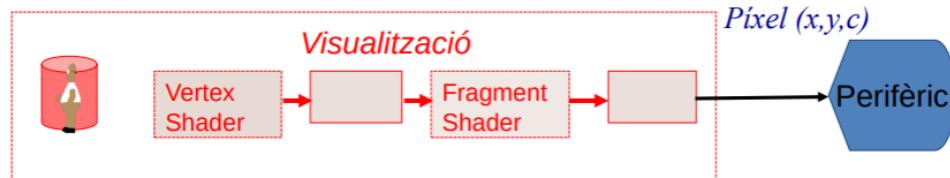
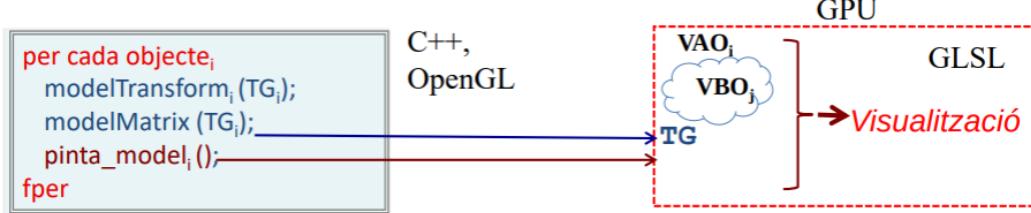
### Fragment Shader

```
#version 330 core
out vec4 FragColor;

void main() {
    FragColor = vec4(0, 0, 0, 1);
}
```



## Pintar en OpenGL 3.3: “core” mode

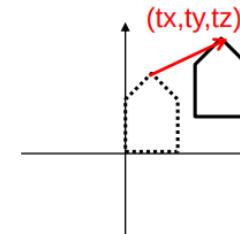


```
#version 330 core
in vec3 vertex;
uniform mat4 TG;

void main() {
    gl_Position = TG * vec4(vertex, 1.0);
}
```

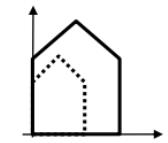
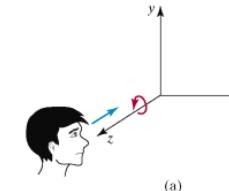
## Transformacions Geomètriques

Transformació geomètrica



$$x' = x + tx; \quad y' = y + ty; \quad z' = z + tz$$

Matriu 4x4  
TG



$T(tx,ty,tz)$

$$\begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$R_z(\text{angle})$

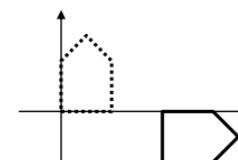
$$\begin{bmatrix} \cos \alpha & -\sin \alpha & 0 & 0 \\ \sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$S(s_x,s_y,s_z)$

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

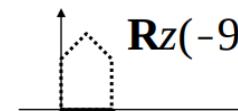
## Composició de Transformacions

- Imaginem que volem

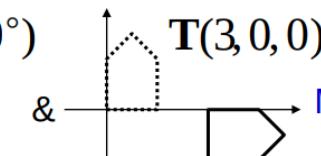


No es pot fer amb cap de les matrius anteriors

- Cal compondre/efectuar dues transformacions



$$\mathbf{P}' = R_z(-90^\circ) \cdot \mathbf{P}$$



$$\mathbf{P}'' = T(3,0,0) \cdot \mathbf{P}'$$

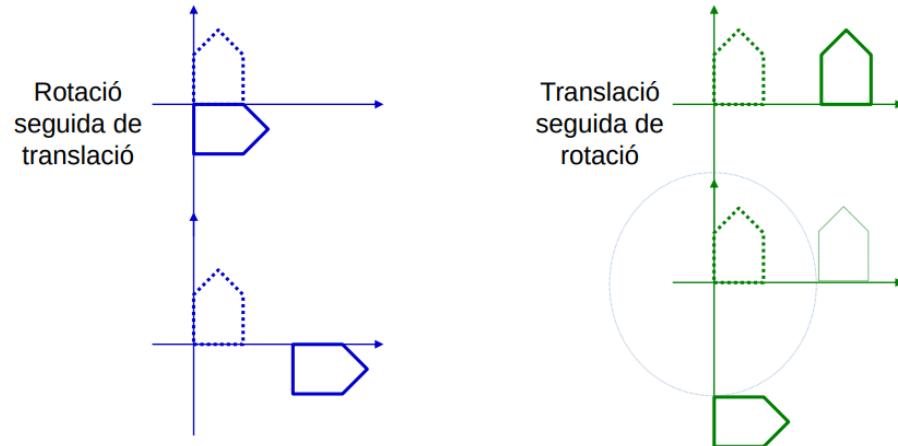
$$\mathbf{P}'' = T(3,0,0) \cdot (R_z(-90^\circ) \cdot \mathbf{P}) = (T(3,0,0) \cdot R_z(-90^\circ)) \cdot \mathbf{P} = \mathbf{M} \cdot \mathbf{P}$$

## Composició de Transformacions

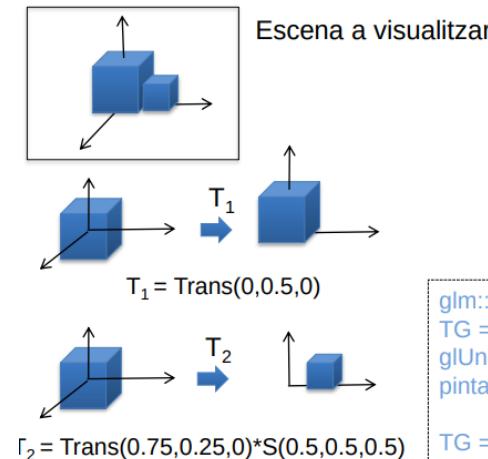
$$\mathbf{T}(3,0) \cdot \mathbf{R}(-90^\circ) \neq \mathbf{R}(-90^\circ) \cdot \mathbf{T}(3,0)$$

②      ①      ②      ①

- Multiplicació de matrius no és commutativa



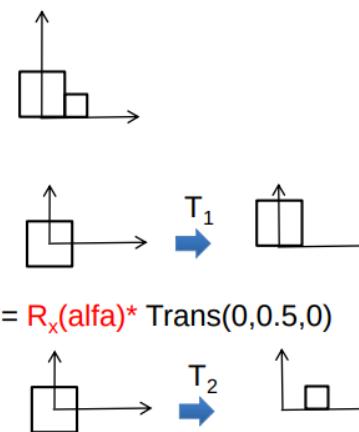
### Exemple simple de TG (1)



Exemple codi per pintar en CPU

Com faríeu per girar tota l'escena alfa graus respecte l'eix x?

### Exemple simple (2): Girar escena repetit eix X alfa graus



```
AUX = Rotate (alfa, (1,0,0));
TG = AUX * Translate(0, 0.5, 0);
modelMatrix (TG);
pinta_cub();
```

```
TG = AUX * Translate(0.75, 0.25, 0);
TG = TG * Scale(0.5,0.5,0.5);
modelMatrix (TG);
pinta_cub();
```

Sobre els tres tipus de transformacions:

- Translació:  $\text{Translate}(tx, ty, tz)$ .
- Escalat:  $\text{Scale}(sx, sy, sz)$ . Normalment  $sy = sz$  per no deformar.
- Rotació: a teoria:  $\text{Rotatex}(\varphi_x)$ ,  $\text{Rotatey}(\varphi_y)$ ,  $\text{Rotatez}(\varphi_z)$ , tot i que també es fa servir  $\text{Rotate}(\varphi, (x, y, z))$ , on  $(x, y, z) = (0, 0, 0)$  excepte un dels tres que és 1.
- Si volem fer translació + rotació, la matriu a aplicar és  $R \cdot T$ . Importa l'ordre!

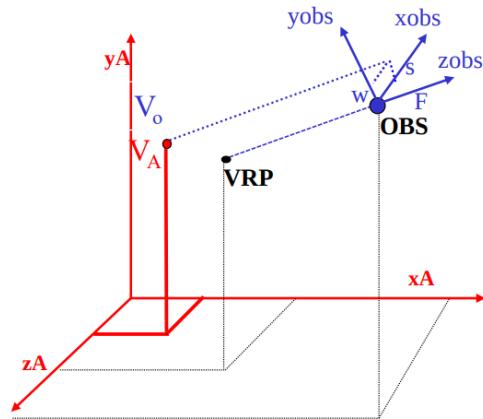
Sobre rotacions i escalats:

- Un cop estem en l'origen (hem fet Translació( $-t$ ), on  $t$  és el centre/base de la capsella mínima contenidora), podem aplicar rotació + escalat o escalat + rotació, no importa!

```
glm::mat4 TG = glm::mat4(1.f);
TG = glm::translate (TG, glm::vec3(0,0,0.5));
glUniformMatrix4fv (transLoc, 1, GL_FALSE, &TG[0][0]);
pinta_cub();

TG = glm::mat4(1.f);
TG = glm::translate (TG, glm::vec3(0.75,0.25,0));
TG = glm::scale(TG, glm::vec3(0.5,0.5,0.5));
glUniformMatrix4fv (transLoc, 1, GL_FALSE, &TG[0][0]);
pinta_cub();
```

## Pas a SCO amb VRP, OBS i Up: Càcul View Matrix



```
VM = lookAt(OBS, VRP, Up);
viewMatrix(VM);
```

$$VM = \begin{bmatrix} s.x & s.y & s.z & 0 \\ w.x & w.y & w.z & 0 \\ F.x & F.y & F.z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -OBS.x \\ 0 & 1 & 0 & -OBS.y \\ 0 & 0 & 1 & -OBS.z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$F = OBS - VRP = (F.x, F.y, F.z) \quad F = F / \|F\|$$

$$s = Up \times F \quad s = s / \|s\|$$

$$w = F \times s$$

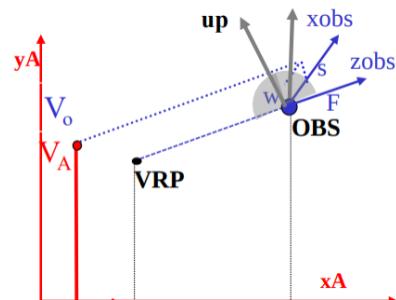
viewMatrix

$$\downarrow$$

View Transform

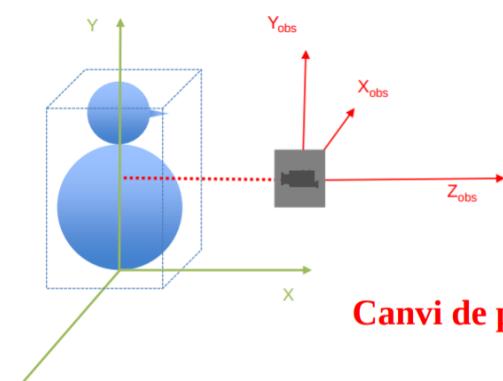
$$V_o = VM * V_A$$

$$V_o = (x_o, y_o, z_o, 1)$$



**OBS** = Observador  
**VRP** = View Reference Point  
**up** = View Up Vector

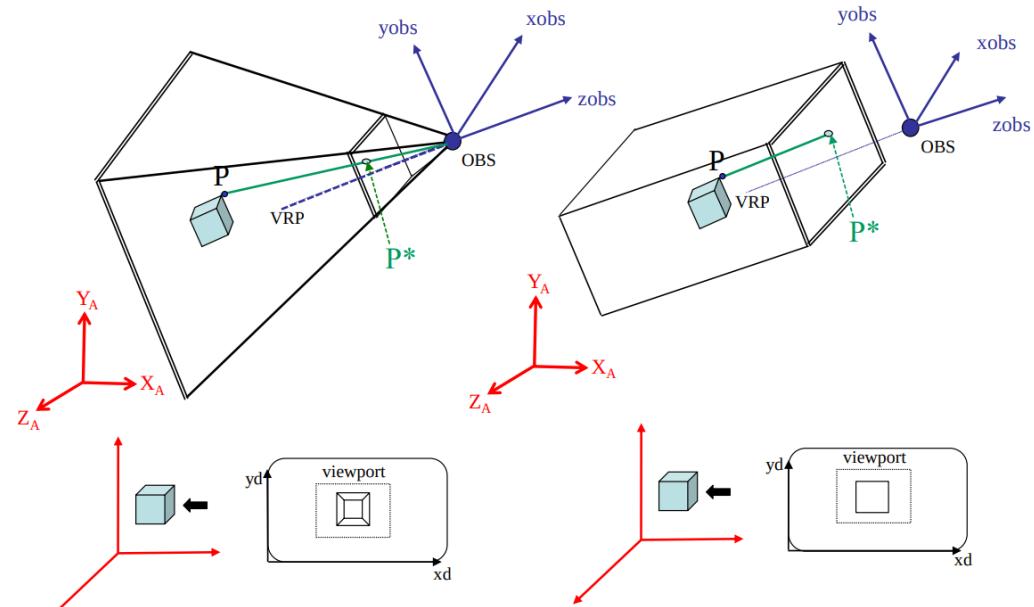
**up** "indica" la direcció de l'eix vertical de la Càmera



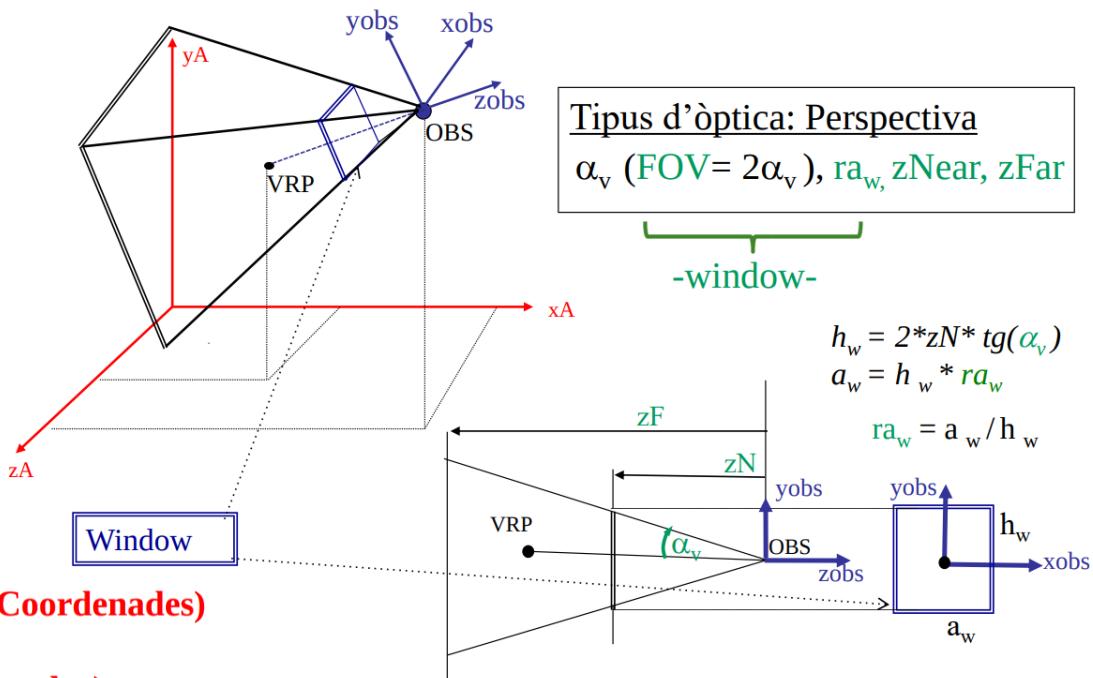
**Canvi de punt de vista (canvi de Sistema de Coordenades)**

**Pas de SCA (Aplicació) a SCO (Observador)**

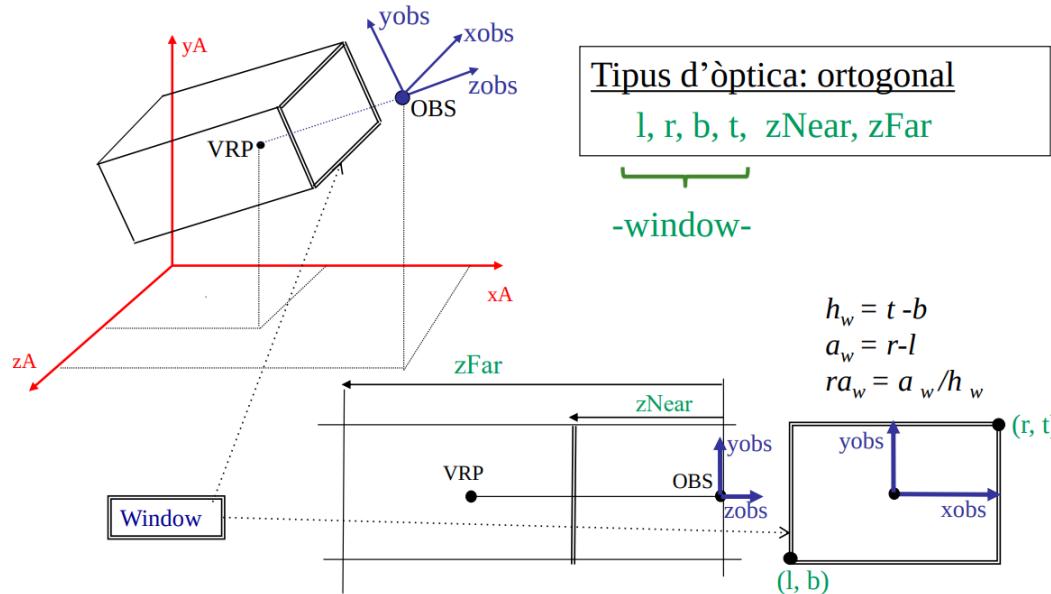
## Projecció: Perspectiva o Ortogonal



### Optica perspectiva: paràmetres



# Òptica ortogonal: paràmetres



## Càcul matriu Project Matrix

$$PM = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & d \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

a=2/(r-l) b=2/(t-b)  
c=2/(zf-zn)  
d=(zn+zf)/(zf-zn)

$$V_o = (x, y, z, 1)_o \xrightarrow{\text{ProjectMatrix}} V_c = PM * V_o$$

### Òptica Ortogonal

$V_c = (x_c, y_c, z_c, w_c)$  on  $w_c = 1$   
 $PM = \text{ortho}(l, r, b, t, zN, ZF);$   
 $\text{projectMatrix}(PM);$

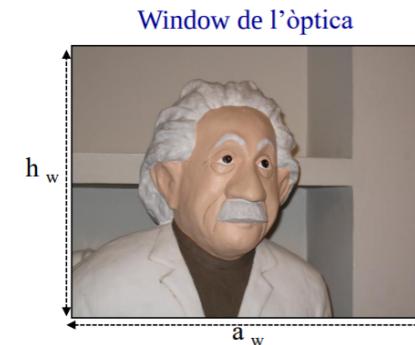
### Òptica Perspectiva

$V_c = (x_c, y_c, z_c, w_c)$  on  $w_c = -z_0$   
 $PM = \text{perspective}(FOV, ra, zN, ZF);$   
 $\text{projectMatrix}(PM);$

$$PM = \begin{pmatrix} 1/r * a & 0 & 0 & 0 \\ 0 & 1/a & 0 & 0 \\ 0 & 0 & c & d \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

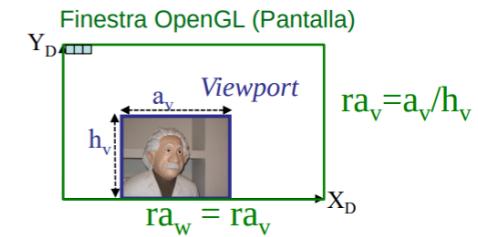
a = tg(FOV/2)  
c = (zf+zn)/(zn-zf)  
d = 2\*zn\*zf/(zn-zf)

# Sobre la relació d'aspecte del window i del viewport

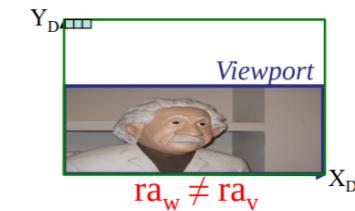


Per a no tenir deformació en la imatge

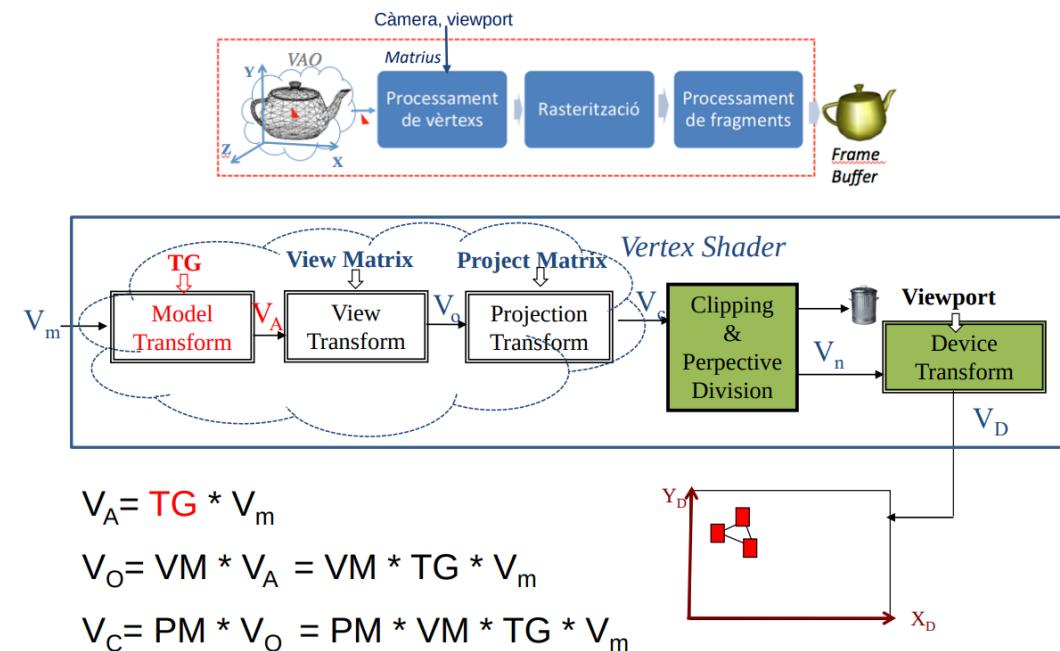
$$ra_w = ra_v$$



glViewport (ox, oy, av, hv)



## Vertex Shader – afegim TG



# Programació bàsica del vertex shader

#version 330 core

```
in vec3 vertex;
uniform mat4 PM;
uniform mat4 VM;
uniform mat4 TG;

void main() {
    gl_Position = PM*VM*TG*vec4 (vertex, 1.0);
}
```

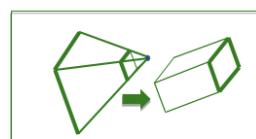
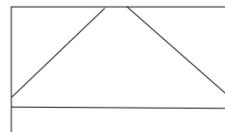
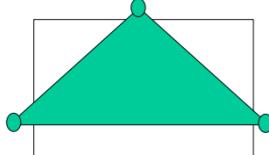
## Vertex Shader

## Clipping

Condició per a que un Vèrtex sigui interior al volum de visió:

$$\begin{aligned} -w_c &\leq x_c \leq w_c \\ -w_c &\leq y_c \leq w_c \\ -w_c &\leq z_c \leq w_c \end{aligned}$$

$V_c = (x_c, y_c, z_c, w_c)$  on  $w_c=1$  en ortogonal  
 $V_c = (x_c, y_c, z_c, w_c)$  on  $w_c=-z_o$  en perspectiva



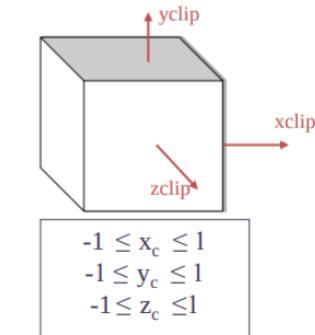
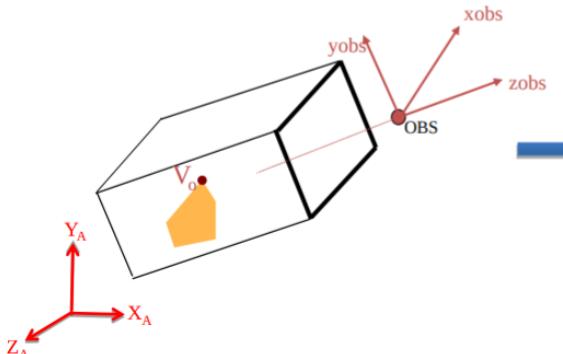
$$V^* = V_c / w_c \rightarrow V_n$$

$$\begin{aligned} -1 \leq x_n &\leq 1 \\ -1 \leq y_n &\leq 1 \\ -1 \leq z_n &\leq 1 \end{aligned}$$

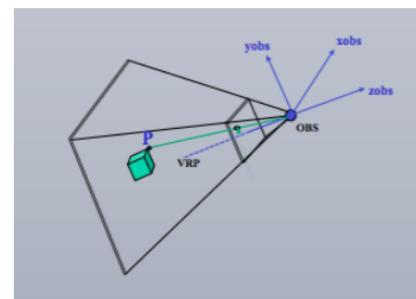
## Projecció: Ortogonal

$$PM = \begin{pmatrix} a & 0 & 0 & e \\ 0 & b & 0 & f \\ 0 & 0 & c & d \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{aligned} a &= 2/(r-l) & e &= (r+l)/(r-l) \\ b &= 2/(t-b) & f &= (t+b)/(t-b) \\ c &= 2/(zf-zn) & d &= (zn+zf)/(zf-zn) \end{aligned}$$

$$V_c = (x_c, y_c, z_c, w_c) \text{ on } w_c=1$$

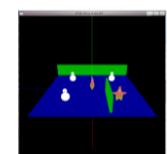


## Projecció: Perspectiva



$$\begin{aligned} \text{Vèrtex projectat:} \\ V^* &= V_c / w_c = -V_d / z_0 \\ x^* &= -x_c / z_0 \quad y^* = -y_c / z_0 \quad z^* = -z_c / z_0 \end{aligned}$$

Inversament proporcional a distància a observador 😊



$$PM = \begin{pmatrix} 1/r_a * a & 0 & 0 & 0 \\ 0 & 1/a & 0 & 0 \\ 0 & 0 & c & d \\ 0 & 0 & -1 & 0 \end{pmatrix} \quad \begin{aligned} a &= \tan(\text{FOV}/2) \\ c &= (zf+zn)/(zn-zf) \\ d &= 2*zn * zf / (zn-zf) \end{aligned}$$

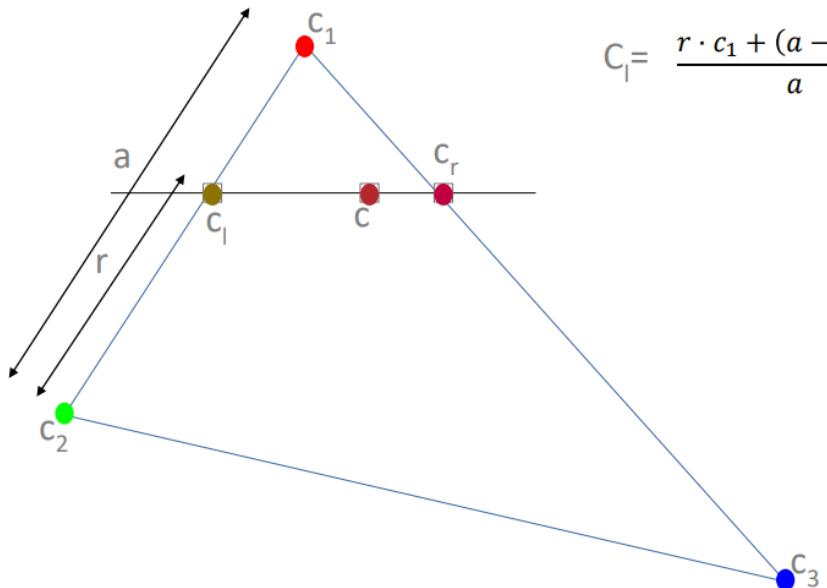
$$V_c = (x_c, y_c, z_c, w_c) \text{ on } w_c=-z_0$$

$$\begin{aligned} -w_c &\leq x_c \leq w_c \\ -w_c &\leq y_c \leq w_c \\ -w_c &\leq z_c \leq w_c \end{aligned}$$

# Rasterització: interpolació



$$C_l = \frac{r \cdot c_1 + (a - r) \cdot c_2}{a}$$



## Processament de fragments: El fragment Shader

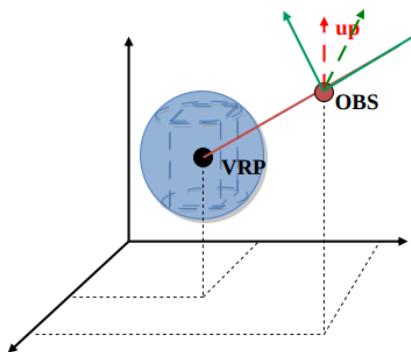
### Fragment Shader

```
#version 330 core
in ...
out vec4 FragColor;
void main() {
    FragColor = vec4(0, 0, 0, 1);
}
```

Sobre càmera perspectiva:

- El volum de visió sempre és respecte l'OBS!!!
- Recordem que, en perspectiva, tenim l'angle d'obertura  $\alpha$  (la meitat del FOV),  $zN$ ,  $zF$  i relació d'aspecte del window ( $raw$ ).
- El càcul de l'altura del window és  $aw = 2 zN \tan(\alpha)$ , ja que  $\tan(a) = (hw/2) / zN$ .
- L'amplada del window és  $raw \cdot aw$ .
- Per no tenir deformació cal  $raw = rav$  (window és la meva imatge, viewport és el lloc de la pantalla que tinc per aquesta imatge).
- Perspective( $FOV, raw, zN, zF$ ); projectMatrix( $PM$ );
- Quan multipliquem per la project matrix, tenim vèrtexs en coordenades de clipping. Per què això simplifica els càlculs? Perquè només hem de mirar, donat un  $VC = (xc, yc, zc, wc)$  que  $xc$  estigui entre  $-wc$  i  $+wc$ ,  $yc$  entre  $-wc$  i  $+wc$ , etc. Notem que  $wc$  és l'oposat a la coordenada  $z$  del vèrtex en coordenades d'observador (just abans de multiplicar per la Project Matrix!).
- Després d'aquest pas, es divideix el vector per  $wc$ , de manera que, un cop descartat els vèrtexs fora del rang de visió, tenim  $Vn = (xc/wc, yc/wc, zc/wc, 1)$ . Això fa més grans els vèrtexs a prop de l'OBS i més petits si estan més allunyats: efecte de les vies del tren.
- El següent pas és el Device transform: simplement és la regla de tres de transformació del meu window al meu viewport: la cantonada de dalt a l'esquerra es correspon amb la cantonada de dalt a l'esquerra, etc. (sol ser en el rang  $[-1, 1]$  a  $[0, 1]$ ).

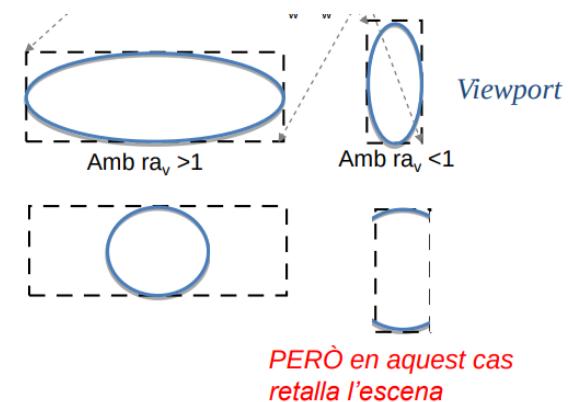
## Càmera 3a persona (1): Inicialització posicionament amb OBS, VRP, up



- Centrat => **VRP=CentreEscena**
- Per assegurar que l'escena es veu sense retallar des d'una posició arbitrària CAL que **OBS** sempre fora capsà mínima contenidora; per assegurar-ho CAL que **OBS** fora de l'esfera englobant de la capsà => distància "d" de l'**OBS** a **VRP** superior a R esfera.
  - CapsaMinCont=(xmin,ymin,zmin,xmax,ymax,zmax)
  - CentreEscena=Centre(CapsaMinCont) =  $((xmax+xmin)/2,(ymax+ymin)/2,(zmax+zmin)/2)$
  - $R=\text{dist}((xmin,ymin,zmin),(xmax,ymax,zmax))/2$
  - $d>R$ ; per exemple  $d=2R$
  - **OBS=VRP+ d\*v;**  $v$  normalitzat en qualsevol direcció;  
per exemple  $v=(1,1,1)/\|(1,1,1)\|$
  - **up** qualsevol que no sigui paral·lel a  $v$ ; si volem escena vertical (eix Y es vegi vertical)  $up=(0,1,0)$

$$\begin{aligned} ZN &= d-R; \quad ZF=d+R \\ \alpha_v &= \arcsin(R/d) \\ \Rightarrow \text{FOV} &= 2\alpha_v \\ ra_w &= a_w/h_w = 1 \end{aligned}$$

Per a què **no hi hagi deformació**, cal modificar  $ra_w$  i forçar una

$$ra_w^* = ra_v$$


- Si  $ra_v > 1$  ( $>$  que la  $ra_w$  mínima requerida que és 1) => No es retalla l'escena al fer  $ra_w^* = ra_v$  no cal modificar  $\alpha_v$  (FOV)

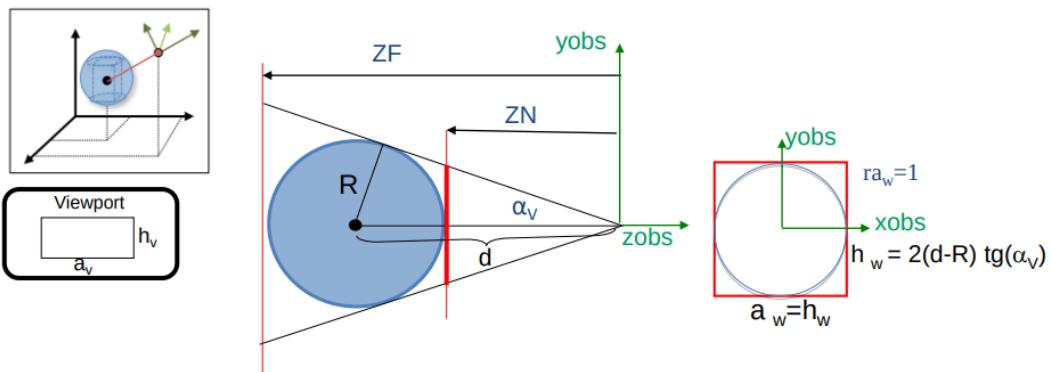


- Si  $ra_v < 1$  ( $<$  que la  $ra_w$  mínima requerida que és 1) => al fer  $ra_w^* = ra_v$  es retalla escena



Cal incrementar l'angle d'obertura  
 $\text{FOV}=2\alpha_v^*$  on  
 $\alpha_v^* = \arctg(\tan(\alpha_v)/ra_v)$

## Càmera en 3a persona (2): tota l'escena, sense deformar i òptica perspectiva



- Si tota l'esfera englobant està dins la profunditat del camp de visió, no retallem l'escena.

Per tant,  $ZN \in [0, d-R]$     $ZF \in [d+R, \dots]$

$ZN = d-R; \quad ZF = d+R$  per aprofitar la precisió en profunditat

- Per a aprofitar al màxim la pantalla, el viewport, el window de la càmera s'ha d'ajustar per veure tota l'escena; una aproximació és ajustar el window per veure l'esfera englobant.

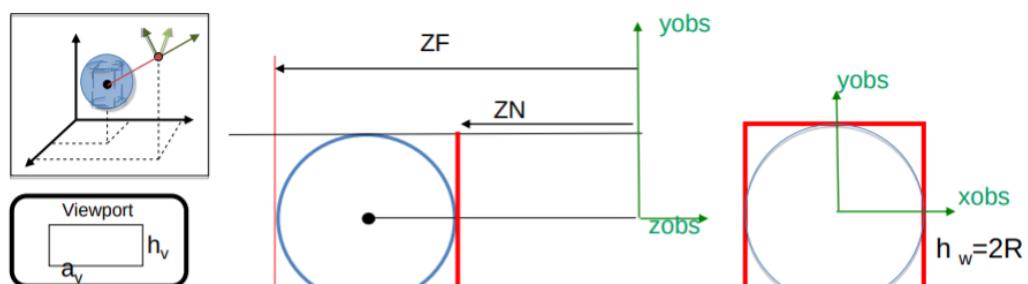
-  $R = d \sin(\alpha_v)$  ;  $\alpha_v = \arcsin(R/d) \Rightarrow \text{FOV}=2\alpha_v$

- com el window està situat en ZN,  $\alpha_v$  determina que la seva alçada sigui:

$$h_w = 2(d-R) \tan(\alpha_v)$$

- $ra_w = a_w/h_w = 1$  ( $\alpha_H$  hauria de ser igual a  $\alpha_v$  per assegurar que esfera no resulta retallada)

## Càmera 3a persona (5): tota l'escena, sense deformar i òptica ortogonal



- **ZN i ZF** mateix raonament que en càmera perspectiva.

- **Window mínim requerit (centrat)= (-R,R,-R,R)** => una  $ra_w = 1$  (per què?)

- Si  $ra_w \neq ra_v \Rightarrow$  deformació (per què?)

- Si  $ra_v > 1 \Rightarrow$  cal incrementar la  $ra_w \Rightarrow$  modificar window com  $ra_w = a_w/h_w \Rightarrow$  podem incrementar  $a_w$  o decrementar  $h_w$  (és retallaria esfera!!)

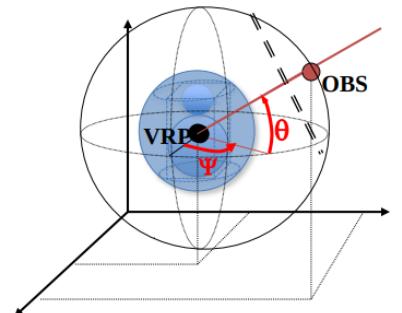
Per tant, modifiquem  $a_w$ :

$$a_w^* = ra_v * h_w = ra_v * 2R$$

$$\text{window} = (-R \ ra_v, R \ ra_v, -R, R)$$

- Raonament similar per recalcular window quan  $ra_v < 1$

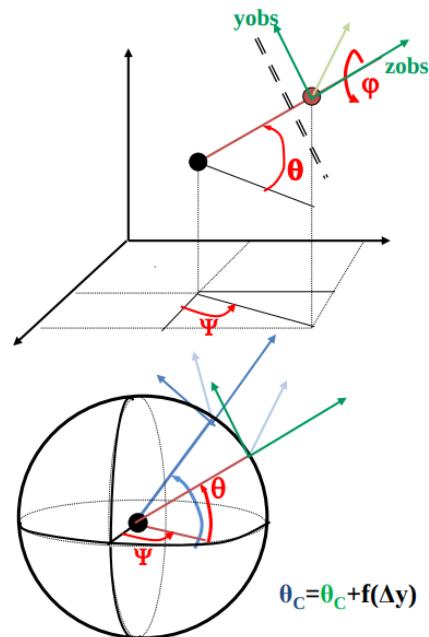
## Moure la Càmera per inspeccionar l'escena



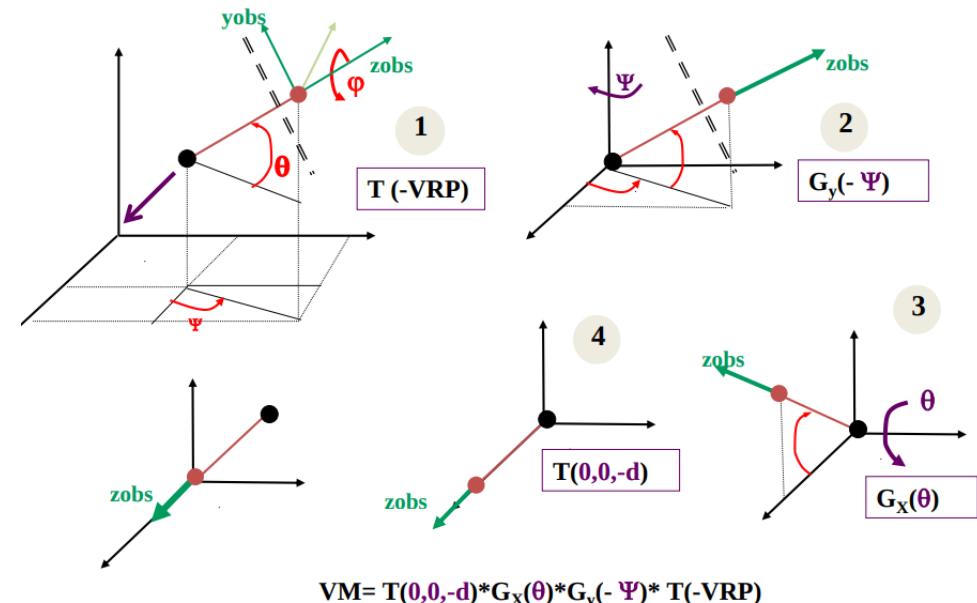
- Els angles (d'Euler) determinen la posició d'un punt en l'esfera
- Des de la interfície d'usuari desplaçem el cursor dreta/esquerra ( $\Psi$ ) i pujar/baixar ( $\theta$ ); per moure OBS sobre l'esfera
- No cal canviar l'òptica

### Com calculem OBS, VRP, up?

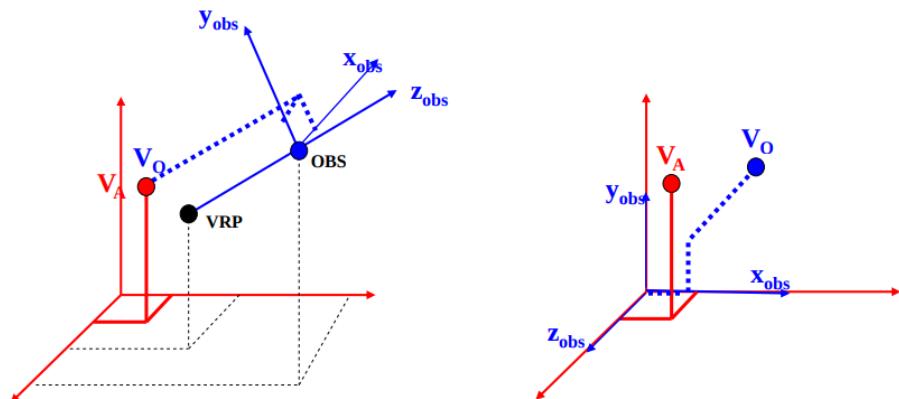
$VM = \text{lookAt}(\text{OBS}, \text{VRP}, \text{up});$   
 $\text{viewMatrix}(VM);$



## Càcul VM directe a partir d'angles Euler, VRP i d (3)



## Càcul viewMatrix directe a partir d'angles Euler, VRP i d (1)



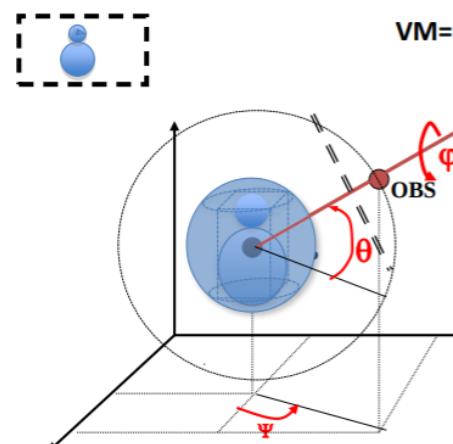
RECORDEU: La  $\text{viewMatrix}$  serveix per tenir la posició dels vèrtexs respecte del SCO

$$V_o = VM * V_A$$

Es pot calcular la VM:

- Pensant que movem la càmera (OBS, VRP, Up)
- Pensant que tenim una càmera fixa al SCA i ubiquem tota l'escena respecte d'ella → realitzar una mateixa  $TG_{VM}$  a tots els objectes. Si vèrtexs queden respecte a la càmera en la mateixa posició →  $TG_{VM}$  serà igual a la VM calculada amb OBS, VRP, Up

## Exemple: Posicionament amb angles Euler (4)



$$VM = T(0,0,-d) * G_z(-\phi) * G_x(\theta) * G_y(-\Psi) * T(-VRP)$$

$VM = \text{Translate}(0,0,-d)$   
 $VM = VM * \text{Rotate}(-\phi, 0, 0, 1)$   
 $VM = VM * \text{Rotate}(\theta, 1, 0, 0)$   
 $VM = VM * \text{Rotate}(-\Psi, 0, 1, 0)$   
 $VM = VM * \text{Translate}(-VRP.x, -VRP.y, -VRP.z)$   
 $\text{viewMatrix}(VM)$

Compta amb signes:

- Si s'ha calculat  $\psi$  positiu quan càmera gira cap a la dreta, serà un gir anti-horari respecte eix Y de la càmera, per tant, matemàticament positiu; com girem els objectes en sentit contrari, cal posar  $-\psi$  en el codi.
- Si s'ha calculat  $\theta$  positiu quan pugem la càmera, serà un gir horari; per tant, matemàticament un gir negatiu; com objecte girarà en sentit contrari (anti-horari), ja és correcte deixar signe positiu.

Sobre càmera en 3a persona + Euler:

- Tenim un OBS fora de la capsà mínima, un VRP definit al centre de la capsà i un up adient (normalment  $(0, 1, 0)$ ).
- Per rotar l'escena, pensem els moviments a fer amb la capsà per tenir la visió "per defecte" (OBS a l'origen mirant cap a Z negatives).
- Assumim angles positius cap a la dreta i cap a dalt.
- Els passos són: Translació(centre capsà), RotacióY( $-\psi$ ), RotacióX( $\theta$ ), Translació $(0, 0, -d)$ . Normalment  $d = 2 R$ .<sup>3</sup>

Càmera perspectiva sense retallar:

- Si  $rav > 1$ , no es retalla, no cal modificar el FOV. Només  $raw^* = rav$ .
- Si  $rav < 1$ , fem  $raw = rav$  i incrementem l'angle d'obertura amb  $FOV = 2 \alpha_V^*$ , on  $\alpha_V^* = \text{arctg}(\text{tg}(\alpha_V) / rav)$ .

Sobre òptica ortogonal:

- Paràmetres:  $(l, r, b, t), zN, zF$ .
- Única diferència respecte perspectiva: el  $wc$  és sempre 1 (si el punt en coordenes de clipping està a Z més pròximes o més llunyanes a l'observador no importa).

Càmera ortogonal sense retallar:

- Si  $rav > 1$  (massa ample), passem de  $(-R, R, -R, R)$  a  $(-rav \cdot R, rav \cdot R, -R, R)$ .
- Si  $rav < 1$  (massa alt), passem de  $(-R, R, -R, R)$  a  $(-R, R, -rav \cdot R, rav \cdot R)$ .



- Basat en estímuls R, G i B als cons i bastons.
- És una funció  $f(R, G, B)$ ; s'acota en el pla  $X + Y + Z = 1$

### 3. Color

Diferenciem:

	Síntesi additiva	Síntesi subtractiva
Mitjà	Llum	Paper
Color base	Negre	Blanc
Esquema de colors	RGB	CMY(K)
Esquema visual		
	(R, G, B)	(C, M, Y)
Blanc	(1, 1, 1)	(0, 0, 0)
Negre	(0, 0, 0)	(1, 1, 1)
Red	(1, 0, 0)	(0, 1, 1)
Green	(0, 1, 0)	(1, 0, 1)
Blue	(0, 0, 1)	(1, 1, 0)
Cyan	(0, 1, 1)	(1, 0, 0)
Magenta	(1, 0, 1)	(0, 1, 0)
Yellow	(1, 1, 0)	(0, 0, 1)
Conversió	$(A, B, C) \leftrightarrow (1 - A, 1 - B, 1 - C)$	

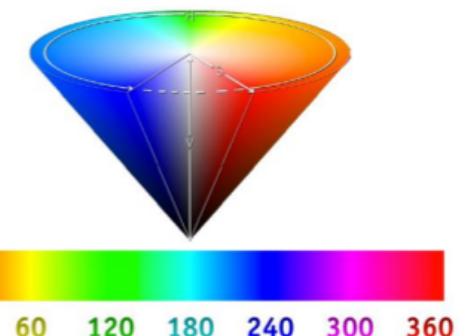
Sobre CMY(K):

- La component K (blacK) cal perquè les tintes C, M i Y solen tenir impureses i, juntes, no fan un negre pur a la vida real.

Model HSV:

- Hue (matiz) + Saturació + Valor.
- Hue és el color (roig / blau / ...).
- La saturació indica la quantitat de gris del color.
- El valor (o intensitat).

Model CIE:



## 4. Usabilitat i més coses

### Introducció a la HCI

HCI:

- Human-Computer Interface: com interaccionen humans i computadors.
- Va d'entendre tecnologies interactives i pràctiques dels humans.
- La HCI és principalment (però no només!!) usabilitat.

User Interfaces:

- User Interfaces: eines i mètodes que es fan servir per comunicar humans i sistemes.

Usabilitat:

- La usabilitat és (def. ISO): "habilitat en què un producte es pot fer servir per un grup específic d'usuaris (!!!) per dur a terme tasques específiques (!!!) de manera efectiva, eficient, i amb satisfacció en un entorn d'ús específic (!!!)".
- La usabilitat sempre es refereix a un grup concret d'usuaris i a un usuari concret d'una aplicació!!!!
- Per tant: usabilitat = eficàcia (es fa allò que volem) + eficiència (bon ús de recursos) + satisfacció.

User experience:

- La UX és quant a dispositiu: volem crear una bona sensació al fer servir el nostre dispositiu (!!!).

Interaction design:

- Com interactuem amb els dispositius (per exemple: mida de pantalles dels mòbils, teclat físic / virtual en pantalla, etc.).
- Regla del polze (imatge).



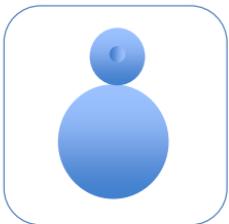
Més d'HCI:

- En escriptoris: pantalles grans, espai per tot, ratoli, teclat, resolució gran.
- En mòbils: pantalla menuda, pensar com ajustar contingut a l'espai reduït. Problemes amb les notificacions. Interacció amb dit/stylus. Sense teclat. Poca resolució i limitacions de software.
- En tablets: mida més gran, però hem de pensar encara com fer cabre les coses. Semblant als mòbils.

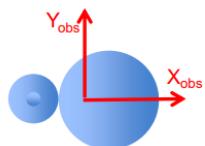
Sobre aplicacions:

- Diferenciem web apps i apps natives.
- Web apps: develop once & deploy everywhere, fàcil d'actualitzar, llenguatges fàcils; però, no és tan rica en UI, protocols ineficients i insegurs. Solen estar dissenyades per a grans pantalles amb ratolins.
- Aplicacions natives: UI més rica, molts controls, accessos a recursos locals fàcils i segurs; menys varietat de llenguatges i eines de programació, dissenyades per pantalles petites i amb control de touch. Però, no hi ha accés universal (Android vs iOS), difícil d'actualitzar, menys general que el development d'escriptori.

## Exemple 1 bis:

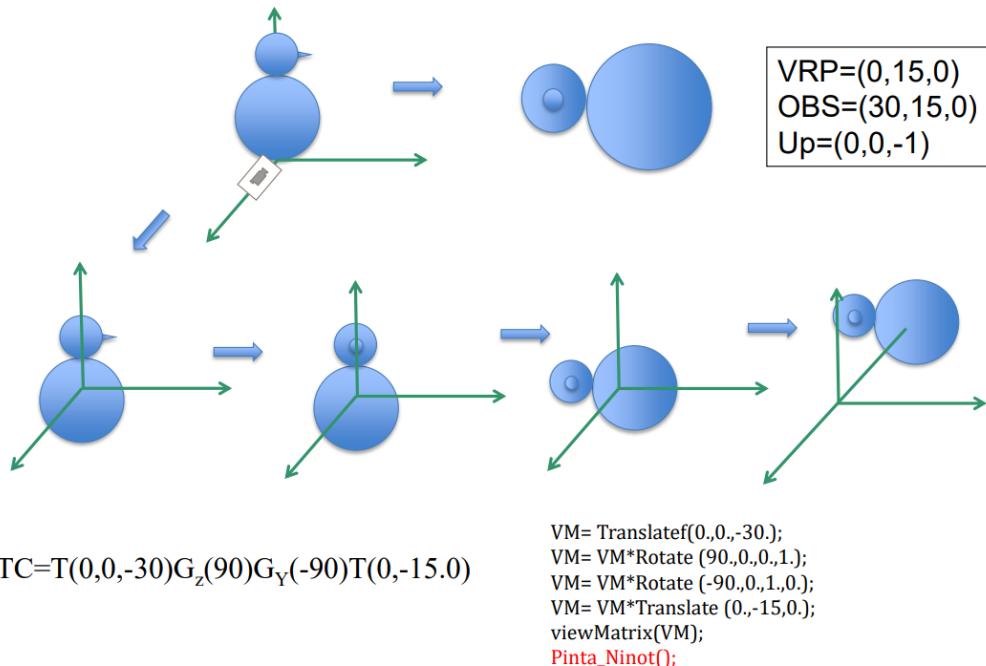


VRP=(0,15,0); OBS=(30,15,0), up=(0,1,0)

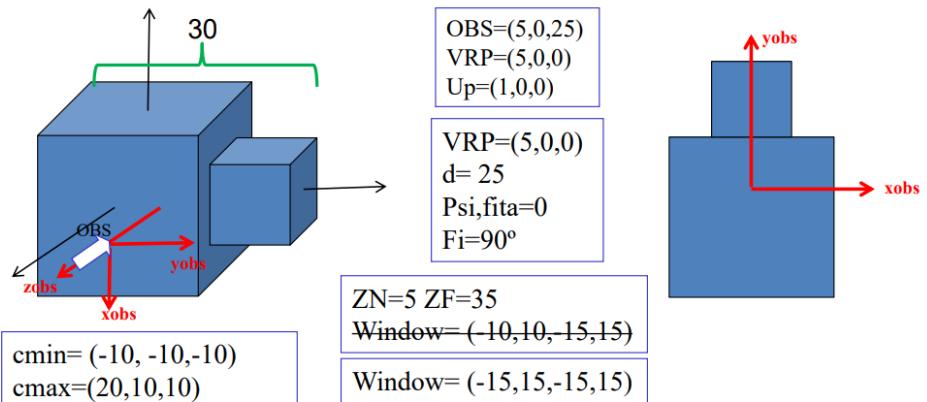


*Quins paràmetres si volem que quedí així?  
amb lookAt() i amb Euler*

## Solució Exemple 1 bis.



**Solució exemple 7:** Una escena està formada per dos cubes, un de costat 20 centrat al punt (0,0,0), i l'altre de costat 10 centrat al punt (15,0,0). Indiqueu TOTS els paràmetres d'una càmera **ortogonal/perspectiva** que permeti veure a la vista dos quadrats, un damunt de l'altre (el més gran a sota), de manera que ocupin el màxim de la vista (*viewport*). Cal que indiqueu la posició i orientació de la càmera especificant; a) VRP, OBS i up b) **angles Euler**. El *viewport* és quadrat.



**Solució exemple 2:** Tenim una escena amb un cub de costat 2 orientat amb els eixos i de manera que el seu vèrtex mínim està situat a l'origen de coordenades. La cara del cub que queda sobre el pla  $x=2$  és de color vermell, la cara que queda sobre el pla  $z=2$  és de color verd i la resta de cares són blaves.

Indica TOTS els paràmetres d'una càmera perspectiva que permeti veure completes a la vista només les cares vermella i verda. La relació d'aspecte del *viewport* (vista) és 2. Fes un dibuix indicant la imatge final que s'obtindria. **Posiciona la càmera amb angles d'Euler.**

