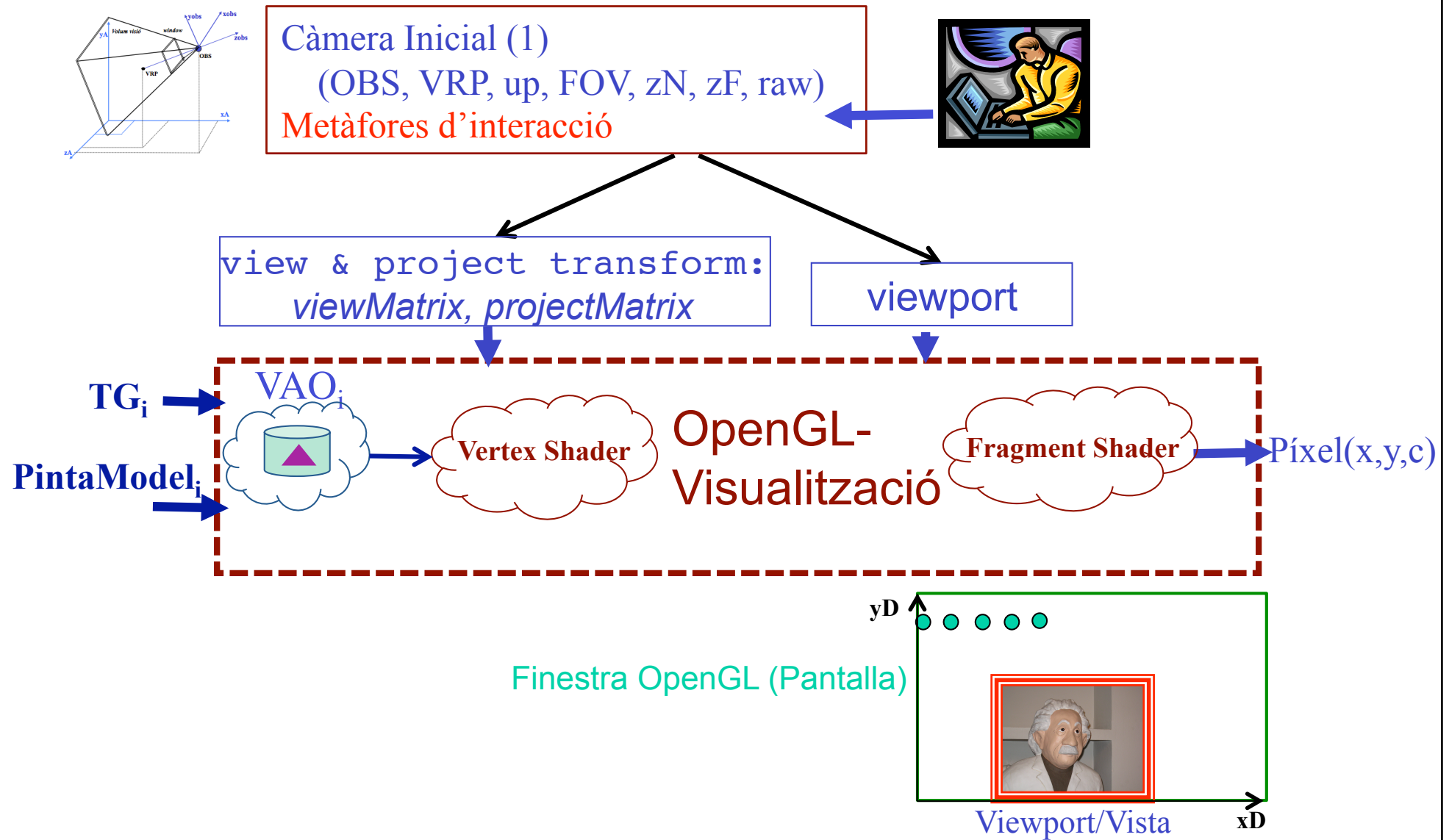


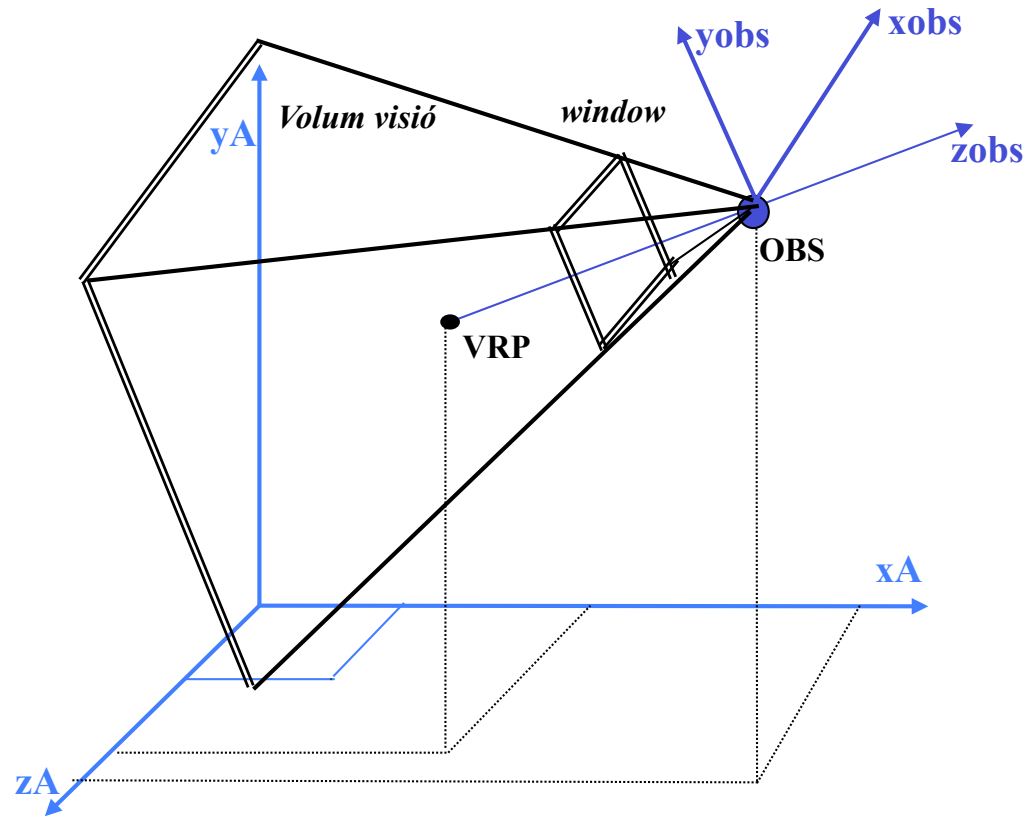
Classe 4: contingut

- Càmera (2):
 - Recordatori: PV, Matrius, definició de càmera + Exercicis
 - Exercici: càmera en 3ra persona
 - Moure Càmera i Angles Euler
- Alguns exercicis càmera

Càmera i procés de visualització



Càmera: OBS, VRP, up, zN, zF, FOV raw



Exercici: Quan s'inicialitza la càmera, en quin ordre cal indicar les transformacions de càmera i el viewport a OpenGL?

- a) No importa l'ordre en què s'indiquen.
- b) Transformació de posició + orientació, transformació de projecció, *viewport*.
- c) La transformació de projecció, transformació de posició + orientació, *viewport*.
- d) *Viewport*, transformació de projecció, transformació de posició + orientació.

```

/* CreateBuffers(); Crear VAO del model
(un cop)*/

...
/*IniCamera() calcular paràmetres càmera i
matrius cada cop que es
modifiquin */
//viewTransform()
VM = lookAt(OBS,VRP,UP);
viewMatrix(VM);
//projectTransform()
PM=perspective (FOV,ra,zN,ZF);
projectMatrix(PM);
//resize(...)
glViewport (0,0,w.h);
/*PaintGL(); cada cop que es requerix
refresc*/
/*per cada model: modelTransform()
Calcula TGi i passar a OpenGL*/
modelTransform_i(TG);
modelMatrix(TG);
Pinta_model(VAO);

```

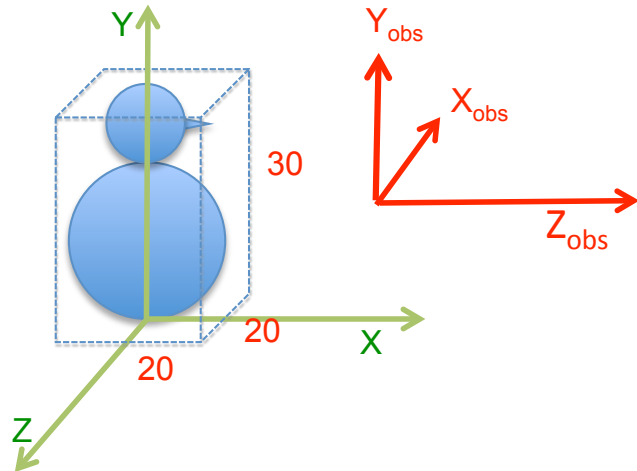
Vertex Shader

```

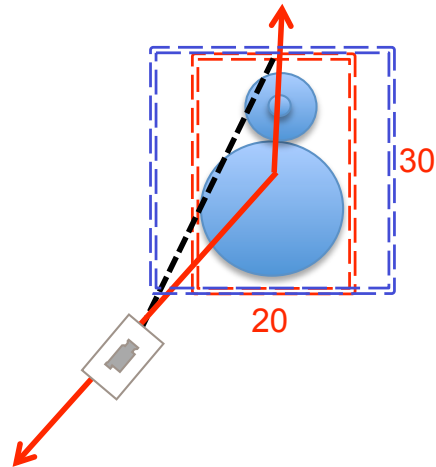
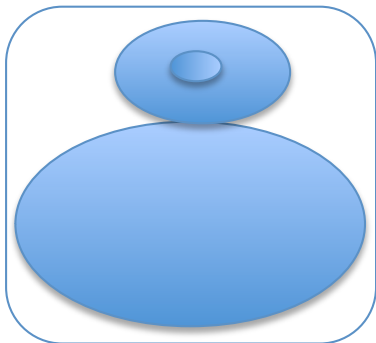
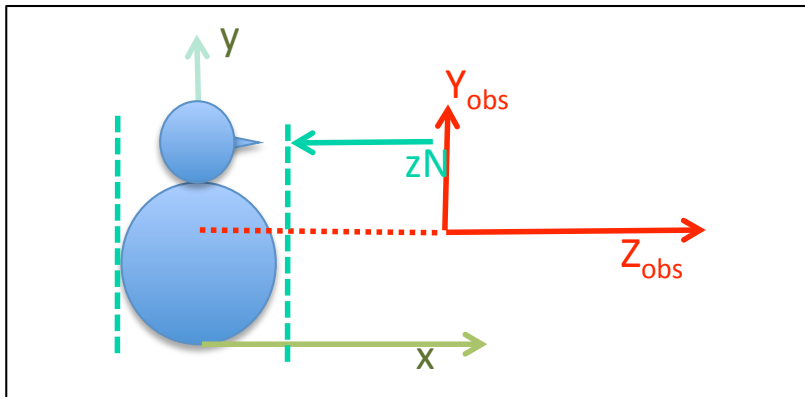
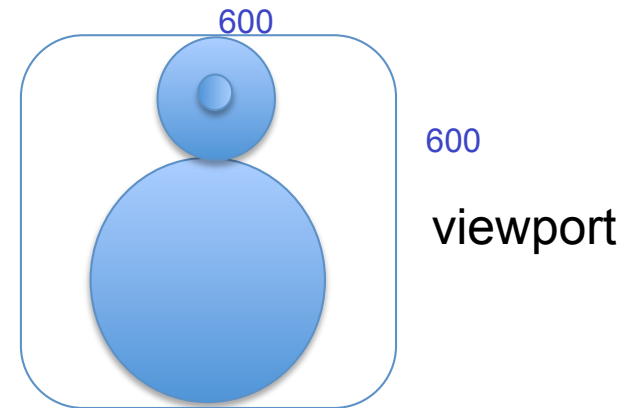
in vec3 vertex;
uniform mat4 TG, VM, PM;
void main ()
{
    gl_Position =
        PM*VM*TG*vec4(vertex,1.0);
}

```

Exemple 1: Òptica perspectiva



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



$zN = 20$; $zF = 40$

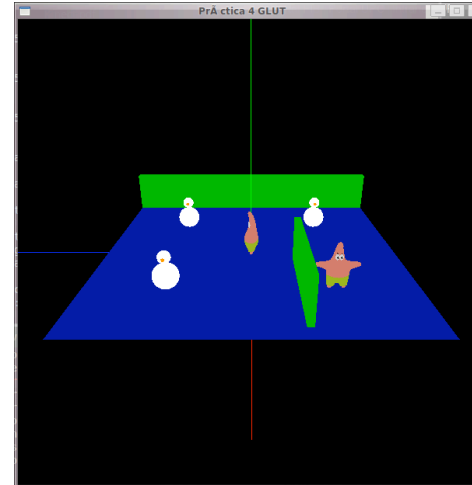
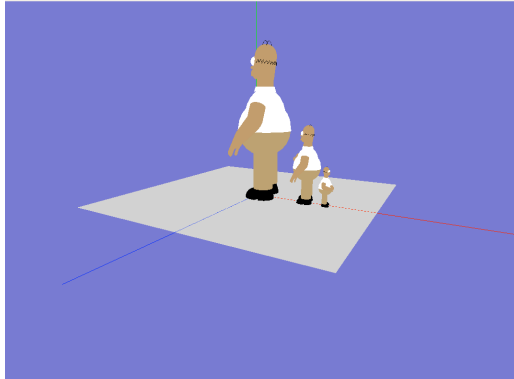
$\alpha = \arctg(15/20) \rightarrow \alpha = 36,8^\circ$

$ra_W = 20/30 = 0,66$

Com $ra_v = 1 \rightarrow$ deformació

Solució $ra_W = 1$

Càmera 3ra persona

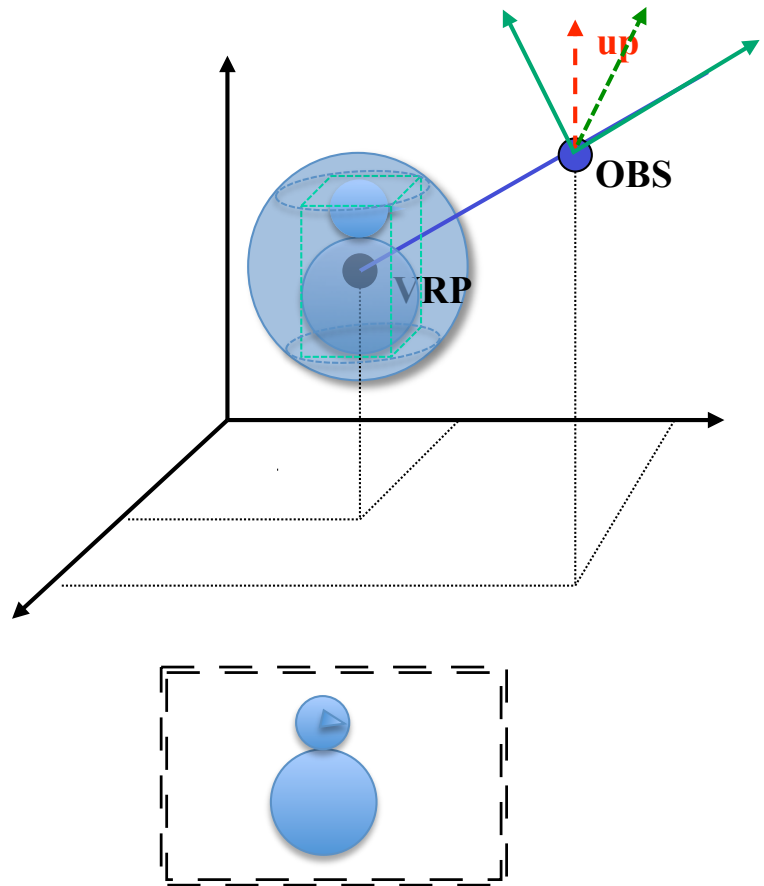


Quins paràmetres de posició, orientació i òptica per càmera en 3ra persona? →
imatge inclogui tota l'escena, ocupant el màxim del viewport.

Dada: capsa mínima contenidora d'escena

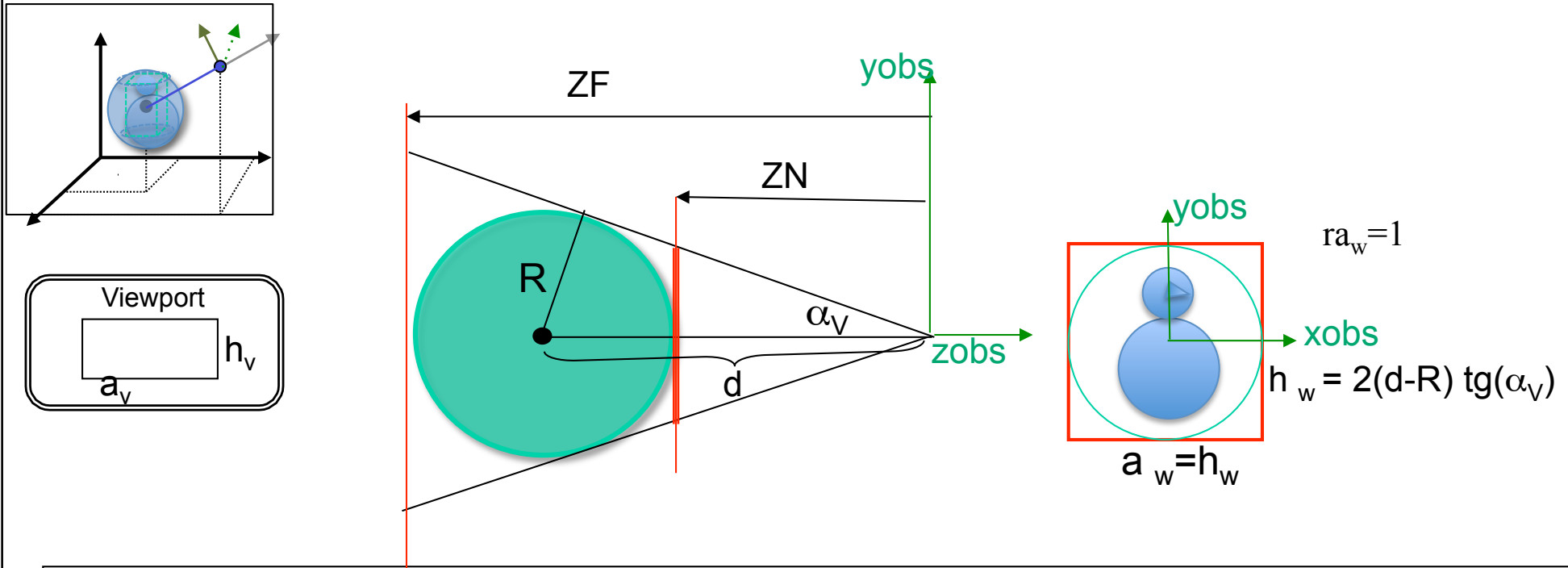
$(x_{\min}, y_{\min}, z_{\min}) - (x_{\max}, y_{\max}, z_{\max})$

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 capsa mínima contenidora; per assegurar-ho CAL que **OBS** fora de l'esfera englobant de la capsa => distància "d" de l'**OBS** a **VRP** superior a R esfera.
 - CapsaMinCont=(xmin,ymin,zmin,xmax,ymax,zmax)
 - CentreEscena=Centre(CapsaMinCont) =
 $((x_{max}+x_{min})/2, (y_{max}+y_{min})/2, (z_{max}+z_{min})/2)$
 - $R = \text{dist}((x_{min}, y_{min}, z_{min}), (x_{max}, y_{max}, z_{max}))/2$
 - $d > R$; per exemple $d = 2R$
 - **OBS**=**VRP**+ $d \cdot \mathbf{v}$; \mathbf{v} normalitzat en qualsevol direcció;
per exemple $\mathbf{v} = (1, 1, 1) / \|(1, 1, 1)\|$
- **up** qualsevol que no sigui paral·lel a \mathbf{v} ; si volem ninot vertical (eix Y es vegi vertical) **up**=(0,1,0)

Tota escena en la vista, sense deformar i càmera perspectiva



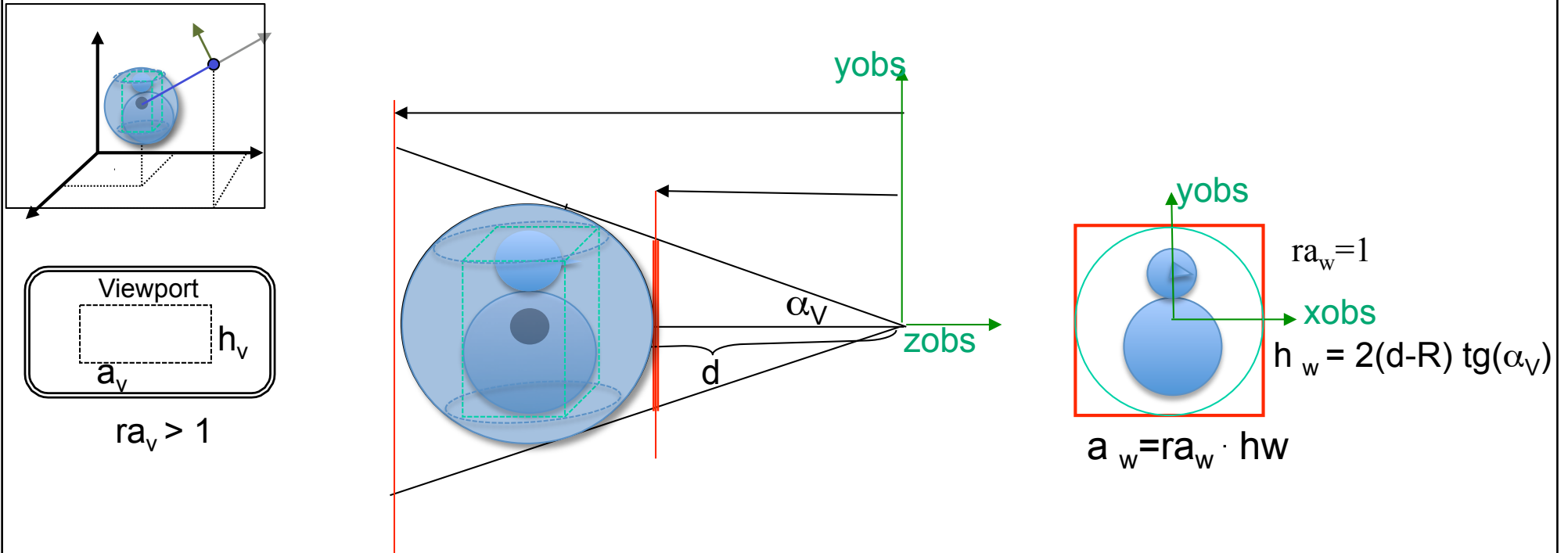
- Si tota l'esfera englobant està dins la profunditat del camp de visió, no retallem l'escena.
Per tant, $Z_N \in]0, d-R]$ $Z_F \in]d+R, \dots]$; per a aprofitar precisió profunditat: $Z_N = d-R$; $Z_F = d+R$
- Per a aprofitar al màxim la pantalla (de fet el viewport), el window de la càmera s'ha d'ajustar a l'escena; una aproximació és ajustar el volum de visió (piràmide) de la càmera a l'esfera englobant.
 - $R = d \sin(\alpha_V)$; $\alpha_V = \arcsin(R/d)$ $\Rightarrow FOV = 2 * \alpha_V$
 - com window està situat en Z_N , α_V determina que la seva alçada sigui: $h_w = 2(d-R) \tan(\alpha_V)$
- $ra_w = a_w/h_w = 1$ (perquè α_H hauria de ser igual a α_V per assegurar que esfera no retallada)

- **PERÒ** per a què no hi hagi deformació, cal que $ra_w = ra_v$, per tant, si no volem modificar el viewport cal forçar una

$$ra_w^* = ra_v$$

Pregunta: amb aquesta nova ra_w^* es retallarà l'esfera? (estarà tota l'esfera/escena dins del volum de visió?)

Tota escena en la vista, sense deformar i càmera perspectiva

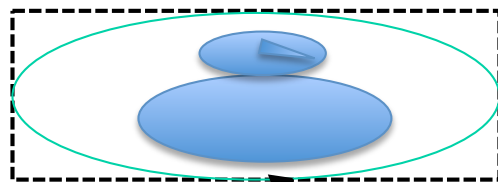


Si $ra_v > 1$ i fem **$ra_w^* = ra_v$** \Rightarrow si no modifiquem h_w ,

la nova $a_w^* > a_w$ mínima requerida,

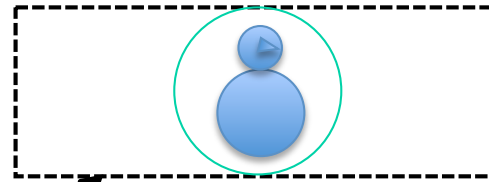
per tant, no es retalla

no cal modificar α_v (FOV)



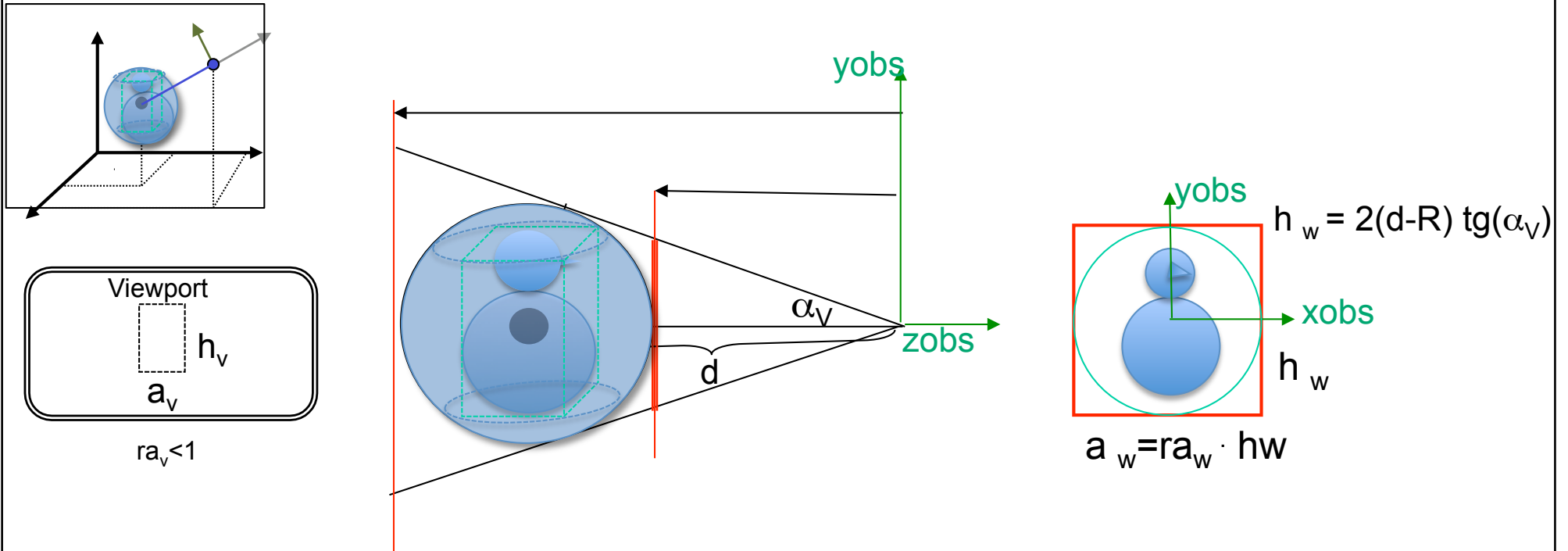
Amb $ra_w = 1$

Viewport
 $ra_v > 1$

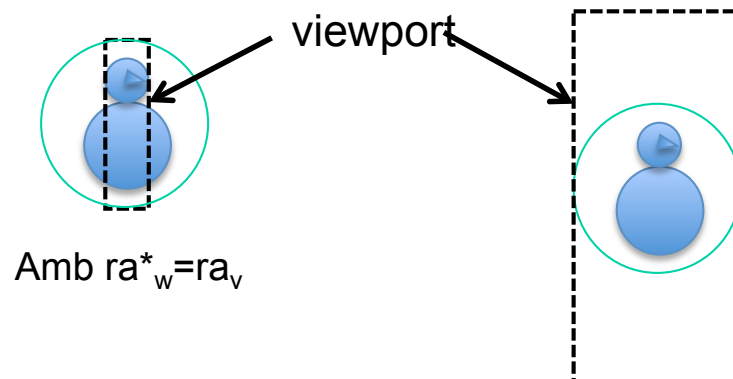


Amb $ra_w^* = ra_v$

Tota escena en la vista, sense deformar i càmera perspectiva



Si $ra_v < 1$ i només fem $ra_w^* = ra_v$, no hi haurà deformació però com $ra_w^* < ra_w$, implícitament estem $a_w^* < a_w$
 → retallarà esfera

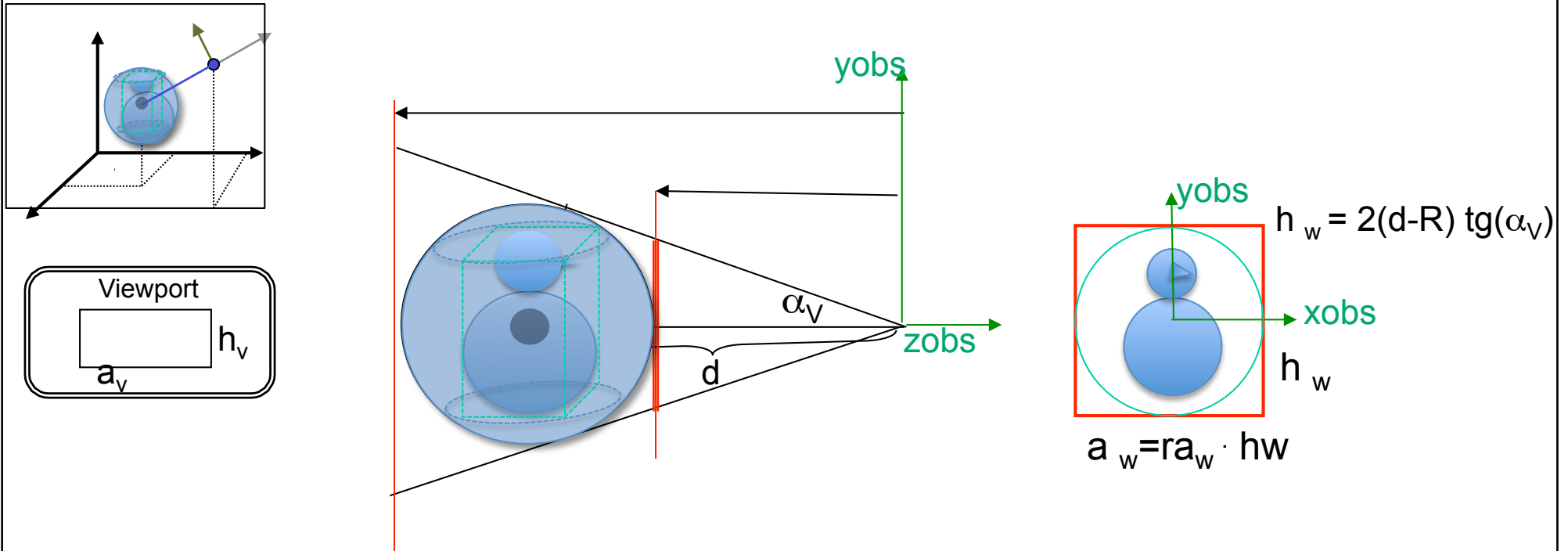


Per evitar-ho cal incrementar l'angle d'obertura, per incrementar proporcionalment l'amplada i englobar tota l'esfera. CAL $ra_w^* = ra_v$ i nou FOV

$$\text{FOV} = 2 \alpha_v^* \text{ on } \alpha_v^* = \arctg(\operatorname{tg}(\alpha_v) / ra_v)$$

- Sempre cal calcular el nou angle a partir de l'inicial (window quadrat). Penseu que passaria si no ho feu i modifiqueu interactivament el viewport (finestra gràfica) fent-ho >1 i <1 molts cops seguits.

Tota escena en la vista, sense deformar i càmera perspectiva



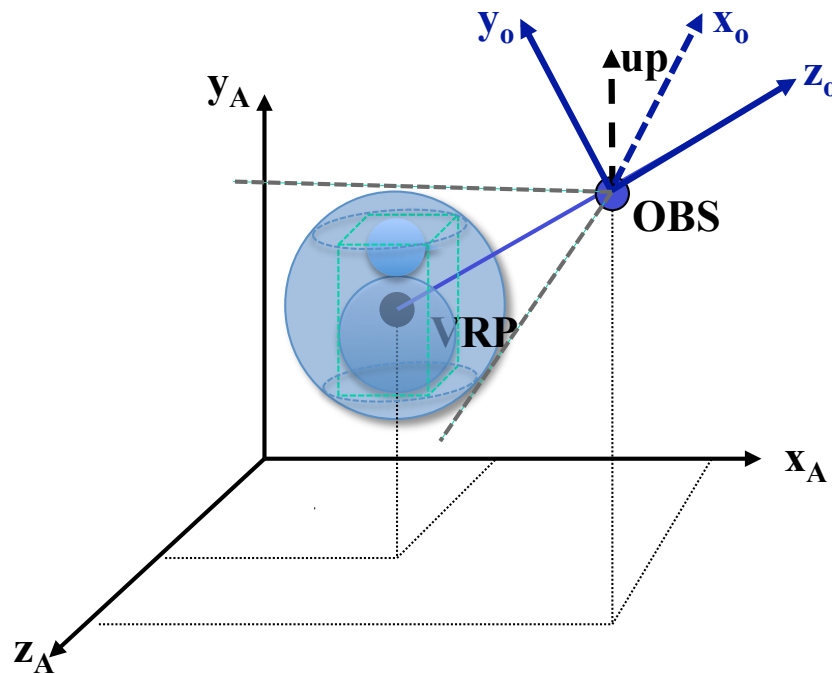
- Si $ra_v > 1$ ($>ra_w$ mínima requerida 1) \Rightarrow No es retalla, **no cal modificar α_v (FOV), només fer $ra_w^* = ra_v$**
Justificació: ra_w^* serà superior a 1; si no modifiquem l'angle FOV, h_w no canvia \Rightarrow
 $a_w^* = ra_w^* \cdot h_w$ i com $ra_w^* > ra_w \Rightarrow a_w^* > a_w$ i, per tant, serà més gran del necessari però es veurà tota l'esfera i quedarà espai pels laterals.
- Si $ra_v < 1$ ($<ra_w$ mínima requerida 1) \Rightarrow cal **fer $ra_w^* = ra_v$ i incrementar l'angle d'obertura i**
- **$FOV = 2 \alpha_v^*$ on $\alpha_v^* = \arctg(\operatorname{tg}(\alpha_v) / ra_v)$**

Justificació: com $a_w^* = ra_w^* \cdot h_w$, si no modifiquem angle, h_w no varia; com $ra_w^* < ra_w \Rightarrow a_w^* < a_w$ i l'esfera quedaria retallada (en horitzontal). Per tant, cal incrementar l'angle α_v (i, per tant, h_w^*) per a garantir una amplada del window igual a la mínima requerida.

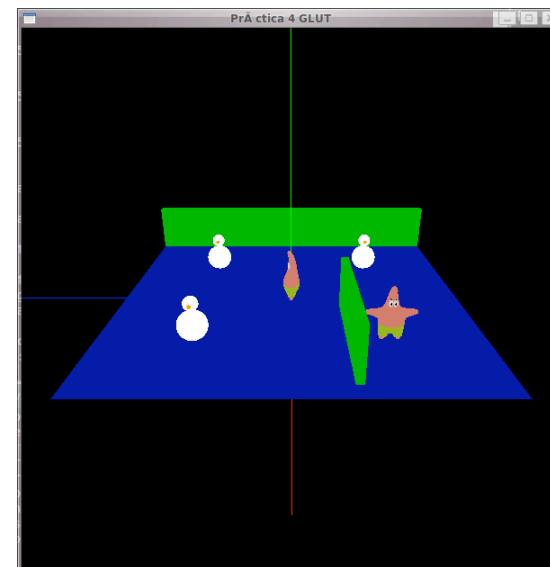
- Com $h_w^* = a_w / ra_v$ i per trigonometria $h_w^* = 2(d-R) \operatorname{tg}(\alpha_v^*)$, igualant les equacions
 $\alpha_v^* = \arctg(\operatorname{tg}(\alpha_v) / ra_v)$

Vist...

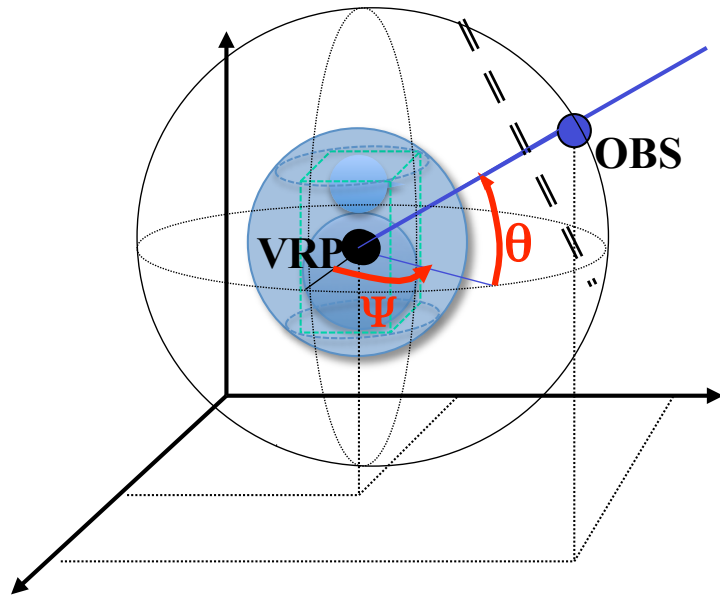
- Posicionament: OBS, VRP, up \rightarrow viewMatrix
- Òptica perspectiva: z_N , z_F , FOV, ra \rightarrow projectionMatrix
- Càmera en 3ra persona: posició inicial



**Com Moure la Càmera
per inspeccionar escena?**



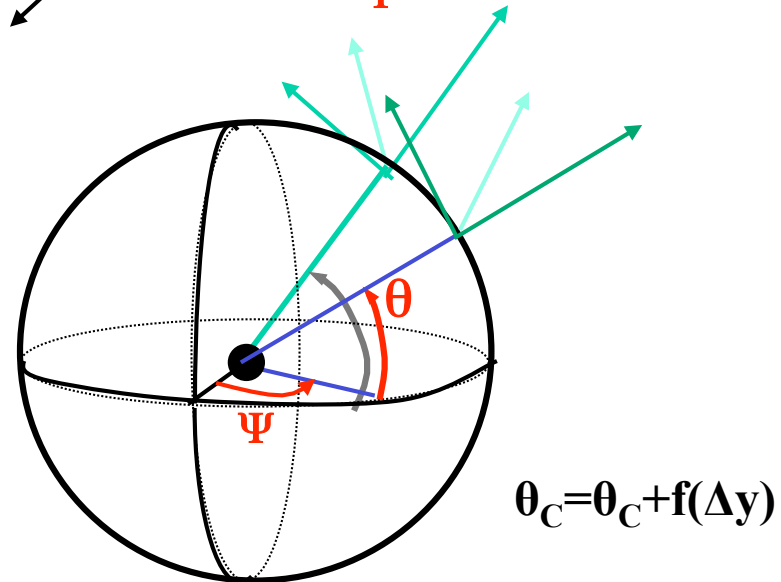
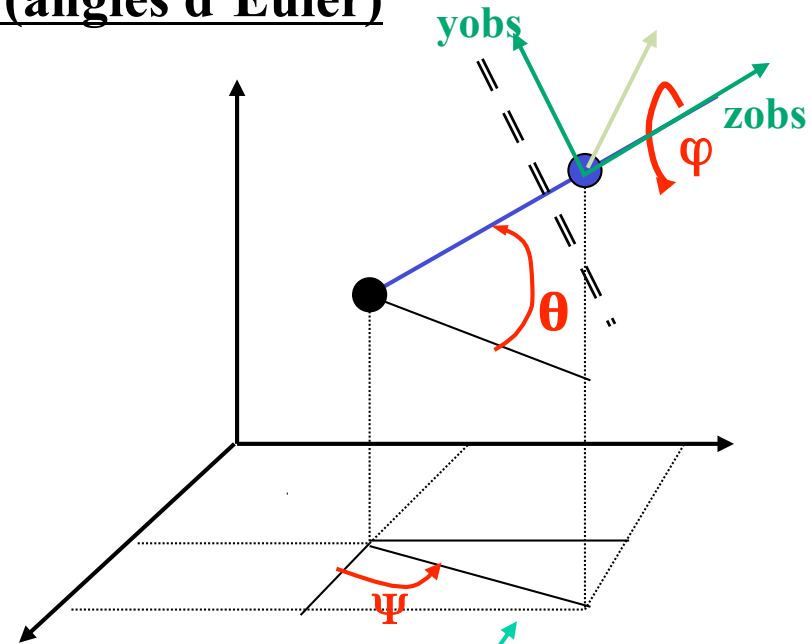
Moure la Càmera (angles d'Euler)



- Els angles (d'Euler) determinen la posició d'un punt en l'esfera
- Des de la interfície d'usuari desplacem el cursor dreta/esquerra (Ψ) i pujar/baixar (θ); per moure **OBS** sobre l'esfera

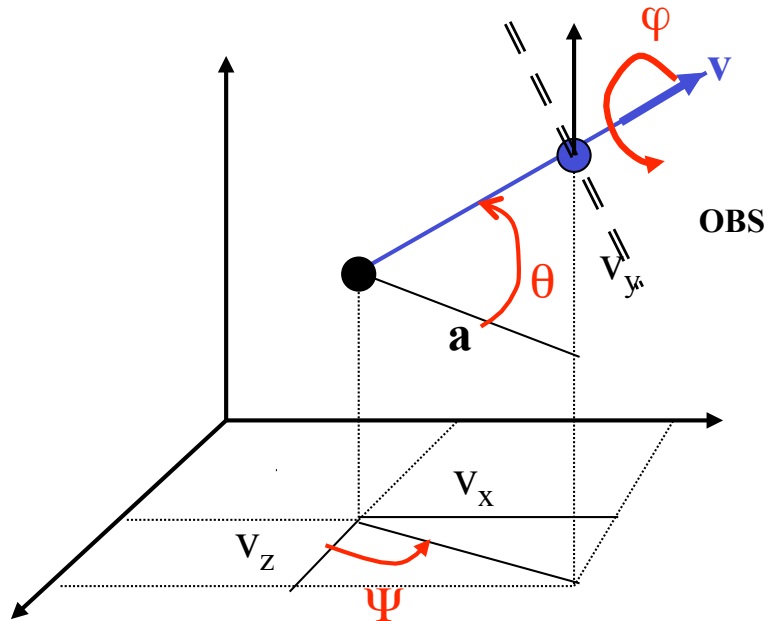
Com calculem **OBS**, **VRP**, **up**?

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



$$\theta_c = \theta_c + f(\Delta y)$$

Càlcul VRP, OBS a partir d'angles Euler



VRP = Punt d'enfoc

OBS = **VRP** + d **v**

d > **R** ; per exemple: d = 2R

$v_y = \sin(\theta)$; $a = \cos(\theta)$;

$v_z = \cos(\theta) \cos(\Psi)$;

$v_x = \cos(\theta) \sin(\Psi)$;

Un possible **up**: **up** = (0,1,0) ($\varphi = 0^\circ$)

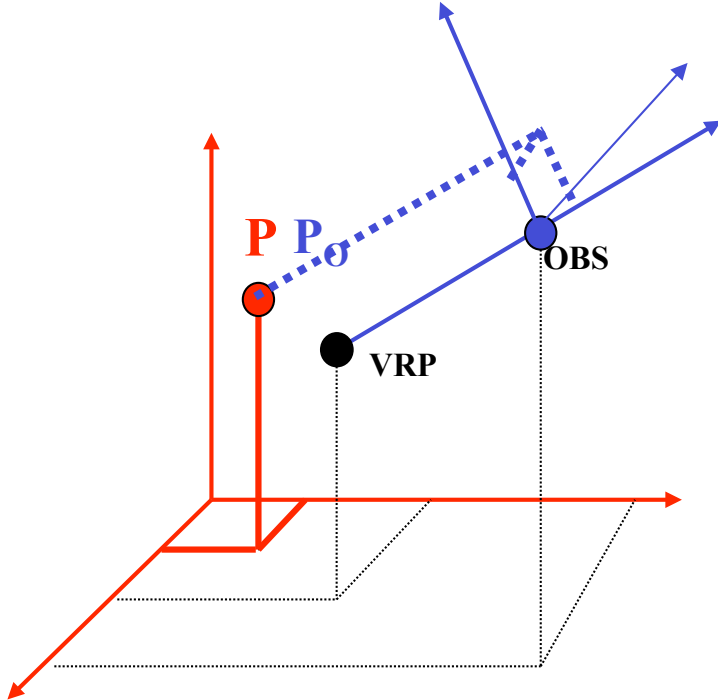
Es podria calcular la view matrix directament a partir dels angles?

Noteu que estem considerant els angles d'orientació de la càmera:

Ψ en $[-180, 180]$, θ en $[-90, 90]$

positius quan movem la càmera cap ➔ i quan la movem cap ⬆

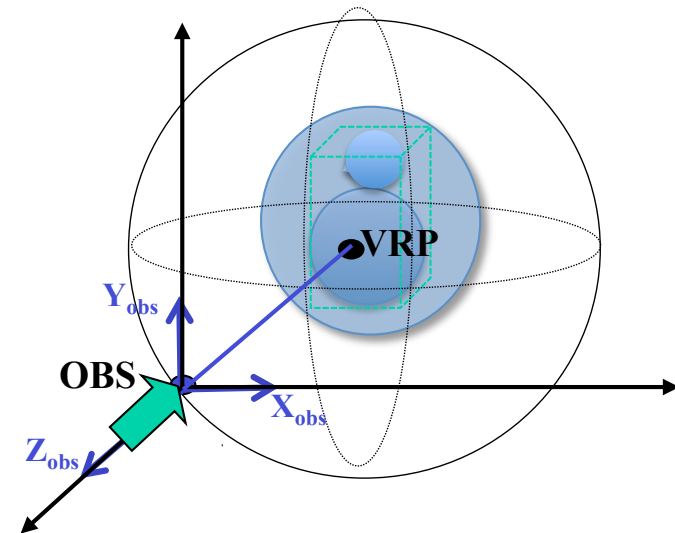
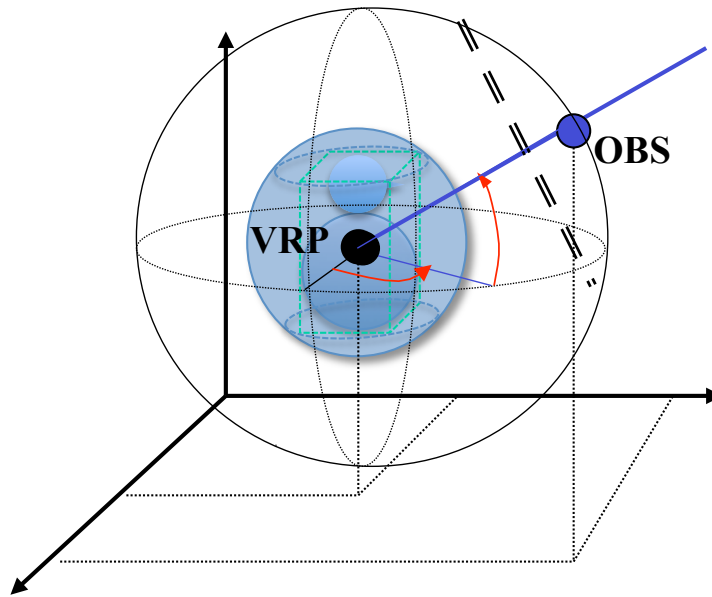
Càlcul view Matrix directe a partir d'angles Euler, VRP i d



RECORDEU:

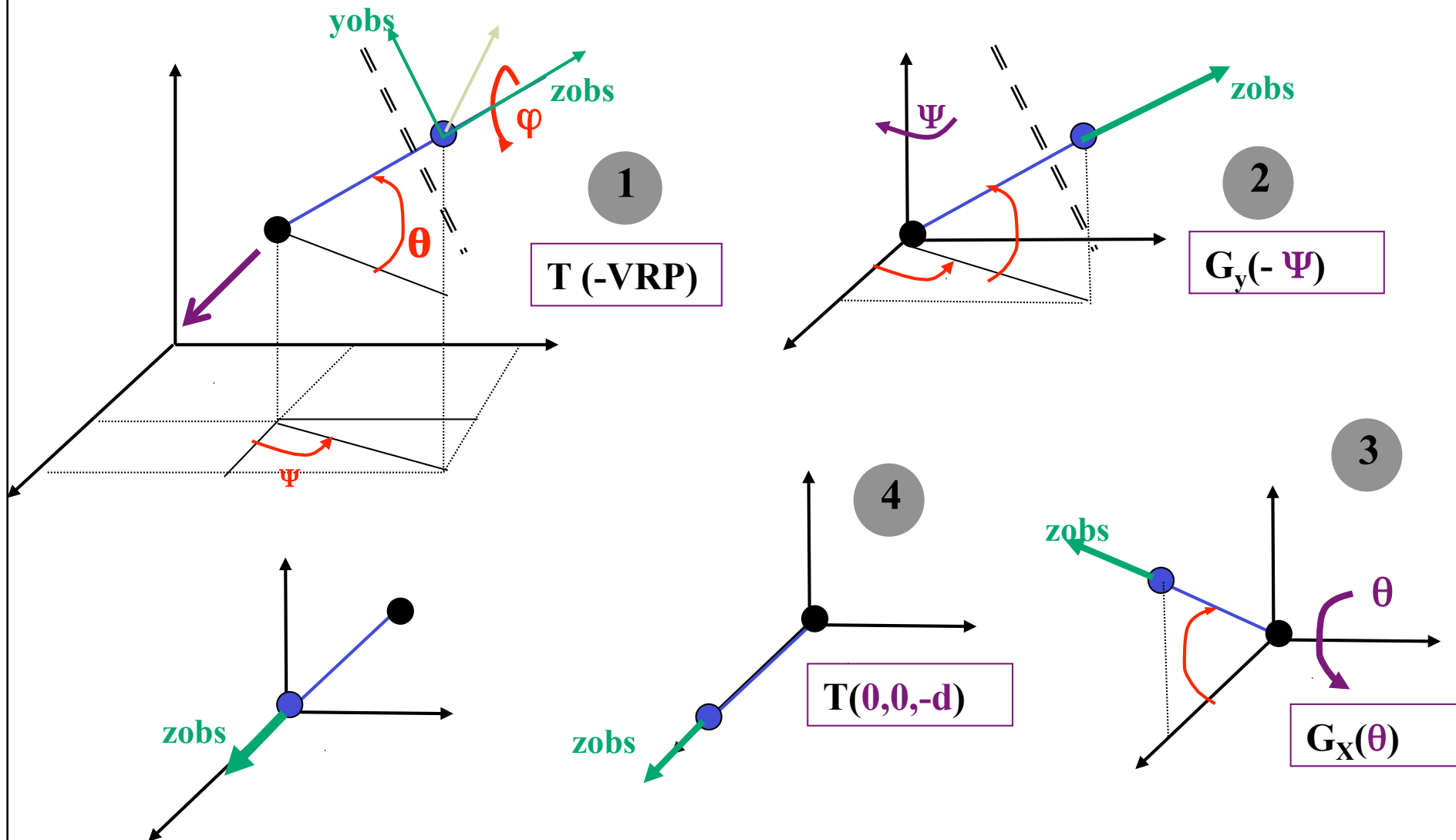
La viewMatrix serveix per tenir posició de punts respecte observador

Càlcul VM directe a partir d'angles Euler, VRP i d



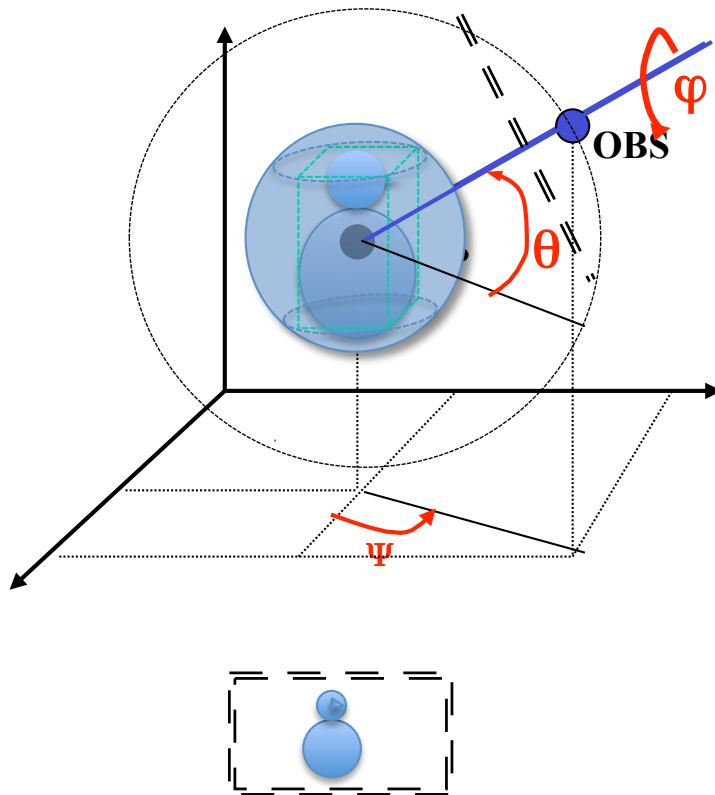
- Ho podeu pensar com si girem l'esfera per a què la seva posició respecte la càmera de defecte sigui la mateixa. Agafar l'esfera i posicionar-la.
- Noteu que z_{obs} passarà a ser coincident amb z_A (SCO i SCA coincidiran)
- **Pensarem el moviment tenint en compte que sabem calcular matrius de gir només si girem entorn d'eixos que passen per origen de coordenades.**

Càlcul MV directe a partir d'angles Euler: exemple més complexe



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

Exercici d'inicialització càmera: Posicionament amb angles Euler (TG)



$$VM = T(0,0,-d) * G_Z(-\varphi) * G_X(\theta) * G_Y(-\psi) * T(-VRP)$$

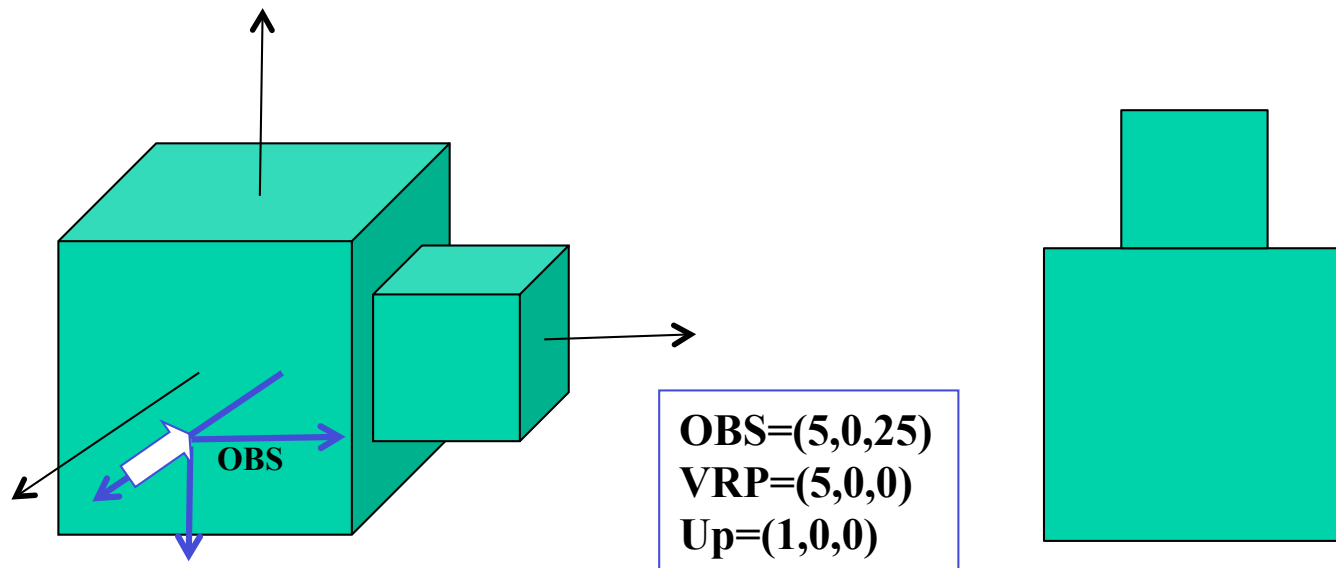
```
VM=Translate (0.,0.,-d)
VM=VM*Rotate(-\varphi,0,0,1)
VM= VM*Rotate (\theta,1,0,0.)
VM= VM*Rotate(-\psi,0,1,0.)
VM= VM*Translate(-VRP.x,-VRP.y,-VRP.z)
viewMatrix(VM)
```

Ull amb signes:

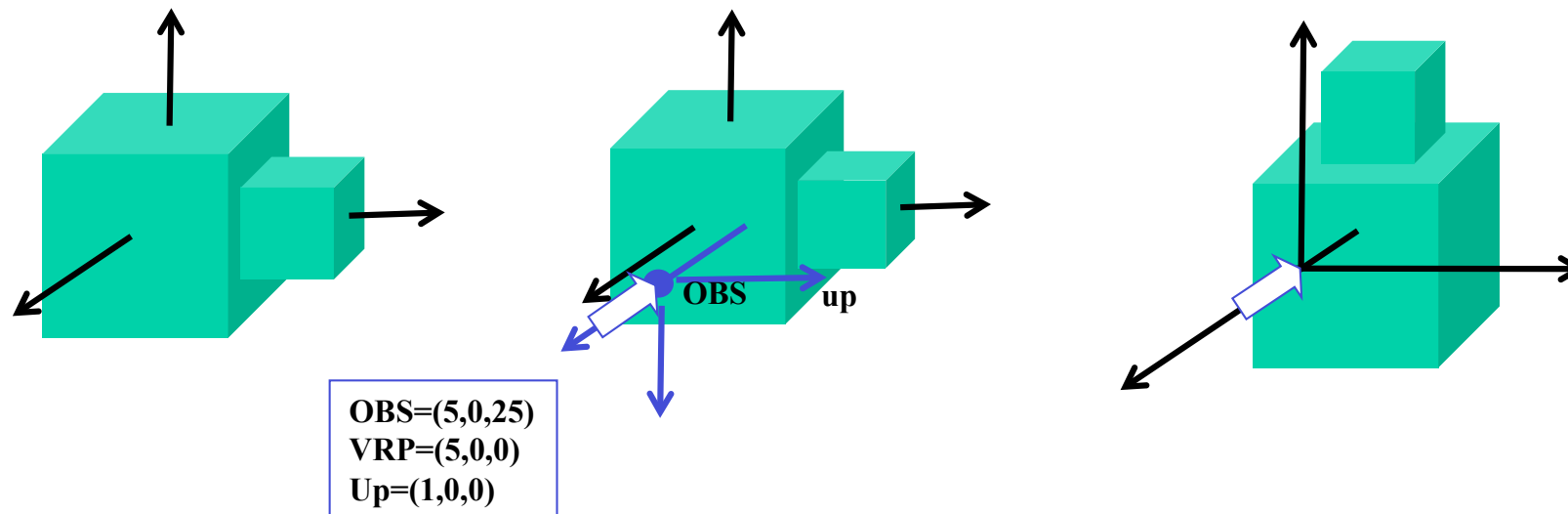
- Si s'ha calculat ψ positiu quan càmera gira cap a la dreta, serà un gir anti-horari respecte eix Y de la càmera, per tant, matemàticament positiu; com girem els objectes en sentit contrari, cal posar $-\psi$ en el codi.
- Si s'ha calculat θ positiu quan pugem la càmera, serà un gir horari; per tant,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

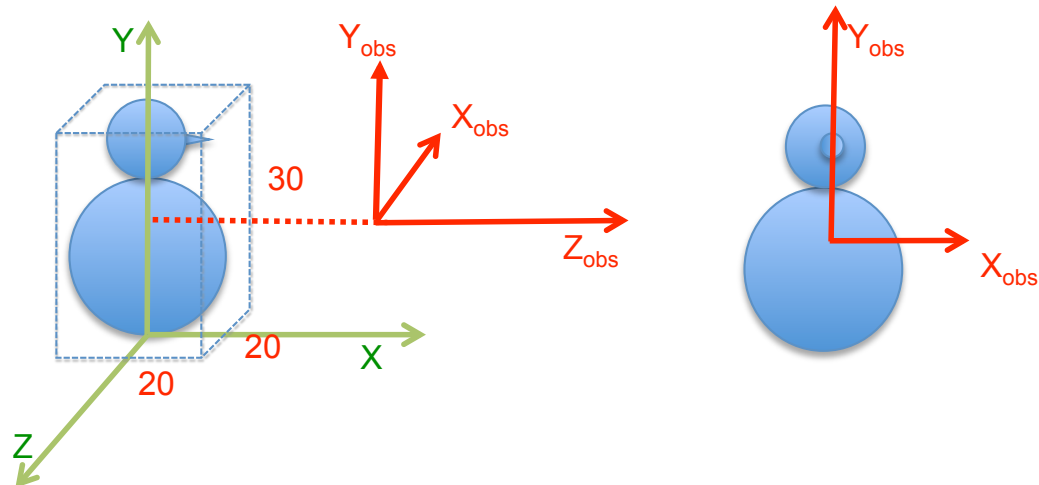
Alguns exercicis

Exercici 18. Una escena està formada per dos cubs, 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 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 **VRP**, **OBS** i **up**.



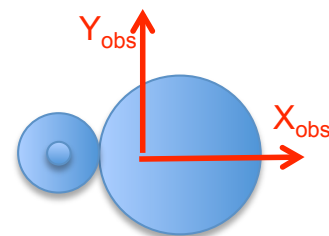
Exercici 18(bis). Una escena està formada per dos cubs, un de costat 20 centrat al punt $(0,0,0)$, i l'altre de costat 10 centrat al punt $(15,0,0)$. Indiqueu TOTS els paràmetres d'una càmera que permeti veure a la vista dos quadrats, un damunt de l'altre (el més gran a sota), de manera que ocupin el màxim de la vista (*viewport*). Cal que indiqueu la posició i orientació de la càmera especificant els **angles d'Euler** i indicant l'expressió de la viewMatrix.





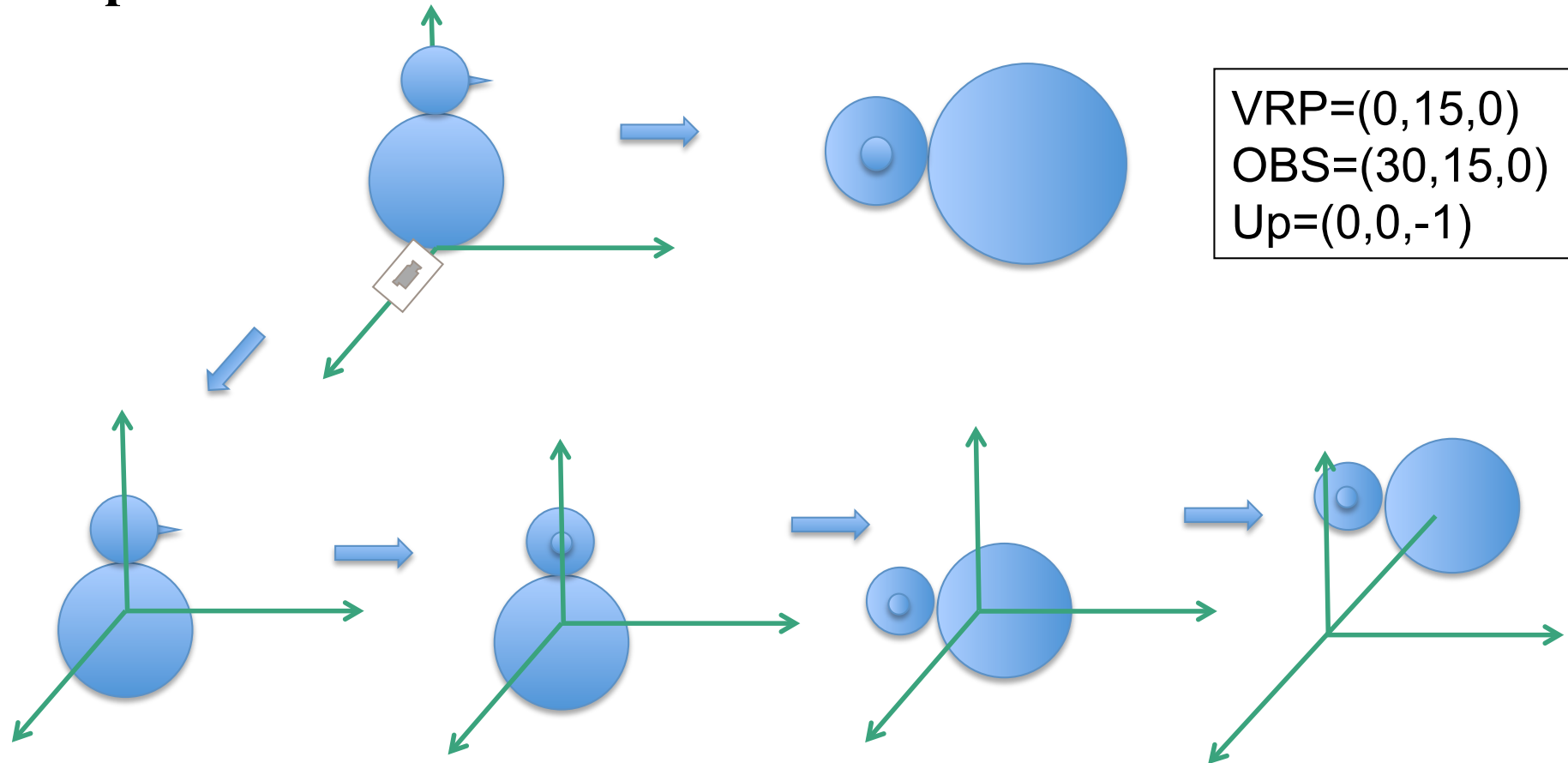
VRP=(0,15,0)
OBS=(30,15,0)
Up=(0,1,0)

Penseu en càmera i com ha de quedar la imatge



Quins paràmetres si volem que quedi així?

Exemple Ninot: càlcul de VM amb TG



$$TC = T(0,0,-30)G_z(90)G_y(-90)T(0,-15.0)$$

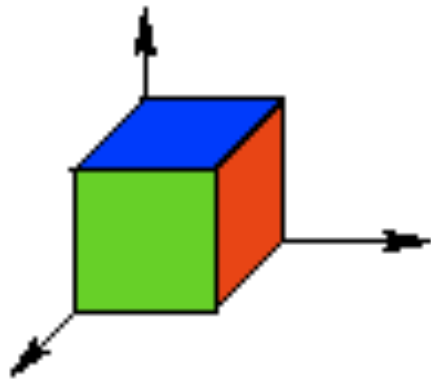
```

VM= Translatef(0.,0.,-30.);
VM= VM*Rotate (90.,0.,0.,1.);
VM= VM*Rotate (-90.,0.,1.,0.);
VM= VM*Translate (0.,0,-15.);
ViewMatrix(VM);
Pinta_Ninot();
    
```


109. (2015-2016P Q1) 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.

a) 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'obtingria.

b) Quin efecte tindria en la imatge final modificar l'òptica a axonomètrica?



Exercici 62. Una esfera de radi 1 es visualitza en un viewport quadrat de 400 per 400, amb una càmera posicionada correctament per poder veure tota l'esfera, i on el mètode per a definir la projecció de la càmera utilitza la següent crida:

```
TP = Perspective (60.0, 1.0, 1.0, 10.0);  
projectMarix (TP);
```

L'usuari ha redimensionat la finestra a 500 d'amplada per 400 d'alçada. Digues què cal canviar de la càmera per tal que es vegi l'esfera correctament (sense retallar-la ni deformar-la).

- a. Incrementar l'angle d'obertura vertical (FOV) i la relació d'aspecte del window.
- b. Augmentar la relació d'aspecte del window i la distància al ZNear.
- c. Només augmentar la relació d'aspecte del window.
- d. Només canviar l'angle d'obertura vertical (FOV).

Per pensar: Òptica per veure tota l'escena
Càmera en primera persona

Exercicis de la llista a fer (mínims):

- 35
- 42
- 45
- 22
- 48
- 52
- 70