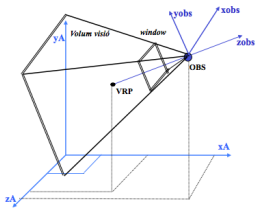
Classe 4: contigut

- Càmera (2):
  - Recordatori: PV, Matrius, definició de càmera
  - Càmera en 3ra persona
  - Angles Euler
- Exercicis Càmera

Camera i procés de visualització



Càmera Inicial (1)
(OBS, VRP, up, FOV, zN, zF, raw)
Metàfores d'interacció

view & project transform:
viewMatrix, projectMatrix

viewport

model
Matrix$_i$

PintaModel$_i$

VAO$_i$

Vertex Shader

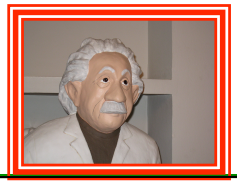
OpenGL-
Visualització

Fragment Shader

Píxel(x,y,c)

yD

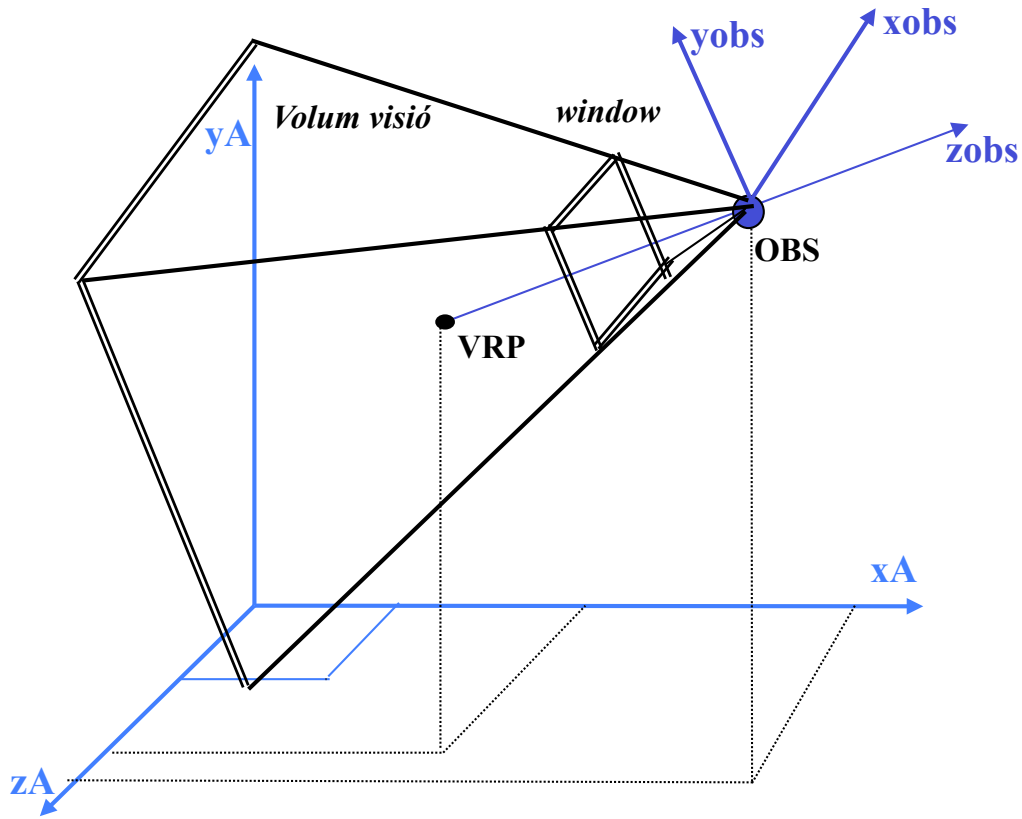
Finestra OpenGL (Pantalla)



Viewport/Vista

xD

Càmera: OBS, VRP, up, zN, zF, FOV raw



```
// Crear VAOs de models (un cop);
…
/* calcular paràmetres càmera i
    matrius cada cop que es
   modifiquin */
VM = lookAt(OBS,VRP,UP);
viewMatrix(VM);
PM=perspective (FOV,ra,zN,ZF);
projectMatrix(PM);
glViewport (0,0,w.h);
….
// cada cop que es requerix  refresc
per cada objecte_i
  // calcular TG i passar uniform a OpenGL
    modelMatrix (TG_i);
    pintaModel (VAO_i);
fper
```

Vertex Shader

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

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

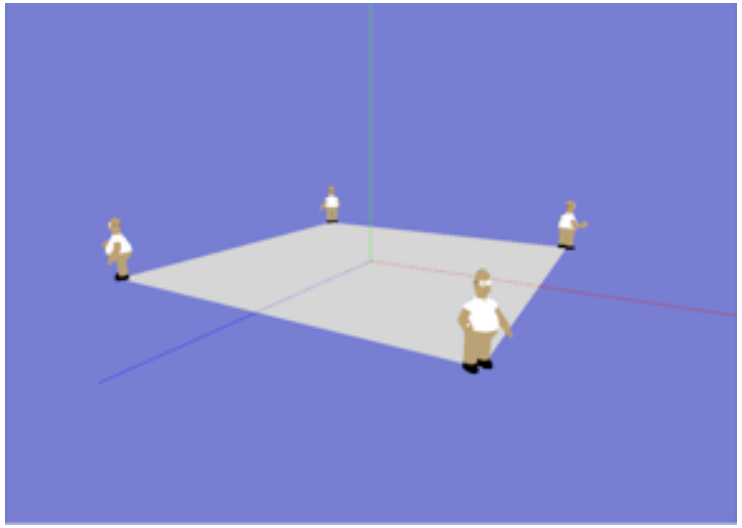
a) No importa l'ordre en què s'indiquen.

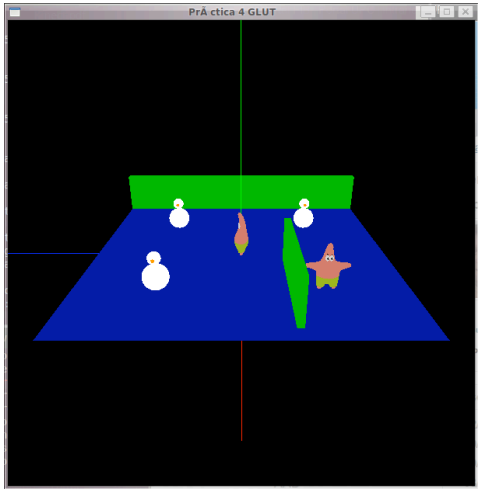
b) Transformació de posició + orientació, transformació de projecció, *viewport*.

c) La transformació de projecció, transformació de posició + orientació, *viewport*.

d) *Viewport*, transformació de projecció, transformació de posició + orientació.

Càmera 3ra persona



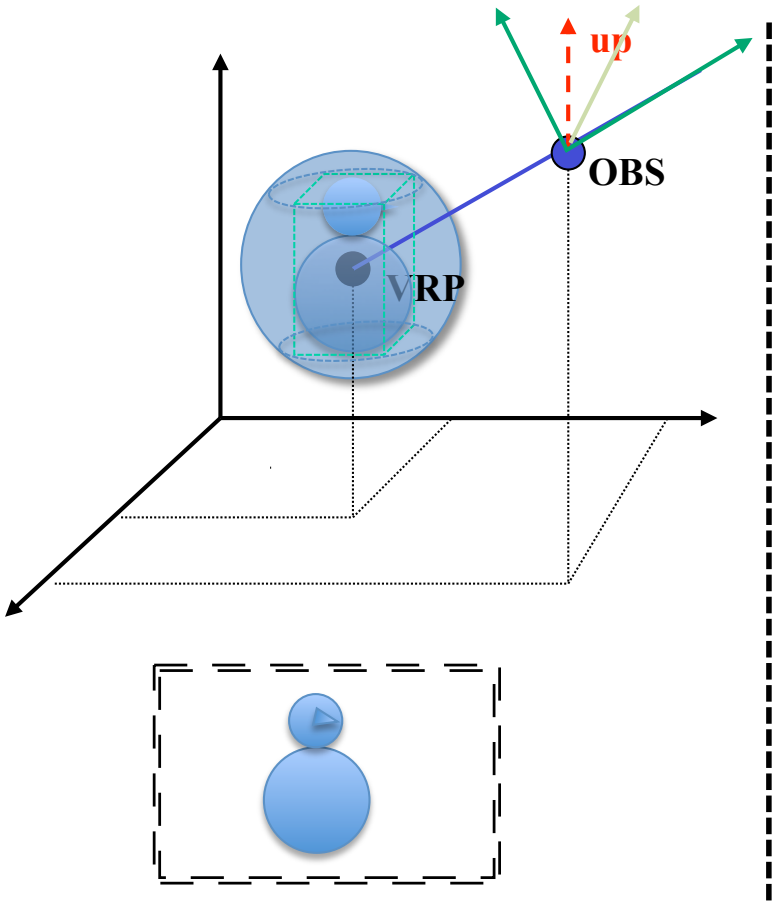


Quins paràmetres de posició, orientació i òptica per càmera en 3ra persona? ➔ imatge inclogui tota l'escena, ocupant el màxim del viewport.

Dada: capsa mínima contenidora d'escena

(xmin, ymin, zmin) - (xmax, ymax, zmax)

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)

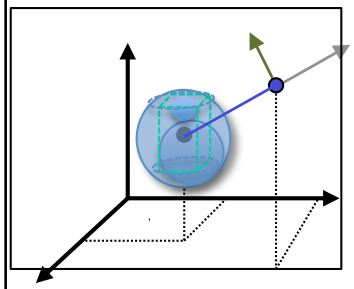
  - CentreEscena=Centre(CapsaMinCont) = ((xmax+xmin)/2,(ymax+ymin)/2,(zmax+zmin)/2))

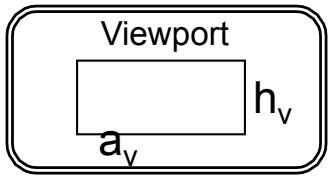
  - R=dist((xmin,ymin,zmin),(xmax,ymax,zmax))/2
  - d>R; per exemple d=2R

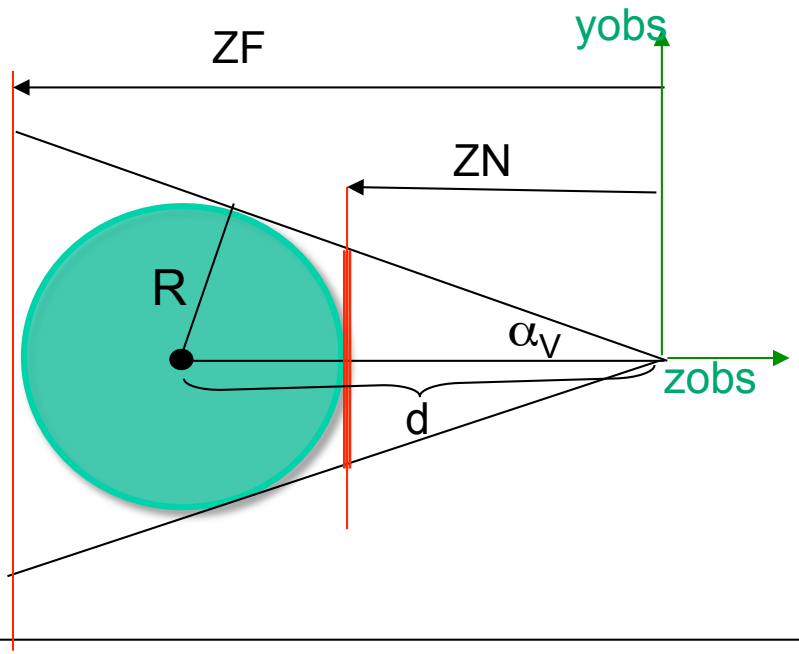
  - **OBS**=**VRP**+ d***v**; **v** normalitzat en qualsevol direcció; per exemple **v**= (1,1,1)/ ||(1,1,1)||

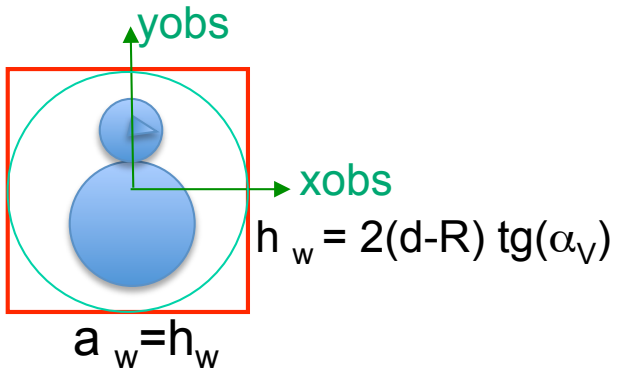
- **up** qualsevol que no sigui paral·lel a **v**; si volem ninot vertical (eix Y es vegi vertical) **up**=(0,1,0)

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, ZN ∈] 0, d-R] ZF ∈] d+R, …];

  per a aprofitar precissió profunditat: ZN= d-R; ZF=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 (α_V) ; α_V = arc sin (R/d) => FOV=2*α_V

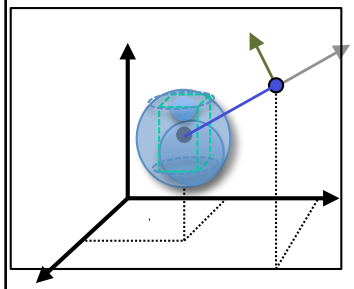
  - com window està situat en ZN, α_V determina que la seva alçada sigui: $h_w = 2(d-R)\ tg(\alpha_V)$

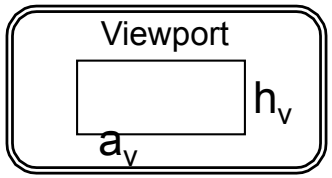
- $ra_w = a_w/h_W = 1$ (perque α_H hauria de ser igual a α_V per asegurar que esfera no retallada)

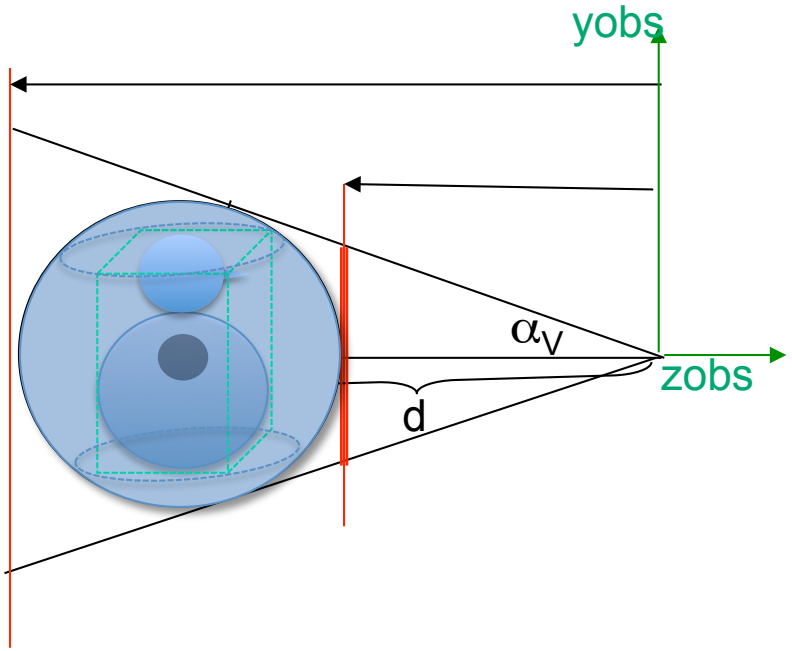
  - Però ULL!! per a què no hi hagi deformació, cal que ra_w sigui sempre igual a ra_v, per tant, si no volem modificar el viewport: $ra^*_w = ra_v$ i … amb aquesta nova ra^*_w

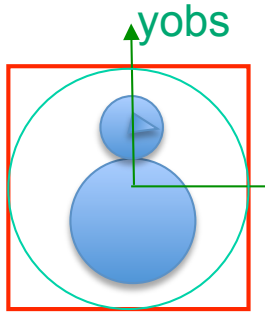
  es retallarà l'esfera? (estarà tota l'esfera dins del volum de visió?)

Tota escena en la vista, sense deformar i càmera perspectiva









$h_w = 2(d-R)\ tg(\alpha_V)$

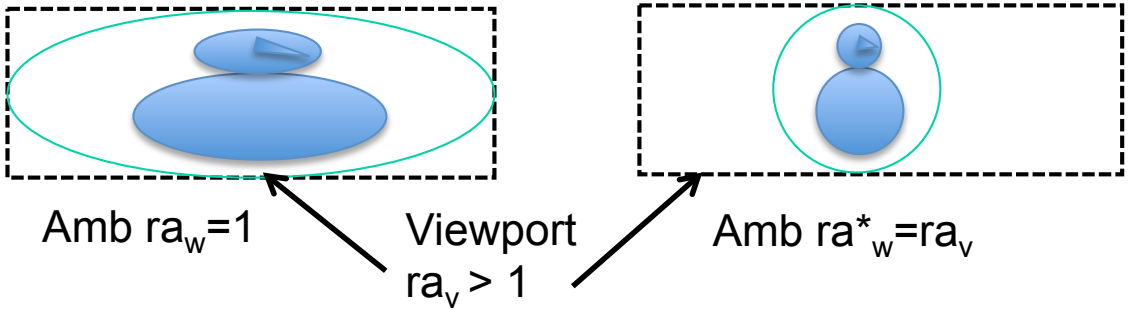
h_w

$a_w = ra_w \cdot hw$

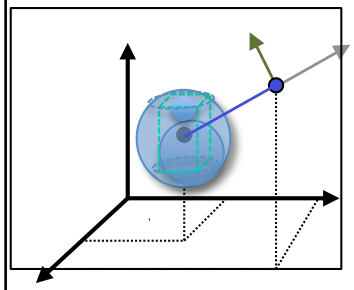
- Si $ra_v > 1$ i $ra_w = ra_v$ => la nova $a^*_w > a_w$ mínima requerida =>

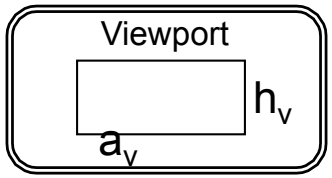
No es retalla

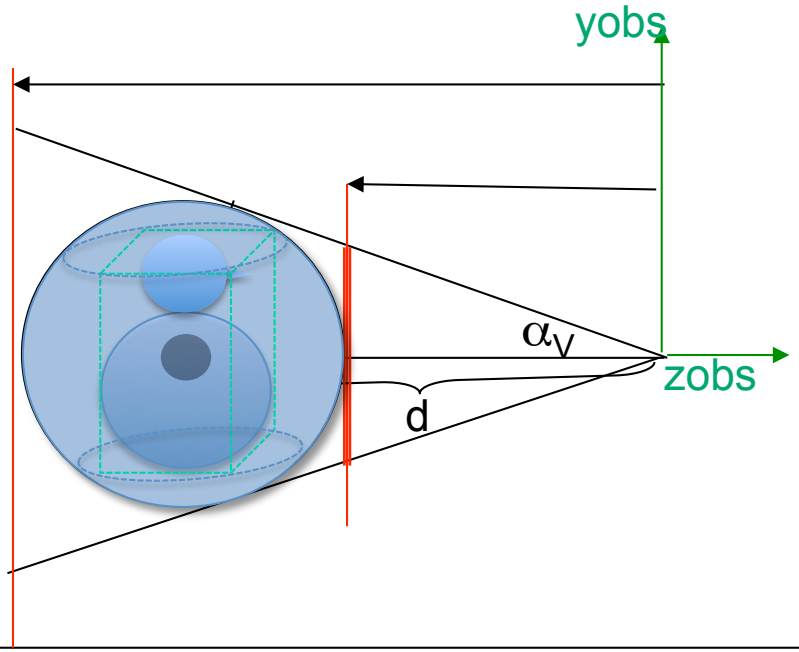
no cal modificar α_V (FOV)

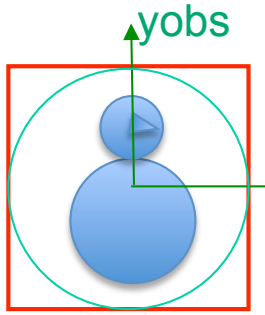


Tota escena en la vista, sense deformar i càmera perspectiva





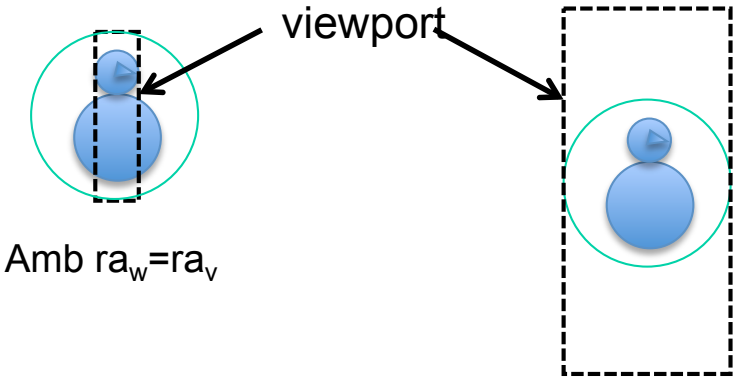




$h_w = 2(d-R)\ tg(\alpha_V)$

$a_w = ra_w \cdot hw$

- Si $ra_v<1$ => $ra^*_w < ra_w$ => $a^*_w < a_w$ =>retallarà; per evitar-ho cal incrementar l'angle d'obertura (quedarà espai lliure a dalt i a baix)

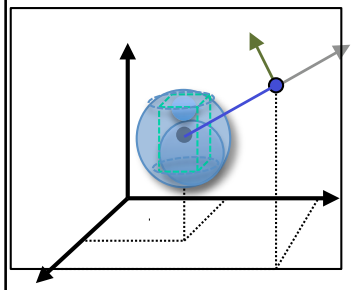


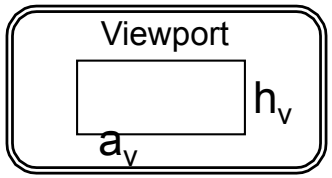
Amb $ra_w=ra_v$

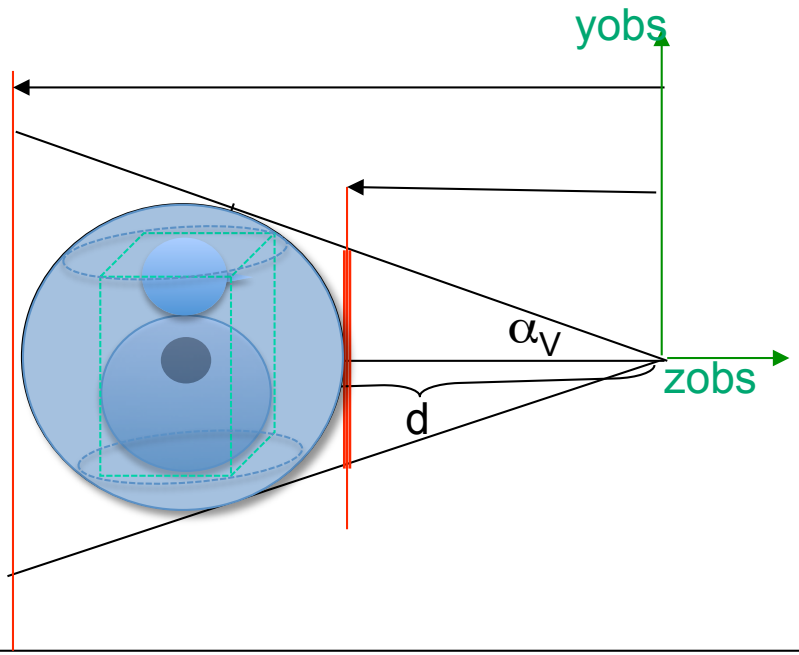
- Amb $ra_w=ra_v$ i nou FOV
- FOV=2 α^*_V on $\alpha^*_V = arctg(tg(\alpha_V) / ra_v)$

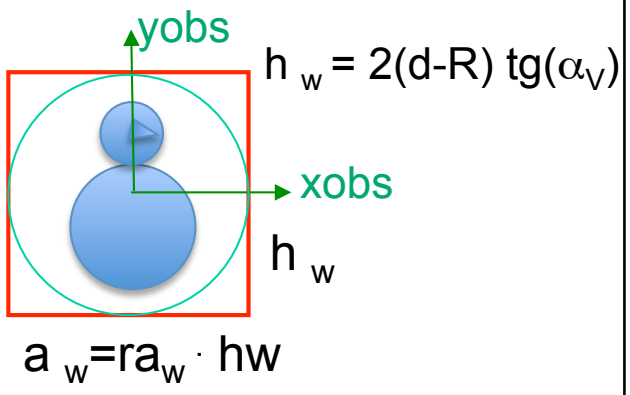
- Sempre cal calcular el nou angle a partir de l'inicial (window quadrat). Penseu que pasaria si no ho feu i modifiqueu interactivament el viewport (finestra gràfica) fent-ho >1 i <1 molts cops seguits.

Tota escena en la vista, sense deformar i càmera perspectiva









- Si $ra_v > 1$ (>ra_w mínima requerida 1) => No es retalla, no cal modificar α_V (FOV)

  Justificació: ra^*_w serà superior a 1; si no modifiquem l'angle FOV, h_w no canvia =>
  $a^*_w = ra^*_w \cdot h_w$ i com $ra^*_w > ra_w$ => $a^*_w > a_w$ 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 mínim requerida 1) => cal incrementar l'angle d'obertura (quedarà espai lliure a dalt i a baix)

  $FOV = 2\ \alpha^*_V$ on $\alpha^*_V = arctg(tg(\alpha_V) / ra_v)$

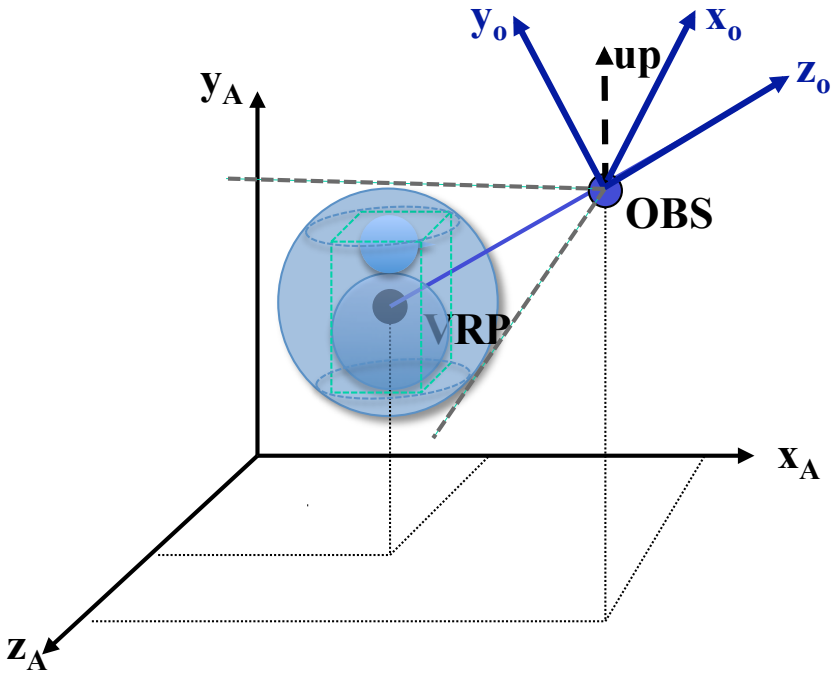
  Justificació: *com* $a^*_w = ra^*_w \cdot h_w$, si no modifiquem angle, h_w no varia; com $ra^*_w < ra_w$ => $a^*_w < a_w$ i l'esfera quedaria retallada (en horitzontal). Per tant, cal incrementar l'angle α_V (i, per tant, h^*_w) per a garantir una amplada del window igual a la mínima requerida (igual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

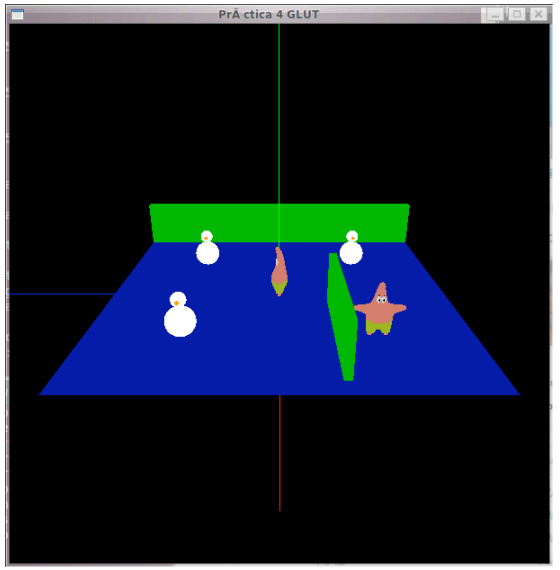
  - sabem: $h^*_w = a_w / ra_v = (2(d-R)tg(\alpha_V)) / ra_v$ i per trigonometria $h^*_w = 2(d-R)\ tg(\alpha^*_V)$
  - per tant: $\alpha^*_V = arctg(tg(\alpha_V) / ra_v)$

Vist…

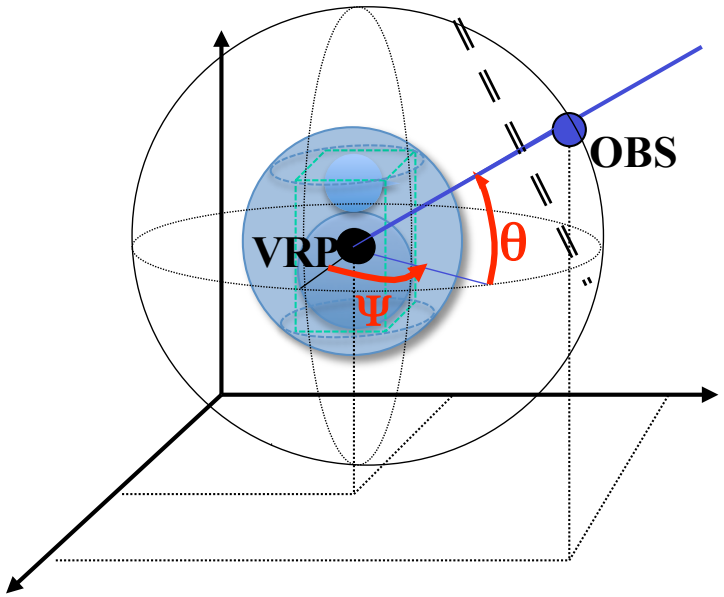
- Posicionament: OBS, VRP, up → viewMatrix
- Òptica perspectiva: zN, zF, FOV, ra →projectionMatrix
- Càmera en 3ra persona: posició inicial

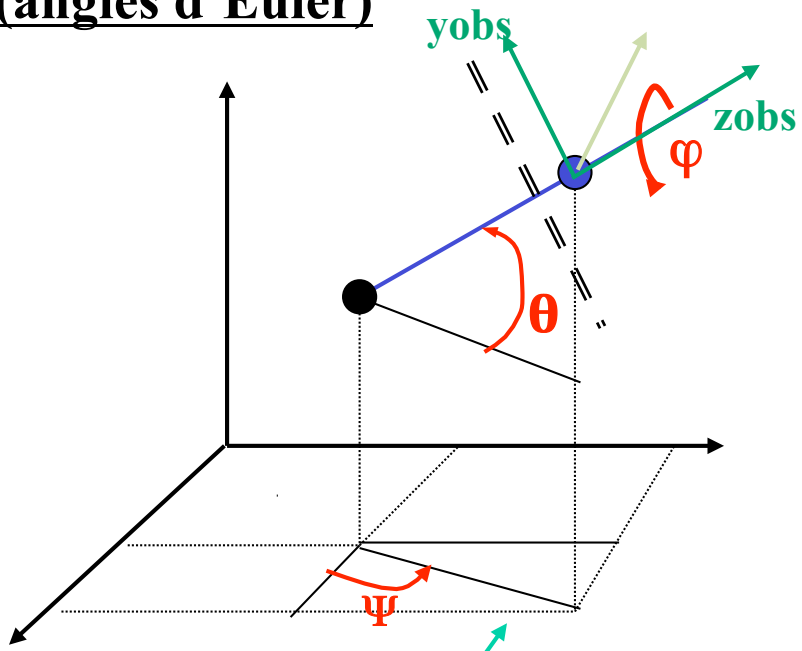
Com Moure la Càmera per inspeccionar escena?



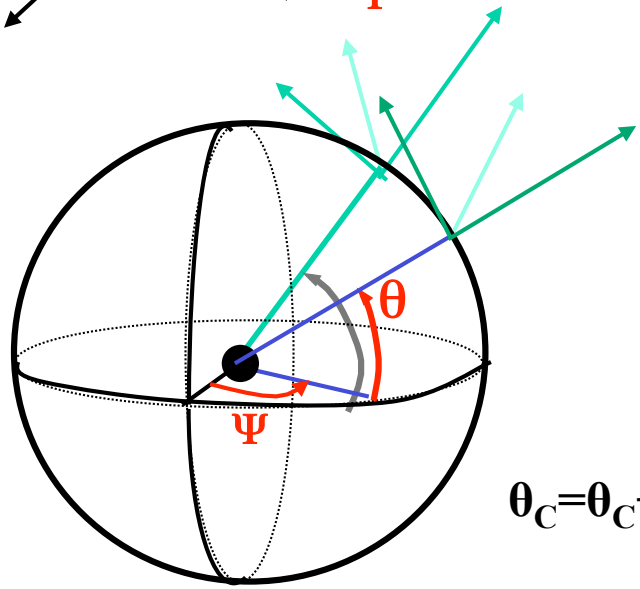


Moure la Càmera (angles d'Euler)





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

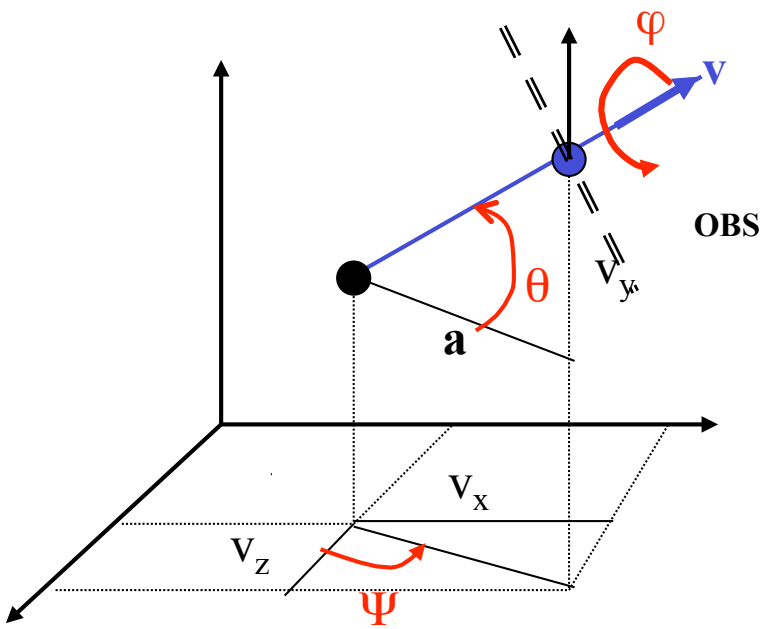


$\theta_C = \theta_C + f(\Delta y)$

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

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

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 (θ); a = cos (θ);

v_z = cos (θ) cos (Ψ) ;

v_x = cos (θ) sin (Ψ);

Un possible **up**: **up** = (0,1,0) (φ = 0º)

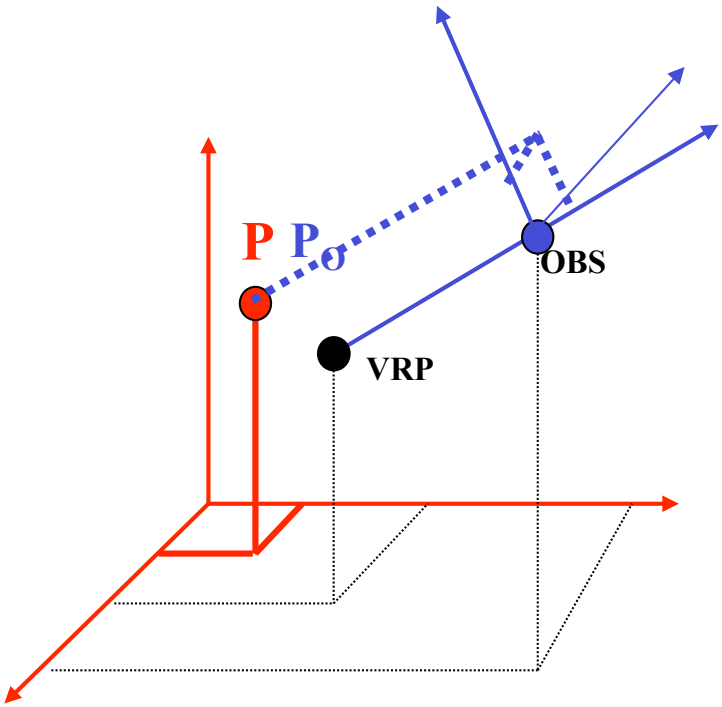
Es podria calcular la view matrix directamente a partir dels angles?

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* ➚

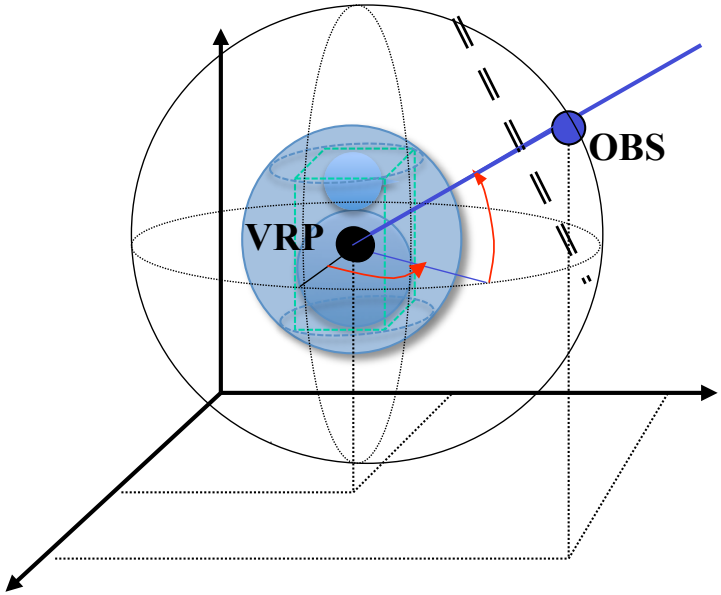
Càlcul view Matrix directe a partir d'angles Euler, VRP i d

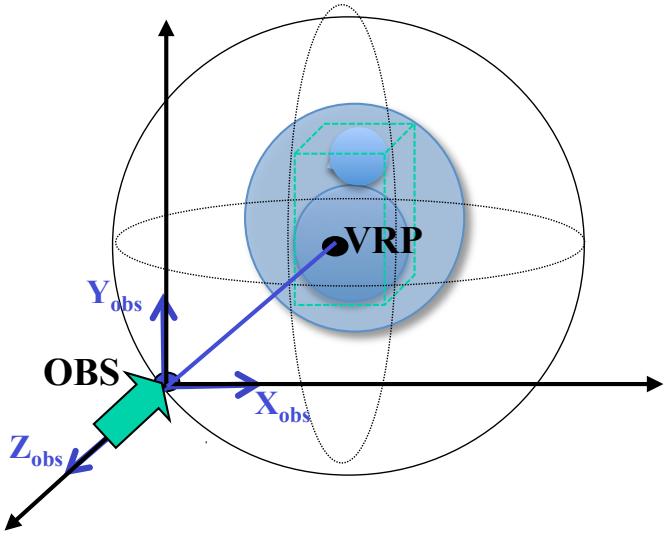


RECORDEU:

La viewMatrix serveix per tenir posició de punts respecte observador

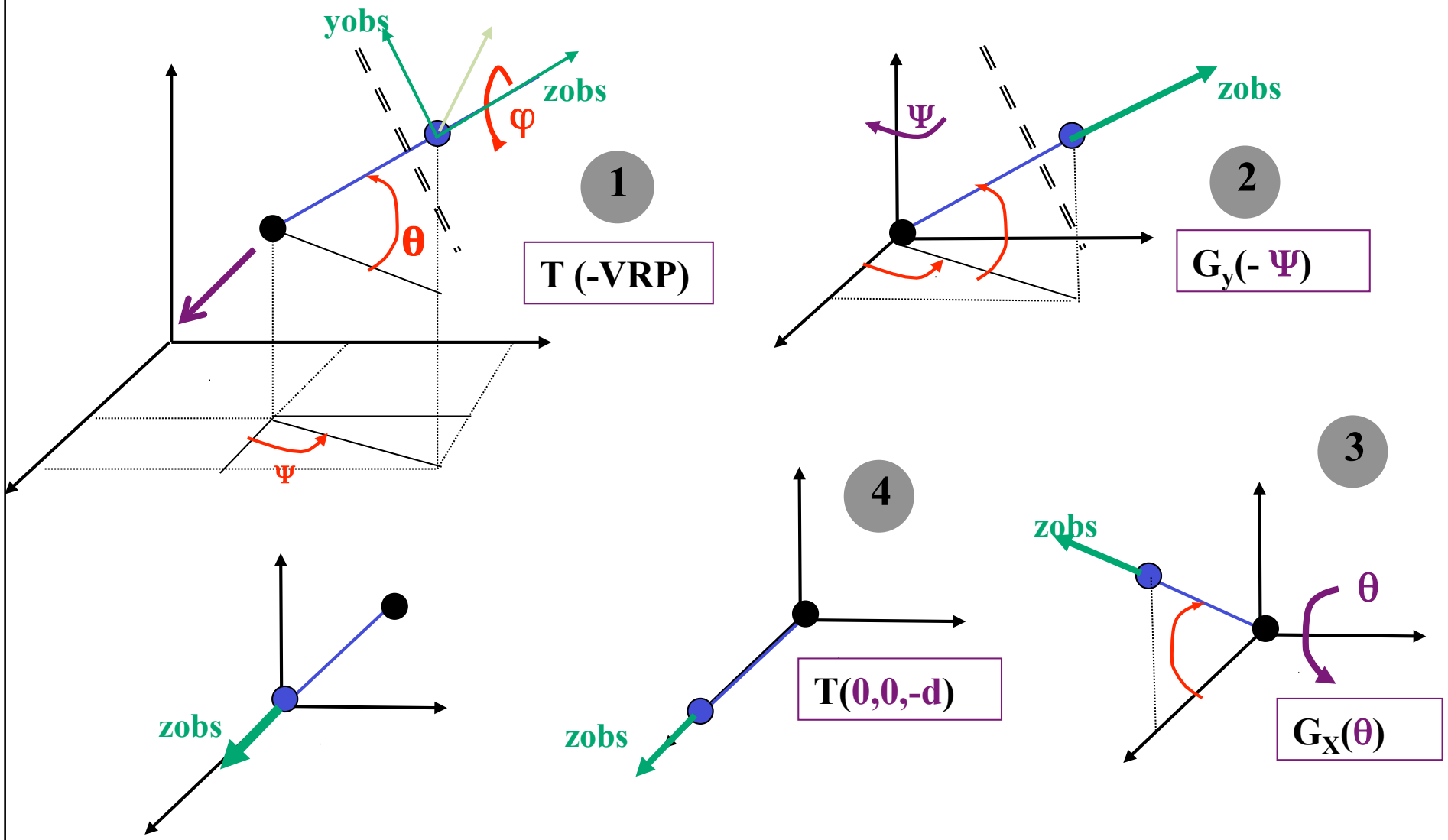
Càlcul VM directe a partir d'angles Euler, VRP i d





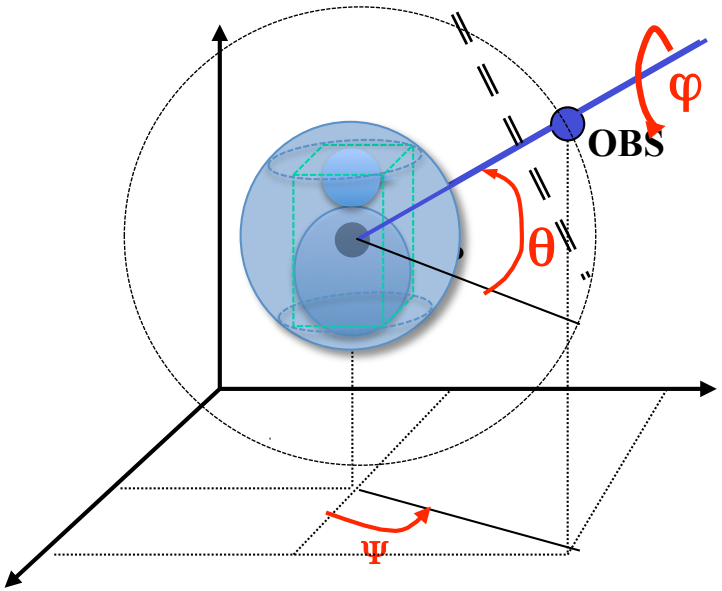
- Ho podeu pensar com si girem l'esfera per a què la seva posició respecte la càmera de defecte sigui la mateixa. Agafar l'esfera i posicionar-la.
- Noteu que zobs passarà a ser coincident amb zA (SCO i SCA coincidiran)
- Pensarem el moviment tenint en compte que sabem calcular matrius de gir només si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

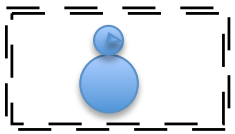
Càlcul MV directe a partir d'angles Euler: exemple més complexe



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

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(-φ,0,0,1)
VM= VM*Rotate (θ,1.,0.,0.)
VM= VM*Rotate(-ψ.,0.,1.,0.)
VM= VM*Translate(-VRP.x,-VRP.y,-VRP.z)
viewMatrix(VM)
```

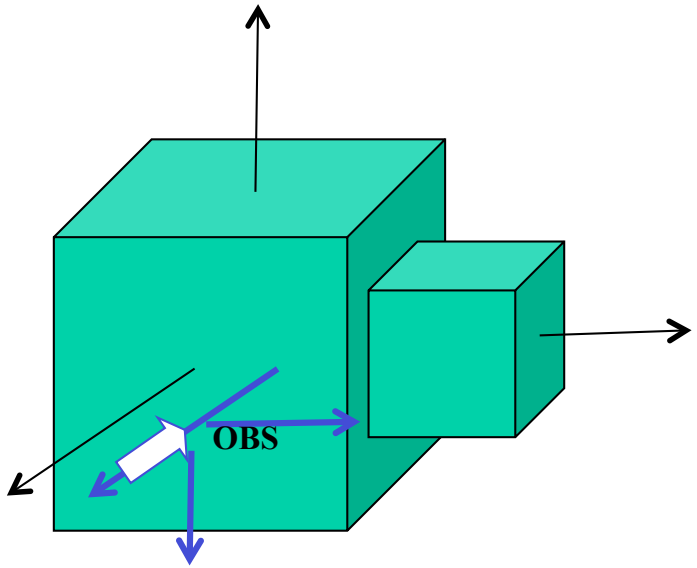
Ull amb signes:

• 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 −ψ en el codi.

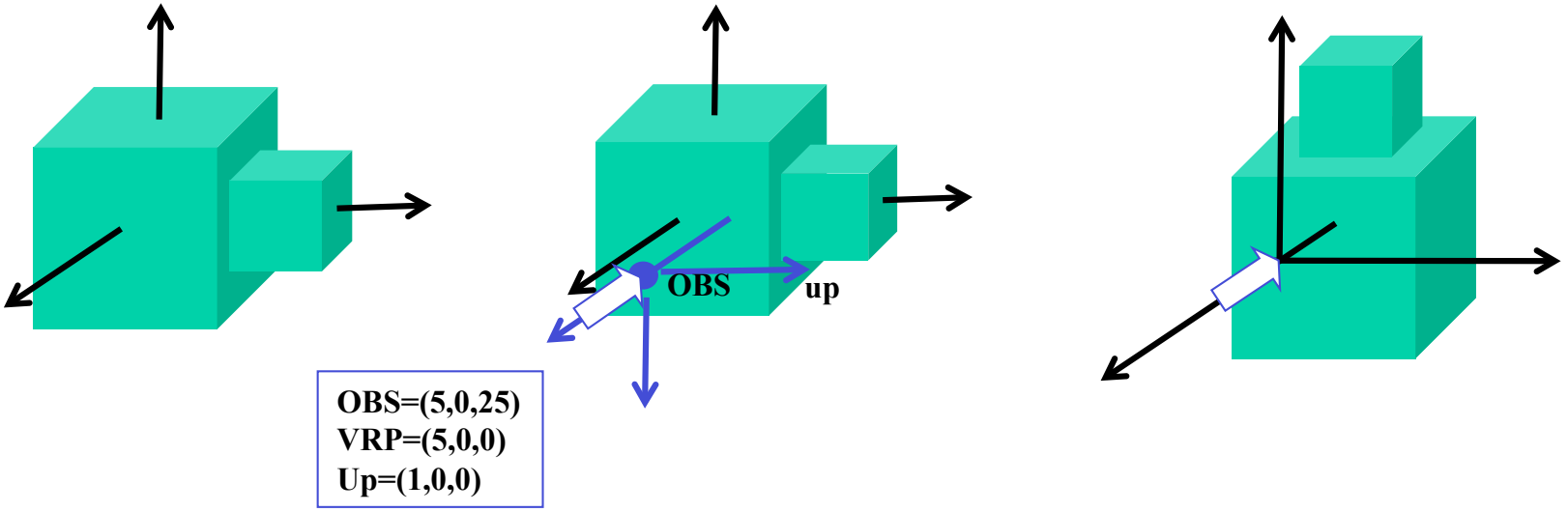
•Si s'ha calculat θ positiu quan pugem la càmera, serà un gir horari; per tant, matemàticament un gir negatiu; com objecte girarà en sentit contrari (anti-horari), ja és correcte deixar signe positu.

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**.



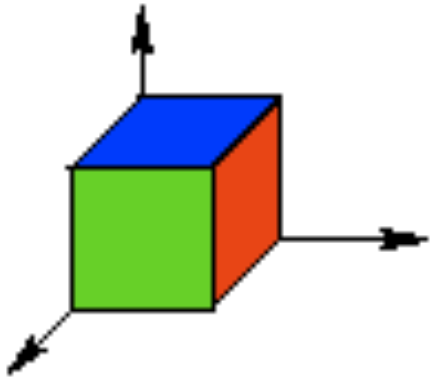
Exercici 18(bis). Una escena està formada per dos cubs, un de costat 20 centrat al punt (0,0,0), i l'altre de costat 10 centrat al punt (15,0,0). Indiqueu TOTS els paràmetres d'una càmera que permeti veure a la vista dos quadrats, un damunt de l'altre (el més gran a sota), de manera que ocupin el màxim 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.

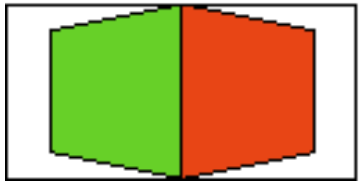


109. (2015-2016P Q1) Tenim una escena amb un cub de costat 2 orientat amb els eixos i de manera que el seu vèrtex mínim està situat a l'origen de coordenades. La cara del cub que queda sobre el pla x=2 és de color vermell, la cara que queda sobre el pla z=2 és de color verd i la resta de cares són blaves.

a) Indica TOTS els paràmetres d'una càmera perspectiva que permeti veure complertes 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.

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



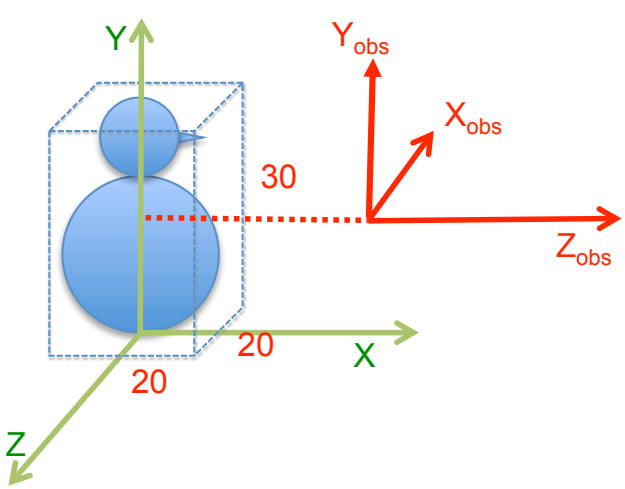


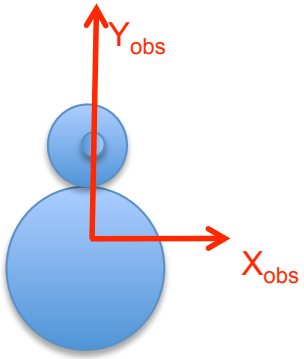
Exercici 62. Una esfera de radi 1 es visualitza en un viewport quadrat de 400 per 400, amb una càmera posicionada correctament per poder veure tota l'esfera, i on el mètode per a definir la projecció de la càmera utilitza la següent crida:

```
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).

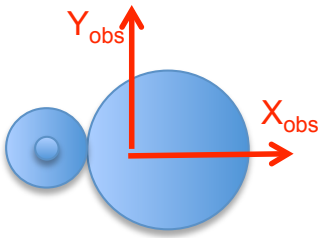
a. Incrementar l'angle d'obertura vertical (FOV) i la relació d'aspecte del window.
b. Augmentar la relació d'aspecte del window i la distància al ZNear.
c. Només augmentar la relació d'aspecte del window.
d. Només canviar l'angle d'obertura vertical (FOV).





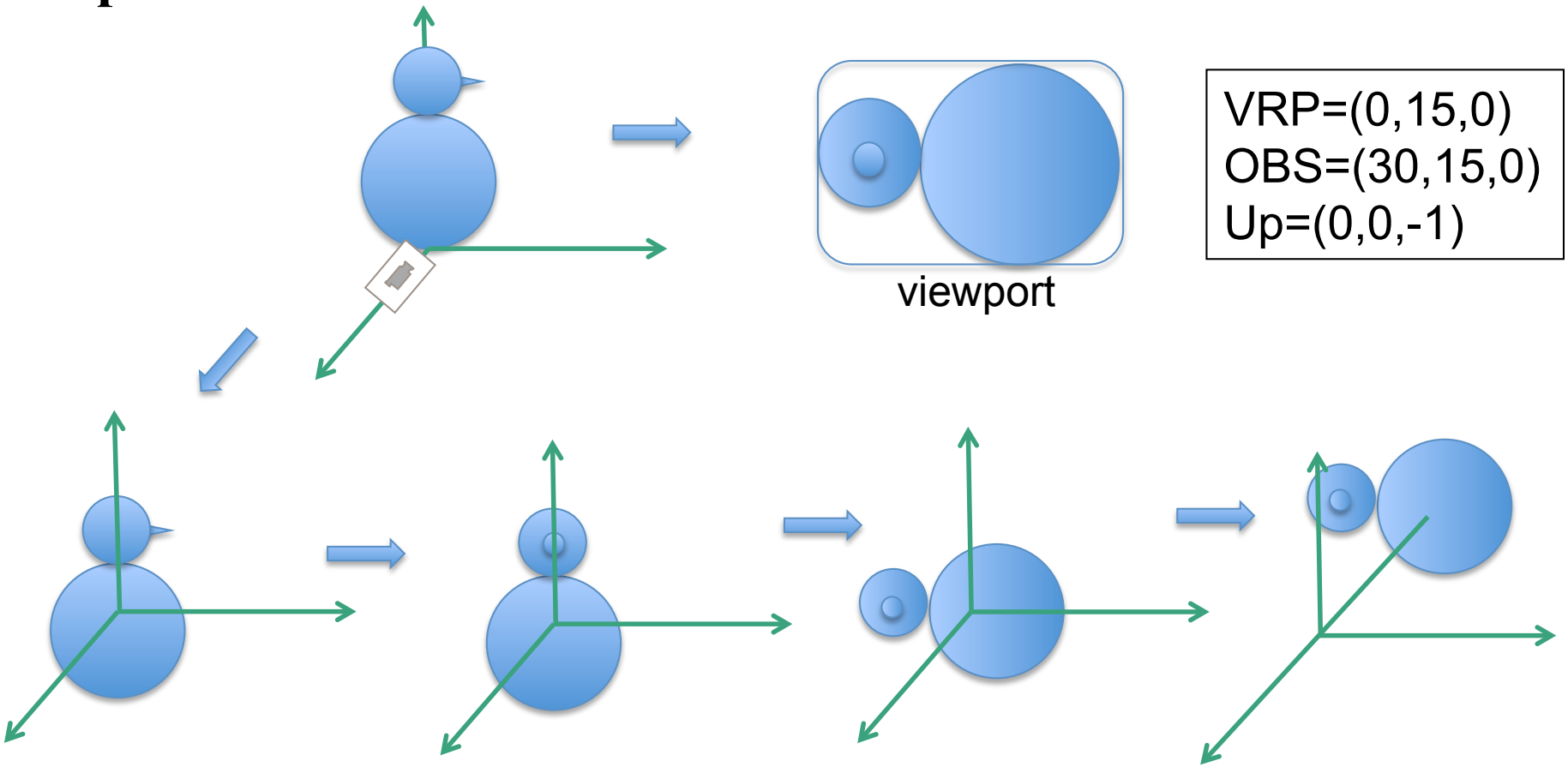
VRP=(0,15,0)
OBS=(30,15,0)
Up=(0,1,0)

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();
```

L'òptica i el ZOOM

a) Modificar l'angle d'obertura (tot mantenint la *ra*) ;
Modificar window en axonomètrica

b) Modificar la distància de l'Obs al VRP
(modificant ZN i ZF adequadament)

c) Modificar Obs i VRP en la direcció –v → travelling