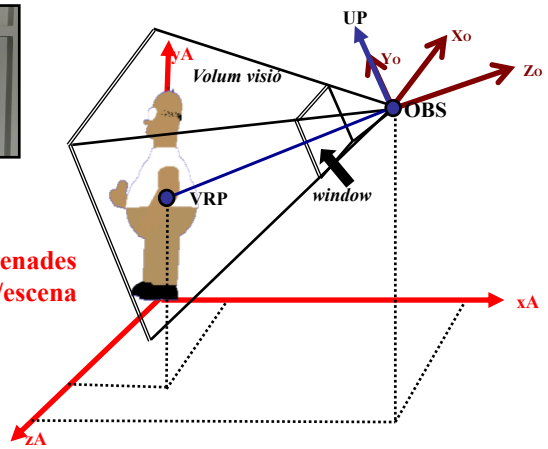
Classe 4: contingut

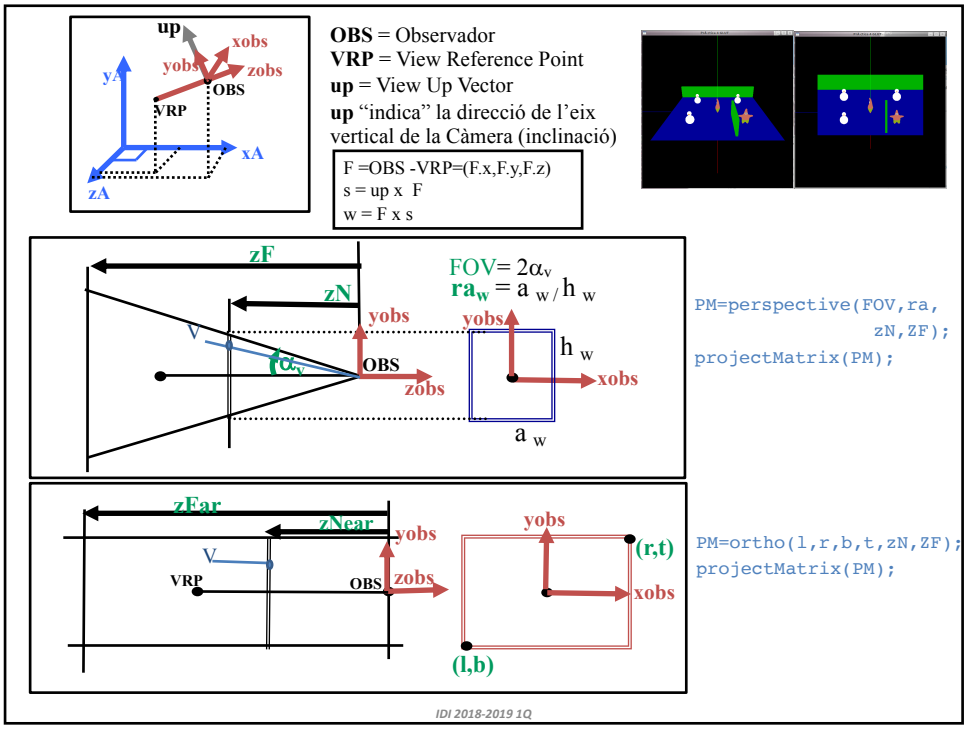
- Breu repàs càmera i el Procés de Visualització d'OpenGL
- Exercicis d'especificació de càmera
- Ubicar models en escena en el Vèrtex Shader
- Càmera en tercera persona

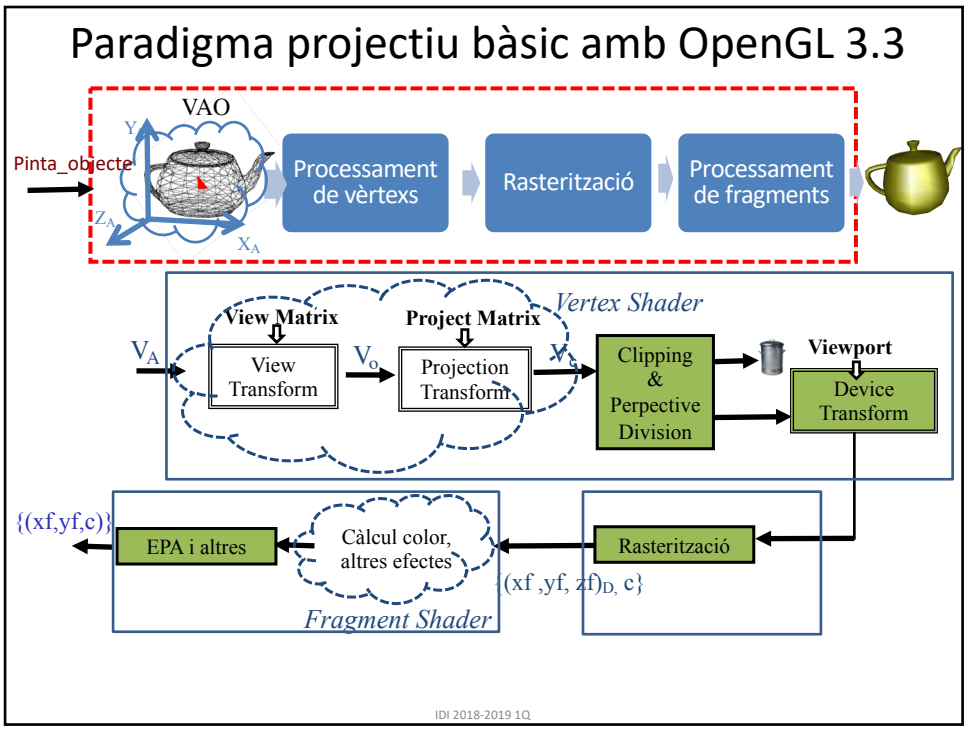
Com especificar la càmera virtual?



Sistema Coordenades Aplicació/món/escena

1. **Ubicació càmera respecte SCA: obs, vrp, up**
2. **Definir òptica: Volum de Visió ➔ window, zNear, zFar**


up
xobs
yobs
zobs
OBS
VRP
yA
xA
zA
OBS = Observador
VRP = View Reference Point
up = View Up Vector
up "indica" la direcció de l'eix vertical de la Càmera (inclinació)
F =OBS -VRP=(F.x,F.y,F.z)
s = up x F
w = F x s
zF
zN
FOV= 2αv
raw = a w / h w
yobs
V
αv
OBS
zobs
h w
xobs
a w
PM=perspective(FOV,ra, zN,ZF);
projectMatrix(PM);
zFar
zNear
VRP
(r,t)
(l,b)
PM=ortho(l,r,b,t,zN,ZF);
projectMatrix(PM);
IDI 2018-2019 1Q


Paradigma projectiu bàsic amb OpenGL 3.3
VAO
Pinta_objecte
Processament de vèrtexs
Rasterització
Processament de fragments
Vertex Shader
View Matrix
Project Matrix
VA
View Transform
Vo
Projection Transform
Vp
Clipping & Perpective Division
Viewport
Device Transform
{(xf,yf,c)}
EPA i altres
Càlcul color, altres efectes
Rasterització
{(xf ,yf, zf)D, c}
Fragment Shader
IDI 2018-2019 1Q

Programació mínima dels shaders

Vertex Shader

```
#version 330 core

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

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

Fragment Shader

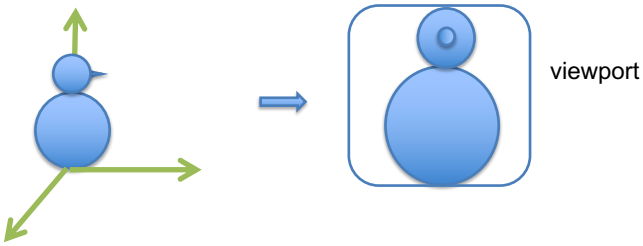
```
#version 330 core

out vec4 FragColor;

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

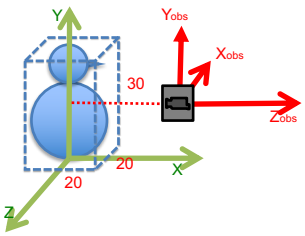

Classe 4: contingut

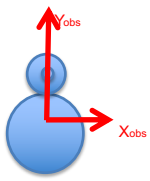
- Breu repàs càmera i el Procés de Visualització d'OpenGL
- **Exercicis d'especificació de càmera**
- Ubicar models en escena en el Vèrtex Shader
- Càmera en tercera persona

Exemple 1: Donada una funció `pinta_ninot()` que pintaria un objecte com el de la figura, format per: una esfera de radi 10 i centre (0,10,0), una altra esfera de radi 5 i centre (0,25,0), i un con de base centrada en (2.5, 25,0), r=2 i llargada 5 orientat segons l'eix X^+

– Indica tots els paràmetres d'una càmera que permeti obtenir la imatge similar a la que s'indica, en un viewport de 600x600 que optimitza espai en la finestra gràfica i amb òptica perspectiva.





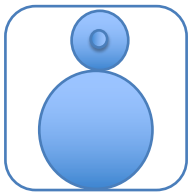


Penseu en càmera i com ha de quedar la imatge

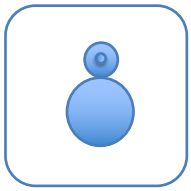
```
VM = lookAt(OBS,VRP,up);
viewMatrix(VM);
```

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

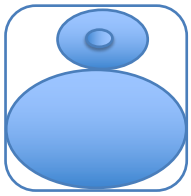
1. Posició, orientació
2. Òptica perspectiva



No

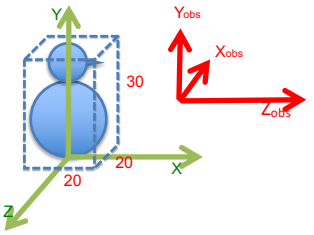


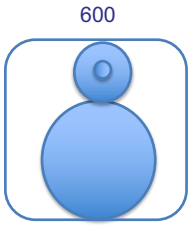
No



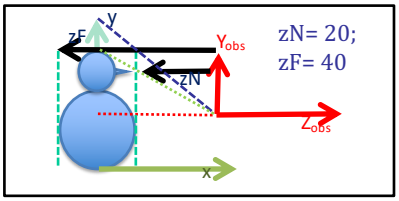
Viewport: tota la finestra gràfica 600x600
- Ninot optimitzi espai en viewport
- Sense deformacions

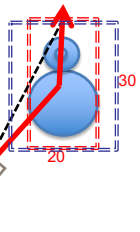
Exemple 1: Òptica perspectiva





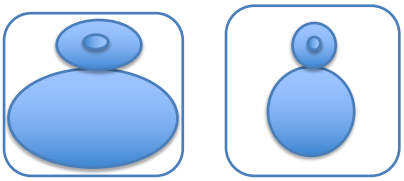
VRP=(0,15,0); OBS=(30,15,0), up=(0,1,0)





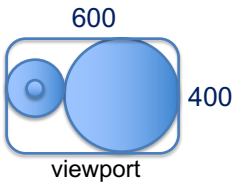
alfa = arctg (15/20) → alfa = 36,8º
ra_W = 20/30= 0,66

Com ra_v =1 → deformació
Solució ra_W =1

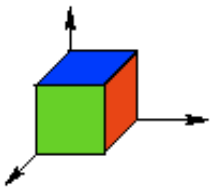


Per pensar…

- Òptica ortogonal
- Quins paràmetres de posicionament i òptica de càmera per a obtenir:



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.
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'obtindria.

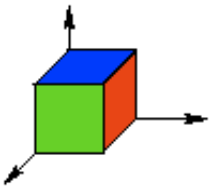


VRP= (2,1,2)
OBS= (3,1,3)
Up = (0,1,0)

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.

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.

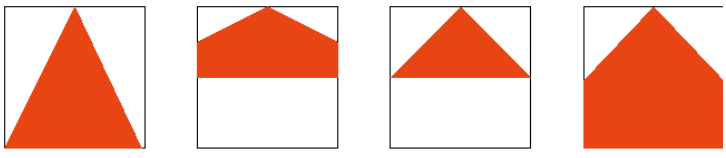
b) Quin efecte tindria en la imatge final modificar l'òptica ortogonal? Defineix la càmera ortogonal.



VRP= (2,1,2)
OBS= (3,1,3)
Up = (0,1,0)

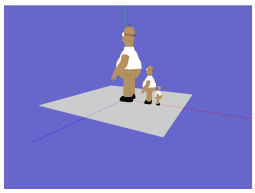
Exemple 3

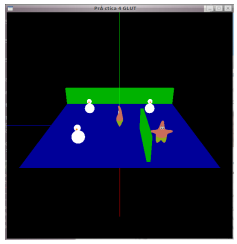
Tenim una escena amb un triangle vermell amb vèrtexs V1=(-2,0,0), V2 = (2, 0, 0) i V3= (0, 1, 0). Suposant que tenim un viewport quadrat de 600x600 píxels, i que hem inicialitzat les matrius de càmera (view) i projecció (proj) a la matriu identitat, indica quina de les següents imatges és la que sortirà en un viewport de 600x600 (sabem que el Vertex Shader i el Fragment Shader estan correctament implementats):



a) b) c) d)

Càmera 3ra persona





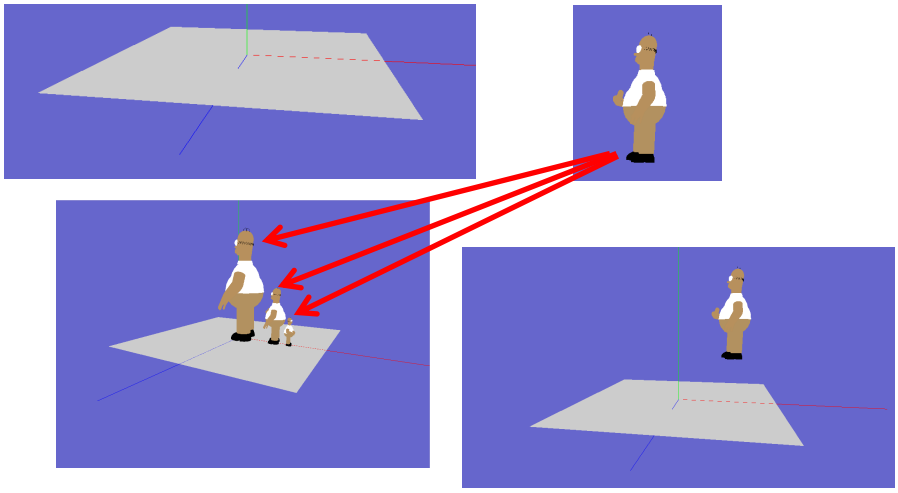
Quins paràmetres de posició, orientació i òptica per càmera en 3ra persona?
➔ - imatge inclogui tota l'escena (no retalli)
- centrada en viewport
- optimitzant ocupació del viewport
- sense deformació

Dades: capsa mínima contenidora d'escena
(xmin, ymin, zmin) - (xmax, ymax, zmax)

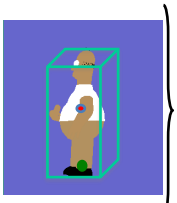
Classe 4: contingut

- Breu repàs càmera i el Procés de Visualització d'OpenGL
- Exercicis d'especificació de càmera
- **Ubicar models en escena en el Vèrtex Shader**
- Càmera en tercera persona

Recordatori de com generem escena



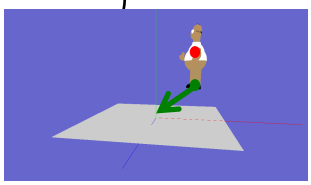
Transformació geomètrica → Matriu 4x4 TG



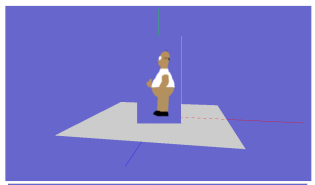
CapsaMinCont = (xmin,ymin,zmin,xmax,ymax,zmax)

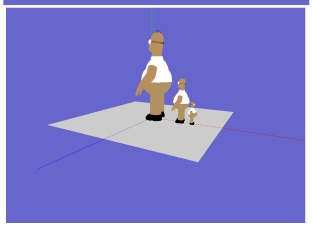
Mides => a = (xmax-xmin) , h= (ymax-ymin), f = (zmax-zmin)

CentBaseCapsa=(cbx,cby,cbz)=(xmin+xmax)/2,ymin,(zmin+zmax)/2)



TG_{H1}=Trans(t)
t = (-cbx,-cby,-cbz)



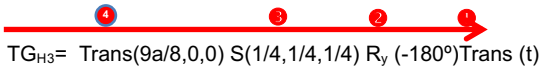


TG_{H2}= Trans(3a/4,0,0) S(1/2,1/2,1/2) Trans (t)

TG_{H3}= Trans(9a/8,0,0) S(1/4,1/4,1/4) R_y(-180) Trans (t)

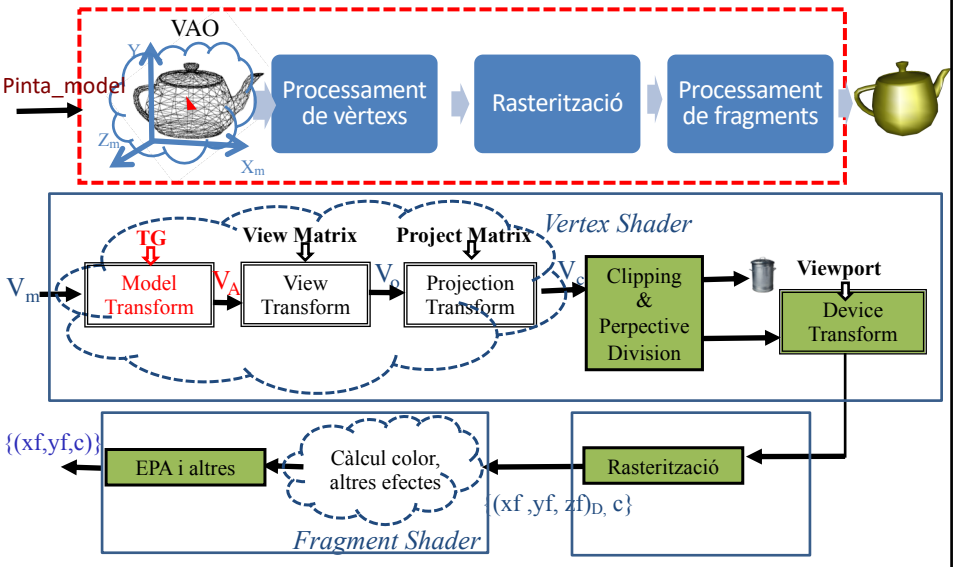
Visualització OpenGL

```
per cada objecte_i
  //Càlcul TG_i i enviar a OpenGL
  modelTransform_i()
  //pinta_homer() o pinta_terra()
  pinta_model_i();
fper
```




```
modelTransform_homer3(){
TG=I;
TG= TG*Translate(posx,posy,posz));
TG= TG*Scale(s,s,s);
TG= TG*Rotate (-180., (0.,1.,0.));
TG= TG*Translate (-cb.x,-cb.y,-cb.z);
modelMatrix(TG); //enviar uniform
}
```


Paradigma projectiu bàsic amb OpenGL 3.3



Programació mínima dels shaders

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);
}
```

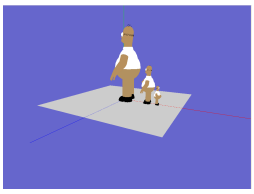
Fragment Shader

```
#version 330 core

out vec4 FragColor;

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


Classe 4: contingut

- Breu repàs càmera i el Procés de Visualització d'OpenGL
- Exercicis d'especificació de càmera
- Ubicar models en escena en el Vèrtex Shader
- **Càmera en tercera persona**

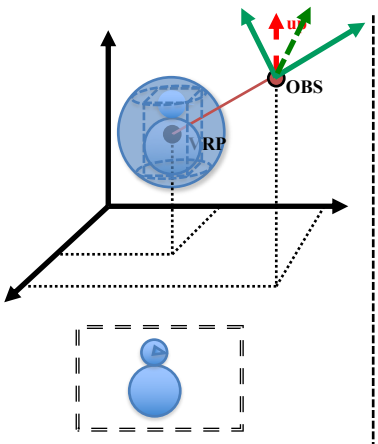
Càmera 3ra persona



Quins paràmetres de posició, orientació i òptica per càmera en 3ra persona?
➔ - imatge inclogui tota l'escena (no retalli)
- centrada en viewport
- optimitzant ocupació del viewport
- sense deformació

Dada: capsa mínima contenidora d'escena
cmin=(xmin, ymin, zmin) i cmax=(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)

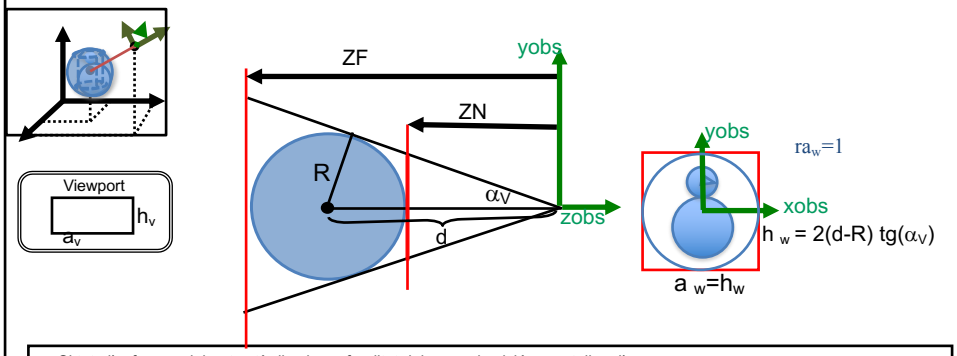
  - CentreEscena=Centre(CapsaMinCont) = ((xmax+xmin)/2,(ymax+ymin)/2,(zmax+zmin)/2))

  - 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)

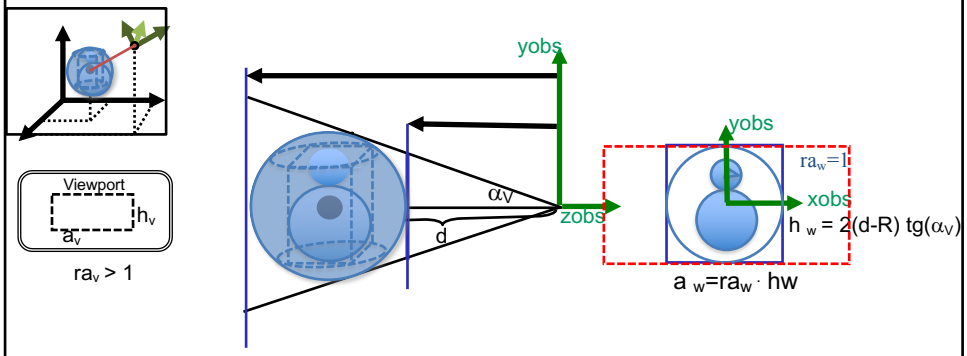
Tota escena en la vista, sense deformar i càmera perspectiva



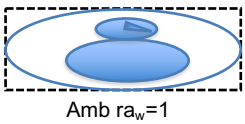
- Si tota l'esfera englobant està dins la profunditat del camp de visió, no retallem l'escena.
  Per tant, ZN ∈] 0, d-R] ZF ∈ [d+R, …]; per a aprofitar precisió profunditat: ZN= d-R; ZF=d+R
- Per a aprofitar al màxim la pantalla (de fet del viewport), el window de la càmera s'ha d'ajustar per veure tota l'escena; una aproximació és ajustar el window de la càmera per poder veure l'esfera englobant.
  - R = d sin (α_V) ; α_V = arc sin (R/d) => FOV=2*α_V
  - com el window està situat en ZN, α_V determina que la seva alçada sigui: $h_w = 2(d-R)\ tg(\alpha_V)$
- $ra_w = a_w/h_w = 1$ (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

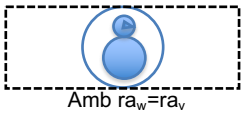


Cas amb Viewport $ra_v > 1$

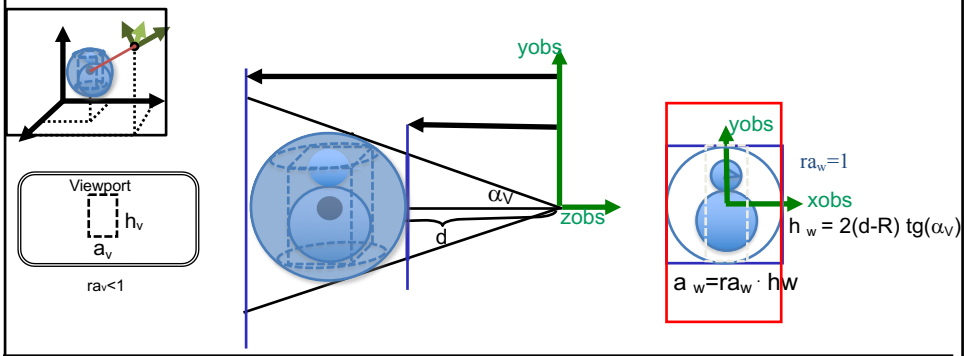

Amb $ra_w=1$

Si només **$ra_w=ra_v$** => no modifiquem h_w, simplement es força una nova $a^*_w > a_w$ mínima requerida, per tant, no es retalla

no cal modificar α_V (FOV)


Amb $ra_w=ra_v$

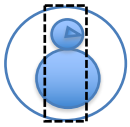
Tota escena en la vista, sense deformar i càmera perspectiva



Cas amb Viewport ra_v <1

Amb ra_w=1

Si només **ra_w=ra_v**
no toquem h_w => reduim a_w



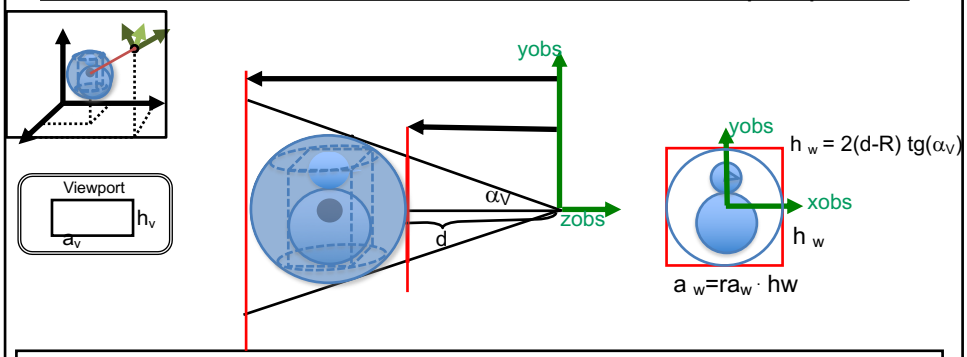
retallarà esfera!!!!

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

FOV=2 α^*_V on α^*_V = arctg(tg(α_V) / ra_v)

IDI 2018-2019 1Q

Tota escena en la vista, sense deformar i càmera perspectiva

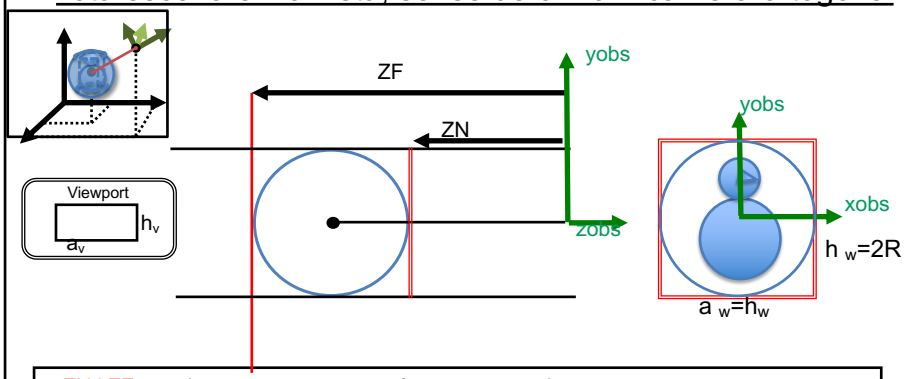


- Si ra_v > 1 (>ra_w mínima requerida 1) => No es retalla, no cal modificar α_V (FOV), només fer ra^*_w = ra_v
  Justificació: ra^*_w serà superior a 1; si no modifiquem l'angle FOV, h_w no canvia =>
  a^*_w= ra^*_w·h_w i com ra^*_w > ra_w => a^*_w > a_w i, per tant, serà
  més gran del necessari però es veurà tota l'esfera i quedarà espai pels laterals.

- Si ra_v<1 (<ra_w mínima requerida 1) => cal fer ra^*_w = ra_v i incrementar l'angle d'obertura
  FOV=2 α^*_V on α^*_V = arctg(tg(α_V) / ra_v)

  Justificació: *com* a^*_w= ra^*_w·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.
  - Com h^*_w = a_w/ ra_v i per trigonometria h^*_w=2(d-R) tg(α^*_V), igualant les eqüacions
    α^*_V = arctg(tg(α_V) / ra_v)

Tota escena en la vista, sense deformar i càmera ortogonal



- ZN i ZF mateix raonament que en càmera perspectiva.
- Window mínim requerit (centrat)= (-R,R,-R,R) => una $ra_W = 1$ (per què ?)

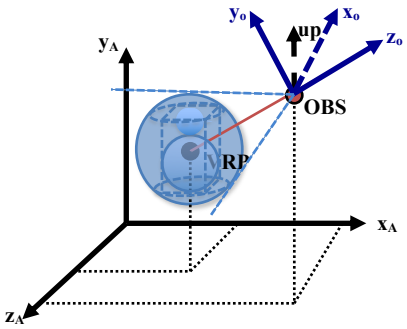
- Si $ra_w \neq ra_v$ ==> deformació (per què?)
  - Si $ra_v > 1$ => cal incrementar la ra_w => *modificar window*
    com $ra_w = a_w/h_w$ => podem incrementar a_w o decrementar h_w (és retallaria esfera!!)
    Per tant, modifiquem a_w:
    $a^*_w = ra_v * h_w = ra_v*2*R$
    window = $(-R\ ra_v, R\ ra_v, -R, R)$
- Raonament similar per recalcular window quan $ra_v < 1$

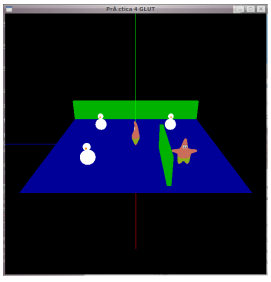
33

Vist...

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

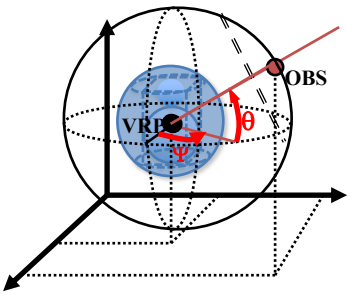
Com Moure la Càmera per inspeccionar escena?

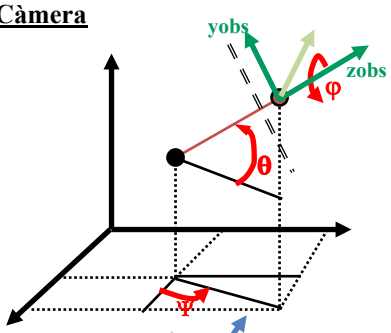


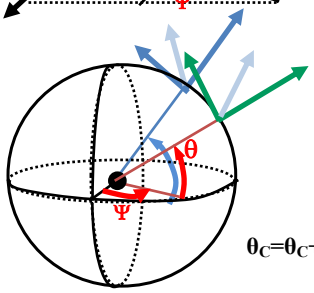


IDI 2018-2019 1Q

Moure la Càmera







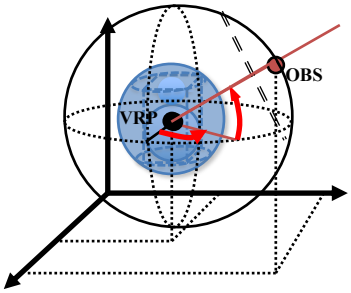
$\theta_C=\theta_C+f(\Delta y)$

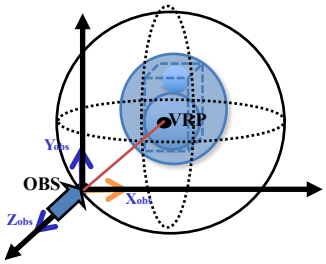
- Els angles (d'Euler) determinen la posició d'un punt en l'esfera
- Des de la interfície d'usuari desplacem el cursor dreta/esquerra (Ψ) i pujar/baixar (θ); per moure **OBS** sobre l'esfera

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

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


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.