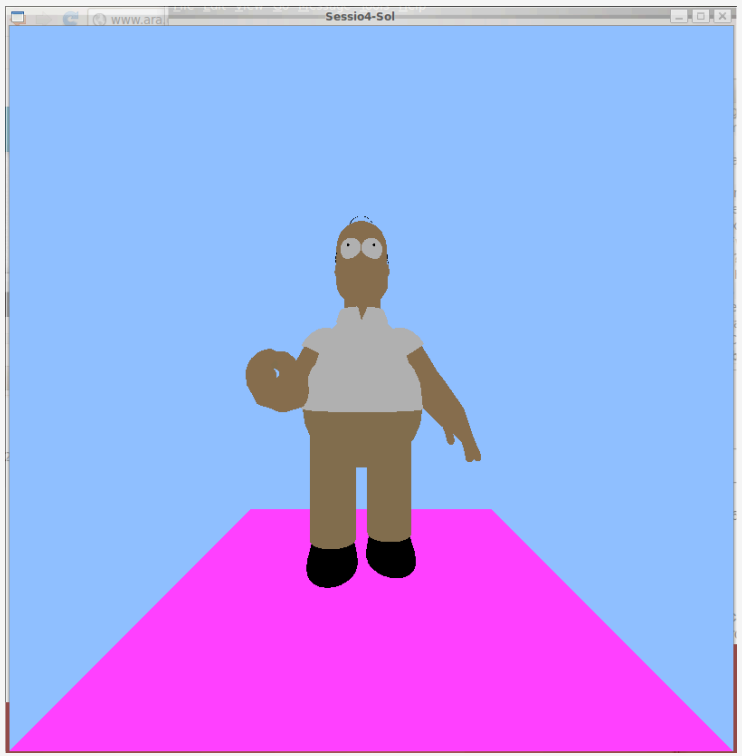
Laboratori OpenGL – Sessió 2.2

- Redimensionat finestra sense deformació (resize)
- Càlcul càmera per a visualitzar escena (càmera 3ª persona)
- Visualitzar objecte qualsevol
- View Matrix amb angles d'Euler
- Interacció per inspecció (amb angles d'Euler)

Redimensionat sense deformació

(exercici 1)

- Quan l'usuari redimensiona la finestra gràfica s'executa automàticament el mètode resizeGL ()
- Si aquest mètode només modifica el *viewport*:

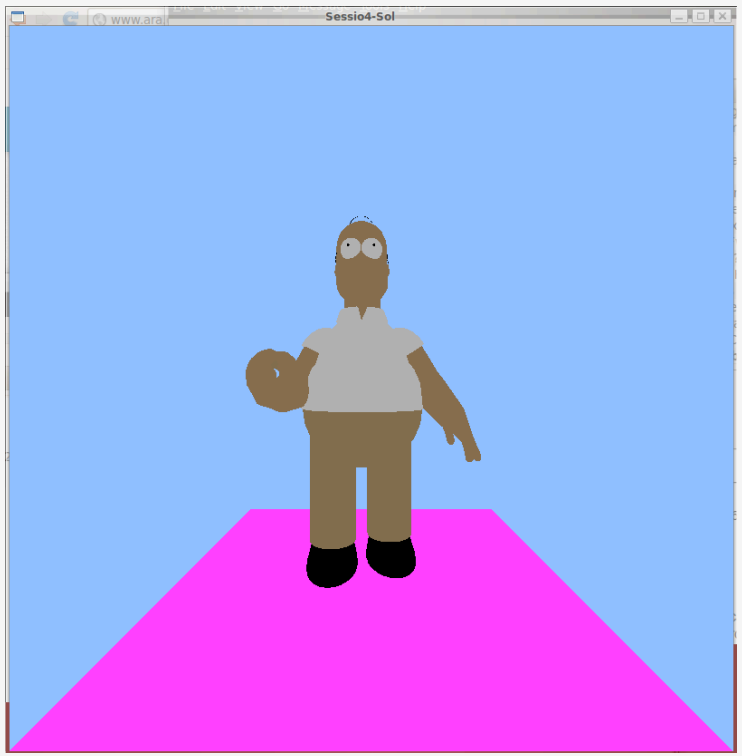


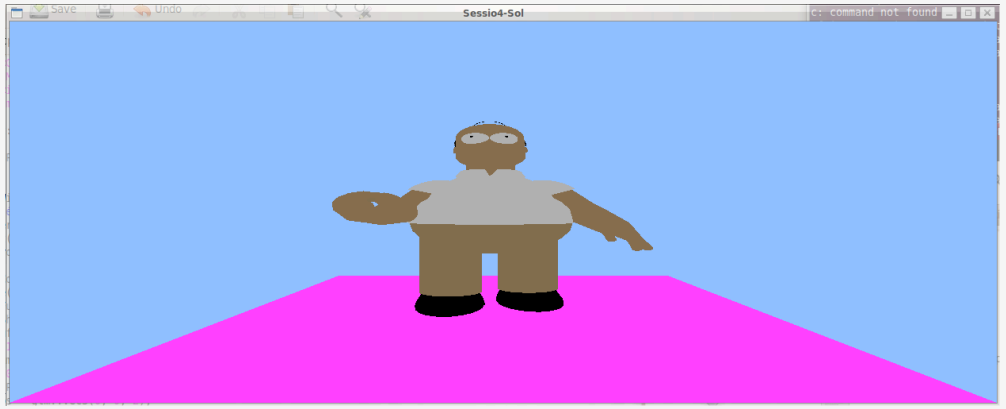

```
void MyGLWidget::resizeGL (int w, int h)
{
   glViewport(0, 0, w, h);
}
```

Redimensionat sense deformació

(exercici 1)

- Quan l'usuari redimensiona la finestra gràfica s'executa automàticament el mètode resizeGL ()
- Si aquest mètode només modifica el *viewport*:





Redimensionat sense deformació (exercici 1.a)

- La relació d'aspecte (ra) del window ha de ser igual que la del viewport: $ra_w = ra_v$
- Per tant si canvia la ra_v ➔ ha de canviar la ra_w ➔ refer perspective (...)

```
void MyGLWidget::resizeGL (int w, int h)
{
   glViewport(0, 0, w, h);
   float rav= float (w)/float (h);
   raw=rav;  //declarar global raw; podeu inicialitzar a 1
   projectTransform();
}
```

```
void MyGLWidget::projectTransform () {
    glm::mat4 Proj;
    Proj = glm::perspective ((float)M_PI/2.0f, raw, 0.4f, 3.0f);
     glUniformMatrix4fv (projLoc, 1, GL_FALSE, &Proj[0][0]);
 }
```

Redimensionat sense deformació
(exercici 1)

- El mètode resizeGL rep com a paràmetres l'amplada i alçada de la finestra gràfica
  - void resizeGL (int width, int height);

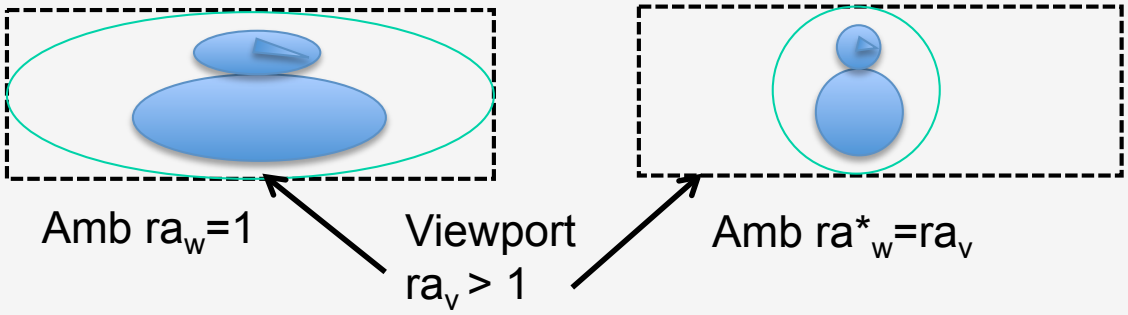
```
// càlcul de la relació d'aspecte del viewport
float ra = float (width) / float (height);
```

- Mètodes de QOpenGLWidget que ens poden ser útils:
  - width () → retorna amplada de la finestra gràfica (int)
  - height () → retorna alçada de la finestra gràfica (int)

Redimensionat sense deformació, ni retallat

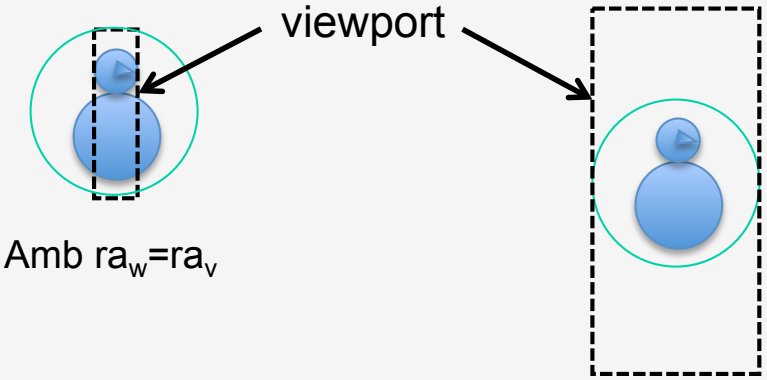
- La relació d'aspecte (ra) del window ha de ser igual que la del viewport: $ra_w = ra_v$
- Per tant si canvia la ra_v ➔ ha de canviar la ra_w ➔ refer perspective (...)

- 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)

- 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 $ra_w = ra_v$ i nou FOV
- FOV* = 2 α^*_V on α^*_V = arctg (tg (α_V) / ra_v)
- Sempre cal calcular el nou angle a partir de l'inicial (*window* quadrat).

Redimensionat sense deformació (exercici 1)

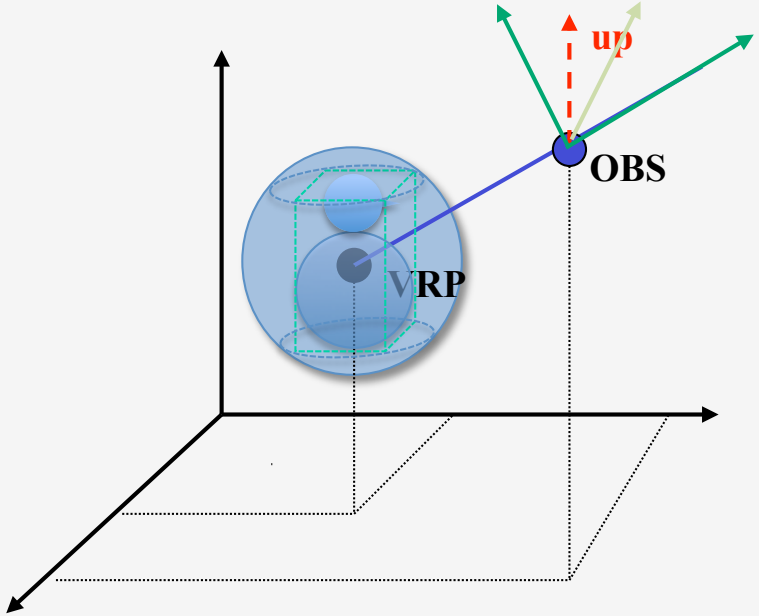
- La relació d'aspecte (ra) del window ha de ser igual que la del viewport: $ra_w = ra_v$
- Per tant si canvia la ra_v ➔ ha de canviar la ra_w ➔ refer perspective (...)

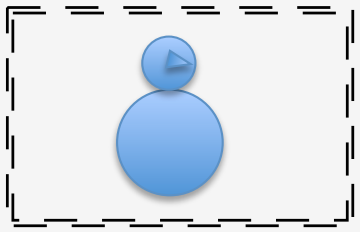
```
void MyGLWidget::resizeGL (int w, int h)  {
   glViewport(0, 0, w, h);
   float rav= float (w)/float (h);
   raw=rav; FOV_a=FOV;
   if (rav < 1.)
      { //declarar globals  FOV i FOV_a i FOV inicialitzat a (float)M_PI/2.0f
        FOV_a = 2* atan (tan (FOV/2.)/rav); //no modifiqueu FOV
      }
   projectTransform();
}
```

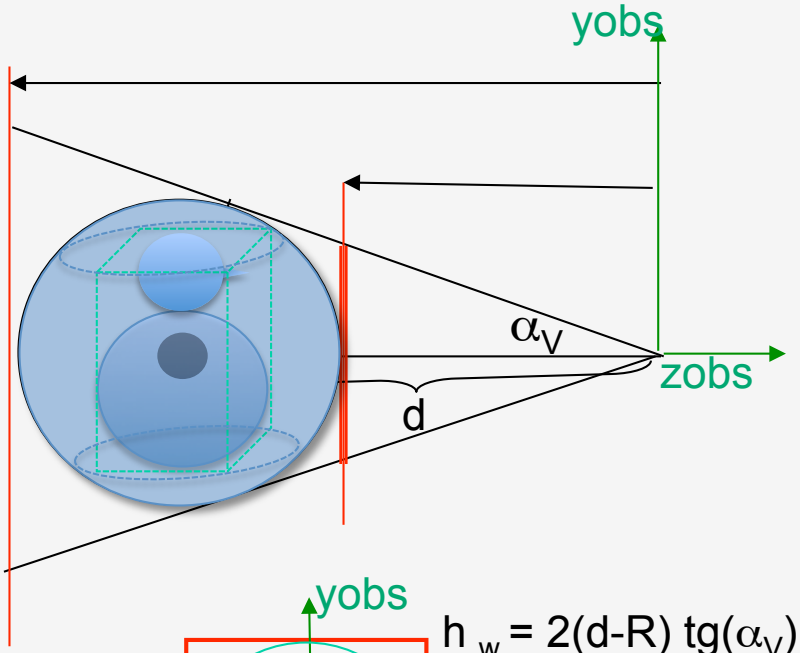
```
void MyGLWidget::projectTransform () {
    glm::mat4 Proj;
    Proj = glm::perspective (FOV_a, raw, 0.4f, 3.0f);
     glUniformMatrix4fv (projLoc, 1, GL_FALSE, &Proj[0][0]);
 }
```

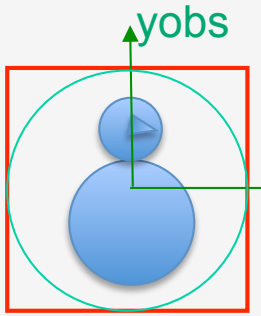
Càmera Inicial en 3ª persona (exercicis 2 i 3)

- Considerar la capsa (i esfera) mínima contenidora de l'escena
- Càlcular els paràmetres de posició i orientació (OBS,VRP,Up)
- Calcular els paràmetres de l'òptica perspectiva (FOV, raw, ZN, ZF)









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

$a_w = ra_w \cdot hw$

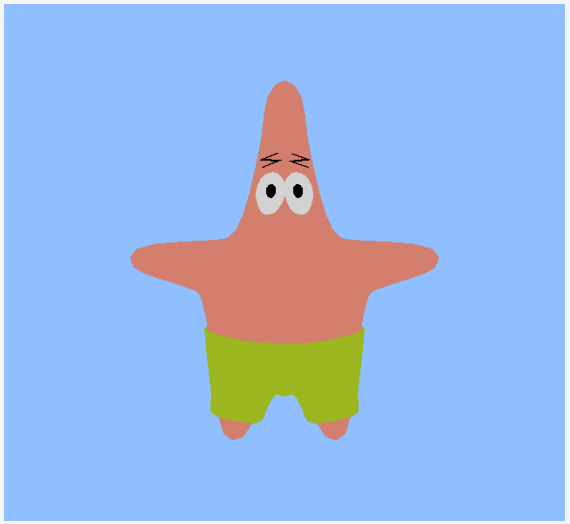
Consells d'estructuració de codi (1)

```
void MyGLWidget::initializeGL ()
{
  initializeOpenGLFunctions();
  glClearColor(0.5, 0.7, 1.0, 1.0);
  carregaShaders();
  createBuffers();
  glEnable (GL_DEPTH_TEST);

    iniEsfera();  //inicialitza el radi i centre esfera a partir de capsa escena
    iniCamera (); //inicialitza càmera inicial OBS, VRP, up, FOV, zN, zF, raw
    viewTransform();
}
```

Proveu com es veu l'escena que tenia-ho, si voleu proveu modificar vector up i/o altres paràmetres i veure efecte

Pintar objecte qualsevol
(exercici 4)

- Pintem el Patricio.obj
  - Model no centrat a l'origen i de mides no controlades (decisió del dissenyador del model)
  - ➢ Cal calcular la capsa contenidora del model
  - Es vol el model **sense escalar** i **centrat a l'origen** de coordenades
  - ➢ Cal afegir transformacions de model necessàries per a centrar el model



Consells d'estructuració de codi (2)

- Feu un mètode:

  void MyGLWidget::CalculCapsaPatr()

  que calcula la capsa mínima contenidora del model (patricio) i paràmetres requerits per a calcular la TG que el porta a la posició requerida.

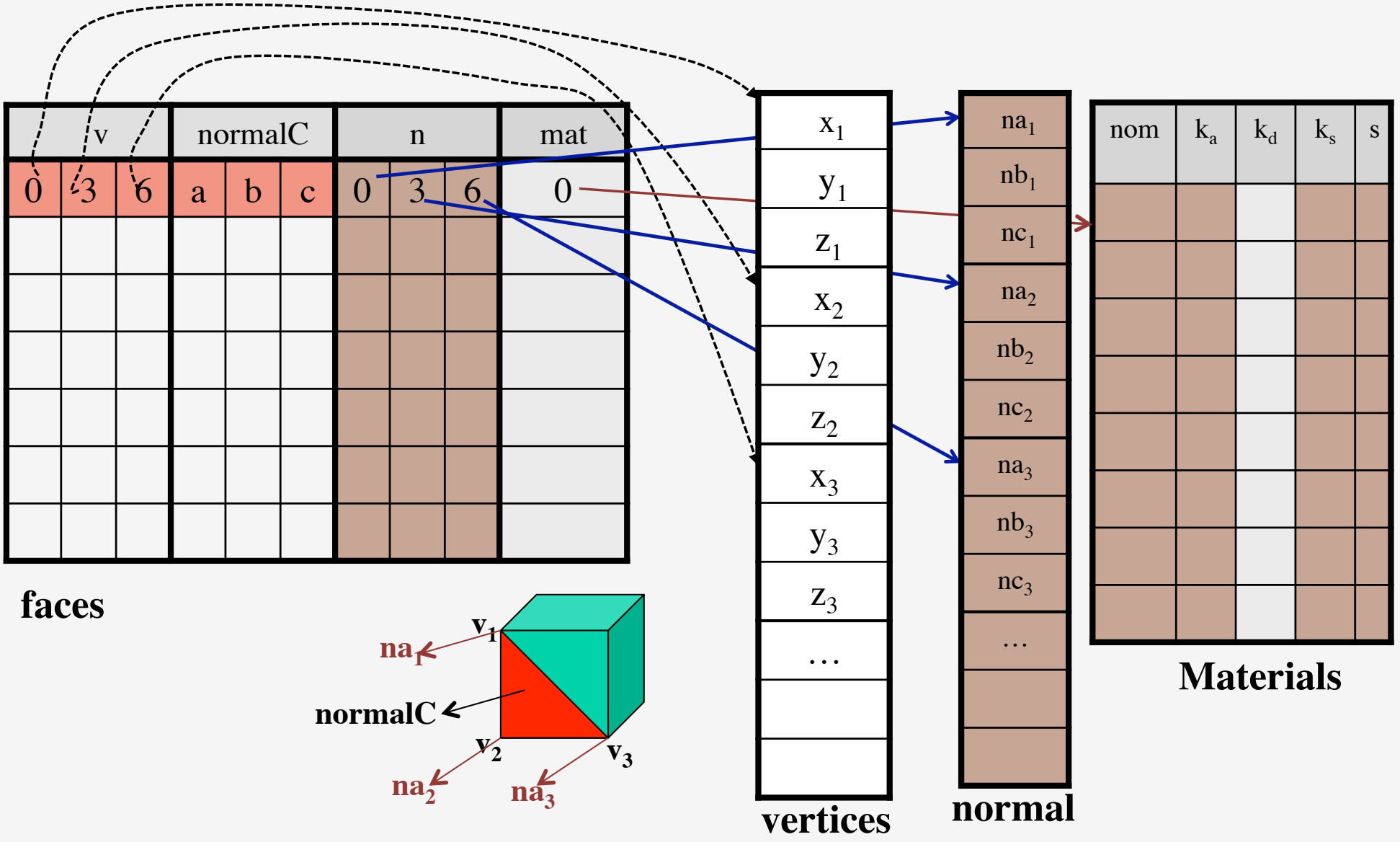
  Pel càlcul de la capsa, recordeu transpes de la sessio 2.1 de la clase Model

- Utilitzeu els paràmetres calcluats per a fer una

  void MyGLWidget::modelTransformPat ()

  que calcula la TG i envia uniform

Representació classe Model



Analitzeu l'arxiu model.h

Compte!! amb el nom dels camps de Material que en l'esquema són simbòlics; p.e. k_d és float diffuse[4]

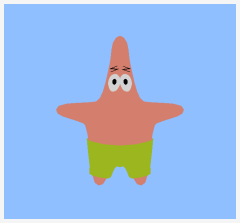
Exemple de recorregut

- Recorregut de la taula de vèrtexs

```
for (unsigned int i = 0; i < m.vertices().size(); i+=3) {
   // escric per pantalla les coordenades del vèrtex
   std::cout << "(x, y, z) = (" << m.vertices()[i] << ", "
                                  << m.vertices()[i+1] << ", "
                                  << m.vertices()[i+2] << ")" << std::endl;
}
```

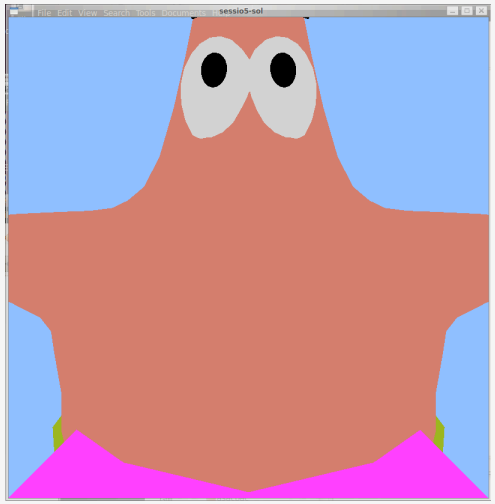
Pintar objecte qualsevol

(exercici 4)



- Pintem el Patricio.obj
  - Model no centrat a l'origen i de mides no controlades (decisió del dissenyador del model)
  - ➢ Cal calcular la capsa contenidora del model
  - Es vol el model **sense escalar** i **centrat a l'origen** de coordenades
  - ➢ Cal afegir transformacions de model necessàries per a centrar el model

Amb càmera actual veureu imatge de la dreta

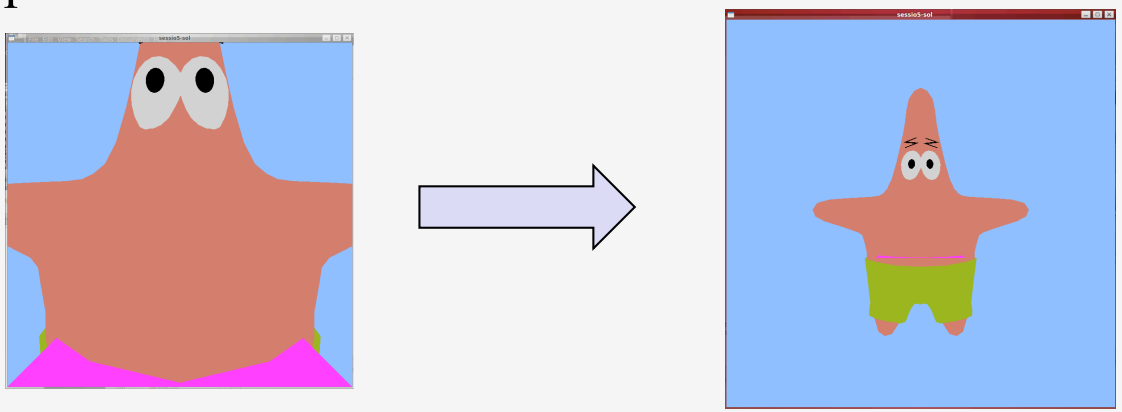


Pintar objecte qualsevol (exercici 4)

- **Recalculem càmera per veure senser**
  - Model del Patricio no hi cap a la càmera que tenim
  - ➢ Cal recalcular els paràmetres (de posició i orientació i òptica) de la càmera perspectiva per a veure'l sencer i ocupant el màxim del *viewport.*

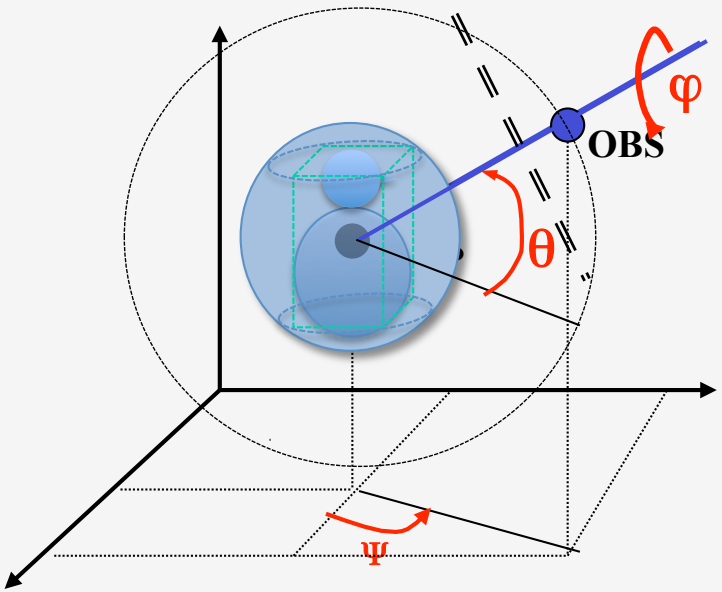
    tingueu en compte que la capsa de l'escena és ara la del Patricio però centrada a l'origen => ***a partir de les mides de la capsa, calculeu la capsa de l'escena***

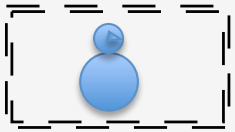
    (si codi ok estructurat, només cal modificar capsa de l'escena)
  - Què ha passat amb el terra?
  - ➢ No el pintem



Transf. *view* amb angles d'Euler

(exercici 5)





```
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.

Consells d'estructuració de codi (3)

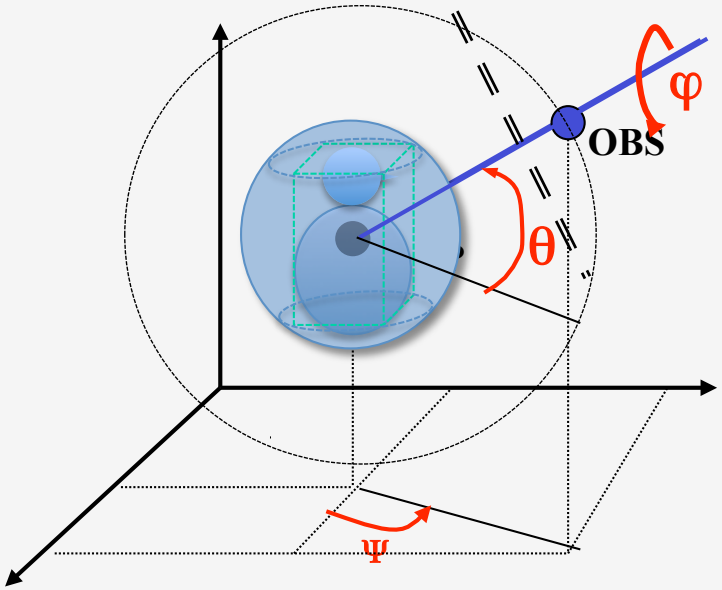
- En ini_càmera inicialitzeu angles i VRP i d si no els teniu inicialitzats

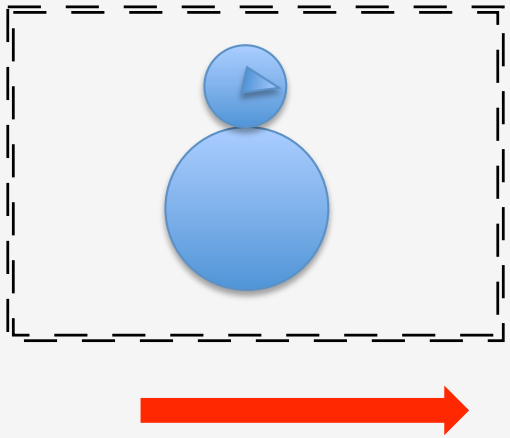
Feu una nova viewTranform()

```
void MyGLWidget::viewTransform()
{
   glm::mat4 VM (1.0f);

   //aquí codi de transformacions

  glUniformMatrix4fv(viewLoc, 1, GL_FALSE, &VM[0][0]);
}
```

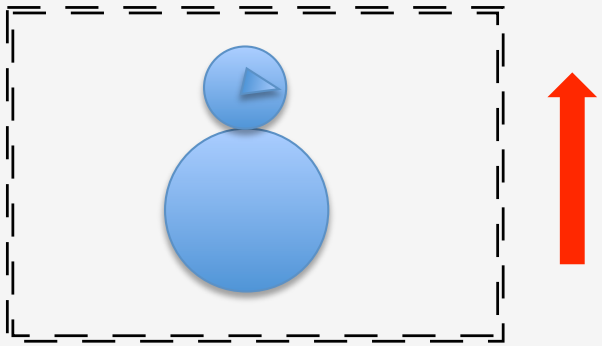
Interacció amb angles d'Euler

(exercici 6)





Moviment del ratolí d'esquerra a dreta ➔ increment angle Ψ



Moviment del ratolí de baix a dalt ➔ increment angle θ

Interacció directa amb Qt

- Per tal de tractar events de baix nivell en una aplicació OpenGL amb Qt cal re-implementar els mètodes virtuals corresponents (a la classe MyGLWidget):

virtual void mousePressEvent (QMouseEvent * e)

virtual void mouseReleaseEvent (QMouseEvent * e)

virtual void mouseMoveEvent (QMouseEvent * e)

virtual void keyPressEvent (QKeyEvent * e)

En MyGLWidget.h caldrà: #include <QMouseEvent>
i declarar el mètodes virtuals

Utilitzem moviment ratolí per Euler, com fer-ho?

Interacció directa amb Qt

- Exemple d'implementació:

```
void MyGLWidget::mousePressEvent (QMouseEvent *e)
{
   makeCurrent();
   xClick=e->x();
   yClick = e ->y();
   if ( e->buttons() == Qt::LeftButton )
       DoingInteractive=ROTATE;

}
```

Tenir diferents "estats" d'interacció, caldrà haver declarat en .h

typedef enum {ROTATE, NONE, ZOOM} InteractiveAction;

InteractiveAction DoingInteractive;

Interacció amb angles d'Euler

Es vol que el moviment de càmera es faci prement el **botó esquerre** del ratolí, i no qualsevol.

- Si volem controlar el botó del ratolí que s'usa:

```
if ( e->buttons() == Qt::LeftButton )  // e és QMouseEvent
```

- Si volem controlar que a més no s'ha usat cap modificador (Shift, Ctrl, Alt):

```
if ( e->buttons() == Qt::LeftButton &&
     ! ( e->modifiers() &
         ( Qt::ShiftModifier | Qt::AltModifier | Qt::ControlModifier ) ) )
// controla que s'ha premut botó esquerre i cap modificador
```

Interacció directa amb Qt

- Exemple d'implementació:

```
void MyGLWidget::mouseMoveEvent (QMouseEvent *e)
{
    makecurrent();
     if (DoingInteraction==ROTATE) {
          psi+=( e->x() – xClick)*M_PI/100.0;
          …
          xclick=e->x();
          viewTransform();
     }

      update();
}
```

Interacció directa amb Qt

- Exemple d'implementació:

```
void MyGLWidget::mouseReleaseEvent (QMouseEvent *e)
{
    makecurrent();
    DoingInteraction=NONE;
}
```

Consells d'interacció amb ratolí

- No oblideu cridar a makeCurrent() sempre que entreu en un mètode de Qt que tracta un event
- 1) mousePressEvent(…)
  - Identificar posició ratolí
  - Identificar el botó premut => identificar l'acció d'interacció al moure el cursor (**inspecció**, escalat, zoom,…)
- 2) mouseMoveEvent ()
  - Recollir posició cursor ; càlcul increment de posició
  - Segons tipus d'interacció: **actualitzar angles**, escalat, FOV,…
- 3) mouseReleaseEvent()
  - No volem cap tipus d'interacció al moure el ratolí

Interacció amb angles d'Euler
(exercici 6)

Es vol que el moviment de càmera es faci prement el **botó esquerre** del ratolí, i no qualsevol.

- Si volem controlar el botó del ratolí que s'usa:

```
if ( e->buttons() == Qt::LeftButton )  // e és QMouseEvent
```

- Si volem controlar que a més no s'ha usat cap modificador (Shift, Ctrl, Alt):

```
if ( e->buttons() == Qt::LeftButton &&
     ! ( e->modifiers() &
         ( Qt::ShiftModifier | Qt::AltModifier | Qt::ControlModifier ) ) )
// controla que s'ha premut botó esquerre i cap modificador
```