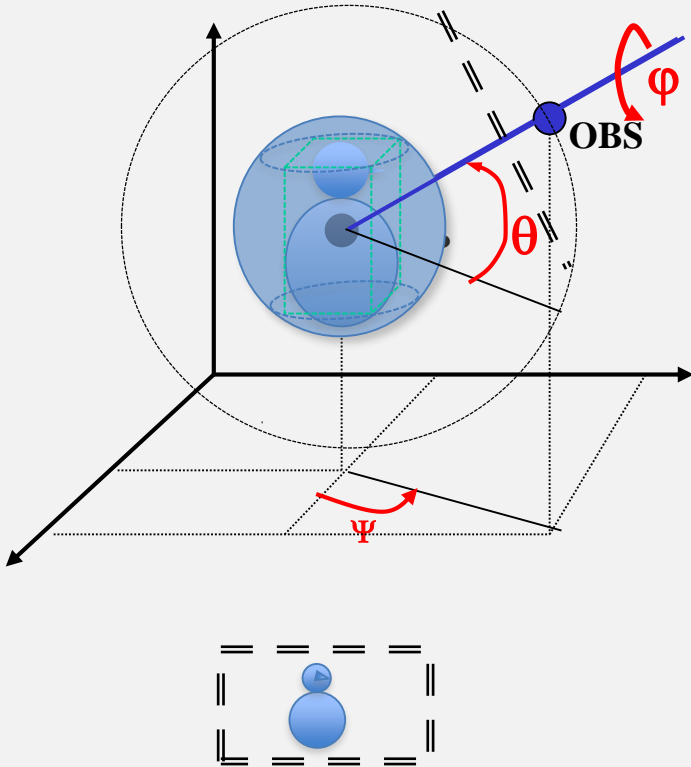


# Laboratori OpenGL – Sessió 2.3

- View Matrix amb angles d'Euler
- Interacció per inspecció (amb angles d'Euler)
- Zoom (òptica perspectiva i ortogonal -opcional-)
- Animacions amb QTimer (opcional)
- Creació d'una escena més complexa

# Transf. *view* amb angles d'Euler

(exercici 1)



```
VM=Translate (0.,0.,-d)
VM=VM*Rotate(-\phi,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)
```

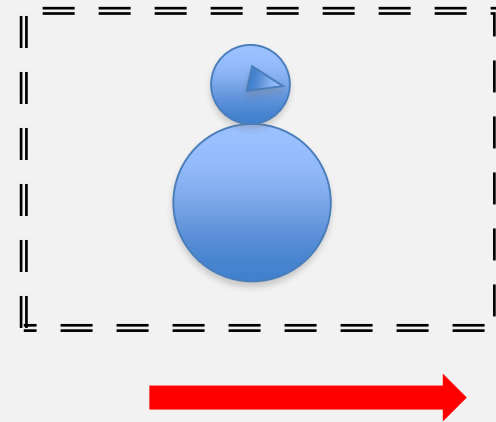
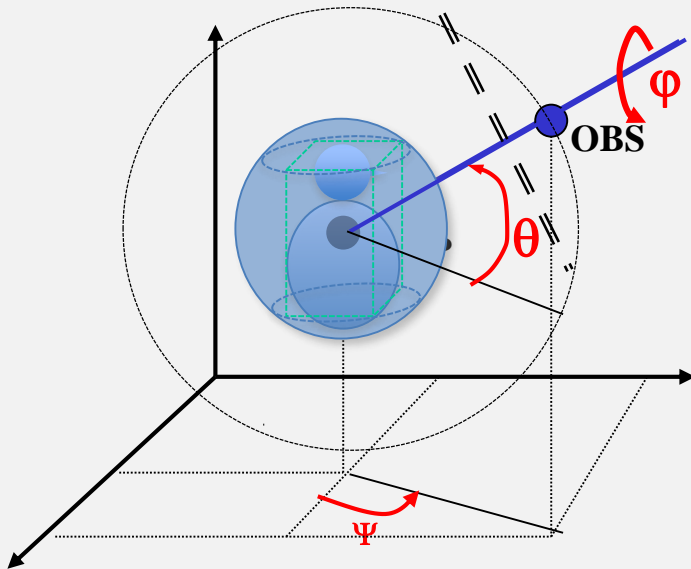
**Atenció a l'ordre!**

Compte amb signes:

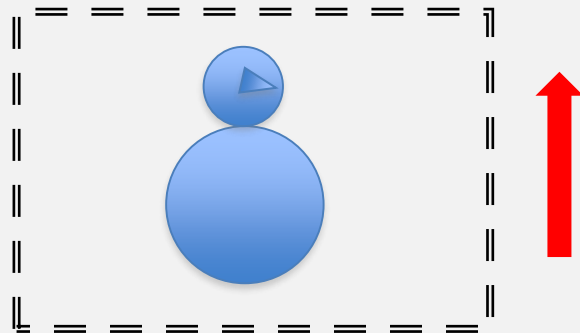
- Si s'ha calculat  $\psi$  positiu quan càmera gira cap a la dreta, serà un gir anti-horari respecte eix Y de la càmera, per tant, matemàticament positiu; com girem els objectes en sentit contrari, cal posar  $-\psi$  en el codi.
- Si s'ha calculat  $\theta$  positiu quan pugem la càmera, serà un gir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

# Interacció amb angles d'Euler

## (exercici 2)



Moviment del ratolí d'esquerra a dreta → increment angle  $\psi$



Moviment del ratolí de baix a dalt → increment angle  $\theta$

# Interacció amb angles d'Euler

## (exercici 2)

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
```

# Zoom

## (exercici 3)

- Per fer un zoom ho farem modificant l'angle d'obertura de la càmera (FOV)
  - Zoom-in → decrementar l'angle FOV (tecla 'Z')
  - Zoom-out → incrementar l'angle FOV (tecla 'X')
- Per a càmera ortogonal (opcional):
  - Modificar el window (left, right, bottom, top) mantenint ra

# Animacions (QTimer)

Per afegir animacions automàtiques a les nostres aplicacions:

- Usarem la classe QTimer de QT

```
#include <QTimer>
```

- Declarem un atribut de la nostra classe d'aquest tipus

```
QTimer timer;
```

- I definim un nou “slot” a la nostra classe

```
public slots:
```

```
void animació ();
```

# Animacions (QTimer)

Per afegir animacions automàtiques a les nostres aplicacions:

- Cal afegir lligam en codi cpp  
(només 1 cop –constructor o initializeGL)  
`connect (&timer, SIGNAL (timeout()), this, SLOT (animació ()));`
- Decidim quan començar i quan acabar animació  
`timer.start (16); // s'activa cada 16 milisegons (60 cops per segon)`  
`timer.stop (); // atura animació`
- Exemple rutina per animació  

```
void animació () {  
    makeCurrent ();  
    angleGir += increment; // exemple del que es vol animar  
    update ();  
}
```

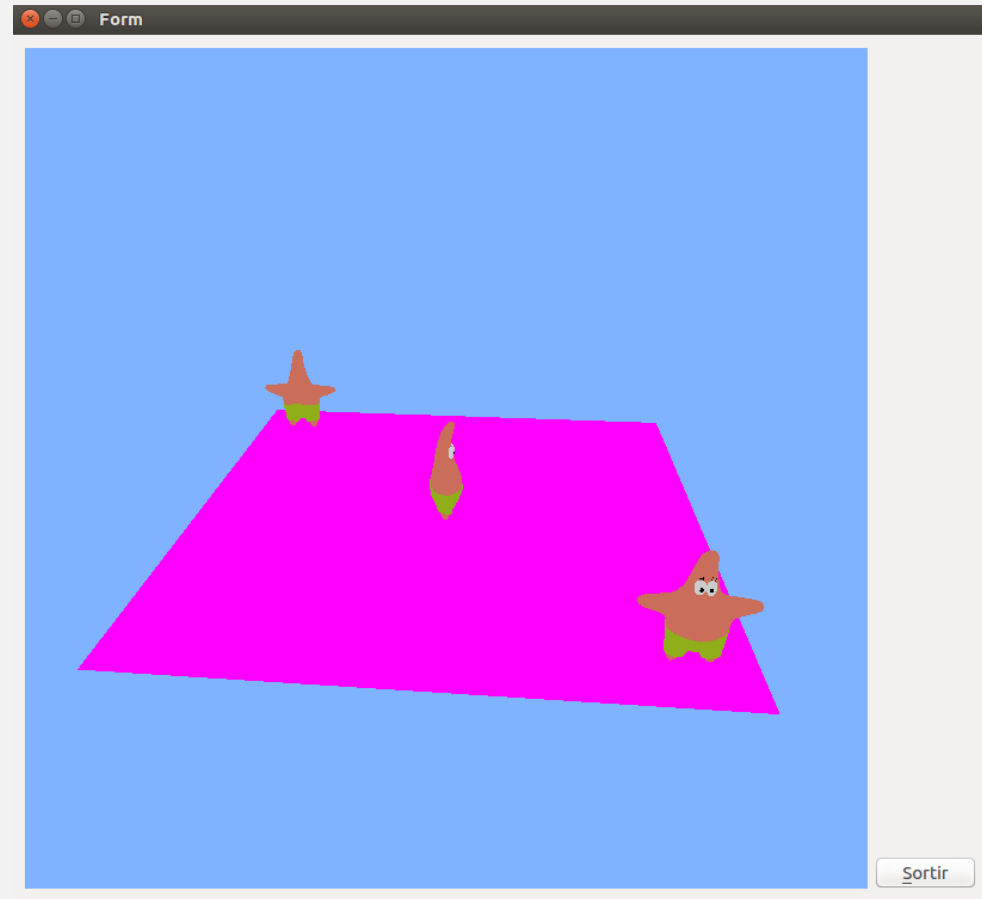
# Escena completa (exercici 4)

Modifiquen la vostra escena per veure el que es veu a la imatge.

Nova escena formada per:

- Terra de 5x5 centrat al  $(0,0,0)$
- Tres Patricios d'alçada 1 amb centres base en  $(2,0,2)$ ,  $(0,0,0)$  i  $(-2,0,-2)$ . El primer direcció Z+, el segon direcció X+ i el tercer direcció Z-

Calen paràmetres de càmera per veure-ho tot (3<sup>a</sup> persona)





# Exercici extra (animació)

Exercici opcional per provar animació amb QTimer:

- a) Afegiu a la nova escena de l'exercici 4 un quart Patricio de la mateixa mida i amb el centre de la seva base a la posició (1.5, 0, 0) mirant cap a Z+.
- b) Fes que aquest Patricio giri constantment al voltant de l'eix Y de l'escena (farà voltes al voltant del Patricio que està al (0,0,0)).