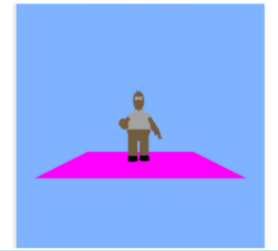
**Laboratori Càmera (I)**

**Objectius**

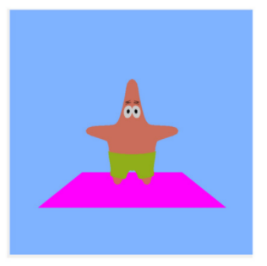
Objectiu 1:

Calcular una càmera per a veure aquesta escena completa, centrada, ocupant el màxim del viewport i sense deformar.



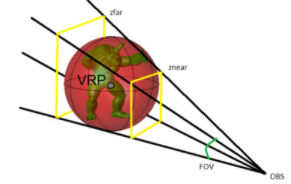
Objectiu 2:

El mateix amb un objecte diferent.



**Paràmetres de càmera en tercera persona**

* Cal calcular els paràmetres de la càmera adequadament.
* Treure els valors hard-coded de lookAt i perspective.
* Els paràmetres han de ser variables (no constants).
* Quan arrenca l’aplicació s’han d’inicialitzar els paràmetres.
* Cada cop que algun paràmetre canvia, caldrà reconstruir les matrius.
* Els paràmetres es calculen a partir de la definició de l’escena: l’esfera englobant.

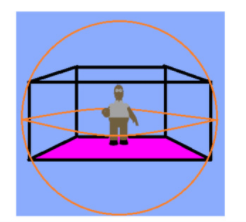
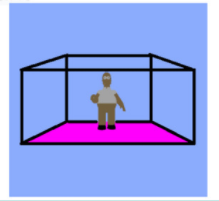


Per **lookAt** cal calcular: glm::lookAt(OBS, VRP, up)

* OBS.
* VRP.

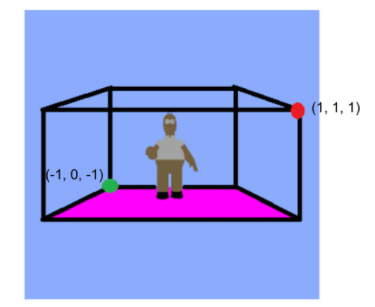
Per **perspective** cal calcular: glm::perspective(FOV, ra, znear, zfar)

* FOV.
* ra del window.
* znear.
* zfar.

****

HomerProves està situat a prop de l’origen, cal afegir un terra a sota dels seus peus. I calcularem els paràmetres a partir de l’esfera englobant de l’escena, que es calcula a partir de la caixa englobant.

La caixa englobant (o contenidora) conté tota l’escena al seu interior. I només calen dos punts per a definir una caixa: el mínim i el màxim.



Punt verd => punt mínim; punt vermell => punt màxim;

A partir dels punts mínim i màxim de la caixa podem calcular l’esfera contenidora.

Centre de l’esfera:

glm::vec3 Pmin, Pmax, centre;

**centre = (Pmin + Pmax) / 2;**

**VRP = centre;**

Radi de l’esfera:

radi = glm::distance (Pmin, Pmax) / 2;

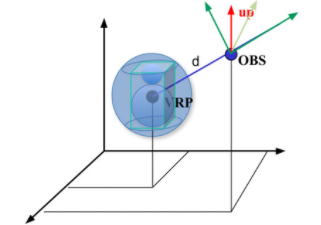
radi = glm::distance (centre, Pmin);

radi = glm::distance (centre, Pmax);

(el radi ha de ser una variable en el codi)

OBS:

L’observador es pot calcular a partir del VRP i donant una direcció i una distància d.

Per exemple: **OBS = VRP + d\*(0, 0, 1)**. OBS quedarà sobre l’eix z.

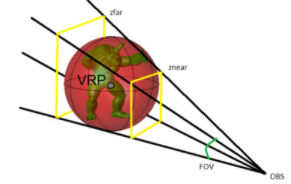
La d ha de ser més gran que el radi de l’esfera.

De moment tenim:

* Pmin, Pmax: punts mínim i màxim de la caixa.
* VRP = centre de la caixa.
* Radi de l’esfera contenidora (R).
* d > R (per exemple, d = 2\*R, d = 3\*R).
* OBS calculat a partir de VRP i d, amb una direcció arbitrària.

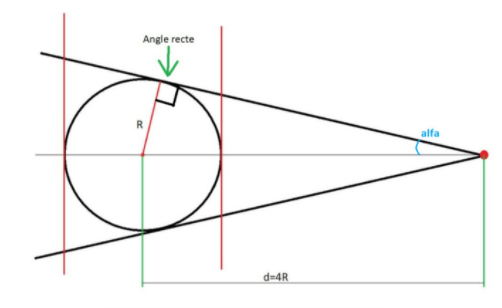
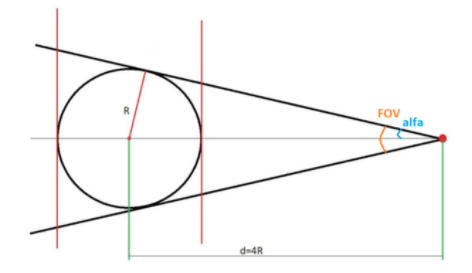
Falta:

* FOV.
* znear i zfar (distàncies des de l’observador).
* ra del window.



Càlcul de FOV:

Posem vista lateral, FOV = 2\*αinicial.



Per calcular α, tenim:

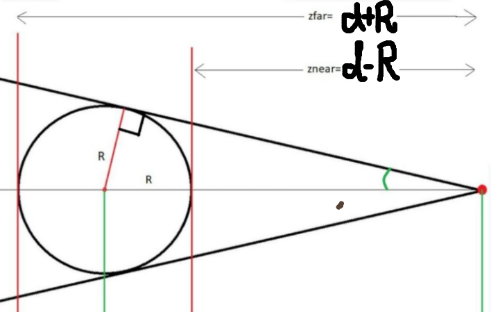
* R catet oposat.
* d hipotenusa.

Per tant:

Sin (α) = R / d.

**αinicial = asin (R / d).**

**FOV = 2\*asin(R / d).**

znear i zfar:

znear = d – R;

zfar = d + R;

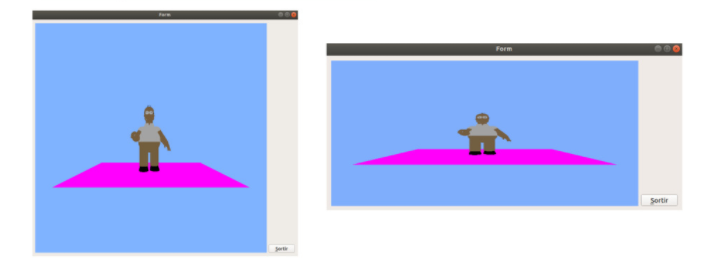
Valors en hard-coded: Pmin, Pmax (els calcularem a mà a partir de l’escena) i d (que pot ser d = 2\*R, d = 3\*R...).

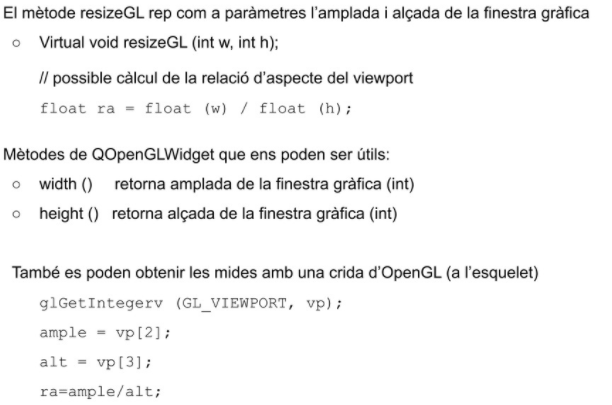
La resta de paràmetres es calcula a partir d’aquests: VRP, R, OBS, FOV, znear i zfar.

**Resize**

Ja es pot veure l’escena completa i centrada, però falta que ocupi tot el viewport i no es deformi quan es redimensioni la finestra.

Quan l’usuari redimensiona la finestra gràfica s’executa automàticament el mètode resizeGL(). Si aquest mètode no fa res, la imatge es deforma:

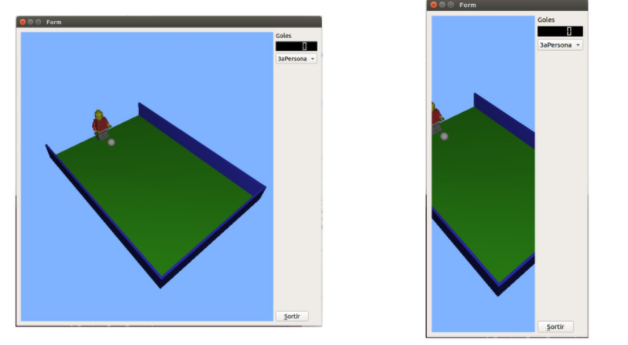




Cal calcular l’aspect ratio del viewport al resize i guardar-lo en una variable (ra). Per evitar deformacions l’única cosa que cal és que raw = rav. Llavors només cal passar l’aspect ratio del VIEWPORT a la funció perspective, perspective (FOW, ra, znear, zfar).

Recordatori: les funciones que inicialitzen la càmera (lookAt i perspective) s’han de cridar almenys un cop per crear les matrius.

Amb això es soluciona les deformacions, però no els retallats (quan ra < 1):



Per evitar retallat mirarem al resize com és ra:

* ra >= 1 (ample > alt):
  + Cal deixar FOV amb el valor inicial.
  + Passar ra a perspective i reconstruir la matriu.
* ra < 1 (ample < alt):
  + Recalcular l’angle d’obertura.
  + FOV = 2 \* αnou αnou = atan (tan (αinicial) / ra).
  + Passar FOV i ra a perspective i reconstruir la matriu.

ATENCIÓ: Per evitar zooms estranys quan ra >= 1, deixarem el FOV amb el valor inicial.

El càlcul de FOV està deduït per la meitat de l’angle. OpenGL demana el doble. FOV = 2 \* αinicial.

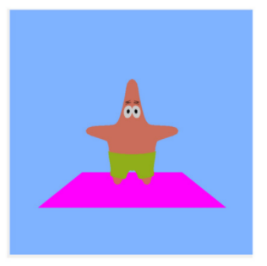
El càlcul del nou alfa es fa a partir del primer alfa que es va calcular per a inicialitzar la càmera (αinicial), no a partir del alfa del FOV del resize anterior.

Al mètode resizeGL() arriba per paràmetre l’amplada (w) i l’alçada (h) del viewport.

Per consultar l’amplada i l’alçada es pot fer servir les crides width() i height() des de MyGLWidget (retornen int):

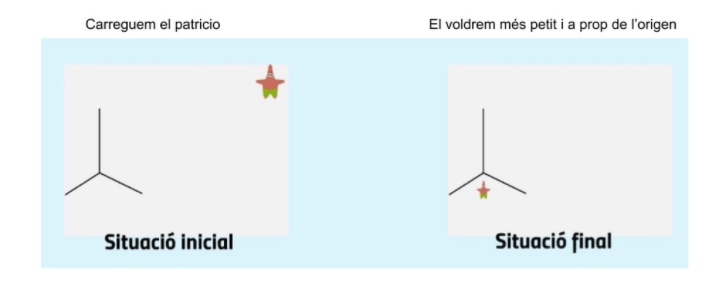
double ra = double(width() / double(height());

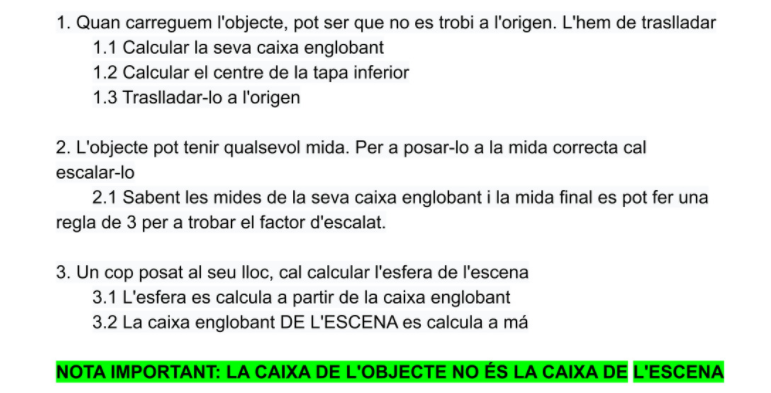
**Visualitzar un objecte qualsevol**

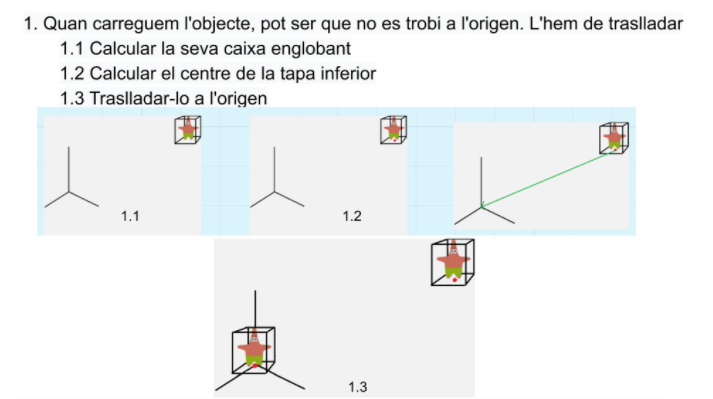


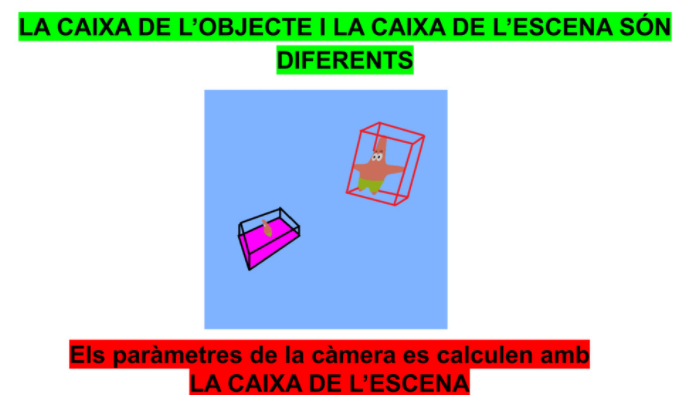
Substitució del HomerProves pel Patricio. Calcular els paràmetres de la càmera per a visualitzar un objecte qualsevol.

Quan es carrega un objecte no té perquè estar centrat a l’origen i pot tenir la mida que el dissenyador vulgui: pot estar situat molt lluny de l’origen i molt gran.

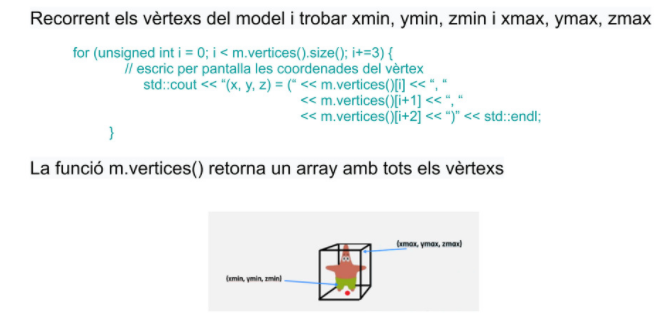


****





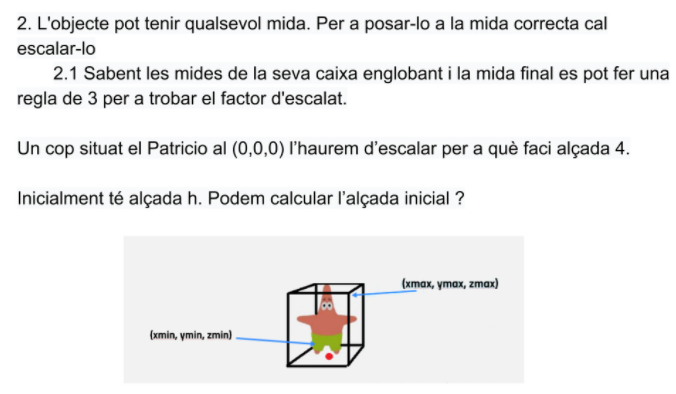
Càlcul dels punts mínim i màxim de la caixa englobant d’un model qualsevol:



El centre de la tapa inferior té coordenades:

(xmin + xmax) / 2, ymin, (zmin + zmax) / 2

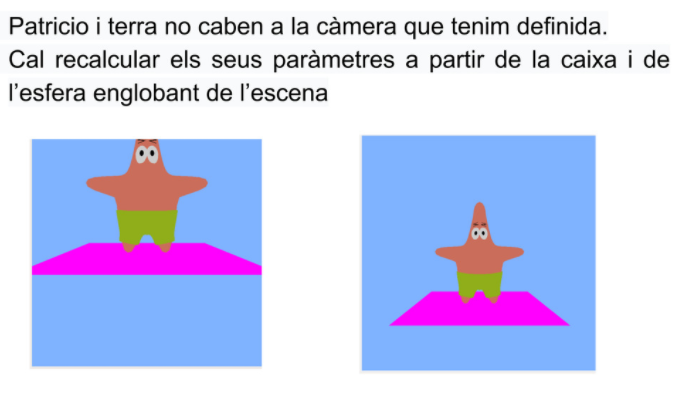
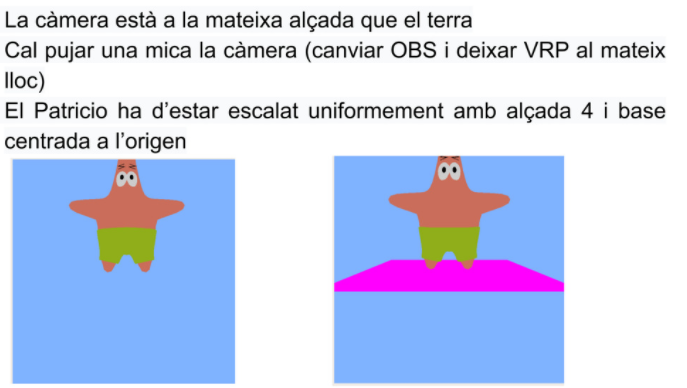
Aquests punts ens serveix per calcular la matriu de translació per situar els peus dels Patricio al punt (0, 0, 0). Es fa a modelTransformPatricio.



Un cop situat el Patricio al (0, 0, 0) l’haurem d’escalar per a què faci alçada 4.

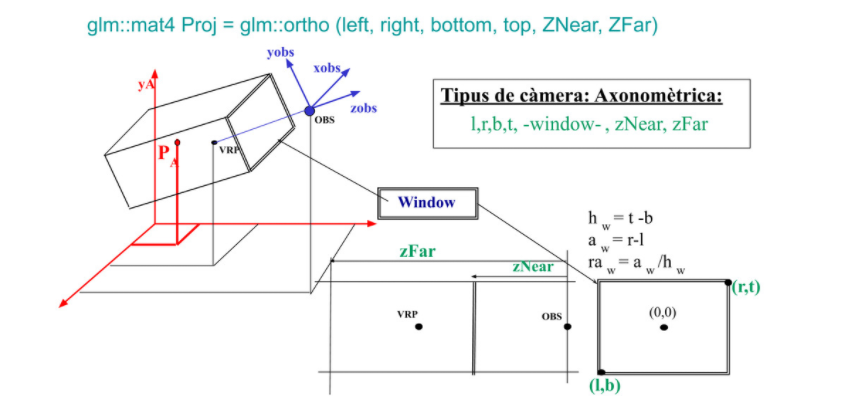
Els factors d’escalat són sx = 4 / h; sy = 4 / h; sz = 4 / h;

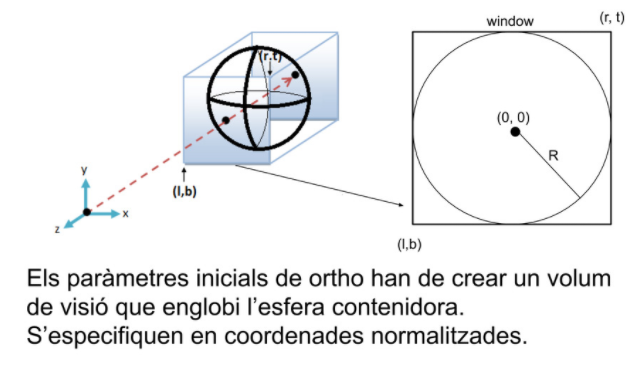
Aquest escalat s’ha d’afegir també a modelTransformPatricio.



**Càmera ortogonal / Òptica ortogonal**

La matriu de projecció que cal per càmera ortogonal la calcula la funció glm::ortho:



****

(l, b) = (-R, -R); (r, t) = (R, R);

Afegir la possibilitat de tenir les dues òptiques possibles i decidibles amb la tecla ‘O’:

* Inicialment tenim òptica perspectiva i canviarem d’òptica cada cop que l’usuari premi la tecla ‘O’.
* Cada cop que canviem d’òptica caldrà reconstruir i enviar la matriu de projecció al Shader.

**Resize amb òptica ortogonal**

Afegir / modificar el mètode resizeGL el necessari per a que no deformi ni retalli tampoc amb aquesta òptica. En un exemple on R és el radi de l’esfera tenim:

