**Transformacions de la càmera**

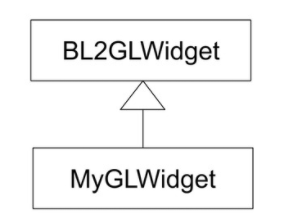


**Introducció**

Nou esquelet

Classe nova: BL2GLWidget

* Hereda de QOpenGLWidget.
* Inclou codi que no es pot modificar.

Classe MyGLWidget

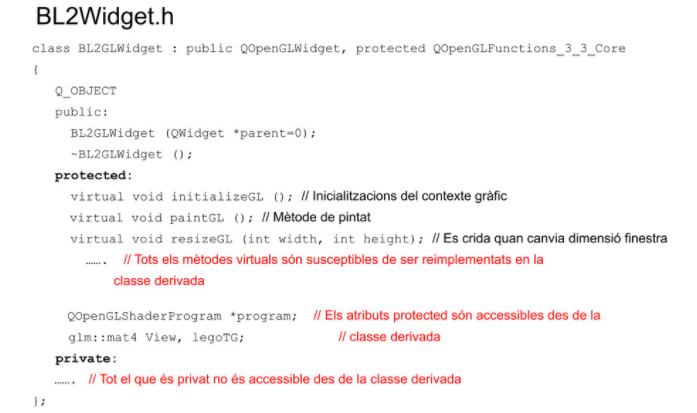
* Hereda de BL2GLWidget.
* Pot usar mètodes i propietats **protected** de BL2GLWidget.
* No pot usar mètodes ni propietats **private** de BL2GLWidget.
* Pot reimplementar mètodes virtuals de BL2GLWidget.

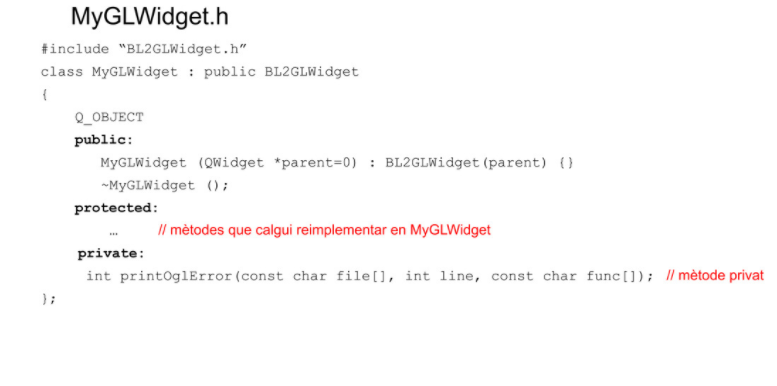
Mètode virtual

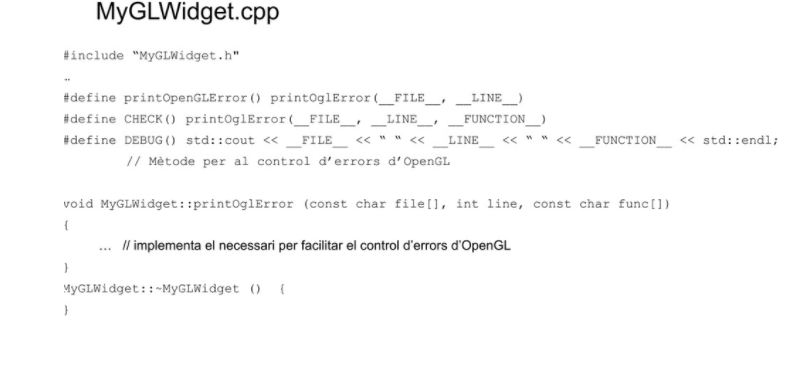
* Un mètode virtual és un mètode que es pot reimplementar a la classe derivada.
* Tant el mètode de la classe base com el de la derivada tenen el mateix nom.
* Si es crida a un mètode declarat virtual a la classe base (BL2GLWidget) i reimplementat a la classe derivada (MyGLWidget), s’executarà el de la classe derivada.
* Si el mètode no fos virtual, es cridaria al de BL2GLWidget.
* El mètode virtual de la classe pare queda substituït pel de la classe derivada.
* A la classe derivada no cal indicar que el mètode és virtual.

Restriccions nou esquelet:

* No es pot canviar la visibilitat de mètodes i propietats de BL2GLWidget.
* No es poden afegir mètodes ni propietats a BL2GLWidget.
* No es poden modificar els mètodes de BL2GLWidget.
* Tots els exercicis s’han de implementar a MyGLWidget.

El formulari de l’aplicació Qt conté un element de tipus MyGLWidget i quan s’executa l’aplicació s’intenta executar els mètodes initializeGL(), resizeGL(), i paint() de MyGLWidget. Com que no estan implementats a MyGLWidget, s’executen els que estan a BL2GLWidget (per herència).





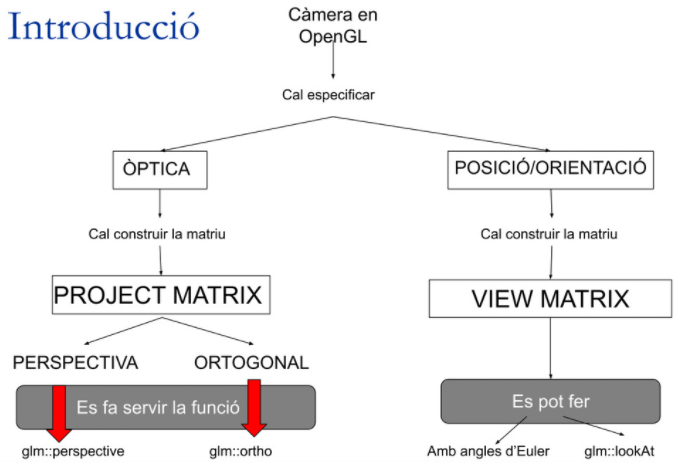
Implementar noves funcionalitats

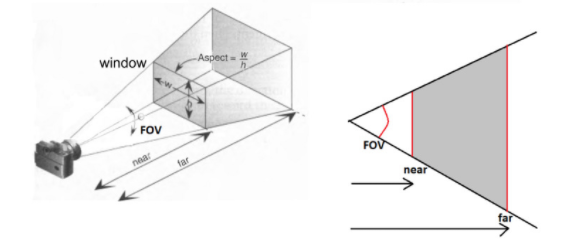
Tres mètodes:

* **Afegir un nou mètode directament a MyGLWidget:**
  + El mètode no ha d’existir ja a BL2GLWidget.
  + Posar la capçalera al MyGLWidget.h i la implementació al MyGLWidget.cpp.
* **Reescriure un mètode de BL2GLWidget:**
  + El mètode ja existeix a BL2GLWidget, però es vol refer sencer a MyGLWidget.
  + Declarar el mètode virtual amb el mateix nom al MyGLWidget.h i implementar-lo al MyGLWidget.cpp.
  + Aquest mètode substitueix el que hi havia a BL2GLWidget.
* **Aprofitar la funcionalitat que ofereix un mètode de BL2GLWidget:**
  + Afegir el mètode amb el mateix nom a MyGLWidget i declarar-lo com a virtual al .h.
  + A la implementació (al .cpp) es pot afegir implementació pròpia i cridar al principi o al final el mètode pare.
  + Per exemple: BL2GLWidget::initializeGL();

Per saber si una crida a OpenGL ha donat error:

* OpenGL en cas d’error en una de les seves funcions activa la variable d’entorn GL\_ERROR i deixa en ella el codi de l’error que s’ha produït.
* En el MyGLWidget podem usar la crida CHECK() allà on vulguem controlar si crides anteriors a funcions d’OpenGL han produït un error. Si s’ha produït un error ens escriurà per pantalla: glError in file <file> @line <line>: <error> function: <function> on <error> és la constant d’error (per exemple, GL\_INVALID\_ENUM) i cal anar al manual de la crida d’OpenGL per saber què l’ha produït.



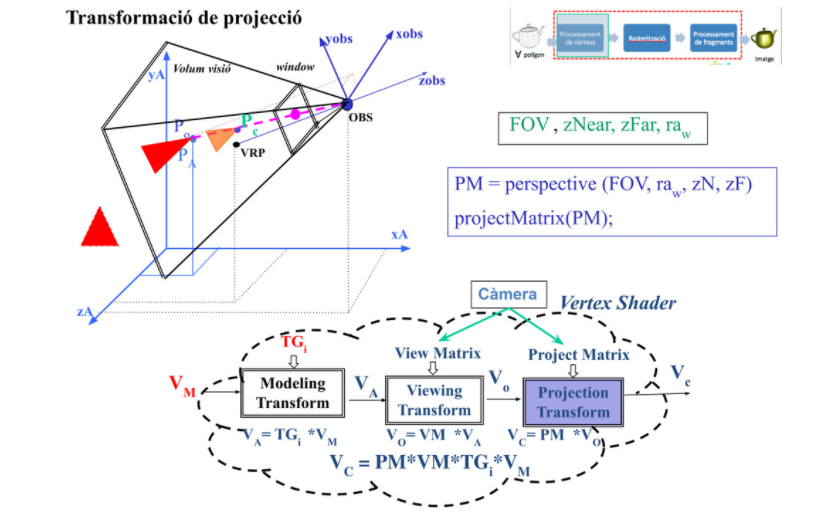


**Transformacions de projecció**

Paràmetres necessaris per càmera perspectiva:

* FOV: angle d’obertura.
* znear, zfar: distàncies.
* raw: aspect ratio del window.

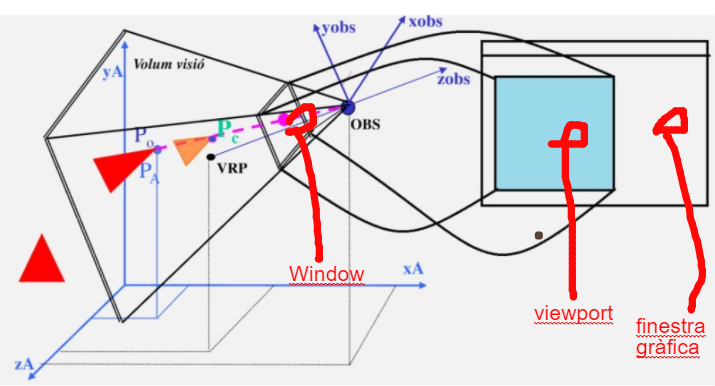
En una càmera perspectiva les línies paral·leles es creuen a l’infinit, mentre que a l’ortogonal es veuen paral·leles.



La finestra gràfica la genera el sistema de finestres del sistema operatiu. A dins es dedica un espai, el viewport, que és on OpenGL dibuixa l’escena.

El window és un tros del pla de retallat anterior on els objectes 3D es projecten.

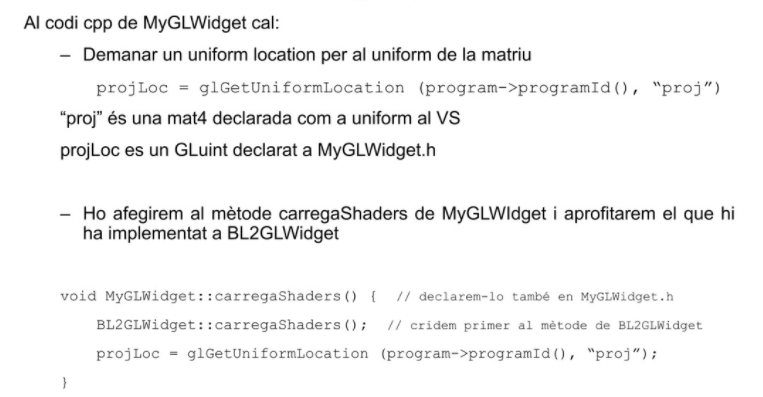
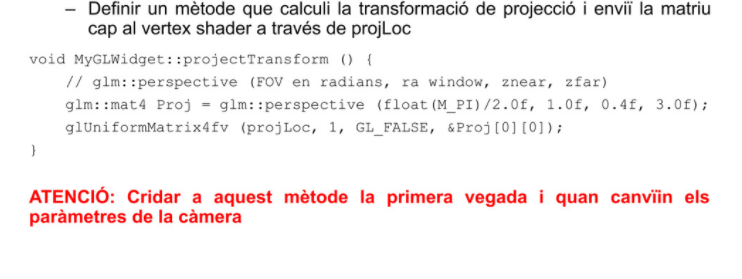
Tot el que es projecta al window es mapeja al viewport.

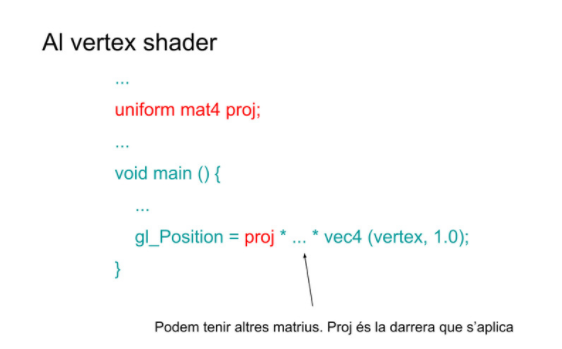


Inicialment, per defecte, OpenGL tracta amb càmera ortogonal. Per programar l’aplicació amb càmera perspectiva cal:

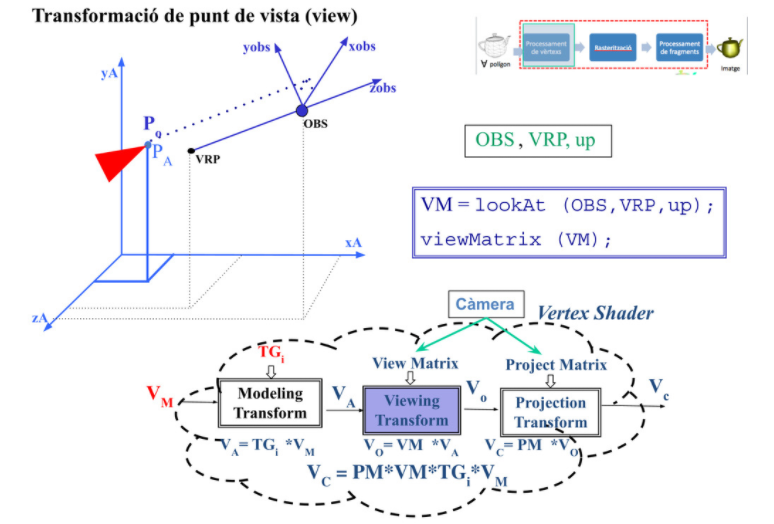
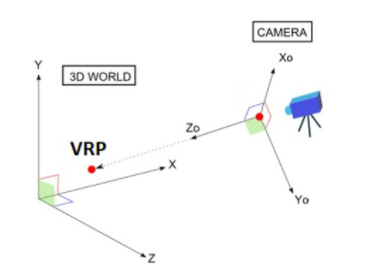
1. Crear la matriu de projecció a l’aplicació.
2. Passar al vèrtex Shader la matriu.

Al VS cal multiplicar la matriu de projecció pels vèrtexs.





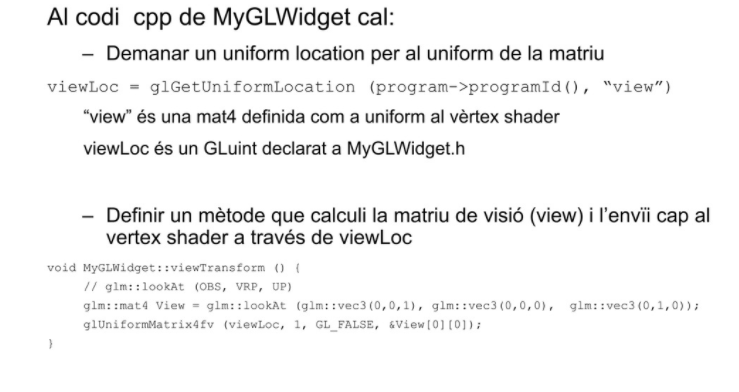
**Transformació de visió**

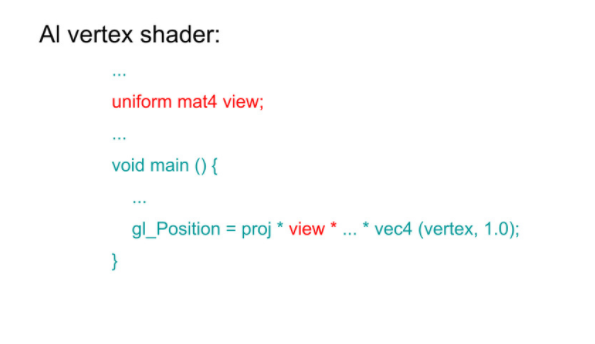


Defineix la posició i orientació de la càmera.

Paràmetres necessaris per lookAt:

* Posició de la càmera.
* VRP: on mira la càmera.
* View-up vector.





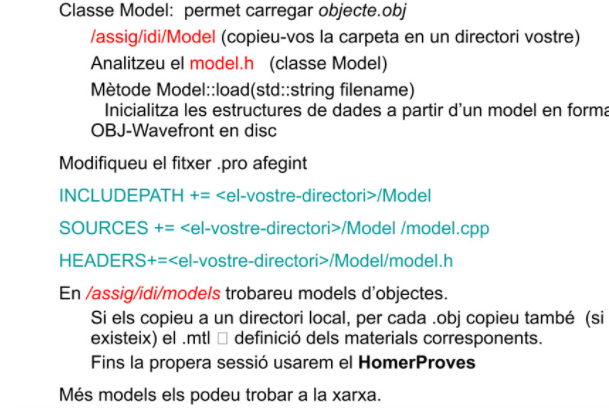
Amb la matriu projection matrix s’especifica el tipus de càmera que volem fer servir (ortogonal o perspectiva). I amb la matriu view matrix s’especifica la posició i la orientació de la càmera.

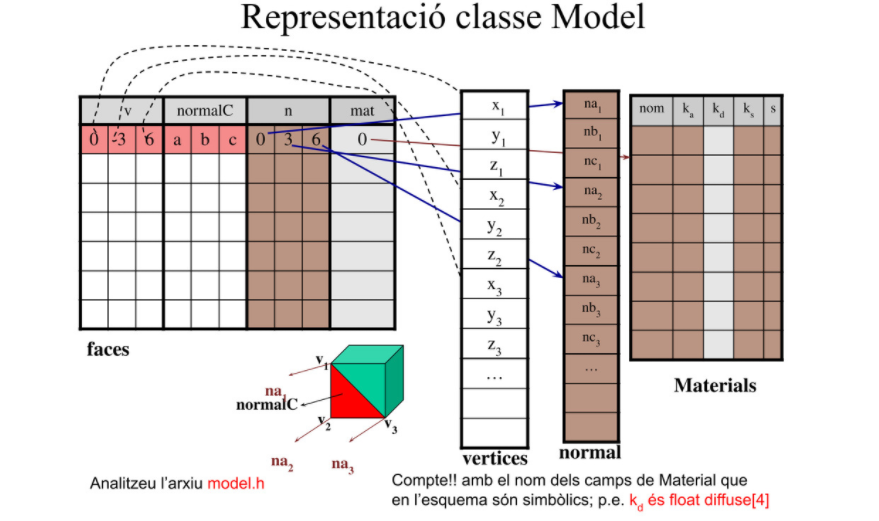
Per crear la matriu de projecció de la càmera, es necessita l’angle d’obertura, aspect ratio del window, distància al pla de projecció near i la distància al pla de projecció far. I per crear la matriu de visió de la càmera, es necessita la posició de l’observador, view reference point i el view-up vector.

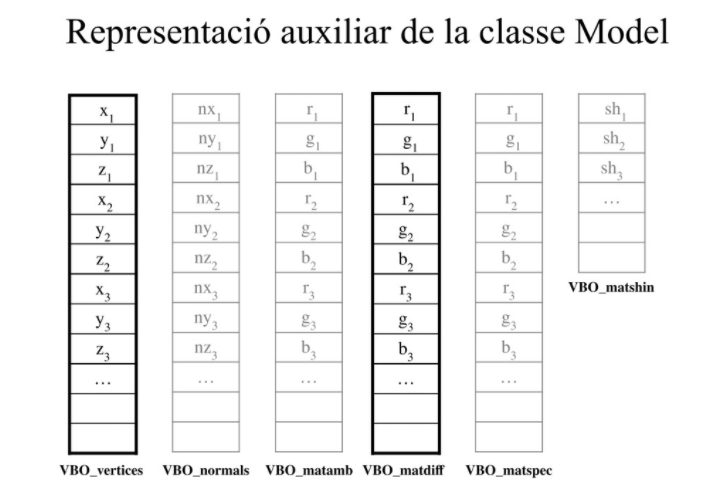
La funció per generar la matriu de projecció es glm::perspective i la funció per generar la matriu de visió es glm::lookAt.

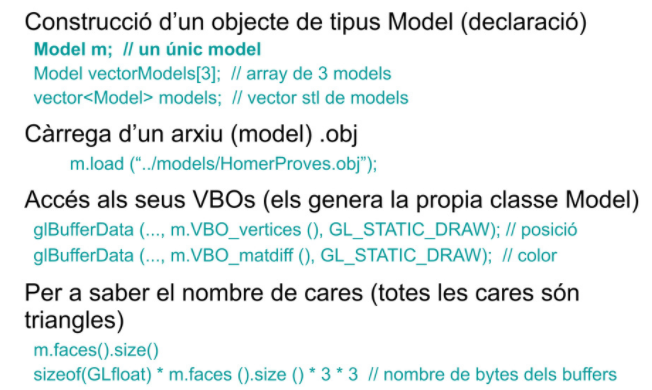
Per usar la càmera perspectiva cal especificar l’angle d’obertura, el znear, el zfar i l’aspect ratio. Mentre que per usar la càmera ortogonal, cal crear una matriu que es multiplica per a tots els vèrtexs.

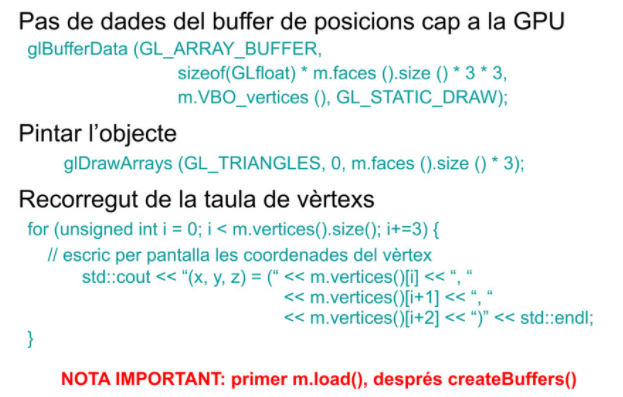
**Càrrega de models**



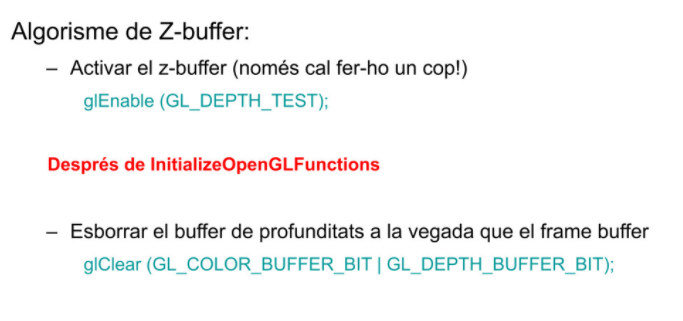


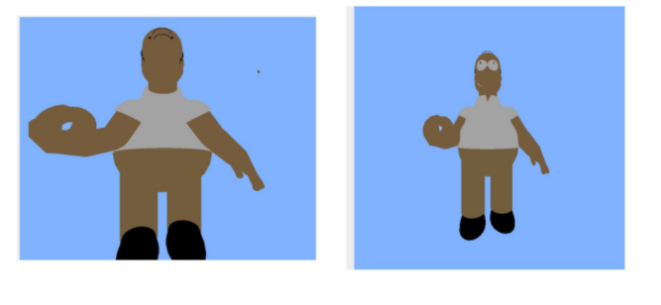






**Z-buffer**

****

****

Pinta l’objecte del tal manera que els colors més propers a l’observador sobreescriuen sobre els que es troben més lluny.