Laboratori IDI: OpenGL, bloc 2

Professors d'IDI, 2025-26.Q1

10 d'octubre de 2025

En aquest segon bloc partirem d'un codi d'esquelet que organitza la feina d'una manera diferent. Tenim una classe `BL2GLWidget` que hereta de `QOpenGLWidget` i que serà la classe on tindreu aquell codi que us donem implementat. **No es pot modificar aquesta classe `BL2GLWidget` sota cap concepte!** Vosaltres haureu de modificar la classe `MyGLWidget` que hereta d'aquesta `BL2GLWidget` i que és on us demanarem que implementeu els exercicis.

El codi de l'esquelet pinta un polígon amb forma de caseta (amb un color diferent per a cada vèrtex). La idea és que primer de tot et familiaritzis amb el codi que et passem i entenguis què fa. Aquest codi el pots trobar a `~/assig/idi/blocs/bloc-2`.

La durada d'aquest bloc és de 4 sessions de laboratori, però es farà un parèntesi entre la tercera i la quarta on farem una sessió per explicar el disseny d'interfícies amb Qt.

Els **temes que tractarem en aquest segon bloc** són els que fan referència al tema de **càmera**, paràmetres de posició i orientació (**View Matrix**) i paràmetres d'òptica (**Project Matrix**). Hauríeu de tenir fresc tot el que s'ha explicat a teoria sobre aquest tema per poder seguir amb èxit el laboratori i sobretot per entendre els exercicis que en aquest laboratori es proposen.

Sessió 2.1: Transformacions de càmera i càrrega d'OBJs 5 hores

Un cop hem vist a classe la sessió 2.1 de laboratori, per començar amb els exercicis, ► el primer que et caldrà fer és compilar i executar l'exemple que et passem per veure què fa, i estudiar amb detall el codi a partir de l'explicació que s'ha fet a classe.

Quan ja hagis mirat i entès l'exemple d'aquest bloc, fes **tots** els següents exercicis **en l'ordre en què estan**, perquè defineixen una guia a seguir per poder entendre tots els passos.

2.1 Exercicis:

1. ► Afegeix al codi de l'exemple un mètode `projectTransform ()` que implementi la crida a perspective amb paràmetres FOV = M_PI/2, ra = 1, znear = 0.4, zfar = 3, i enviï el uniform corresponent de la matriu de projecció al vertex shader, que l'haurà de fer servir. Recorda que tots els paràmetres de la crida perspective han de set floats.

   És normal el que es veu considerant que la posició de la càmera és el punt (0,0,0) (matriu identitat en view transform)?

2. ► Afegeix al codi de l'exemple un mètode `viewTransform ()` que implementi la crida a lookAt amb paràmetres OBS = (0,0,1), VRP = (0,0,0), UP = (0,1,0), i enviï el uniform corresponent de la matriu view al vertex shader, que l'haurà de fer servir.

   Un cop tenim tota la càmera perspectiva definida (View Matrix i Project Matrix) podem convertir els paràmetres (OBS, VRP, up, FOV, ra, znear i zfar) en variables que s'inicialitzen en un mètode

`ini_camera ()`. Aquest mètode a més d'inicialitzar aquestes variables haurà de fer les crides necessàries per inicialitzar les matrius (`projectTransform ()` i `viewTransform ()`). Pensa el lloc del codi on has de posar la crida a `ini_camera()`.

3. ► Volem veure la caseta tombada de costat de manera que el sostre ens quedi a l'esquerra de la imatge i el terra a la dreta. Com has de modificar el paràmetre del vector UP per poder assolir aquesta imatge?

   Fes proves amb altres valors per al vector UP i mira si l'efecte que fan en la imatge que veus és el que t'esperaves. Pots provar, per exemple, de posar-li valors (0,-1,0), (-1,0,0), (1,1,0), etc...

4. ► Modifica l'exercici eliminant la caseta i pintant en el seu lloc l'objecte definit pel fitxer `HomerProves.obj`. Carrega l'objecte (load) abans de construir els buffers i construeix dos VBOs per a aquest objecte, un a partir de les dades dels seus vèrtexs que ens retorna el mètode `VBO_vertices()` de la classe Model i l'altre a partir de la informació del seu material que ens retorna el mètode `VBO_matdiff()` i que podem usar com a informació de color per passar-la al vertex shader. Analitza i estudia el que s'ha explicat a classe (laboratori) per entendre les crides a realitzar, els canvis necesaris al fitxer .pro, etc...

   Recorda que per veure correctament qualsevol objecte cal activar l'algorisme d'eliminació de parts amagades. Activa el Z-buffer com s'ha explicat a classe.

5. ► Afegeix una interacció de teclat de manera que amb la tecla R el Homer roti cada vegada 45 graus (M_PI/4 radians) respecte l'eix vertical Y.

6. ► Afegeix a l'escena un terra quadrat de mides 4 en X i en Z, sobre el pla Y=-1 i centrat al punt (0,-1,0).

   Assegura't que el terra no rota, és a dir, hauràs de tenir una transformació de model diferent per al terra i per al Homer. Quina transformació li cal al terra?

Sessió 2.2: Objecte qualsevol i càmera ortogonal 4 hores

Un cop hem vist a classe la sessió 2.2 de laboratori, pots continuar amb la llista d'exercicis següents. T'aconsellem que et guardis el resultat de l'exercici de la sessió anterior.

2.2 Exercicis:

1. ► Fes un mètode que a partir de dos punts (punt-mínim i punt-màxim) d'una capsa contenidora d'una escena, calculi i guardi en atributs de la classe (o en variables de sortida del mètode) el centre de l'escena (centre d'aquesta capsa contenidora) i el radi de l'esfera que conté la capsa (que passarà a ser el radi de l'escena).

   Utilitza aquest mètode per calcular el centre i radi de l'esfera que conté l'escena que estem visualitzant, en aquest cas, els punts mínim i màxim de l'escena es poden posar a mà perquè són coneguts sabent l'escena que es visualitza (punts extrems del terra + alçada del HomerProves, que és 2).

2. ► Modifica els paràmetres de les crides a lookAt del mètode `viewTransform ()` i a perspective del mètode `projectTransform ()` per a què usin les dades de l'esfera contenidora de l'escena calculada en l'exercici anterior. Amb això aconseguim una càmera en tercera persona que ens permet veure tota l'escena sencera i ocupant el màxim del viewport.

   Recorda el que s'ha explicat a teoria sobre el càlcul dels paràmetres d'una càmera en tercera persona i aplica-ho en aquest exercici amb les dades de l'escena que has calculat en l'exercici anterior.

3. ► Afegeix a l'exercici que mostra el terra i el HomerProves la implementació necessària per a que, quan l'usuari fa una redimensió de la finestra (*resize*), no hi hagi deformació de l'escena ni es retalli el que s'estava veient.

   Recorda el que s'ha explicat a teoria sobre com han de variar els paràmetres de l'òptica de la càmera quan es varia la relació d'aspecte del viewport.

4. ► Ara volem visualitzar una instància del model `Patricio.obj` enlloc del Homer que teníem fins ara, però aquest model no està creat per a que els seus vèrtexs estiguin ubicats en el volum de visió que tenim definit (si el pintes no es veurà). Has d'afegir al teu codi el càlcul de la **capsa contenidora del model** (a partir dels seus vèrtexs) i pintar el Patricio amb la seva base centrada al punt (0,0,0) i escalat de manera que la seva alçada sigui 4.

   Modifica també el terra i fes que faci 5x5 de mida i estigui centrat a l'origen de coordenades (al punt (0,0,0)).

   Tanmateix, l'escena que acabem de construir no hi cap al volum de visió que tenim, caldrà doncs que calculis també els paràmetres d'una càmera perspectiva que et permeti veure aquesta escena centrada, sencera, i ocupant el màxim del viewport (càmera en tercera persona per a aquesta nova escena).

   Nota: Fixa't que si abans has fet bé la programació del càlcul dels paràmetres de càmera a partir de les dades de l'escena, ara només hauries de modificar els punts mínim i màxim de l'escena i tot s'hauria de calcular correctament.

5. ► Modifica el mètode `projectTransform ()` per afegir la possibilitat de canviar el tipus d'òptica entre l'òptica perspectiva i l'ortogonal, i afegeix el codi necessari per poder implementar l'òptica ortogonal. Fes que els paràmetres d'aquesta òptica ortogonal també siguin els adients per a que l'esfera contenidora de l'escena estigui completament dins del volum de visió (com amb la perspectiva).

   L'usuari ha de poder decidir entre l'òptica perspectiva i l'ortogonal mitjançant la tecla O. Inicialment l'òptica serà perspectiva i canviarà cada cop que l'usuari premi la tecla O.

6. ► Fes que el redimensionament de la finestra (*resize*) no deformi ni retalli tampoc quan s'usa aquest tiput de càmera (ortogonal).

Sessió 2.3: Euler i nova escena 4 hores

En aquesta tercera sessió del bloc 2, continua amb la llista d'exercicis següents. T'aconsellem que et guardis el resultat de l'exercici de la sessió anterior.

2.3 Exercicis:

1. ► Modifica el mètode `viewTransform ()` per a que faci el càlcul de la matriu *view* a partir de les transformacions mitjançant angles d'Euler. Inicialitza aquests paràmetres per veure exactament el que es veia a l'exercici 4 de la sessió del dia anterior, és a dir, mateixa posició relativa entre escena i càmera.

   Recorda el que s'ha explicat a teoria sobre el càlcul de la View Matrix a partir dels paràmetres dels angles d'Euler (VRP, dist, Ψ, Θ i φ).

2. ► Afegeix la possibilitat que l'usuari pugui interactuar amb el ratolí per modificar els angles que giren la càmera respecte els eixos Y i X (en les transformacions d'Euler). Quan l'usuari mou el ratolí en horitzontal d'esquerra a dreta la càmera s'ha de moure sobre l'esfera de visió en direcció a la dreta. Quan l'usuari mou el ratolí en vertical de baix a dalt la càmera s'ha de moure sobre l'esfera de visió en direcció cap a dalt (considerem l'esfera de visió com aquella esfera virtual

centrada en VRP i de radi dist sobre la que es mou l'observador quan es modifiquen els angles Ψ i Θ d'Euler).

3. ► Afegeix a la implementació del teu programa (del que tens fins ara) la possibilitat de fer zoom-in (amb la tecla Z) i zoom-out (amb la tecla X) de manera que es modifiqui l'angle (FOV) de la càmera perspectiva. *També ho podeu fer (opcionalment) per a la càmera ortogonal modificant la mida del window.*

4. ► Ara, per fer una escena una mica més complexa, modifica el teu codi (pots fer una còpia i guardar el que tenies) de manera que la teva aplicació pinti una escena que té un terra centrat a l'origen de mida 5x5 (el que ja tens), amb 3 patricios que estan escalats de manera que la seva alçada és 1 i posicionats sobre el terra de manera que el primer té el centre de la base de la seva capsa contenidora al punt (2,0,2), el segon el té al punt (0,0,0) i el tercer el té al punt (-2,0,-2). El primer Patricio estarà mirant cap a les Z positives (sense cap rotació), el segon estará mirant cap a les X positives i el tercer estarà mirant cap a les Z negatives.

   Cal tenir definida una càmera perspectiva que vegi l'escena sencera, centrada, sense deformació i ocupant el màxim del viewport (fixa't que les mides de l'escena sencera són conegudes, per tant són coneguts els punts de la seva capsa contenidora).

5. ► Afegeix a l'escena anterior un quart Patricio de la mateixa mida que els altres tres i amb la seva base a la posició (1.5, 0, 0) i mirant cap a Z+. Fes que aquest quart Patricio, utilitzant l'animació amb QTimer, giri constantment al voltant de l'eix Y de l'escena (farà voltes al voltant del Patricio que està al (0,0,0)).

Després d'aquesta tercera sessió, farem un parèntesi d'una sessió en què veurem com podem dissenyar interfícies usant Qt. Veurem també amb detall com podrem estendre funcionalitats de widgets propis de Qt.

Sessió 2.4: Afegim elements d'interfície amb Qt 4 hores

En aquesta última sessió del bloc el que farem és afegir elements d'interfície que permetin fer algunes de les coses que ja hem fet mitjançant interacció directa i d'altres. Continua amb els següents exercicis guardant-te els que tenies com a resultat de la sessió anterior.

Nota: *D'aquesta sessió considerem important que facis com a mínim els exercicis 1, 2, 5, 6 i 7. La resta també però els pots considerar opcionals.*

En aquesta sessió, apart d'exercitar el que es va veure a les sessions anteriors sobre com implementar elements d'interfície usant Qt en la nostra aplicació d'OpenGL, també es pretén que pensis els exercicis que es demanen tenint en compte els **principis de disseny d'interfícies** que s'estan veient a teoria.

2.4 Exercicis:

1. ► Afegeix a la interfície del teu programa un *Slider* i un *Spinbox* que estiguin sincronitzats entre ells i que permetin controlar l'angle d'obertura de la càmera (FOV) per fer el Zoom. Caldrà que els dos elements de la interfície estiguin límitats en els seus valors de forma adient.

2. ► Afegeix a la interfície del teu programa un *Radio Button* que permeti canviar el model entre el Patricio i el Legoman. Fixa't que cal construir els buffers (VAO i VBOs) per a cadascun dels dos models, però **aquesta construcció només s'ha de fer una vegada!** Pots decidir quin dels dos pintes en el moment de pintat.

3. ► Afegeix a la interfície del teu programa (del que tens fins ara) un *Radio Button* que permeti decidir entre usar l'òptica perspectiva o l'ortogonal.

4. ► Afegeix a la interficie del teu programa un *Spinbox* que permeti modificar el factor d'escala del model. Fes que l'ús d'aquest element de la interfície sigui compatible amb l'ús de les tecles 'S' i 'D' que ja tenies implementades per fer el mateix.

5. ► Afegeix a la interfície del teu programa dos *Dials* que permetin controlar i modificar els angles d'Euler (Ψ i Θ) de la càmera. Afegeix també la possibilitat que si els angles es modifiquen de manera directa amb el ratolí es vegin afectats també els valors dels *Dials*.

6. ► Implementa una classe derivada de *QLabel* (MyLabel) com a classe pròpia que permeti mostrar mitjançant el color del seu *background* el color representat per 3 *Spinbox* que defineixen els valors de R, G i B del color mostrat.

   Afegeix aquesta MyLabel i els tres *Spinbox* a la teva interfície per permetre decidir amb ells el color del terra.

7. ► Utilitza el MyLabel i els tres *Spinbox* que has definit a l'exercici anterior per afegir a la interfície un control sobre el color de fons de l'aplicació. Aquest control ha de permetre a l'usuari modificar aquest color de fons. Fes que inicialment aquests elements de la interfície estiguin inicialitzats amb els valors del color de fons inicial de l'aplicació.

8. ► Afegeix a la interfície del teu programa dos *Sliders*, un d'horitzontal col·locat sota de la finestra gràfica i ocupant tot l'espai que aquesta ocupa, i un de vertical, col·locat al costat de la finestra gràfica i ocupant tot l'espai que aquesta ocupa. Aquests dos *Sliders* volem que controlin les dimensions del viewport en el que pintem (paràmetres amplada i alçada de la crida a glViewport), de manera que el *Slider* horitzontal tindrà valors entre 0 i l'amplada de la finestra gràfica (ample) i el *Slider* vertical tindrà valors entre 0 i l'alçada de la finestra gràfica (alt).

   Quan l'usuari mou un d'aquests *Sliders* s'ha de modificar la dimensió corresponent del viewport en què es pinta la nostra imatge final, sense modificar l'origen del viewport que continuarà sent el píxel de la cantonada inferior esquerra.

   Recorda la crida a glViewport s'ha de posar dins del mètode paintGL() per a què sigui efectiva.

   Nota: Fixa't que des del moment en què el teu viewport passa a tenir mides fixes (nombre de pixels concret) l'efecte que tenia el mètode resize en aquesta mida del viewport desapareix. L'usuari és ara qui decideix sobre la mida del viewport, i per tant, sobre els canvis que calguin en els paràmetres de càmera. No es demana que resolguis aquest problema, però depenent de com hagis estructurat el teu codi pot ser que et sigui fàcil tenir-ho en compte.