Pràctica 2 – Fase 0 (i II)

Introducció al Z-Buffer

GiVD - curs 2022-23

Objectiu general de la Pràctica 2: Realitzar visualitzacions de dades utilitzant tècniques projectives (Z-Buffer), basades en la targeta gràfica (o GPU) programant *shaders* amb glsl.

FASE 0 (2ª part): Primers programes d'exemple. Com funcionen?

Suposant que ja has comprovat la teva instal·lació i que tots els projectes dels cubs et funcionen bé, aquesta fase es composa de 3 passos:

1. Exploració de la visualització d'un model poligonal amb OpenGL
2. Exploració de la visualització del model re-programant la GPU (glsl)
3. Exploració de la visualització del model utilitzant textures a la GPU (glsl)

Després cadascun dels passos 1, 2 i 3, es fan unes preguntes per a reflexionar o conèixer millor tota l'arquitectura de les aplicacions. Al final d'aquesta fase, trobareu un exercici voluntari proposat per a reforçar els conceptes de la pràctica 0.

En aquesta primera part, **NO s'ha de lliurar cap codi**. Cal comprendre bé l'arquitectura que hi ha darrera d'una aplicació gràfica basada en la GPU. Per això també pots llegir els capítols 1.7 i 2.8 del llibre de referència bàsica de l'assignatura[1]. Llegeix-los un cop hagis baixat i explorat el codi.

Índex

PAS 1: Exploració del projecte CubGL

El projecte CubGL codifica la visualització d'un cub utilitzant la llibreria OpenGL directament, sense reprogramar la GPU directament, utilitzant només crides d'alt nivell d'OpenGL per enviar dades i fer les gestions a la GPU des del codi de C++. és una llibreria que treballa en la memòria de la CPU i encapsula tot l'enviament de dades i gestions a la GPU.

Si no la tens encara baixada, baixa l'aplicació del campus CubGL.tgz de l'enllaç del campus virtual, i obre-la amb el QtCreator.

Les principals classes d'aquesta aplicació (CubGL) són:

- **main**: conté el programa principal
- **mainwindow**: que conté el disseny de la interfície (hereta de la classe de Qt anomenada QWidget). Es creen els scrollbars i es connecten el seus events (signals) a mètodes concrets que serviran l'event (slots). Podeu trobar més informació del funcionament dels signals i slots en Qt a l'enllaç

[1] Interactive computer graphics : a top-down approach with shader-based OpenGL, https://cercabib.ub.edu/permalink/34CSUC_UB/18sfiok/alma991009933539706708

http://doc.qt.io/qt-5/signalsandslots.html

- **glwidget**: conté el widget que permetrà dibuixar amb OpenGL en el seu interior (deriva de QGLWidget). Els mètodes principals a destacar en aquest widget són mètodes que es deriven de la classe abstracte QGLWidget, que són:
  - *initializeGL()*: es crida automàticament només el primer cop que s'instancia el widget
  - *paintGL()*: mètode que es crida automàticament cada cop que la finestra es refresca, ja sigui per que una altra finestra la tapa, o per qué hi ha un canvi de refresc, per exemple per que es canvia un angle de visió.
  - *resizeGL()*: mètode que es crida automàticament cada vegada que hi ha un canvi en la mida del widget, per exemple quan s'amplia la finestra.

  En aquesta classe també es poden sobrecarregar els mètodes de tractament del moviment del ratolí, per exemple.
- **Cub**: classe que conté la informació de la geometria 3D, topologia i els colors del cub Qt. D'aquesta classe només cal remarcar, per ara, que és important realitzar el mètode *draw()*, mètode que permetrà pintar tota la geometria en el widget de GL. Els mètodes per a construir la geometria es tractaran més endavant en l'assignatura. Per ara, només ens fixarem que és necessari saber els punts, la seva connexió (o topologia) i els colors associats a cadascun d'ells.

Per a més explicacions, mira el vídeo que tens en el canal de stream en el següent enllaç. Analitza el funcionament del mètode *draw()*:

- Des d'on es crida? Per què? **des del mètode paintGL**
- Dins de quina classe està definit?
- Quina diferència hi ha entre el constructor i el mètode *draw()*? On es defineix la geometria? On es dibuixa? Qui ho envia a la GPU?
- Quina geometria defineix el cub? Les seves cares estan representades per triangles o per quadrilàters?
- Com estan ordenats els punts? Per què?
- Com s'aconsegueix que roti el cub?

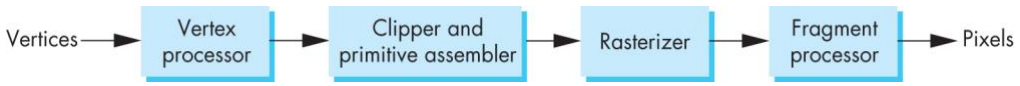
Mira si has entès tots els conceptes fent l'autoqüestionari del campus habilitat per aquesta part.

PAS 2: Com re-programar la GPU? Projecte CubGPU

En els darrers anys, s'ha obert la possibilitat de treballar i programar directament en la GPU mitjançant el llenguatge GLSL. Això significa la possibilitat de fer la reprogramació directa d'algunes funcions que segueix oferint OpenGL, evitant les transferències innecessàries de memòria entre els dos processadors (CPU i GPU) i optimitzant així el temps de visualització (veure secció 1.7 i 2.8 del llibre de referència bàsica).

Concretament es poden reprogramar algunes parts del pipeline gràfic i per això s'utilitzen uns programes anomenats *shaders*. Hi ha diferents tipus de *shaders* depenent de quina part del pipeline es reprograma: *geomètric shader*, *vèrtex shader* i *fragment shader*. A l'assignatura veurem els *vèrtex shaders* i els *fragment shaders*.

Una vegada es tenen els vèrtexs de l'objecte i la seva topologia, es poden realitzar les transformacions geomètriques en paral·lel usant el processador de vèrtexs de la GPU. Quan es treballa a nivell de vèrtexs, es programa el *vertex shader.* A més a més, després de rasteritzar els vèrtexs en píxels, es pot programar el càlcul del color de cada píxel en el processador de la GPU de fragments (el programa que s'usa és el *fragment shader*).



Si encara no la tens baixada, baixa ara l'aplicació CubGPU.tgz, descomprimeix el fitxer i obre el projecte amb l'IDE QtCreator. En la versió per a Qt 5.0 s'utilitzen unes estructures auxiliars per a passar informació a la GPU que són els **Vertex Buffer Objects (VBO)**, que són buffers que contenen els vèrtexs de l'Objecte**.** Amb aquestes estructures permeten passar els vèrtexs i la informació associada a cada vèrtex, com per exemple el color, la normal, etc. per a ser usada en el *vertex shader*. Tens el vídeo explicatiu del projecte aquí.

Pintat dels arrays definits del cub: activació del pipeline de GL. Es crida des de glwidget.cpp.

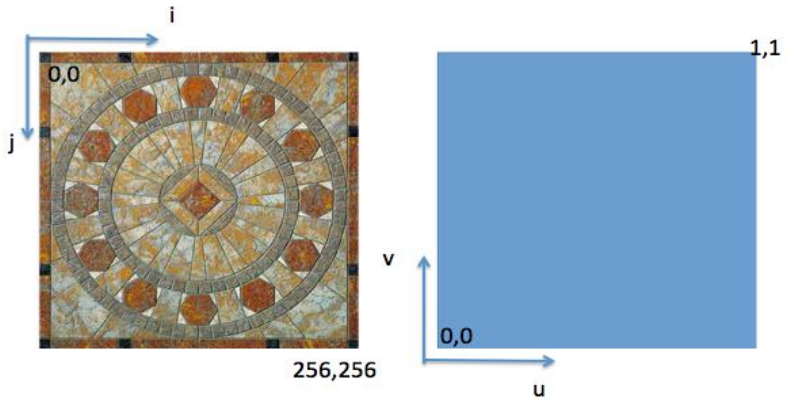
Mira de pensar les següents preguntes:

- Que fa el mètode draw()? Des d'un es crida? Per què?
- Dins de quina classe està definit?
- Quina diferència hi ha entre el constructor i el mètode *draw()*?
- On es defineix la geometria? Cada cara del cub està formada per triangles o per quadrilàters? Quin ordre segueixen els punts?
- On es dibuixa?
- Qui ho envia a la GPU?
- Què s'envia a la GPU?
- Quin shader processa els vèrtexs?
- Quin shader s'encarrega dels colors?
- Per què surten els colors purs en els vèrtexs i en les arestes i cares surten degradats?
- Com s'aconsegueix que es roti el cub?

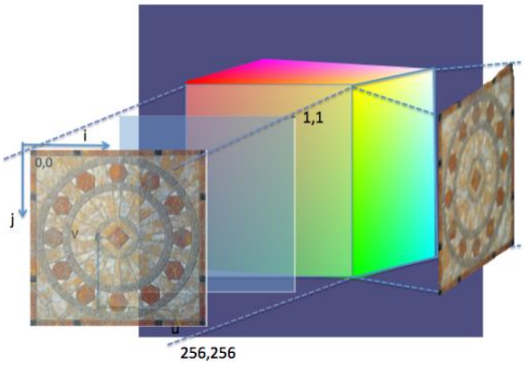
Prova de contestar l'autoqüestionari d'aquest projecte per comprovar que ho has entès tot.

PAS 3: Explorant les textures a la GPU

Disposem de la textura (o imatge mosaic.png), tal que cadascun dels seus píxels es mapegen en un espai 2D entre el (0,0) i el (1,1). Aquestes noves coordenades de píxel entre 0 i 1 són les **coordenades de textura**:



Per a mapejar la textura a cadascun de les cares del cub, cal definir per a cada vèrtex del cub, quin coordenada de textura li correspon.



Així, per a usar una textura cal realitzar els següents passos:

1. Carregar la textura a GL: explora el mètode **InitTextura** de la classe cub

2. Cal passar la textura al *fragment shader* quan es passen les dades a la GPU i lligar la textura amb la variable corresponent del fragment shader.

3. Cal passar les coordenades de textura corresponents a cada vèrtex.

4. Cal activar el mode GL_TEXTURE_2D del GL per a que ho activi en el seu *pipeline*.

Si encara no la tens baixada, baixa ara l'aplicació, descomprimeix el fitxer i obre el projecte amb l'IDE QtCreator. Executa'l i mira el que obtens. Fixa't en la classe cub. Com ha canviat en relació al projecte anterior?

Si, hem afegit la següent linia: glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

- Ha canviat el mètode draw()?
- On es defineixen les coordenades de textura? Quin mètode les calcula?
- Qui les envia a la GPU?
- Què s'envia a la GPU?
- Quin/s shader/s té com a entrada els vèrtexs?
- Quin shader té en compte les coordenades de textura?
- Quin shader utilitza la textura?
- Com canviaries la imatge de la textura mapejada al cub?

Mira de respondre l'autoqüestionari d'aquesta part per veure que ho has entès tot.

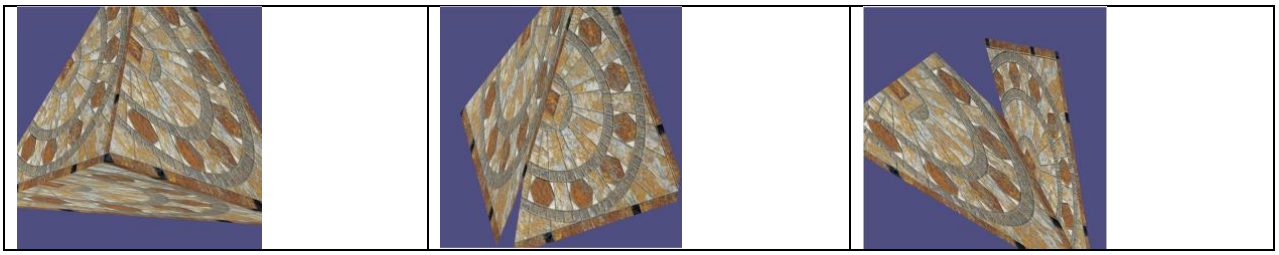
PAS 4: Exercici de Reforç

1. En el projecte cubGPUTextures substitueix el cub per un tetraedre. El tetraedre està format per a quatre vèrtexs (0,0,1), (0, $2\sqrt{2}/3$, -1/3), ($-\sqrt{6}/3$, $-\sqrt{2}/3$, -1/3) i ($\sqrt{6}/3$, $-\sqrt{2}/3$, -1/3).

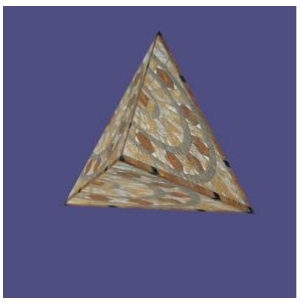
Col·loca una textura a cadascuna de les seves cares. Calcula les coordenades de textura corresponents als seus vèrtexs per a poder situar el dibuix a cada cara.

Quina representació de la malla poligonal és la que estàs utilitzant per a passar els punts a la GPU? Fixa't com i quan es passen els punts i les coordenades de textura.

Veuràs que el tetraedre surt tallat:



2. Això es deu a que GL pinta sempre en un cub de valors entre -1 i 1. Com canviaries el tetraedre per veure'l sense tallar?



3. Com faries per a que a cada cara es veiés una repetició de la textura, com passa a la figura següent?