GiVD - curs 2015-2016 – Pràctica 1

Pràctica número 1

Tema: Il·luminació de l'escena: Implementació de la il·luminació d'una escena composada de diferents objectes amb diferents models locals de *shading*: *Gouraud*, *Phong* i *toon shading*, tenint en compte les textures.

Objectiu: Visualització d'una escena amb diferents *shaders* activats en temps d'execució que permetin els diferents tipus d'il·luminació. Aprendre a utilitzar els shaders per a programar els models d'il·luminació.

La pràctica té com objectiu aprendre a il·luminar l'escena amb diferents models de *shading*. Per això s'aprendrà a passar diferents valors al *vertex shade*r i al *fragment shader.* S'inclouran dues noves classes: la **classe Llum**, que representa tots els atributs d'una llum i la **classe Material**, que codificarà els atributs bàsics que codifiquen un material (component difusa, component especular i component ambient).

En aquesta pràctica s'inclou en l'escena varies llums de diferents tipus i a cada objecte, el seu material associat. Per a començar a desenvolupar el codi d'aquesta pràctica, es partirà de la base del codi que està penjat en el campus virtual de l'assignatura. Si l'executes el projecte i carregues l'objecte esfera.obj de la carpeta *"resources*" veuràs que es pinta una esfera de diferents colors.

La pràctica 1 es composa dels següents exercicis:

- Implementació d'una nova classe **Material** que permeti representar les característiques del material d'un objecte.
- Implementació d'una nova classe anomenada **Llum**, que permeti representar tots els atributs necessaris per a codificar una llum puntual, una llum direccional, una llum de tipus spot-light i amb possible atenuació en profunditat.
- Modificació de la classe Objecte per a que la lectura dels objectes (read_obj) permeti el càlcul de les normals associades als vèrtexs dels objectes i per a poder incloure una textura a l'objecte.
- Implementació dels càlculs d'il·luminació segons les següents tècniques: *gouraud, phong-shading* i *toon-shading* tenint en compte el perfilat de les arestes. Aquests tècniques es podran seleccionar en temps d'execució.
- Incorporació de mètodes realistes basats en textures.

NOTA IMPORTANT: Durant tota aquesta pràctica la **càmera** està fixe i l'observador està situat al punt (0,0, 10). El **frustum** de visió és el cub amb els extrems (-1,-1,-1) i (1,1,1). Tot objecte que carreguis a l'escena ha d'estar situat dins d'aquest cub.

Per a fer les proves disposeu d'una esfera en el fitxer sphere.obj, encara que podeu carregar altres fitxers .obj generats amb blender, per exemple. Han de complir que estiguin en el frustum visible per a que els pugueu veure sense retallar.

1. Etapes de desenvolupament de la pràctica (part obligatòria):

PAS 1. Creació d'una nova classe material

Creació d'una nova classe Material

Descripció:

Aprofiteu la classe Material del codi base de la pràctica per tal que representi el **material** d'un objecte. Aquesta classe definirà els atributs de les propietats òptiques d'un objecte (component ambient, component difús, component especular i coeficient de reflexió especular). Cada objecte tindrà associat un material.

Implementeu també el mètode per a passar els seus valors a la GPU:

```
void Material::toGPU(QGLShaderProgram *program)
```

Utilitzeu també "*structs*" per a estructurar la informació tant a la CPU com a la GPU. Aquest mètode es cridarà des de cadascun dels objectes, dins dels mètodes **toGPU** corresponents.

On es declaren els materials?

Cada objecte ha de tenir associat el seu material. Quan afegiu el material a l'objecte, observeu que ara l'objecte que es carrega té colors que es coloquen en el vector de colors associats a cada vèrtexs per passar-los a la GPU. Ara aquest vector ja no té sentit i per tant ja no és necessari passar-lo a la GPU.

Modifiqueu la classe objecte per a tenir en compte el material enlloc dels colors i feu que aquests canvis es reflecteixin també en el pas de dades a la GPU.

Codifica el *vèrtex shader* per a que el pugui rebre convenientment i comprova que les dades estiguin ben passades.

Com fer-ho?

Els valors dels materials es posaran en la constructora de l'objecte. No es llegeixen de cap fitxer en el codi base.

Per estructurar la informació en els *shaders*, s'utilitzarà un "*struct*" per un material. Recordeu que per a comunicar la CPU amb la GPU s'han de fer diferents passos. A continuació es detallen els passos concrets per a passar un material a la GPU, si s'utilitza un "*struct*" en els *shaders*.

a. Com es defineix i s'utilitza un struct en els shaders?

En la GPU, en glsl:

```
struct Exemple
{
    vec4 exemple1;
    vec3 exemple2;
    float exemple3;
    ...
};

uniform Exemple test;
```

Fixeu-vos que es la variable **test** és de tipus "uniform". Això vol dir que serà constant a tots els vèrtexs de l'objecte.

b. Com es fa el lligam entre les variables de la CPU i la GPU?

En la CPU, farem el lligam de les variables de la GPU de la següent forma:

```
  // 1. Per a passar els diferents atributs del shader declarem una estructura amb els identificadors de la GPU
    struct {
        GLuint ex1;
        GLuint  ex2;
        GLuint ex3;
        ....
    } gl_IdExemple;

   // 2. obtencio dels identificadors de la GPU
    gl_IdExemple.ex1 = program->uniformLocation("test.exemple1");
    gl_IdExemple.ex2 = program->uniformLocation("test.exemple2");
    gl_IdExemple.ex3 = program->uniformLocation("test.exemple3");
    ...

   // 3. Bind de les zones de memòria que corresponen a la GPU a valors de les variables de la CPU
```

```
glUniform4fv(gl_IdExemple.ex1, 1, vectorProva);      // vectorProva és una variable de tipus vec4
glUniform3fv(gl_IdExemple.ex2, 1, vector3D );        // vector3D és una variable de tipus vec3
glUniform1f( gl_IdExemple.ex3, unFloat);             // unFloat és una variable de tipus GLfloat
```

Quines proves pots fer per saber que funciona?

1. Posa la component ambient del material de l'objecte a color vermell

2. En el *vertex shader* dóna com a color de sortida el valors de la component ambient que estàs passant a la GPU.

3. Comprova que l'objecte es visualitza en vermell.

4. Fes el mateix amb la resta de propietats òptiques per a comprovar que estan ben passades a la GPU.

PAS 2. Creació d'una nova classe llum

Creació d'una nova classe Llum

Descripció:

Aquesta classe haurà de permetre codificar una llum i totes les seves propietats, segons el tipus de llum que codifiqui. En principi, en aquesta classe (o jerarquia) heu de definir els atributs que permetin representar:

- llums puntuals (posició)
- llums direccionals (direcció)
- llums spot-light (direcció, angle d'obertura)

Addicionalment, cal que cada llum codifiqui la intensitat en la què emet (ambient, difusa i especular) i els coeficients d'atenuació en profunditat (constant, lineal i quadràtica).

Les llums podran activar-se o desactivar-se per a influir en el càlcul de la il·luminació.

En la classe **Mon** s'ha afegit un vector de llums i cal afegir un mètode anomenat **toGPU()** que permeti passar la informació de totes les llums actives a la GPU, segons la següent capçalera:

```
void Mon::llumsToGPU(QGLShaderProgram *program)
```

Com fer-ho?

Per estructurar la informació de les llums en els *shaders*, s'utilitzarà un "*vector*" per que guardi un "struct" per a cada llum. A continuació es detallen els passos concrets per a passar aconseguir passar un vector d'elements a la GPU, si s'utilitza un "*array*" en els *shaders*.

a. Com es defineix i s'utilitza un array en els shaders?

En la GPU:

```
struct Exemple
{
    vec4 exemple1;
    vec3 exemple2;
    float exemple3;
    ...
};

uniform Exemple conjunt[3];
```

b. Com es fa el lligam entre les variables de la CPU i la GPU?

En la CPU, farem el lligam de les variables de la GPU de la següent forma:

```
// 1. Es declara un vector d'identificadors

struct {

        GLuint ex1;
        GLuint  ex2;
        GLuint ex3;
        ....
    } gl_IdExemple;

 gl_IdExemple gl_IdVect [MAX];

// 2. obtencio dels identificadors de la GPU: Suposem i l'index de l'i-èssim element del vector

gl_IdVect[ i ].ex1 =  program->uniformLocation(QString("conjunt[%1]. exemple1").arg( i ));


// 3. Bind de les zones de memoria que corresponen
   glUniform4fv(gl_IdVect[ i ].ex1, 1, vectorProva);          // vectorProva és una variable de tipus vec4
```

c. On es declara la llum?

Finalment, per a que es pugui utilitzar una llum en l'escena, teniu un vector de llums a la classe **món**, juntament heu de definir la intensitat d'ambient global al món. Aquesta intensitat global, caldrà passar-la també a la GPU per a ser utilitzada pels *shaders*. Per això cal que implementeu un mètode a la classe **Mon** anomenat setAmbientToGPU, amb la següent capçalera:

```
void escena::setAmbientToGPU(QGLShaderProgram *program)
```

Com comprovo que es passen bé els valors?

Comenceu definint només una llum puntual i comproveu que tots els seus atributs es passen correctament a la GPU.

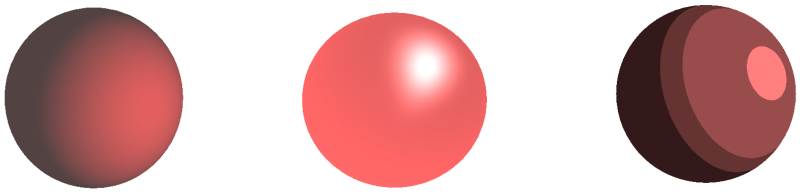
Per a validar que passeu bé les dades a la GPU, com no es pot imprimir per pantalla des del *shader*, utilitzeu la variable color del *vertex shader,* com feieu pel material. Si per exemple, li assigneu al color del *vertex shader*, la intensitat difusa de la llum, hauríeu de visualitzar els objectes del color que heu posat a la intensitat difusa des de la CPU.

Definiu ara tres llums puntuals que il·luminin l'esfera. Comproveu que es passen bé els seus valors.

Definiu els altres tipus de llum (direccionals i spot-light). Comproveu que es passen bé els seus paràmetres. En el vector de llums posa una llum de cada tipus i comprova que es passen bé els tres tipus de llums. Què contindrà el "struct" de la GPU?

PAS 3: Implementació dels diferents tipus de *shading*

Creació de diferents tipus de shading



Descripció:

En aquest apartat es tracta d'implementar els següents diferents tipus de *shading*:

- *Gouraud* (suavització del color)
- *Phong shading* (suavització de les normals)
- *Toon shading* (shading discret segons el producte del vector de la llum direccional amb la normal al punt i siluetes remarcades)

Aquests tipus d'il·luminació estan definits conceptualment a les transparències de teoria. Suposarem que per aquests *shadings* poligonals usarem el model de **Blinn-Phong.**

Teniu en compte que les variables de tipus **out** del *vertex shader* són les que es rebran com a **in** en el fragment shader, ja interpolades en el píxel corresponent.

Es vol activar cada shader via menú. En la pràctica trobaràs els menús ja dissenyats i els slots o mètodes que els serveixen estan buits en la classe **glwidget**. També ja estan predefinits els shortcuts en la interficie: la tecla Alt+1 activarà el *Gouraud*, la Alt+2 el *Phong shading* i la Alt+3 el *Toon-shading*.

Breu explicació de cada shader:

Gouraud (suavització del color a partir de les normals calculades a cada vèrtex): En el *Gouraud shading* es calculen les normals a cada vèrtex i s'interpola el color a nivell de píxels. En aquest tipus de *shading* heu de calcular les normals "reals" a cada vèrtex dels objectes. Fixa't que quan es llegeix un objecte, cada cara té la seva normal. Ara cal que cada vèrtex tingui associada la normal promig de totes les cares a les que pertany.

Phong shading (suavització de les normals a nivell de píxels): En el *Phong shading* les normals s'interpolen a nivell de píxels. Raoneu on és calcula la il·luminació i modifiqueu convenient els fitxers de la pràctica.

Toon shading (*shading* discret segons el producte del vector de la llum direccional amb la normal al punt): Calculeu a nivell de píxel quatre intervals de valors on donareu diferents tonalitats d'un mateix color a l'esfera. Per fer l'efecte de remarcat de les siluetes 2D dels objectes, com en el còmics, tenen color més intens el píxels tals que l'angle entre la normal i la direcció de visió sigui diferent de 0. El color del píxel es calcula directament amb la component difusa del material de l'objecte multiplicada per (1- cos (α)), sent α l'angle entre el vector de visió i la normal associada al píxel.

Si vols utilitzar diferents *shaders* en temps d'execució raona on s'inicialitzaran els *shaders* i com controlar quin *shader* s'usa? Cal tornar a passar l'escena a la GPU quan es canvia de shader?

Com es tenen diferents parells de shaders? Com s'activen i desactiven?

a. S'inicialitzen tots els parells de shaders amb crides successives a InitShader(..)

b. Per a canviar de shader en la GPU cal linkar i fer bind del nou parell de shaders

```
program->link();    // muntatge del shader en el pipeline grafic
                    // per a ser usat
program->bind();    // unió del shader al pipeline gràfic
```

Estructura els programes que necesitaràs per anar activant i desactivant. A quin classe els has de posar?

Com fer-ho? Passos a seguir:

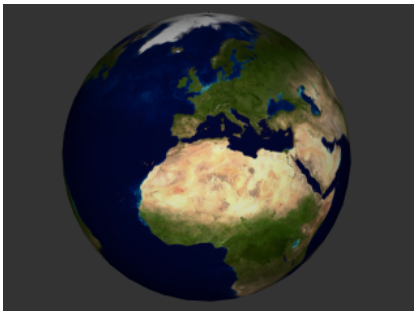
1. Implementa inicialment el *Gouraud shading* amb una llum puntual per aplicar-lo a l'esfera. Per implementar el *flat shading* de cada objecte cal tenir definides a cada vèrtex de cada triangle la mateixa normal. Per això, a cada vèrtex cal calcular la seva normal, segons les cares a les que pertanyi. Cal passar aquestes normals a la GPU, modificant els mètodes **toGPU** i **draw** dels objectes i el *vertex shader*. Cal implementar el model de *Phong-Blinn* en el *vertex shader*.
2. Prova amb diferents materials canviant les diferents components de les propietats òptiques.
3. Implementa el *Phong-shading* quan es premi la tecla 2. Quina diferència hi ha amb el *Gouraud-shading*? On l'has de codificar? Necessites uns nous *vertex-shader* i *fragment-shader*? Raona on

és calcula la il·luminació i modifica convenientment els fitxers de la pràctica.

4. Implementa el *Toon-shading* quan es premi la tecla 3. Cal que la llum direccional funcioni per a poder realitzar el *toon-shading.* Quina diferència hi ha amb el *Gouraud-shading*? On l'has de codificar? Necessites uns nous *vertex-shader* i *fragment-shader*? Raona on és calcula la il·luminació i modifica convenientment els fitxers de la pràctica.
5. Després implementa l'atenuació amb profunditat a cadascun dels mètodes.

PAS 4: Inclusió de textures

Inclusió de textures



Descripció:

S'inclouran textures en la visualització del objecte que s'ha carregat. Suposem que l'objecte que es llegeix de fitxer estan centrats en el punt (0,0,0). Es pot pensar que la textura embolicarà l'objecte com si l'objecte estigués dins d'una esfera (*indirect texture mapping*).

Per a generar les coordenades de textures associades a cada vèrtex del objecte, s'agafa la seva normal i es considera el punt d'intersecció amb una esfera unitària on es consultarà la textura. Es pot utilitzar la següent fórmula, tot suposant que x, y, z és el vector unitari que va des del punt p de l'objecte en direcció (la normal al punt) fins a la superfície de l'esfera. Si interpretem aquest punt en coordenades esfèriques, es pot calcular la coordenada u,v de l'espai de textures.

$$u = 0.5 + \arctan2(z, x)/ 2\pi$$

$$v = 0.5 – \arcsin(y)/ \pi$$

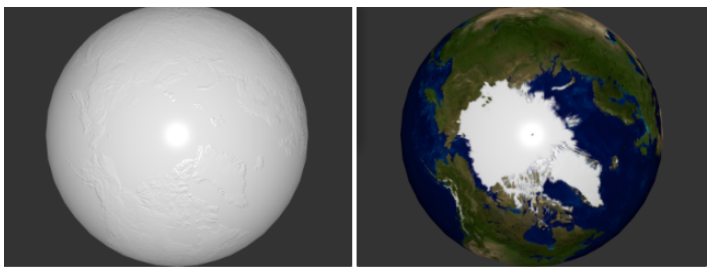
Recorda que (u,v) son valors entre 0 i 1.

Com fer-ho?

Carrega una textura per l'objecte en la seva constructora (recorda com es feia en la classe **cub** del projecte CubGPUTextures).
Per a cada vèrtex calcula la seva coordenada de textura de forma automàtica.
Després passa els vectors de coordenades de textures a la GPU tal i com es feia en el projecte CubGPUTextures. Recorda que la textura es pren com a la component difusa del material o es fa una ponderació amb el material base de l'objecte. *Per exemple, un 75% del color final potser de la textura i un 25% el color del material base.*

2. Part opcional: extensions

2.1 Afegir una segona textura a l'objecte que té associada una pertorbació de la normal a cada punt. Pots trobar la textura associada a les normals de la terra en la carpeta de recursos. Hauràs de passar-la també al shader pel canal 1 enlloc del canal 0, com feies fins ara.



2.2. Afegir una tercera textura per a fer *environmental mapping*, es a dir, fer l'efecte d'un material reflectant en l'objecte. Trobaràs una textura d'un mapa esfèric d'un possible escena reflexada en la carpeta de recursos del projecte.

2.3. Inclou els botons de rotació amb les fletxes del teclat per a poder rotar l'objecte.

2.4. Llegeix els materials des del .obj per a poder definir els materials des de blender.

3. Planificació del desenvolupament de la pràctica 3

Les dates límit de lliurament de la pràctica 11 i 12 d'abril són els dies límit segons el teu grup de pràctiques. A continuació es detalla la planificació aconsellada per a desenvolupar-la.

Febrer	**29**	**1**	**2**	**3**	**4**	Enunciat
Març	**7**	**8**	**9**	**10**	**11**	Material: Implementació i test
	14	**15**	**16**	**17**	**18**	Llums: Implementació i test. Càlcul de normals
Setmana Santa	**21**	**22**	**23**	**24**	**25**	Shaders: Gouraud
	28	**29**	**30**	**31**	**1**	Shaders: Phong i toon-shading
Abril	**4**	**5**	**6**	**7**	**8**	Textures
	11	**12**	**13**	**14**	**15**	Lliurament de la Pràctica 1