Pràctica 2: ZBuffer - Fase 1

GiVD - curs 2016-2017

Pràctica número 2: Fase 1

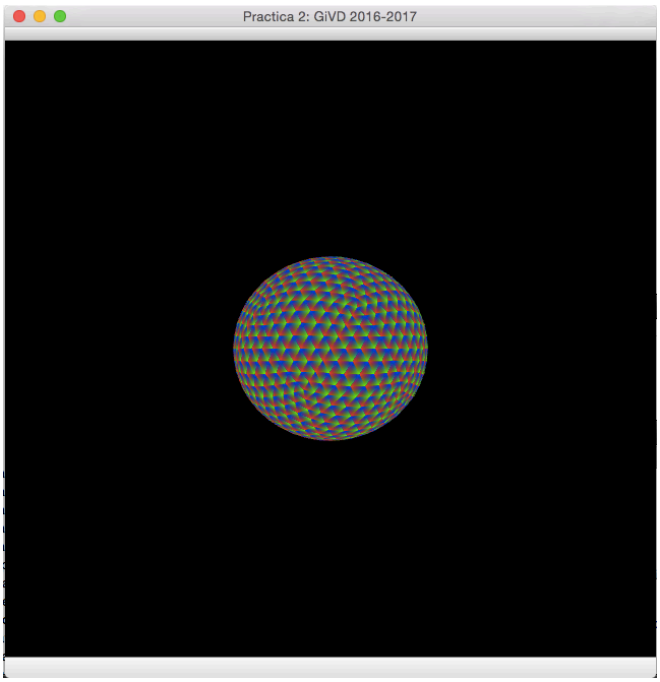
Tema: Il·luminació de l'escena: Implementació de la il·luminació d'una escena composada de diferents objectes amb diferents models locals de *shading*: *Gouraud*, *Phong*, tenint en compte textures. Inclusió de *toon-shading* com il·luminació no realista.

Objectiu de la fase 1: Adaptació de la pràctica 1 a l'algorisme de ZBuffer usant *shaders (o programes executats en la GPU)*, de forma que es visualitzarà l'escena amb diferents *shaders* activats en temps d'execució que permetran els diferents tipus d'il·luminació. Aprendre a utilitzar els *shaders* per a programar els models d'il·luminació. Per això s'aprendrà a passar diferents valors al *vertex shade*r i al *fragment shader.*

S'inclouran dues classes de la pràctica anterior: la **classe Light**, 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, component ambient i transparència). NO s'importaran els tipus de Materials Lambertian, ni Metall, ni Transparent.

En aquesta fase s'inclouran en l'escena varies llums de diferents tipus (PointLight, DirectionalLight i SpotLight).

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, inicialment el projecte permet carregar objectes de tipus .obj (es donen objectes a la carpeta *"resources*"). Si carregues l'objecte sphere0.obj, veuràs que es pinta una esfera de diferents colors. L'esfera està modelada per un model poligonal basat en triangles. Cada triangle es pinta segons els colors associats als seus vèrtexs. Pots explicar per què l'interior de cada triangle es veu de diferents colors?



En aquesta fase has de fer els següents exercicis:

- Inclusió en el projecte d'una nova classe **Material** que permeti representar les característiques del material d'un objecte. Pots adaptar la classe Material de la pràctica 1.
- Inclusió d'una nova classe anomenada **Light**, 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. Pots adaptar i estendre la classe Llum de la pràctica 1.

- Modificació de la classe **Object** per a que la lectura dels objectes (read_obj) permeti llegir del fitxer .obj 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 i phong shading* Aquests tècniques es podran seleccionar en temps d'execució.
- Implementació de la tècnica no realista de *toon-shading*.

NOTA IMPORTANT: Durant tota aquesta fase la **càmera** està fixa 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 sphere0.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. Moure la càmera amb el ratolí s'inclourà en la fase 2 de la pràctica.

En el codi bàsic que se't facilita, s'han comentat els mètodes que s'han d'implementar a la fase 1 i a la fase 2 de la pràctica 2.

Fixa't que pots veure un menú per facilitar afegir llums a l'escena. Aquest menú permet afegir llums i considera l'actual, la darrera que s'ha afegit. Així, els paràmetres que es modifiquen són els de la darrera llum. Pots considerar millorar el comportament de la interfície (codificada en la classe *GLWidget*).

1. Etapes de desenvolupament de la fase 1 (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 tal i com passava a la pràctica 1.

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 a 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 dades 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 per a tots els vèrtexs shaders.

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

```
void Scene::LightsToGPU (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]; // En aquest cas serà un array de 3 elements
```

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 gl_IdExemple;

{

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

 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 memòria 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 el món, teniu un vector de llums a la classe **Scene**, 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 **Scene** anomenat setAmbientToGPU, amb la següent capçalera:

```
void Scene::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 de sortida 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 no realista)

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* i la Alt+2 el *Phong 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.

També necessitaràs la posició de l'observador a la GPU per a calcular el vector H del model Blinn-Phong. En aquesta pràctica suposa que l'observador està situat en el punt (0,0, z_{obs}), on z_{obs} és un valor positiu gran. En la classe **Camera** utilitza el mètode **toGPU** per a passar l'observador als shaders.

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.

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? I també la càmera?

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 necessitaràs 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 *Gouraud 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 també el *vertex shader* o el *fragment shader* per a que les tracti a la GPU.
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 es 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: Emfatitzar les siluetes i objectes transparents

Èmfasi de siluetes en objectes transparents

Descripció:

En aquest apartat es tracta de visualitzar objectes transparents amb els *shaders* ja implementats i afegir-lis l'èmfasi de les siluetes.

Breu explicació:

Objectes transparents: En GL s'utilitza la quarta component del color per codificar la transparència. Si és 0 vol dir que és totalment transparent, si és 1 és totalment opac. Per activar les transparències en OpenGL cal activar en el context de GL el Blending i la funció que s'utilitzarà per fer el blending de colors[1].

Siluetes: Emfatitzar les siluetes suposa més èmfasi en els els contorns de les projeccions 2D dels objectes que s'estan visualitzant. Per això és necessari detectar els píxels que estan en aquests contorns.

[1] http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-10-transparency/

Com fer-ho? Passos a seguir:

1. Per a les transparències, quan es fa la inicialització de GL, cal desactivar el flag de GL_CULL_FACE i activar el flag GL_BLEND. Un cop activat aquest flag s'ha d'activar la funció de blending a utilitzar, fent servir la funció:
   ```
   glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
   ```
2. Inclou l'atribut alpha en els materials i passa'l als shaders. Prova que ara, si poses algun alpha menor a 1.0, es pinten transparents.
3. Per fer l'efecte de remarcat de les siluetes 2D dels objectes, 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. Implementa aquest efecte als tres shaders que has implementat en el pas anterior.

A continuació es detalla la planificació aconsellada per a desenvolupar aquesta fase.

	17 Semana Santa	18 Inici Fase 1	19	20	21	L8: **Enunciat de la pràctica 2** (Sessió conjunta)
April 2017	24	25	26 **Matefest/Infofest**	27	28	Fase1: Llums i Materials implementats. Calcul de les normals i pas a la GPU
	1	2	3	4	5	Fase 1: Shadings implementats. Inici del pas 4. Descripció de la Fase 2: Shadings avançats
May 2017	8	9	10	11	12	Fase 2: Shadings avançats
	15	16	17	18	19	**L12: Entrega P2 i prova P2. Enunciat Pràctica 3**