Pràctica número 3: Il·luminant l'escena

Implementació de la il·luminació de l'escena amb diferents models locals de *shading*: flat, *Gouraud* i *Phong.* Afegir textures per incrementar el realisme.

Objectiu: Visualització del circuit de cotxes amb diferents *shaders* 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* i saber aplicar textures. Per això s'aprendrà a passar diferents valors al *vertex shade*r i al *fragment shader.*

S'inclouran quatre 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), la classe **conjuntLlums**, que representa el conjunt de llums de l'escena i la classe **conjuntMaterials**, que conté el conjunt de materials de l'escena.

En aquesta tercera pràctica s'inclou en l'escena una o varies llums i a cada objecte el seu/s material/s associat/s. Per a començar a desenvolupar el codi d'aquesta tercera pràctica, utilitzeu el codi realitzat a la pràctica 2.

La pràctica 3 es composa dels següents exercicis:

- Implementació d'una nova classe anomenada **llum**, que permeti emmagatzemar 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.
- Implementació d'una nova classe **material** que permeti representar les característiques del material d'un objecte.
- Implementació d'una nova classe **conjuntLlums**, que permetrà guardar les diferents llums de l'escena.
- Implementació de la nova classe **conjuntMaterials**, que guardarà tots els materials dels objectes de l'escena. Així, si dos objectes tenen el mateix material, aquest material només es guardarà una vegada.
- Implementació dels càlculs d'il·luminació segons les següents tècniques: flat, *gouraud* i *phong-shading*.
- Implementació d'una nova classe **textura,** que permetrà definir una textura per utilitzar-la per donar més realisme a la visualització final.

A continuació, en aquest document, es descriuen els conceptes associats als diferents exercicis a realitzar.

1. Creació d'una nova classe llum

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 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).

En la classe **llum** cal afegir un mètode anomenat toGPU() que permeti passar la informació d'una llum a la GPU, segons la següent capçalera:

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

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

1. En la GPU (en els *shaders*) s'ha de fer un "*struct*" amb informació de la llum:

```
struct tipusLlum
{
    vec4 LightPosition;
    vec3 Ld;
    float coef_a;
    ...
};

uniform tipusLlum light;
```

Fixeu-vos que es la variable **light** és de tipus "uniform" ja que serà constant a tots els vèrtexs de l'objecte.

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

```
   // Suposem que tenim les variables posicioLlum, difusa, coef_a a la CPU. Son diferents
exemples amb diferents tipus de dades, no són tots els atributs de les llums
          vec4 posicioLlum;
          vec3 ld;
          float coef_a;

 // Per a passar a la GPU declarem una estructura amb els identificadors  de la GPU

    struct {
        GLuint posicio;
        GLuint ld;
        GLuint a;
        ....
    } gl_IdLlum;

   // obtencio dels identificadors de la GPU

    gl_IdLlum.posicio = program->uniformLocation("light.LightPosition");
    gl_IdLlum.ld = program->uniformLocation("light.Ld");
    gl_IdLlum.a = program->uniformLocation("light.a");
    ...
```

```
// Bind de les zones de memoria que corresponen a les variables de la CPU
        glUniform4fv(gl_IdLlum.posicio, 1, posicioLlum); //posicioLlum és un vec4
        glUniform3fv(gl_IdLlum.Id, 1, difusa );        // difusa és un vec3
        glUniform1f(gl_IdLlum.a, coef_a);              // coef_a és un Glfloat
```

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*. Si per exemple, li assigneu la intensitat difusa de la llum, hauríeu de visualitzar els objectes del color que heu posat a la intensitat difusa des de la CPU.

En l'escena es poden tenir diferents llums. Per això cal implementar una classe que contingui el conjunt de llums de l'escena. Per a passar-les al vèrtex shader, no es poden passar directament com una taula de structs, sinó que cal passar-les d'una a una. Fes un màxim de tres llums.

Finalment, per a que es pugui utilitzar una o varies llums en l'escena, cal que afegiu un atribut conjunt de llums a la classe escena, juntament amb la intensitat d'ambient global a l'escena. 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 **ConjuntLlums** anomenat setAmbientToGPU, amb la següent capçalera:

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

2. Creació d'una nova classe material

Implementeu una nova classe per a representar 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) i un nom de material. 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.

Quan afegiu el material a l'objecte, observeu que el vector de colors associats a cada vèrtexs ja no té sentit i per tant ja no és necessari passar-lo a la GPU. En canvi, pel càlcul de la il·luminació és necessari passar les normals associades a cada vèrtex.

Per tal que no hi hagi materials repetits en l'escena, creeu una classe amb el conjunt de materials, de forma que cada objecte tindrà un identificador de material que correspondrà a l'índex del material en el conjunt de materials.

3. Implementació dels diferents tipus de *shading*

En aquest apartat s'explica com implementar els següents diferents tipus de *shading*:

- Flat shading (color constant a cada triangle)
- Gouraud (suavització del color)
- Phong shading (suavització de les normals)

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.**

Els shaders instal·lats a laboratori són superiors a la versió 150, que no permeten l'ús de variables de tipus *varying*. 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.

A continuació, es detalla una breu explicació de cada tipus de *shading* i els fitxers que cal lliurar en cadascuna de les implementacions:

- Flat shading (Color constant a cada triangle): S'usarà per a visualitzar el terra del circuit.

A) Per implementar el flat shading d'un objecte cal tenir definides a cada vèrtex de cada cara, la normal de la cara.

B) Cal passar les normals a la GPU, modificant els mètodes toGPU i draw de l'objecte i el vertex shader.

C) Cal implementar el model de Blinn-Phong en el vertex shader.

D'aquesta forma cada vèrtex d'una cara tindrà el mateix color, ja que té la mateixa normal i només en el pas al fragment shader, es farà la interpolació del color.

- Gouraud (suavització del color a partir de les normals calculades a cada vèrtex): s'utilitzarà per a visualitzar els obstacles.

En el Gouraud shading es calculen les normals a cada vèrtex i s'interpola el color a nivell de píxels. En aquest apartat heu de calcular les normals "reals" a cada vèrtexs de l'objecte. Raoneu on és calcula la il·luminació i modifiqueu convenient els fitxers de la pràctica.

- Phong shading (suavització de les normals a nivell de píxels) : s'utilitzarà per visualitzar el cotxe.

En el Phong shading les normals s'interpolen a nivell de píxel. Raoneu on és calcula la il·luminació i modifiqueu convenient els fitxers de la pràctica.

Implementeu inicialment tots els mètodes amb una llum puntual i atenuació en profunditat. Proveu amb diferents materials canviant les diferents components de les propietats òptiques.

4. Guió de la pràctica.

1. Implementeu la classe **Llum** per a que pugui representar una llum puntual, direccional o spot-light, i que pugui representar la seva atenuació en profunditat. Considereu també que pot estar activa o no. Implementeu el mètode que passa la llum a la GPU i comproveu que el vèrtex shader la rep correctament.

2. Un cop tingueu funcionant la llum, implementeu la classe **ConjuntLlums**, afegint el mètode ConjuntLlums::toGPU(QGLShaderProgram *program), que passa les dades de les llums a la GPU i el mètode void ConjuntLlums::setAmbientToGPU(QGLShaderProgram *program) que permet passar les llums actives a la GPU i la llum ambient global a la GPU. Implementa el codi del vèrtex shader necessari per rebre un màxim de dues llums. Primer, prova-ho inserint només una llum a l'escena.

3. Implementa la classe **Material** que representa tots els valors d'un material d'un objecte de l'escena. Aquesta classe definirà els atributs de les propietats òptiques d'un objecte (component ambient, component difús, component especular i coeficient de reflexió especular) i un nom de material. Cada objecte tindrà associat un material. Afegeix el mètode Material::toGPU(QGLShaderProgram *program) que passa les dades d'un material al vèrtex shader. Codifica el vèrtex shader per a que el pugui rebre convenientment i comprova que les dades estiguin ben passades. Posa un material gris al terra per a provar-ho.

4. Implementa la classe **ConjuntMaterials** que permet guardar un conjunt de materials de l'escena per a no tenir repeticions. Cada objecte tindrà l'índex a la posició corresponent al conjunt de materials. Des d'on hauràs de passar el material de cada objecte a la GPU? Prova que els materials es passen correctament posant un material gris i difús al terra, un material vermell i especular al cotxe i un material negre i difús a les rodes del cotxe.

5. Per tal d'implementar els diferents shadings en la GPU, cal que es passin les normals dels objectes a la GPU i no el color, com es feia fins ara. Per això cal que modifiquis el mètode draw() de l'objecte i el vèrtex shader.

6. Implementa el flat shading per a aplicar a tots els objectes de l'escena, seguint les indicacions de l'enunciat. Col·loca una llum puntual per a que es vegi l'efecte del flat shading.

7. Implementa el Gouraud shading per a que es calculin les normals ponderades a cada vèrtex dels objectes. Prova-ho amb el cotxe o amb els obstacles.

8. Implementa el Phong shading per a visualitzar el cotxe. On l'has de codificar? En el vèrtex shader o en el fragment shader? Inclou una altra llum que sigui una llum del cotxe. De quin tipus serà? Modifca el vèrtex shader i el fragment shader per a que puguin suportar dues llums.

9. Si vols utilitzar diferents shaders pels diferents objectes, per exemple el flat shading pel pla, Gouraud als obstacles i Phong shading al cotxe, com pots activar i desactivar els shaders? Des d'on ho fas?