Pràctica 2: ZBuffer - Fase 2

GiVD - curs 2016-2017

Pràctica número 2: Fase 2

Tema: Interacció amb la càmera i *shaders* avançats: Implementació de la interacció amb la càmera i il·luminació basada en textures.

Objectiu: Visualització d'una escena amb interacció amb la càmera i aprendre a utilitzar textures per a realitzar diferents efectes d'il·luminació.

La fase 2 de la pràctica 2 es composa dels següents exercicis:

- Implementació del pas de les transformacions de la càmera als *shaders* per a permetre modificar la càmera amb el ratolí (canvi d'angles i zoom).
- Incorporació de mètodes realistes basats en textures.

En aquesta fase, la posició de la càmera és variable i també es permetrà fer zoom. Al principi de l'aplicació, la càmera visualitzarà tota l'escena, sense retallar-la. En carregar un nou objecte, la càmera visualitzarà tota l'escena mostrant tots els objectes que la composen. Això s'aconsegueix calculant la finestra que conté la projecció de la capsa mínima contenidora de tota l'escena.

S'ha programat el moviment de la càmera fent *drag* amb el butó esquerra del ratolí. Es pot fer zoom in i zoom out fent *drag* amb el butó dret del ratolí.

1. Etapes de desenvolupament de la fase 2 (part obligatòria):

PAS 1: Interactivitat amb la càmera

La classe GLWidget del codi base de la pràctica 2 ja està preparat per a fer la interacció amb la càmera i crida els mètodes de la classe Càmera que has d'implementar.

La classe càmera conté tota la informació relativa a la càmera (hi ha alguns atributs que són redundants i complementaris).

```
typedef enum {PARALLELA = 0, PERSPECTIVA = 1} TipProj;

 vec4 origin; // Posició de l'observador
vec4 vrp;    // View Reference Point

vec4 u, v, w; // Sistema de coordenades de la càmera

double angX; // Angles de rotacio de la camera
double angY;
vec4  vUp;    // equivalent a v

TipProj typeProjection; // tipus de projecció: Paral·lela o Perspectiva
float dant, dpost;      // Plans de clipping anterior i posterior
Capsa2D window;         // window

Capsa2D vp;             // Viewport
```

Com a atributs privats, es tenen dues matrius, que caldrà passar al shader (i per tant, també es guarden els seus identificadors de memòria de la GPU):

```
 mat4  modView;     // Matriu model-view de la CPU
mat4  proj;        // Matriu projection de la CPU
GLuint  model_view;  // model-view matrix uniform shader variable (GPU)
GLuint  projection;  // projection matrix uniform shader variable (GPU)
```

És important mantenir la coherència entre aquests atributs. Per exemple, si es canvien els angles que defineixen la posició de l'observador, caldrà canviar la matriu *modView* en CPU per a que sigui consistent amb aquests canvis.

Dins de la classe càmera trobareu mètodes que permeten el càlcul de diferents paràmetres necessaris en el càlcul de la càmera.

Modifica la classe **Càmera**, seguint els següents passos:

1. Implementació del pas a la GPU de la matriu model-view/projection des de la classe càmera

Per a poder passar les matrius en el *vertex shader* cal realitzar els següents passos (estan explicats amb la matriu *model-view* però també caldria fer-los per la matriu *projection*).

Com fer-ho?

Cal implementar els mètodes:

```
void setModelView(QGLShaderProgram *program, mat4 m);
```
Connecta la matriu model-view (modView) de la CPU amb la matriu de la GPU (model-view) usada en el *vertex shader.*

```
void setProjection(QGLShaderProgram *program, mat4 p);
```
Connecta la matriu de projecció (proj) de la CPU amb la matriu de la GPU (projection) usada en el *vertex shader.*

```
void toGPU(QGLShaderProgram *program);
```
Connecta les matrius de model-view (modView) i de projection (proj) de la CPU amb les matrius de la GPU usades en el *vertex shader.*

Com es defineix i s'utilitza la matriu model-view en el vertex shader?

Si es tenen definides a la CPU les variables:

```
mat4  modView;
GLuint  model_view;
```

I a la GPU, la matriu està definida amb el nom "model_view". El pas de la matriu a la GPU des de la CPU és:

```
model_view = program->uniformLocation("model_view");
glUniformMatrix4fv( model_view, 1, GL_TRUE, modView );
```

2. Definició i ús de la matriu model-view/projection en el vertex shader:

Cal modificar el *vertex shader* per a que apliqui les transformacions geomètriques *model_view* i *projection* a cadascun dels vèrtexs. Recordeu que la sortida dels *vertex shaders* ha de donar coordenades homogènies per tot tipus de projeccions, ja siguin paral·leles o perspectives.

Com fer-ho?

Les matrius de visualització i de projecció són constants per tots el vèrtexs de tots els objectes. Aquests tipus de variables s'anomenen uniform en els vertex shaders. A continuació s'explica com definir i usar la matriu model-view en el vèrtex shader, però també cal implementar-ho per la matriu projection.

Com es defineix i s'utilitza la matriu model-view en el vertex shader?

Definició:

```
uniform mat4 model_view;
```

Utilització:

```
void main()
  {
   .....
   gl_Position = model_view*vPosition;
```

```
.....
color = vColor;
}
```

3. Pas de la posició de l'observador des de la classe Camera:

Fixa't que en alguns *shaders* es necessari passar la posició de l'observador per a calcular correctament la il·luminació de *Gouraud* i de *Phong*. Modifica el mètode **toGPU** de la classe **Camera** per a que es passi la posició de l'observador cada vegada que s'actualitza la posició de la càmera amb el ratolí. Com serà aquesta variable al *shader*? Uniform? O IN?

PAS 2: Inclusió de textures

Inclusió de textures a les esferes

Descripció:

S'inclouran textures en la visualització del objecte que s'ha carregat.
Suposem inicialment 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 **p** del objecte, es considera el vector unitari que va des del centre, **c**, de la capsa mínima contenidora de l'objecte, fins a **p**. Aquest vector serveix per a considerar el punt d'una esfera centrada en **c** i que embolica l'objecte. Les coordenades de textura **(u, v)** es calculen a partir de les coordenades esfèriques de l'esfera seguint la següent fórmula:

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

$v = 0.5 – \arcsin(y)/ \pi$

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

Com fer-ho?

Carrega una textura per l'objecte en la seva constructora (consulta el projecte CubGPUTextures per veure com es feia a la classe **cub**). Recorda que la textura sempre s'acaba consultant en el fragment *shader*.

Implementa els mètodes de la classe Object:

```
void Object::initTextura()

void Object::toGPUTexture(QGLShaderProgram *pr)

void Object::drawTexture()
```

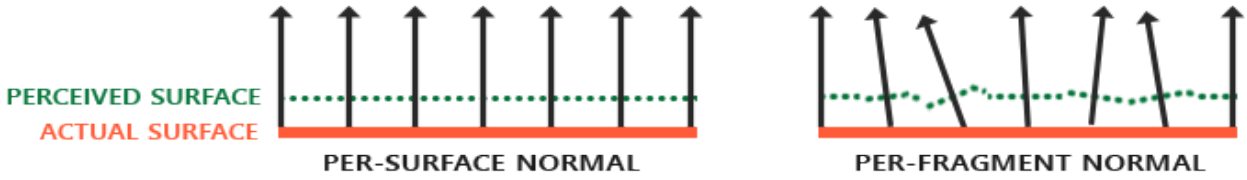
Abans de començar a implementar, planteja't les següents preguntes:
- com guardaràs la textura de l'objecte? Quan es carregarà la textura de l'objecte?
- necessites guardar les coordenades de textura de cada punt de l'objecte?
- Per a cada vèrtex calcula la seva coordenada de textura. On faràs aquest càlcul en el cas que volguessis fer-lo molt ràpid? I en el cas que volguessis una visualització més suau?

Implementa els parells de *shaders* corresponents a *Gouraud* i a *Phong shading* per a que es cridin des dels menús de l'aplicació de forma que s'obtinguin visualitzacions més suaus.

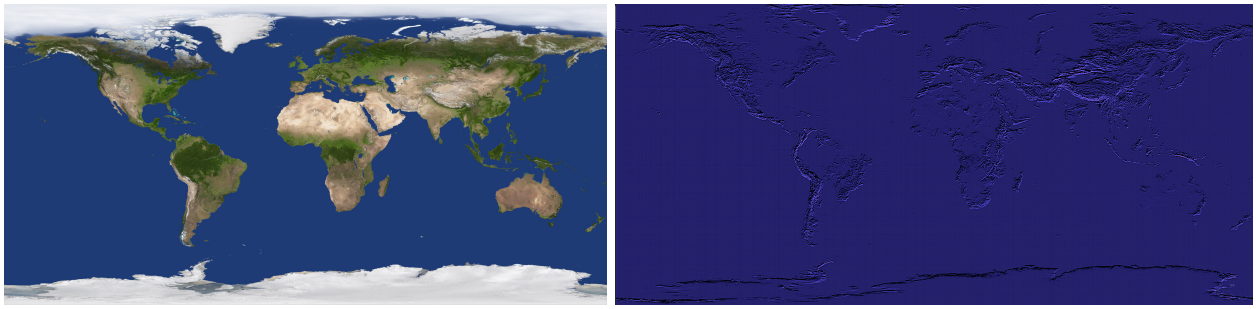
Recorda que la textura es pren com a la component difusa del material o es fa una ponderació amb el material base de l'objecte. *Per exemple, un 75% del color final potser de la textura i un 25% el color del material base.*

PAS 3: Efectes de relleu amb textures

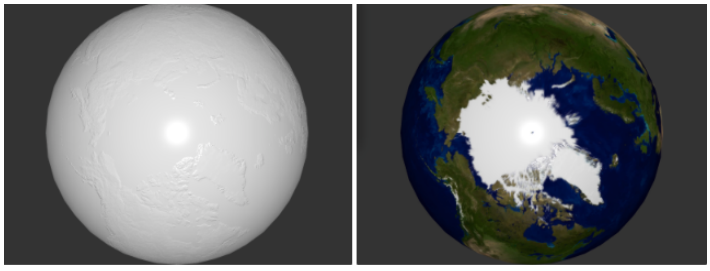
Afegir una segona textura a l'objecte que té associada una pertorbació de la normal a cada punt. Així doncs es carrega una textura amb les pertorbacions de les normals de forma que es modifica lleugerament la normal dels objectes per a calcular la il·luminació per donar sensació de rugositats:



Així, donades dues textures, una de color difús i una altra de les normals, en aplicar-les en els valors del material difús i en les normals, acaben donant efectes més realistes.



En la textura es tenen les pertorbacions codificades entre 0 i 1, com si fossin colors,quan en canvi han de ser pertorbacions de les normals que han de estar entre -1 i 1. Quan es carrega una textura de normals cal ajustar els seus valors.



Per ara considera la simplificació de sumar la pertorbació directament a la normal que ja tenies de l'objecte abans de calcular la il·luminació. A quin *shader* ho has de fer? En el *vertex shader* o en el *fragment shader*?

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. Aixi, quan passis la textura 0 faràs un bind(0) de la textura del color difús i quan vulguis passar la textura 1, faràs un bind(1) de la textura de les normals.

El càlcul que es demana a la pràctica de la pertorbació de la normal és aproximat. No es demana fer el càlcul exacte. Tot i així, si vols explorar més en detall com s'aplica del tot correctament i **opcionalment** t'animes a implementar-lo, pots consultar l'enllaç: http://www.opengl-tutorial.org/intermediate-

tutorials/tutorial-13-normal-mapping/

PAS 4: Decoració de l'entorn amb textures (OPCIONAL)

En la carpeta de recursos trobaràs una esfera que engloba la resta i que té les cares orientades cap al seu interior. Pots decorar l'entorn carregant aquesta esfera directament a l'escena i mapejant-li una textura de cel. Per fer aquest efecte, crea *shader* que s'activi a cada visualització just després de visualitzar l'escena. Aquest *shader* serà l'encarregat de visualitzar l'esfera amb la seva textura corresponent, sense tenir en compte les llums, però en canvi si que tindrà en compte la càmera.

Pots trobar altres idees d'efectes obtinguts amb els *shaders* a: http://qmlbook.github.io/ch09/index.html - qt-graphicseffect-library

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	Fase 2: Shadings avançats
	22	23	24	25	26	**L12: Entrega P2 i prova P2. Enunciat Pràctica 3**