Pràctica 2 – Fase 2

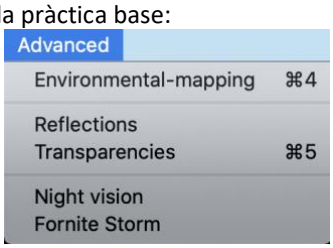
Visualització Projectiva: Animacions i *shaders* avançats

GiVD - curs 2022-2023

Pràctica 2 - Fase 2

Tema: *Exercicis sobre shaders, Animacions* i ***Shaders* avançats:** Animacions i Il·luminació basada en textures per a realitzar reflexions i transparències i realització d'animacions.

En aquesta Fase es proposa una part OBLIGATÒRIA i diferents parts OPCIONALS de la pràctica 2. Moltes d'elles es troben en el Menú Advanced de la pràctica base:



La part obligatòria consta de dos exercicis per veure com usar els shaders:

- Night vision (Visió nocturna)
- Fornite storm (La tempesta de Fornite)

Les possibles parts opcionals són:

- Animacions
- Decoració de l'entorn amb textures (environmental mapping).
- Reflexions
- Transparències
- Normal mapping
- Múltiples materials i textures per objectes
- Mapeig en dues fases (descrit al pas 4 de la Fase 1)
- Altres (mira el darrer apartat d'aquest document)

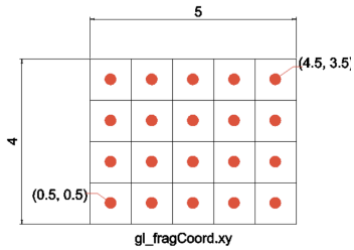
1. Part obligatòria: Exercicis sobre shaders

1.1. Visió Nocturna o Target amb cercle verd:

Es volen visualitzar tots els objectes que estan en un cercle de mida la mitat del viewport (per exemple, si es té un viewport de 512x512, el radi seria 128) mitjançant Phong Shading i només tenint en compte el canal verd. La resta es pintarien de color negre.

Detalla on es faria el càlcul? Amb quines coordenades? Amb coordenades de món? De càmera? O de viewport? Si decideixes que el càlcul es farà a nivell de viewport, pots usar la variable gl_fragCoord que indica en quin píxel s'està pintant l'objecte, tal i com indica la figura de sota. Pensa com pots aconseguir el valor del viewport

actual de l'escena.



Com aconseguiries que els píxels de fons inclosos en el cercle de visió nocturna es pintessin també de color verd?

1.2. La tempesta de Fornite:

Durant el joc de Fornite, existeix una tempesta que afecta a una regió exterior d'una esfera centrada en el (0,0,0) de món de radi 90. Durant el joc, la tempesta va creixent però en aquest exercici la farem de forma estàtica (és a dir l'esfera de la tempesta sempre tendrá radi 90).

Les regions afectades es visualitzen de colors tintats de blaus amb *Gouraud Shading* i les que no estan afectades es veuen en *Phong Shading*.

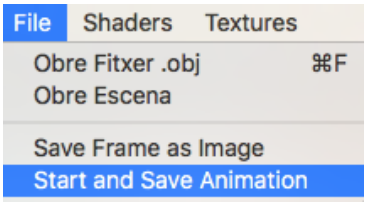
Soluciona a la teva pràctica aquest exercici de forma que en carregar una escena puguis veure aquest efecte. Si cal canviar el radi de la tempesta per què l'escena és molt petita, canvia'l adequadament.

Considera quants parells de vèrtex-fragment shaders has d'usar, a on cal considerar el test amb l'esfera, etc. Detalla-ho en el teu informe explicant la solució pensada.

2. Parts Opcionals:

2.1. Animacions [OPCIONAL 1]:

Les animacions funcionen de forma similar a la pràctica 1 (veure fase 2). Es tracta de visualitzar dades geolocalitzades en diferents instants de temps. Fixa't que per a facilitar l'enregistrament d'una animació, l'aplicació base de la pràctica porta un menú que activa i grava les animacions.



Aquest menú està lligat al mètode de la classe GLWidget anomenat saveAnimation():

```
void GLWidget::saveAnimation() {
  // Comença el timer de l'animació
```

```
    timer = new QTimer(this);
    currentFrame=0;
    currentImage=0;
    connect(timer, SIGNAL(timeout()), this, SLOT(setCurrentFrame()));
    timer->start(1000);

}
```

que engega un *Timer* i grava el *frame* actual. El mètode start del Timer està posat cada 1000 tics que equivalen a 1 segon. Pots canviar-lo per a fer actualitzacions més freqüents. Fixa't que el mètode setCurrentFrame() de la mateixa classe GLWidget crida al mètode de la classe Scene que fa el update:

```
void GLWidget::setCurrentFrame(){

    scene->update(currentFrame);
    updateGL();
    this->saveFrame();
    currentFrame++;

    if (currentFrame == MAXFRAMES)
        timer->stop();

}
```

Cal que implementis el mètode update a la classe Mesh, o dels elements que vulguis fer update, per a que apliqui la TG que calgui o canviï les propietats que vulguis animar.

PAS 1: Crea una escena amb una nova classe SceneAnimation, que defineixi la teva escena i tots els seus paràmetres i animacions.

Quan es crea l'escena cal que els objectes que es volen animar tinguin animacions. En el següent exemple s'afegeix l'animació a l'esfera per a traslladar-la (0.2, 0.2, 0.2) a cada frame:

```
void SceneFactoryBasica::OneSphere(Scene *s) {
    QString s1(":/resources/models/sphere1.obj");
    Mesh * obj1 = new Mesh(100000, s1);

   // ... Troç de codi just per a afegir animació a l'objecte
    Animation *anim = new Animation();
    anim->transf = new Translate(vec3(0.2));
    obj1->addAnimation(anim);

    s->objects.push_back(obj1);
}
```

PAS 2: Completa el codi de Builder per a que pugui visualitzar sèries temporals de dades.

2.2. Il·luminacions avançades

[OPCIONAL 2]: Environmental Mapping: Decoració de l'entorn amb textures

En la carpeta de recursos trobaràs un CubeMap d'un cel (skybox) que pot definir el background de la teva escena. Pots decorar l'entorn carregant un cub prou gran que englobi directament tota l'escena i mapejant-li una textura de cel (segons les transparències de teoria Tema 3d).

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 el cub amb la seva textura corresponent, sense tenir en compte les llums, però en canvi si que

tindrà en compte la càmera.

Trobaràs més decoracions a l'enllaç: http://www.humus.name/index.php?page=Textures

[OPCIONAL 3]: Reflexions

Utilitza la tècnica d'environmental mapping definida a les transparències de teoria (Tema3d) per a calcular les reflexions d'un objecte calculant els vectors reflectits de l'objecte en un CubeMap. Utilitza el mètode *reflect(..)* definit en glsl per a calcular el raig reflectit i accedir a la coordenada de textura corresponent al CubeMap.

Una extensió possible d'aquest mètode és realitzar les reflexions de forma dinàmica si es mouen els objectes en l'espai. T'hi animes? Per això, hauràs de saber com recuperar un buffer renderitzat en una textura. Mira l'enllaç: https://learnopengl.com/Advanced-OpenGL/Framebuffers per veure com es fa. No es tan difícil com sembla!.

[OPCIONAL 4]: Transparències

Objectes transparents GL:

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].

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);
```

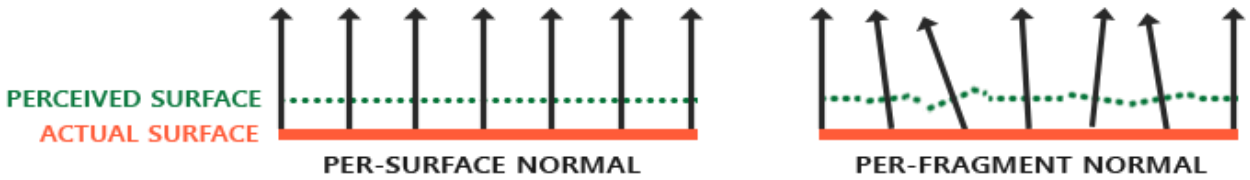
Inclou l'atribut alpha en els materials i passa'l als shaders. Prova que ara, si poses algun alpha menor a 1.0, es pinten transparents.

Objectes transparents via environmental mapping

Utilitza la tècnica d'environmental mapping definida a les transparències de teoria (Tema3d) per a calcular les transparències d'un objecte calculant els vectors refractats de l'objecte en un CubeMap. Utilitza el mètode refract(..) definit en glsl per a calcular el raig refractat i accedir a la coordenada de textura corresponent al CubeMap.

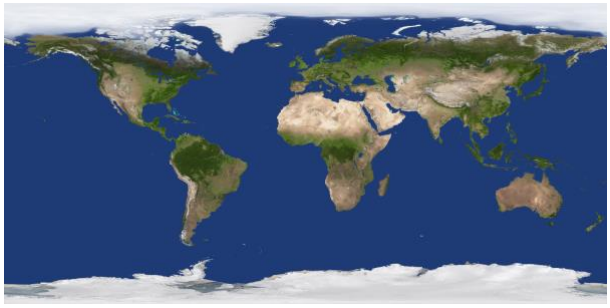
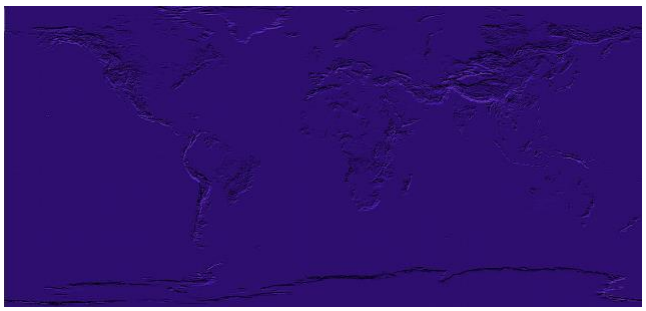
[OPCIONAL 5]: Efectes de relleu amb textures (Normal mapping)

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:

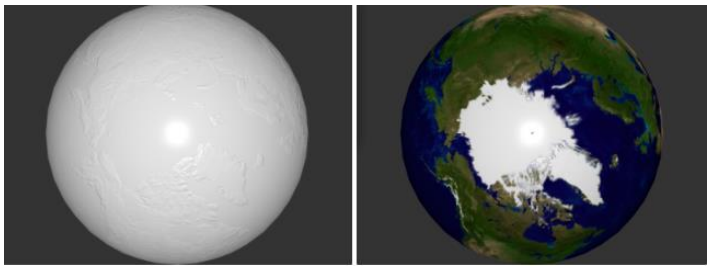


Així, donades dues textures, una de color difús i una altra de les normals, en aplicar-les en els valors del material difús i en les normals, acaben donant efectes més realistes.

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

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. Així 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. Com modificaries ara el càlcul de la nova normal per a passar-lo a l'espai tangent de la superfície? Si t'animes a implementar-lo, pots consultar l'enllaç: http://canvas.projekti.info/ebooks/Mathematics for 3D Game Programming and Computer Graphics, Third Edition.pdf (capítol 7.8)

[OPCIONAL 6] Múltiples materials i textures per objectes

Fins ara un objecte té associat un únic material. Els fitxers .obj en poden tenir diferents per diferents conjunts de triangles. Modifica la teva aplicació per a poder carregar múltiples materials i passar-los a la GPU per a que es pintin correctament.

[ALTRES]

Si aquestes extensions no t'acaben d'agradar, pots trobar altres idees d'efectes obtinguts amb els *shaders* amb glsl a l'enllaç*: https://www.shadertoy.com/browse.*
Anima't a provar d'integrar-ne algun a la teva pràctica!