TP3
FBO et effet

Pour ce TP vous devez à nouveau utiliser l'environnement `QT 5-13` avec le compilateur `mingw 64b` sous `Windows` et la librairie `QGLShaderViewer` que nous vous fournissons.

1 Rendu dans un FBO

Dans ce TP vous allez appliquer la technique de *render to texture* qui consiste à faire une passe de rendu dans un `Frame Buffer Object`.

1.1 Création d'un FBO

La librairie proposée pour ce cours possède une classe FBO qui vous permet d'utiliser un `Frame Buffer Object`.

L'initialisation d'un FBO demande un certain nombre de paramètres :

- `h` et `w` pour la résolution du FBO (hauteur, largeur)
- un format de FBO (par défaut `GL_RGBA8` pour un rendu de couleur)
- un booléen par défaut à `false` pour faire un rendu des couleurs, si ce booléen est à `true`, le FBO stockera la profondeur de la scène rendu et non la couleur.

Attention : votre FBO doit avoir la même taille que la fenêtre de rendu. Ce qui implique que lorsque la fenêtre change de taille, la taille du FBO doit être mise à jour :

- le changement de la taille de fenêtre déclenche l'appel de la méthode `resizeGL(...)` associée à la fenêtre.
- la classe FBO a une méthode `resize(...)`

1.2 Rendu dans un FBO

Le rendu dans un FBO est similaire à un rendu écran, donc dans la méthode `draw()` :

- Un FBO doit être activé avant d'y faire un rendu grâce à sa méthode `bind()`
- Il doit être réinitialisée avant d'y faire un rendu comme pour un rendu classique : `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)`
- Les objets de la scène sont rendus
- Le FBO doit ensuite être désactivé avec la méthode `release()`

1.3 Accéder au contenu du FBO

Maintenant on va afficher un cube avec la scène calculée comme texture. Pour y parvenir vous devez :

1°) Modifier votre matériau de texture avec un FBO en paramètre à sa création.

2°) Modifier la méthode `bindSpecific` afin d'utiliser la texture du FBO comme texture à afficher. Pour cela il faut utiliser la méthode `glBindTextures(...)` qui prend en paramètres :

— le type de texture, ici `GL_TEXTURE_2D`

— le numéro de la texture, ici celle correspondant à votre FBO que l'on obtient avec la méthode `getGetexID()`

3°) Il faut maintenant faire le rendu de notre cube :

— Définisser un mesh spécifique comme attribut de votre classe `TestViewer`, correspondant à votre cube que vous n'ajoutez pas à la liste des entités de la scène.

— Définisser un FBO comme attribut de votre classe `TestViewer` que vous initialisez dans la méthode `init` (relire la partie "1.1 Création d'un FBO").

— Modifier la méthode `drawMono()` pour rendre les objet de la scène dans le FBO (relire la partie "1.2 Rendu dans un FBO") puis n'oubliez pas de réinitialiser la sortie couleur et le tampon de profondeur à nouveau avant d'utiliser la méthode `render(...)` cette fois sur votre cube.

2 Ajout d'un effet

Il est très courant dans les techniques de rendu d'utiliser des étapes de post-traitement ou *post-processing* comme avec les exemples du dernier cours. Un effet consiste à prendre un ou plusieurs FBOs en entrée pour y appliquer des traitements similaires à ceux vu sur `Shadertoy`.

2.1 Création d'un effet

4°) Afin de réaliser cela vous allez créer une classe `MyEffect` :

— Elle héritera de la classe `Effect`.

— Le constructeur d'un effet prend en paramètre le chemin d'accès des shaders associés.

— Vous ajouiterez une méthode `render` qui prendra en paramètre un FBO. Le code générique du rendu d'un effet est le suivant :

```
// activation des shaders
        m_program->bind();
// passage des uniformes, des textures ou texture de FBO
        ... //votre code
// affichage du Quadrangle couvrant l'ecran
        m_geometry->bindVertices( attribPos );
        m_geometry->draw();
// desactivation des shaders
        m_program->release();
```

Pour complèter ce code il faut savoir que aa classe `Effect` dispose d'une méthode qui permet de lier directement un FBO à un canal de texture `attachTexture(n,name,monFBO)` : le canal de texture `n` sera associé à la texture de `monFBO` et correspondra à la texture `name` dans le fragment shader.

2.2 Utilisation de l'effet en version simple

Ici nous allons utiliser l'effet simplement pour afficher sur l'écran le contenu du FBO de la partie 1.

5°) Ajouter votre effet comme attribut de votre classe `TestViewer` que vous initialisez dans la méthode `init`.

6°) Ici vous n'avez plus besoin du cube, dans le méthode `drawMono` vous allez rendre directement votre FBO dans le quad définit par votre effet à l'aide de la méthode `render` de votre effet.

7°) Comme pour les matériaux, il vous faut un vertex shader et un fragment shader pour votre effet.

Le vertex shader sera systématiquement le code qui suit :

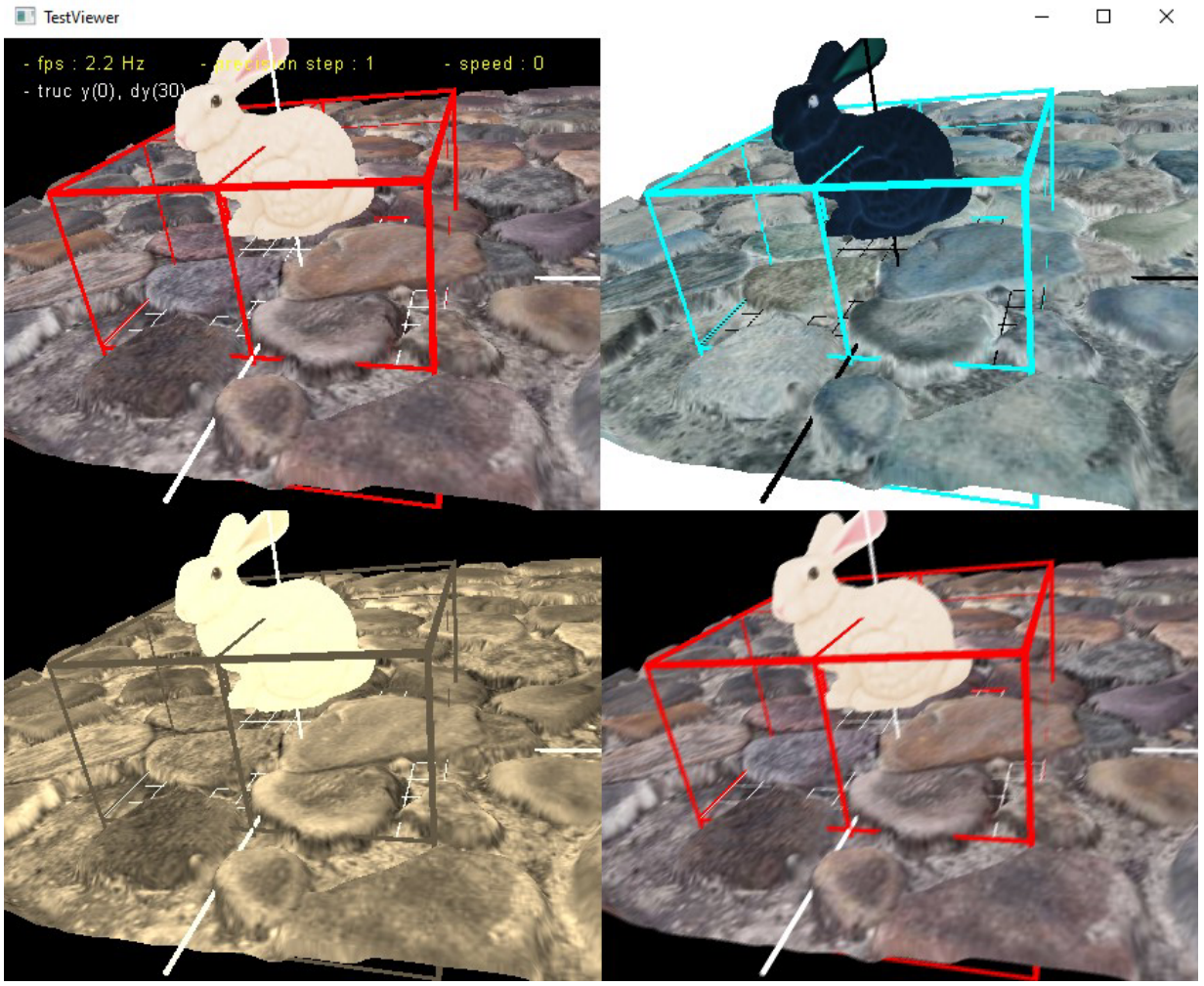
```
#version 430
in vec3 pos;
out vec2 uv;
void main()
{
        gl_Position = vec4(pos,1.0);
        uv= pos.xy/2.0+vec2(0.5);
}
```

Le squelette d'un fragment shader d'effet est :

```
#version 430
in vec2 uv;
void main()
{
        gl_FragColor = ...
}
```

2.3 Création d'un effet plus complexe

Vous allez rendre votre scène dans un FBO. Vous ferez un effet qui donne le résultat suivant :



L'image de votre rendu sera affichée 4 fois (voir exemple ci-dessous) :

— en haut à gauche, l'image de votre FBO

— en haut à droite, l'image de votre FBO en couleur inversée

— en bas à gauche, l'image de votre FBO en ton Sépia

— en bas à droite, l'image floue (avec 25 voisins pour mieux voir le résultat !)

Bonus : une touche du clavier doit permettre de passer d'un rendu normal à un affichage de votre effet.

Conseil :

— Commencer par afficher 4 fois votre texture.

— Pour l'image floue, vous avez besoin de la taille de la texture afin de parcourir les voisins. Dans votre effet, vous pouvez récupérer la taille du FBO en utilisant getSize() puis passer la largeur et la hauteur au fragment shader.

3 TP à rendre

3.1 Deux effets

Vous allez rendre votre scène dans un FBO. Vous ferez les deux effets suivants :

— un effet imposé : un damier transparent : les cases noires seront transparentes (on y voit votre scène) et les blanches s'afficheront blanches

— un effet libre : qui doit donc être différent pour chaque étudiant sinon cela sera pénalisé

3.2 Un matériau évolué

Vous allez combiner le rendu de type Phong avec des textures qui vous sont fournies sur le *moodle* du cours avec le TP3 :

— Une texture "diffuse.jpg" représentera le coefficient `Kd` du modèle de Phong.

— Une texture en niveau de gris "specexponent.jpg" influera sur `Ks` ainsi Ks =128*valeur (avec valeur récupérée dans la texture).

— Une texture "specstrenght.jpg" influera sur l'importance de la composante spéculaire ainsi :

  — Couleur finale = ambient + diffu + valeur2 * spéculaire (avec valeur2 récupérée dans la texture)

  — la couleur ambiante sera ici (0.1,0.1,0.1,1.)

Bonus : vous pouvez utiliser la texture "normal.jpg" pour modifier la normale de l'objet.

3.3 A rendre

Ce qui doit être rendu :

— le code de vos effets (.h, .cpp et shaders)

  — les parties de code venant de vous doivent être commentées

  — une capture de l'application pour chaque effet

— le code source de votre matériau évolué (.h, .cpp et shaders)

  — les parties de code venant de vous doivent être commentées

  — le matériau évolué sera rendu sur un cube

  — 3 captures d'écrans de votre cube vue sous différents angles

— le code source de la fenêtre (.h, .cpp), qui n'utilise que les classes d'origines de la librairie exception faite de votre effet et votre matériau (afin de pouvoir rapidement tester votre code sans avoir de modification à apporter).