GPGPU
Paysage côtier (12h)
CPE

5ETI IMI

1 But

Vous devez pour ce TP créer un paysage côtier.
Exemple de scène réalisable : https://github.com/jbouny/fft-ocean.
Le site https://opengameart.org/ propose des fichiers de texture et de maillage libre de droit.

Dans un objectif d'apprentissage, plusieurs techniques vous serons présentées. Vous pourrez les réutiliser afin de créer votre scène. Les points importants de openGL abordés sont :

1. les Frame Buffers Objects (FBO),
2. les Transform Feedbacks (TF),
3. les Compute Shaders (CS).

Vous apprendrez également à utiliser le bump mapping pour créer du détail à une géométrie grossière.

La qualité du résultat final n'est évaluée, l'important de ce projet est la bonne compréhension des outils proposés et leur juste utilisation. Le rapport devra expliciter les notions abordée, leurs possibles utilisations et leur implémentation dans le cadre du TP. Vous devez détailler dans le rapport ce qui a été mis en place mais ÉGALEMENT les tentatives infructueuses, les difficultées rencontrées et les solutions que vous auriez pu mettre en place.

2 Prise en main de l'environnement

2.1 Compilation

Question 1 *Compilez le code, assurez vous de voir un maillage texturé, vous pouvez bouger la caméra à la souris.*

Question 2 *Observez les fichiers du projet (or dossier external) afin de comprendre le fonctionnement du projet. Vous devez comprendre l'ensemble du code.*

3 Bump mapping

Création d'une surface animée sans géométrie.

Question 3 *Créez une grille (quad) sur le plan (Oxz), une fonction est donnée. Affichez la normal map de l'eau.*

Question 4 *Visualisez la surface de l'eau avec l'éclairage de Phong. Dans un premier temps, la grille étant sur le même plan que la texture, vous n'avez pas besoin de rectifier la normale de la texture avec la normale géométrique .*

Question 5 *Animez votre texture avec une translation dépendante du temps.*

Question 6 *(Option) Dans le cas général, il faut appliquer la transformation de la normale géométrique à la normale de la texture. Effectuez ce travail. Attention, il s'agit d'un changement de repère.*

Question 7 *(Option) Utilisez une carte d'élévation et effectuez vous même le calcul de la normale dans le fragment shader.*

4 Frame Buffers Objects

On souhaite maintenant créer un FBO. Cet objet permet de ne pas afficher à l'écran mais à l'interieur d'une autre *frame*. Une frame est composée d'une couleur, d'une profondeur et d'un stencil ($\sim$ profondeur mais avec plus de contrôle). C'est un premier pas vers le GPGPU. Les FBO permettent notamment d'effectuer du post-processing sur la scène mais aussi de créer des "mirroirs", d'afficher une autre image calculé dans la scène (ex. télévision) ou de créer un affichage de débug.

4.1 À l'initialisation

Pour créer le FBO et le stocker dans une texture, il faut :

- Créer un framebuffer : `glGenFramebuffers(...)`
- Utiliser ce framebuffer : `glBindFramebuffer(...)`
- Créer un buffer de texture : `glGenTextures(...)`
- Utiliser la texture : `glBindTexture(...)`
- Créer la texture vide : `glTexImage2D(...)` (utiliser des octets non signés)
- Configurer la texture :

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

- Attacher la texture au FBO : `glFrameBufferTexture2D(...)`

Il faut ensuite crée un buffer de rendu contenant les informations de profondeur et de stencil:

- Créer un buffer de rendu: `glGenRenderbuffers(...)`
- Utiliser ce renderbuffer : `glBindRenderbuffer(...)`
- Allouer la placer à ce buffer : `glRenderbufferStorage(...)`
- Lier le framebuffer courrant et le renderbuffer (profondeur et stencil) :

```
glFramebufferRenderBuffer(...)
```

Il faut ensuite tester le bon déroulement `glCheckFramebufferStatus(GL_FRAMEBUFFER)` et remettre le framebuffer courant à l'affichage `glBindFramebuffer(GL_FRAMEBUFFER, 0)`

4.2 À l'affichage

Pour créer le FBO avec le bon contenu, il faut effectuer le rendu normalement, seulement il faut utiliser le framebuffer voulu. Pensez à nettoyer l'image (`glClear` et `glClearColor`). Dans certains cas, il peut être nécéssaire de préciser à OpenGL les informations sur la fenêtre `glViewport(...)`

Question 8 *Implémentez un FBO. Une fonction de création d'image à partir du FBO courrant est proposée dans* `glhelper.h`

Question 9 *Utilisez la texture générée par le FBO pour créer un effet mirroir sur un quad.*

Question 10 *(Option) Implémenter le reflet sur l'eau.*

5 Transform Feedbacks

Les transform Feedbacks permettent de modifier les vertices contenus dans un buffer. Seul le vertex (et dans certains cas le geometry shader) sont utilisés. Les transform feedbacks ne servent pas à faire de l'affichage. Il est cependant possible d'utiliser les buffers modifiés pour de l'affichage. Les vertices ne sont pas obligatoirement représentatif d'une géométrie et les TF sont un moyen de faire de GPGPU.

Pour effectuer un TF:

5.1 À l'initialisation

- Créer les shaders
- Créer un programme et lier les shaders
- ▲ IL FAUT PRÉCISER LES ÉLÉMENTS DE SORTIE AVANT DE LINKER LE PROGRAMME.

  ```
  glTransformFeedbackVaryings(...)
  ```

- Linker le programme
- Créer et utiliser les VAO, VBO et EBO normalement (voir `mesh.cpp` pour plus d'information). Il faut créer un/des VBO pour le stocker l'information issue du TF.

5.2 À l'affichage

- Désactiver l'affichage : `glEnable(GL_RASTERIZER_DISCARD)`
- Préciser le programme de TF
- Préciser le VAO
- Préciser le VBO
- Préciser le buffer d'écriture du TF : `glBindBufferBase(...)`
- Faire la transformation :

  ```
  glBeginTransformFeedback(...);
  glDrawArrays(...);
  glEndTransformFeedback();
  ```

- Attendre le buffer `glFlush(...)`
- Réactiver l'affichage `glDisable(GL_RASTERIZER_DISCARD)`

Pour lire le contenu d'un buffer, vous pour utiliser : `glGetBufferSubData(...)`. Dans le cas où un géométrie shader est utilisé, le nombre de vertices peut changer. Il est alors nécéssaire d'utiliser un *query objects*. Plus d'information sont disponible : https://open.gl/feedback.

Question 11 *Implémenter un TF simple permettant de modifier un maillage.*

6 Compute shaders

À voir en TP 4