GPGPU
Geometry shaders et introduction au FBO
CPE

5ETI IMI

1 But

L'objectif de ce TP est d'apprendre à utiliser les geometry shaders.

2 Prise en main de l'environnement

2.1 Compilation

Question 1 *Compilez le code, assurez-vous de voir un maillage gris.*

Question 2 *Dans `glhelper.h`, qui reprend le travail effectué en TP1, assurez-vous de comprendre le role de chaque fonction.*

3 Gestion de plusieurs programmes

Vous pouvez utiliser un `std::vector<GLuint>` pour stocker les différents shaders utilisés par la suite, un compteur permettra de selectionner le shader voulu.

Question 3 *Créez un second programme contenant un fragment shader qui permet d'afficher un maillage avec sa texture.*

4 Gestion des uniformes

Question 4 *Regardez le passage de paramètres uniformes (paramètre commun à tous le programme GPU) pour la camera. Sur le même modèle, modifiez votre fragment shader pour qu'il évolue dans le temps. Vous pouvez utiliser la librairie `chrono` de la stl.*

```
auto t_start = std::chrono::high_resolution_clock::now();
...
auto t_now = std::chrono::high_resolution_clock::now();
float time = std::chrono::duration_cast<std::chrono::duration<float>>(t_now - t_start).count();
```

5 Ajout des geometry shaders

Question 5 *À la manière de `create_program_from_file(...)`, ajoutez une fonction pour créer un programme avec en plus, un geometry shader.*

Question 6 *Modifiez le programme principal afin de prendre en compte le geometry shader `basic.gs`. Compilez, lancez le programme, qu'obtenez-vous, est-ce prévisible ?*

Question 7 *Créez un geometry shader permettant de créer une vue éclatée de l'objet. Il suffit de déplacer les sommets dans le sens de la normale du triangle.*

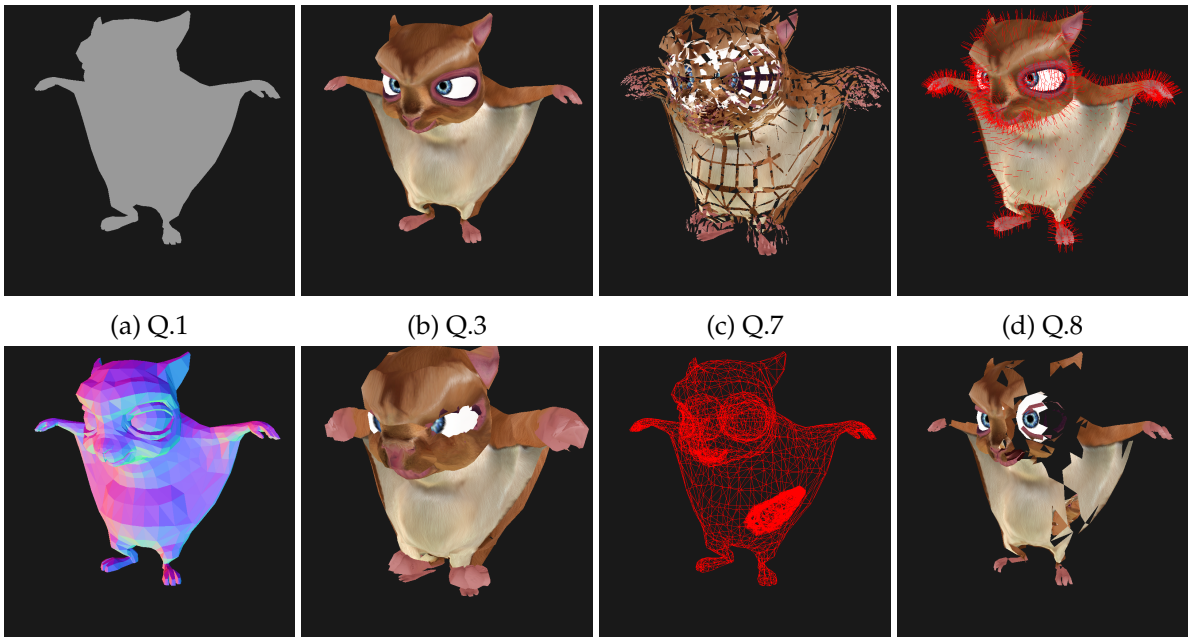
Question 8 *Créez un geometry shader afin de visualiser les normales sous la forme de ligne par-dessus le maillage. Il vous faudra utiliser deux programmes dans la fonction d'affichage, l'un pour le maillage, l'autre pour les normales.*

Question 9 *Créez un geometry shader qui calcule la normale à la surface et permet ensuite d'afficher la couleur associée dans fragment shader (passage de paramètre en shaders).*

Question 10 *Créez un geometry shader qui permet de "gonfler" le maillage : déplacement des sommets dans le sens de la normale de ceux-ci. Est-ce utile d'utiliser un geometry shader ?*

Question 11 *Créez un geometry shader permettant visualiser non plus des triangles plein mais les lignes des triangles.*

Question 12 *Créez un geometry shader permettant d'effectuer un "face culling", c'est-à-dire supprimer les triangles dont la normale n'est pas dans le sens de la caméra.*



(a) Q.1 (b) Q.3 (c) Q.7 (d) Q.8

(e) Q.9 (f) Q.10 (g) Q.11 (h) Q.12

Figure 1: Images attendues pour chaque étape

6 Frame Buffers Objects - Introduction

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 pour chaque fragment : 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 calculée dans la scène (ex. télévision) ou de créer un affichage de débug.

6.1 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 créer un buffer de rendu contenant les informations de profondeur et de stencil:

- Créer un buffer de rendu: `glGenRenderbuffers(...)`
- Utiliser ce renderbuffer : `glBindRenderbuffer(...)`
- Allouer la place 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)`

6.2 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 13 *Implémentez un FBO. Une fonction de création d'image à partir du FBO courrant est proposée dans* `glhelper.h`

Question 14 *Utilisez la texture générée par le FBO pour créer un effet de flou en post-processing sur un quad.*