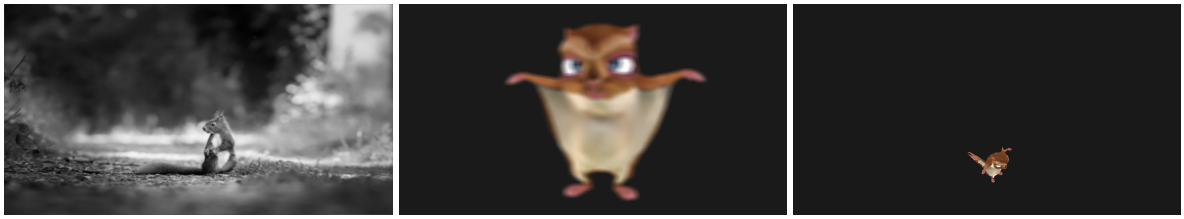
GPGPU
Introduction aux méthodes de programmation générale
CPE

5ETI IMI



(a) Traitement noir et blanc (b) Post-traitement d'une scène, flou moyenneur (c) Maillage en chute, la connectivité est inchangée

Figure 1: Résultats du TP

1 But

Les cartes graphiques ont été conçues pour faire du calcul massivement parallèle. L'objectif de ce TP est de mettre en avant trois utilisations de GPGPU.

1. traitement d'image (simple et convolutif),
2. post-traitement,
3. modification de géométrie

2 Organisation du projet

Vous disposez de trois morceaux de codes :

programme_1 permet d'effectuer un traitement sur une image affichée

programme_2 permet d'effectuer un post-traitement sur une scène

programme_3 permet de modifier un maillage sur la carte graphique

3 Traitement d'image

3.1 Programme 1

Question 1 *Compilez le code, assurez-vous de voir l'image s'afficher.*

Question 2 *Assurez vous de bien comprendre ce que fais le programme.*

Question 3 *Créez un nouveau shader pour effectuer un traitement simple (noir&blanc, ...)*

Question 4 *Créez un nouveau shader pour effectuer un traitement convolutif (filtre moyenneur, ...). On peut soit utiliser la résolution de l'image pour retrouver le voisinage, soit utiliser un rayon avec un nombre de samples prédéfinis.*

3.2 Shadertoy (optionnel)

Question 5 *Utilisez le site* `https://www.shadertoy.com/` *pour réaliser les même traitements.*

4 Post-traitement par RenderToTexture/Framebuffer Object

4.1 Prise en main du programme 2

Question 6 *Compilez le code, assurez-vous de voir un maillage texturé, vous pouvez bouger la caméra à la souris.*

Question 7 *Assurez vous de bien comprendre ce que fait le programme.*

4.2 Objectif

Un FBO est un objet qui permet de ne pas afficher à l'écran mais à l'intérieur 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). Pour pour stocker ces informations, il faut une texture (couleur) et un buffer de rendu (profondeur/stencil).

Dans ce TP, nous utilisons le FBO pour effectuer du post-processing sur la scène. Ils permettent aussi de créer des "mirroirs", d'afficher une autre image calculée dans la scène (ex. télévision) ou de créer un affichage de debug.

4.3 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, la taille de la texture sera la résolution de votre image stockée)
- 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)`. Pour définir le FBO à utiliser, il faut utiliser `glBindFramebuffer(GL_FRAMEBUFFER, fbo_id)`. L'écran est le FBO 0.

4.4 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`). Il faut également préciser à OpenGL les informations de la taille de la frame `glViewport(...)`. Pour le FBO que vous avez crée, il s'agit de la taille de vos texture, pour l'écran il s'agit de la taille de la fenêtre (`cam.width()` et `cam.height()`).

Question 8 *Créez un FBO avec la texture et le buffer de rendu associés ainsi qu'un programme et un quad pour afficher l'image post-processée.*

Question 9 *Rendez la scène dans la texture du FBO et utilisez la pour afficher votre image post-processée.*

5 Modification de buffer par Transform Feedbacks

Les Transform Feedbacks permettent de récupérer dans la VRAM [1], les sorties du vertex shader (ou si présent, du geometry shader) dans un buffer. Ils ne servent pas à faire de l'affichage et le pipeline s'arrête avant la rasterisation. Il est possible d'utiliser les buffers issue du TF pour faire de l'affichage avec une utilisation en ping-pong (un en lecture, un en écriture puis inversement). Les vertices ne sont pas obligatoirement représentatifs d'une géométrie et les TF sont un moyen de faire de la GPGPU.

5.1 À l'initialisation

Il faut créer un programme GPU comme pour l'affichage, cependant il faut préciser les variables à récupérer par le TF :

- Créer les shaders
- Créer un programme et lier les shaders
- ⚠Préciser les éléments de sortie du TF avant de linker le programme. `glTransformFeedbackVaryings(...)`
- Linker le programme
- Créer et utiliser les VAO, VBO et EBO normalement (voir `mesh.cpp` pour s'inspirer). Il faut créer un/des VBO supplémentaires pour stocker l'information issue du TF (de la même façon que les VBO d'affichage).

[1] Vous pour utiliser : `glGetBufferSubData(...)` pour récupérer un buffer sur RAM

5.2 À l'affichage

Attention, après l'utilisation des TF, il faut redéfinir le VAO impacté!

- Désactiver l'affichage : `glEnable(GL_RASTERIZER_DISCARD)`
- Préciser le programme de TF
- Préciser le VAO (et VBO/pointer si besoin)
- Préciser le buffer d'écriture du TF : `glBindBufferBase(...)`
- Faire la transformation sur les primitives POINTS [2]:
```
glBeginTransformFeedback(...);
glDrawArrays(...);
glEndTransformFeedback();
```
- Attendre le buffer `glFlush(...)`
- Réactiver l'affichage `glDisable(GL_RASTERIZER_DISCARD)`

Attention, il faut décrire de nouveau le fonctionnement des buffers après utilisation des TF!

Dans le cas où un geometry shader est utilisé, le nombre de vertices peut changer. Il est alors nécessaire d'utiliser un *query objects*. Plus d'information sont disponible : https://open.gl/feedback.

Question 10 *Observer le code de* tf.vert*, et comprendre comment le maillage se déplacera.*

Question 11 *Créez le programme pour le transform feedback en précisant les variables à capturer et créer le VAO, les 4 VBO et l'EBO et les envoyer sur le GPU.*

Question 12 *Utilisez le programme de TF et le programme d'affichage et les deux VBO en ping pong pour visualiser le déplacement du maillage. Pensez à redéfinir le VAO après le TF.*

6 Questions de réflexions

Question 13 *Les FBO et les TF permettent d'obtenir les sorties du pipeline graphique à différentes étapes, quelles sont-elles?*

Question 14 *En quoi les TF se différencie d'une modification dans le vertex shader pour un affichage classique? Donnez un exemple concret. Pourquoi cela est avantageux par rapport à une modification dans le programme principal ?*

Question 15 *En quoi l'écriture dans une texture par les FBO peut-être préférable à une écriture dans un buffer de type VBO?*

[2] Les TF se font sur les primitives, dans le cas des triangles, on écrirait tous les triangles sans les indexer