TD GPGPU n°1

Cours de GPGPU
MII 2 option SIS

GPGPU grâce à GLSL

Utilisation du pipeline graphique pour des traitements d'images simples à modérément complexes

---

Le but de ce TD est de réaliser dans un premier temps des traitements d'images basiques (recopie, calcul de la luminance, transformation en sépia...) et de comparer les temps d'exécution entre le CPU et le GPU. Après cette prise en main, nous abordons les traitements de type convolution.

Ressource :

Du code source vous est fourni sur le site internet http://www-igm.univ-mlv.fr/~biri/. C'est une application GL / glut simple permettant de réaliser des traitements simples sur une image, que ce soit avec le CPU ou avec le GPU.

Le code est divisé en plusieurs modules :

- *main* est le point d'entré du programme. Notez ainsi que l'image source doit être fournie en argument à l'appel de la fonction du type

```
./aSimpleGLSLprog -f <image>
```

- *interface :* contient toutes les fonctions d'IHM. Si vous ne connaissez pas glut, sachez que c'est une bibliothèque d'IHM fonctionnant par fonctions de callback (fonction appelées lors de l'action de l'utilisateur : appui sur une touche du clavier, d'un clic souris, d'un déplacement souris...). Il existe même une fonction appelée en permanence (idleFunc). Inspectez les fonctions *init* et *kbdFunc*
- *display* : contient le nécessaire pour faire les dessins. Inspectez les fonctions initDisplay et *drawFunc*. *drawFunc* est **la fonction appelée à chaque fois que l'on dessine la fenêtre**. Elle est séparée en deux parties. La première partie réalise un traitement sur l'image (en CPU ou en GPU sur un buffer séparé (un FBO)). La seconde affiche une image : soit l'image initiale (touche F0), soit l'image générée par le CPU (touche F1), soit celle générée par le GPU (touche F2).
- *ImageTexture* : c'est une classe C++. Inutile de connaître le C++ pour la comprendre. Il s'agit d'une encapsulation d'une image et d'une texture GL. Les fonctions intéressantes sont *initTexture*, *chargeTexture* et *dechargeTexture* qui, respectivement, crée la texture, la charge sur une unité de texture et la décharge d'une unité de texture. Grâce à cet objet, vous avez également accès à sa taille (membre *tailu* et *tailv* pour largeur et hauteur), **ainsi qu'à l'identifiant GL lui correspondant nommé *indbind***. Enfin, vous pouvez accéder directement aux pixels de l'image grâce aux fonctions *get* et *set*.
- *rtfbo* : contient le nécessaire pour créer des framebuffers interne à la carte graphique (nous en utiliserons un pour faire nos traitements d'images en « offline » c'est à dire pas sûr le framebuffer d'affichage). rtfbo n'est pas intéressant pour vous.
- *rtshader* : contient un ensemble de fonctions permettant de charger des shaders. Inspectez particulièrement les fonctions *loadShader*, *linkProgram*, *compileProgram*, *compileVertex* et *compileFragment*.
- *TimerManager* : classe C++ permettant de créer et lancer des timers. Les fonctions importantes sont *addOneTimer*, *startOneExecution*, *stopOneExecution* et *getLastTime*. Une utilisation d'un timer est déjà programmé dans la fonction drawFunc de display.cpp. Un seul *TimerManager* par programme est nécessaire et il est déjà créé (voir dans *interface*). Il peut par contre générer de multiples timers.

- *imageComputation :* c'est là que vous allez appeler et « configurer » vos shaders.

L'application s'utilise comme suit :

Lors du lancement du programme, après avoir fourni sur la ligne de commande une image, vous pouvez exploiter les touches suivantes :

- Touche 'escape' : Sort du programme,
- Touches '1','2','3','4' et '5' : Appel les traitements 1, 2, 3, 4 et 5
- Touche 'i' : affichage d'information
- Touche '<' : message d'aide
- Touche 'x' : Switch entre traitement CPU & GPU
- Touche 'f' : Lance les benchmark permettant d'obtenir le FPS total de traitement de l'image
- Touche 't' : Affiche le temps précis d'un traitement (CPU ou GPU) à chaque affichage. Éviter d'utiliser en conjonction avec la touche 'f'.
- Touche 'espace' : Relance un affichage (et donc un traitement). Utile en conjonction avec la touche 't'.
- Touche 's' : Sauvegarde l'image affichée dans le fichier *./essai.ppm*
- Touches F1, F2 et F3 : Affiche respectivement les images source, CPU et GPU

Enfin deux shaders minimalistes, *tstshader.vert* et *tstshader.frag*, sont disponibles dans le dossier *shaders*.

Exercice 1 : Initiation

Après vous être familiarisé avec le code, essayer de réaliser, en CPU, une fonction permettant de faire la transformation RBG => BGR. Pour ce faire, remplissez simplement la fonction *makeColorSwitch* du fichier *imageComputation*.

Puis, en suivant les commentaires dans le code, remplissez la fonction *makeColorSwitchGPU* du fichier *imageComputation*. Pour ce faire, il vous faudra, compléter le shader initial proposé, notamment en introduisant une variable uniforme de type sampler2D qu'il faudra « renseigner » coté CPU (en indiquant le numéro de l'unité de texture soit 0 ici). Ensuite exploiter la fonction *texture2D* permettant de « fetcher » un pixel d'une texture à l'aide des coordonnées de texture.

Vérifiez les résultats et vérifier les temps de calcul.

Exercice 2 : Traitement d'image local à un pixel

2.A : Dans un premier temps, réalisez la transformation d'une image en niveau de gris. Pour cela, transformer chaque pixel en son niveau de luminance donné par la formule suivante :

$$L = 0.3 * R + 0.59 * G + 0.11 * B$$

Faites le en CPU puis en GPU. Pour la version en GPU, construisez une matrice 3x3 et appliquez la au pixel source. Une fois les deux fonctions réalisées, vérifiez et comparez les résultats en temps de calcul.

2.B : Réaliser un effet sépia. Pour ce faire, on ajoute à chaque composante RGB de chaque pixel, une couleur « sepia » multipliée par la luminance. Cela donne :

$$R = (1+sepia\_r) * L$$
$$G = (1+sepia\_g) * L$$
$$B = (1+sepia\_b) * L$$

Il existe une variable globale *sepiaColor* (dans *display*) qui permet de configurer notre shader (en CPU et en GPU). Construisez les deux fonctions *makeSepia* et *makeSepiaGPU* et ensuite, donnez l'opportunité à l'utilisateur de modifier la couleur sepia grâce aux touches du clavier (exemple les touches 'o' et 'p' pourraient diminuer ou augmenter la composante rouge de *sepiaColor* alors que 'l' et 'm' modifieraient la composante verte).

Exercice 3: Convolutions

3.A : Dans cet exercice, nous nous attelons à la conception de convolution. Dans un premier temps, nous abordons la convolution simple de flou. On prend un simple masque 3x3 et ajoute toutes les contributions avant de diviser par 9.

Réaliser la version CPU
Réalisez la version GPU. Il faut prendre soin de transmettre les informations nécessaires pour accéder aux pixels voisins du pixel sur lequel on travaille.

Vérifiez les deux résultats et comparer les temps de calculs.

3.B : Réaliser un masque de sobel.

3.C : Créez des fonctions génériques (CPU & GPU) permettant de réaliser une convolution 3x3 ayant en paramètre n'importe quel *kernel*. Pour la version GPU, on peut s'appuyer sur les structures en GLSL.

3.D : Essayer une convolution 5x5 et comparez les temps de calcul.

3.E : Topologie : réalisez une ouverture (en deux passes donc). Réalisez ensuite une fermeture.