TP 8. Programmation des GPU avec Cuda (suite)

Préparation

Récupérer l'archive compressée `TP8.tar.gz` et extraire les fichiers qui sont contenus dans cette archive.
Se placer dans le répertoire `TP8` :

```
cd TP8
```

et préparer les compilations dans les points suivants avec la commande ci-dessous :

```
./build.sh
```

1 Traitement d'image

Il s'agit de générer à partir d'une image couleur d'origine, une autre en nuances de gris où les contours sont mis en avant.
On fournit un code séquentiel (dans `src/cpu/sequential`) pour processeur CPU qui effectue les opérations suivantes :

- Lecture de l'image d'origine
- Génération d'une image en nuances de gris
- Application d'un filtre gaussien pour obtenir une image plus régulière
- Application d'un filtre de Sobel (calcul de la valeur absolue approchée du gradient en chaque pixel), pour obtenir une image (en nuances de gris) où les contours sont renforcés.
- Enregistrement de cette dernière image dans un fichier.

(une version OpenMP est également fournie pour comparaison de performances).
La compilation avec build.sh fournit les exécutables `./install/image_cpu` et `./install/image_cpu_openmp` qu'il suffit d'exécuter sans paramètres, pour traiter par défault l'image dans le fichier `./install/ecureuil.png`.
Après exécution du code, le fichier qui contient le résultat est dans `./install/res_ecureuil.png`.
Afficher ces deux fichiers.

Question 1.

Comparer les temps d'exécution de la version séquentielle et de la version parallélisée avec OpenMP.

Les temps sont affichés séparément pour les parties lecture et écriture sur fichier et l'application de chaque filtre.

2 Version GPU (Cuda) du code

Dans le répertoire `./src/gpu/cuda`, un code partiellement écrit sera chargé de faire le même calcul sur GPU (type Nvidia) avec Cuda.
Ce qui est déjà écrit :

- la structure `cImage` (dans le fichier `src/common/cImage.h`) pour représenter les données d'une image pour le CPU.

  Ce fichier contient aussi les fonctions de lecture et d'écriture dans des fichiers.

- la structure `cImageGPU` (dans le fichier `src/gpu/cuda/cImageGPU.h`) pour représenter les données d'une image pour le GPU. Si `I` est de type `cImageGPU`

  - `I.width` est le nombre de pixels sur une ligne
  - `I.height` est le nombre de pixels sur une colonne
  - `I.d_coeff[0]` est un vecteur des composantes rouges, `I.d_coeff[1]` les composantes vertes et `I.d_coeff[2]` les composantes bleues de l'image couleur.
  - `I.d_coeff[0]` les composantes de gris pour une image grisée.

  (les vecteurs `I.d_coeff[k]` sont déjà sur GPU)

  Dans ce fichier sont déjà codés les appels de cudaMalloc et de cudaMemcpy pour réserver la mémoire sur le GPU et les transferts entre mémoires du GPU et du CPU.

- Le fichier `./build.sh` et les fichiers `CMakeLists.txt` qui compilent aussi la version cuda

- Les noyaux de calcul copyImageGPU (copie entre 2 structures dans le GPU) et smoothGPU (filtre régularisant) dans le fichier `src/gpu/cuda/process.cu`.

Question 2.

Écrire les noyaux `setGreyGPU` et `sobelGPU` en reprenant les formules dans les fonctions analogues pour le CPU dans `src/cpu/sequential/process.cxx`.

On pourra s'inspirer des autres noyaux déjà écrits dans le même fichier et d'autres vus au cours.