Projet GPGPU 2019
Stéganographie

Contexte

Le but de ce projet est de réaliser un programme de Stéganographie permettant de cacher et récupérer des données dans une image sans que cela ne soit visible à l'œil nu. La parallélisation sur GPU servira à accélérer le processus de traitement de l'image.

Description du protocole

Le protocole utilise la méthode LSB (Least Significant Bit) : c'est à dire que seul le dernier bit servant à coder une couleur pourra être utilisé pour cacher notre information. En effet, cela altère le moins possible l'image d'origine. La méthode LSB la plus simple vient cacher l'information dans les n premiers pixels de l'image et est donc sensible à des techniques de détection.

Ici, nous allons appliquer un filtre sur chaque pixel qui lui associera une valeur. Les pixels qui encoderont de l'information seront ainsi choisis en fonction de la valeur d'application du filtre. En effet, il est possible d'imaginer qu'en fonction du filtre utilisé, les pixels choisis pour cacher l'information rendront la détection plus difficile car l'image ne sera pas altérée de façon visible ou alors le filtre peut être vu comme une clé secrète.

Déterminer les pixels à modifier

Afin de déterminer quels sont les pixels à modifier, nous allons appliquer un filtre **sur les canaux rouge et vert** (le canal bleu servira à cacher l'information) permettant d'associer une valeur à chaque pixel de l'image (sauf les bords). Par exemple, si on applique le filtre suivant sur le pixel situé en (x, y) :

1	0	1
0	2	0
1	0	1

La valeur obtenue sera la somme des valeurs des canaux rouge et vert pour les pixels $(x-1, y-1)$, $(x+1, y-1)$, deux fois (x, y), $(x-1, y+1)$ et $(x+1, y+1)$ (les coefficients peuvent être plus élevés y compris négatifs). Ici pour un filtre de taille 3×3, la valeur pour le pixel à la position i, j peut être exprimée par

$$V_{i,j} = \sum_{m \in [i-1,i+1], n \in [j-1,j+1]} (r_{m,n} + g_{m,n}) * filtre_{m-i,n-j}$$

où r est la valeur du canal rouge, g la valeur du canal vert et $filtre$ le coefficient du filtre.

Vous pouvez vous référer au sujet du projet image pour l'application des filtres.

Cacher un caractère

Afin de cacher un caractère dans l'image (8 bits d'information), on cherche les 8 pixels pour lesquels la valeur associée V par l'application du filtre est la plus grande. On écrit un bit par pixel, en utilisant le bit de poids faible du canal bleu ; le *premier* bit est écrit dans le pixel à la valeur V_1 *la plus grande*, le *deuxième* bit est écrit dans le pixel à la valeur V_2 avec $V_1 \geq V_2$ et ainsi de suite.

En cas de pixels à égalité, on choisit en priorité celui qui apparaît le plus haut dans l'image, et s'ils sont sur la même ligne, le plus à gauche.

Lire un caractère

Pour lire un caractère, il suffit d'appliquer le même filtre à l'image, trier les pixels par leur valeur associée et lire les bits de poids faible du canal bleu de ces pixels.

Gestion des images

Afin de gérer les images dans votre programme, vous utiliserez le format PPM (Portable Pixel Map). Plus d'explications sont disponibles dans le sujet du projet image.

Questions

— Réaliser un programme séquentiel (CPU) permettant de cacher et lire un caractère dans une image. Vous utiliserez des filtres de taille 5×5.

— Écrire une version parallélisée avec CUDA de ce même programme. En effet, l'application du filtre peut tirer avantage des nombreux threads disponibles.

— Améliorer les performances du premier programme CUDA en prenant en compte les accès mémoires concurrentiels pour le traitement de pixels dans une même zone.

Astuce : la mémoire partagée.

— Tester les performances des deux versions du protocole pour des images de différentes tailles mais également pour des applications sur 1000 ou 10000 images.

Vous pouvez simuler 1000 applications avec un nombre moins important d'image.

Pour aller plus loin :

— Permettre au programme de cacher plusieurs caractères.

— Proposer une solution pour accélérer la recherche des n valeurs maximales de façon plus efficace en utilisant CUDA.