Rapport Projet CUDA

Laurien PITROU Anthony MARLIN Mattéo COULON

Introduction :
L'objectif de ce projet est d'écrire plusieurs algorithmes de filtrages et de les améliorer en les exécutant sur un accélérateur NVIDIA.

Présentation des algos :

- Sobel :
  Cet algorithme permet de détecter les contours de l'image grâce aux différences de valeurs liées aux nuances de gris.
- Boxblur :
  Cet algorithme permet de flouter l'image en faisant la moyenne de chaque pixel avec ses voisins.
- Embossing :
  Cet algorithme permet de mettre en évidence les différentes zones de profondeur de l'image en appliquant une matrice de convolution à chaque pixel.
- Gaussian blur :
  Cet algorithme permet de flouter l'image en appliquant une fonction Gaussienne, à la manière de l'embossing, une matrice de convolution est appliquée à chaque pixel afin d'en modifier la valeur selon ses voisins.

Optimisations :
Les programmes CUDA (.cu) ont été optimisés à l'aide de la répartition par bloc.
Tous les filtres ont leurs nombre de blocs et threads par block exprimés à l'aide d'un bloc 2D (un tableau 2x2 de valeurs) avec :

- √(MaxThreads Par Blocs)² pour le nombre de threads par blocs (le max en question vaut ici 1024)
- (((taille horizontale de l'image) + √(MaxThreads Par Blocs) - 1) / taille horizontal de l'image) x (((taille verticale de l'image) + √(MaxThreads Par Blocs) - 1) / taille verticale de l'image)

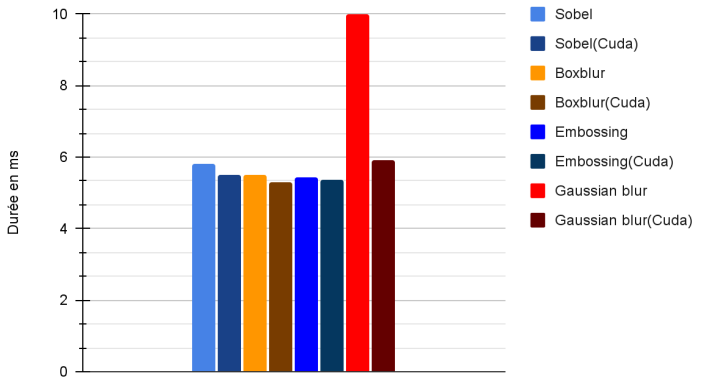
L'optimisation par stream a été réalisée sur le filtre embossing qui est l'un de nos filtres s'exécutant le plus rapidement pour n'importe quelle taille d'entrée (le deuxième plus rapide exactement).
Les résultats en matière de performances se sont révélés décevants par rapport à la version utilisant la répartition par blocs et même la version n'utilisant pas la carte NVIDIA.

Difficultés rencontrées:

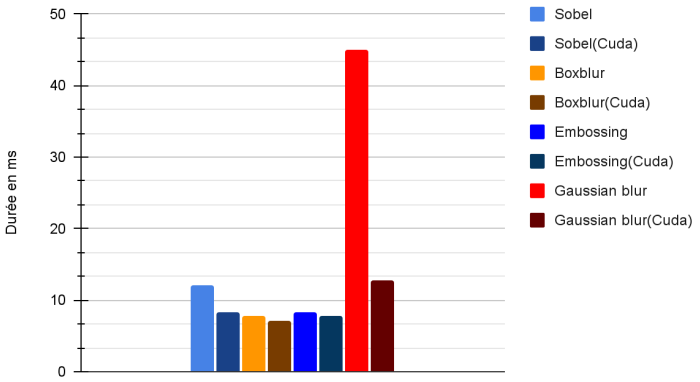
- Différence entre l'image d'origine et l'image avec le filtre appliqué peu visible sans zoomer sur l'image pour les algorithmes Boxblur et Gaussian blur si l'image est très grande.
- N'avoir qu'un seul canal pour les trois composantes rgb pour la version GPU au lieu d'en avoir 3 comme pour la version CPU qui nous a posé problème au début mais nous avons fini par trouver comment mettre les images en couleur après l'application des différents filtres.

Les graphes ci-dessous permettent de constater les différences de temps d'exécution entre la version CPU et les versions GPU (le filtre Gaussian-blur a un kernel size = 5 dans tous les graphes).

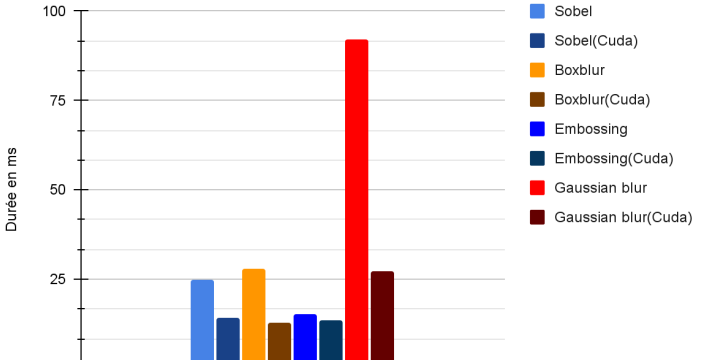
Temps d'exéction en ms des différents algos pour une image de taille 320x176



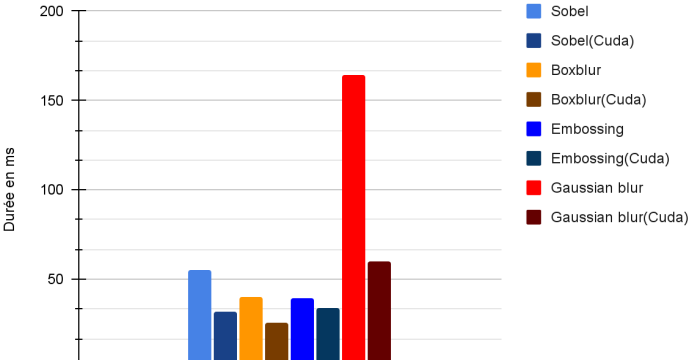
Temps d'exéction en ms des différents algos pour une image de taille 800x532



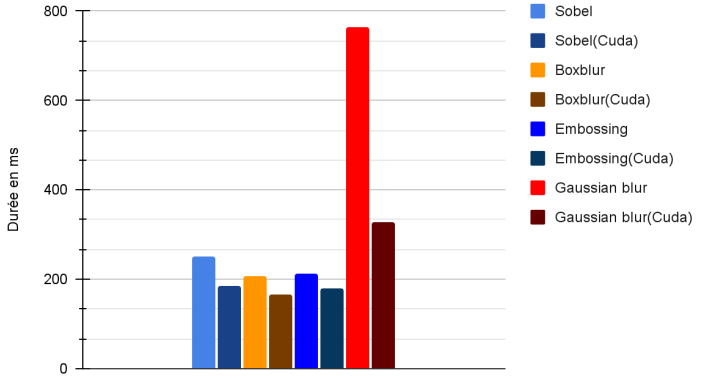
Temps d'exéction en ms des différents algos pour une image de taille 1025x1024



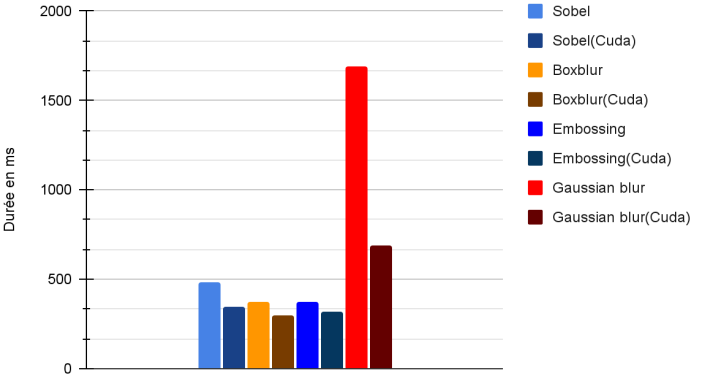
Temps d'exéction en ms des différents algos pour une image de taille 1920x1200



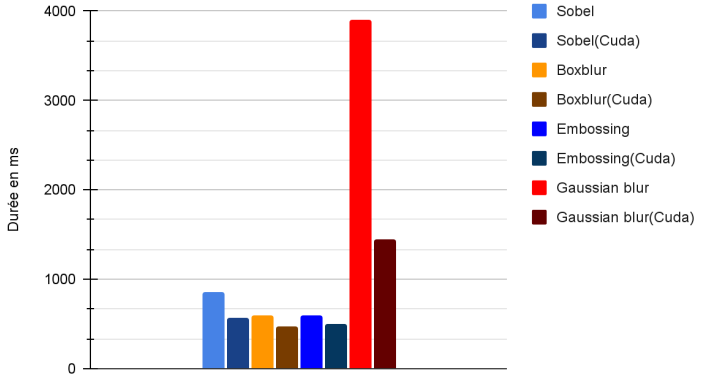
Temps d'exéction en ms des différents algos pour une image de taille 4000x3000



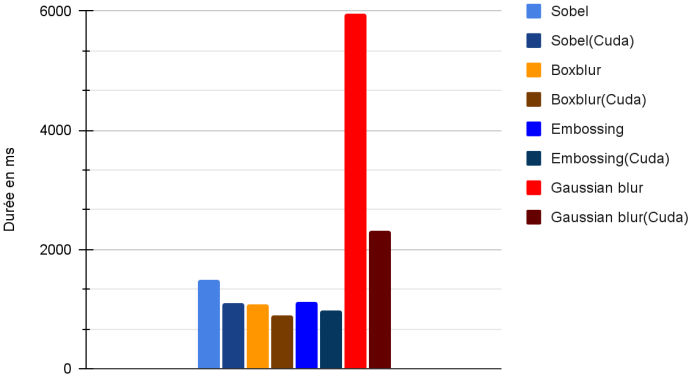
Temps d'exéction en ms des différents algos pour une image de taille 7000x4000

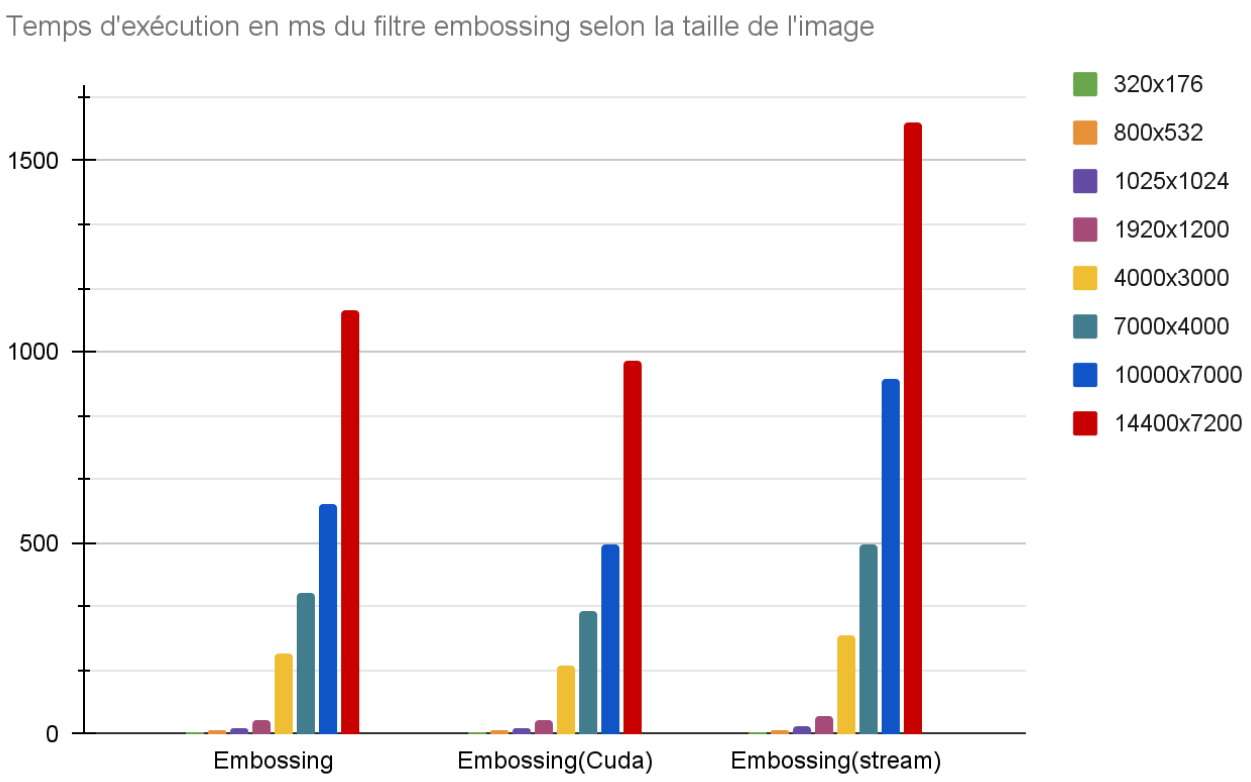


Temps d'exéction en ms des différents algos pour une image de taille 10000x7000



Temps d'exéction en ms des différents algos pour une image de taille 14400x7200



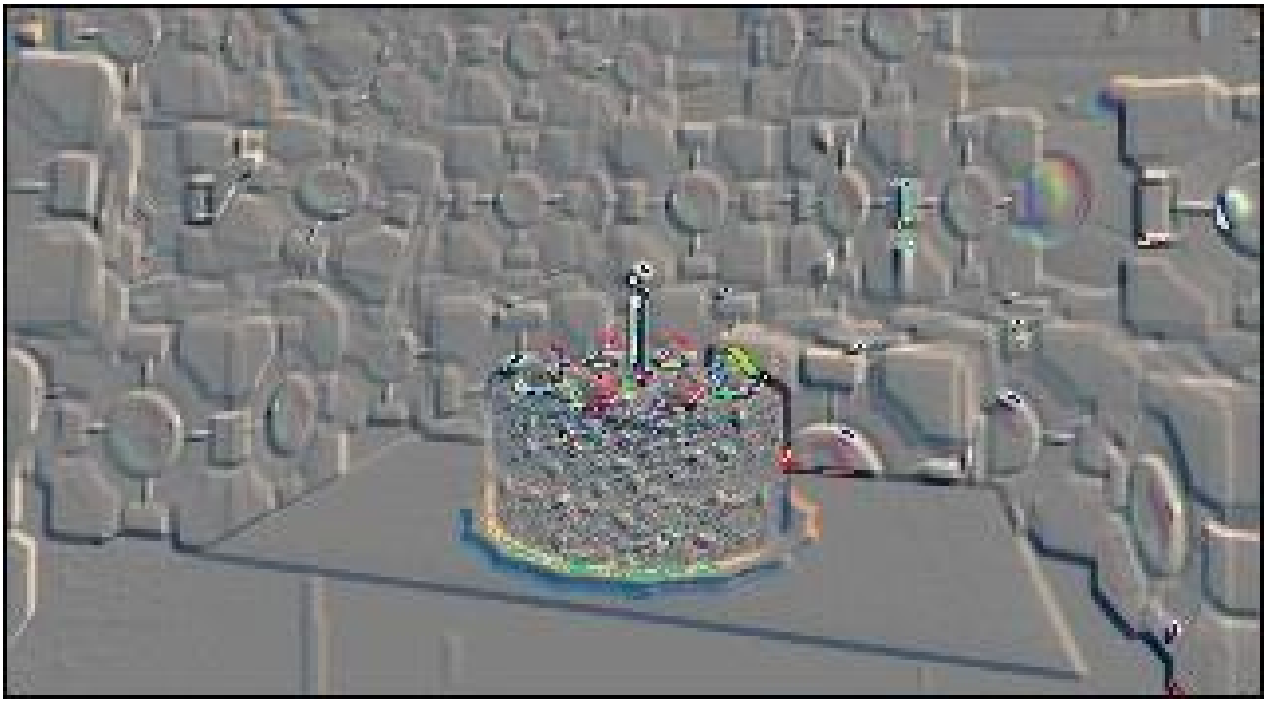


Voici le résultat des différents filtres sur l'image suivante:

Sobel:

BoxBlur:

Embossing:



Gaussian blur:

Discussions :
Une explication du faible niveau de performance de la version utilisant les stream pourrait être le fait que les stream fonctionne non pas sur de l'exécution parallèle mais concurrente.

Le but des streams n'est pas de faire les calculs rapidement mais de les faire en non bloquant, c'est-à-dire permettre au programme de ne pas avoir à attendre que l'accélérateur NVIDIA ait terminé ses calculs pour continuer l'exécution du programme.

Cela peut être utile par exemple dans le domaine du jeux vidéo où la carte graphique travaille en continu pour le rendu visuel du jeu. Si l'exécution était bloquante, le joueur ne pourrait pas interagir avec le jeu car la carte graphique n'aurait jamais fini son exécution.

Dans toutes les versions GPU de nos filtres, on utilise un objet CUDA (deviceProperties) pour obtenir le nombre maximum de threads par blocs pour la carte graphique et on l'utilise dans nos appels de fonctions. Le nombre de blocs et de threads par bloc est donc toujours dépendant de la carte graphique (rien n'est "hard-codé").

Le Gaussian-blur a été testé avec une taille de 5 pour le kernel size à chaque fois car si on avait mit un kernel size plus grand, l'image aurait été plus floue mais aurait prit également plus de temps (la version CPU avec un kernel size=15 met presque 45 secondes avant de terminer sur l'image 14400x7200), le risque aurait été une mauvaise visualisation des données sur le graphe, nous avons donc décidé de le mettre à 5 pour minimiser son résultat mais garder une visualisation du flou afin que les données restent visibles (on constate bien que même avec un kernel size bas, le Gaussian-blur a un temps d'exécution nettement supérieur aux autres filtres quelque soit la taille de l'image).