# Parallélisation sur GPU et Généralisation en 3D d'un Algorithme de Débruitage d'Images

Paul Godbert

Date : 25 avril 2024

**Résumé**

Ce rapport présente la parallélisation d'un algorithme de débruitage d'image ainsi que la généralisation de cet algorithme à un cube d'image en trois dimensions, en utilisant la librairie CUDA, développée par Nvidia. Le but de ce projet est d'améliorer les performances de l'algorithme originel et prenant pleinement avantage de la puissance offerte par le GPU. Pour cela nous verrons comment concevoir et implémenter des noyeaux CUDA adaptés à cet usage précis.

# Table des matières

# 1 Introduction

## 1.1 Contexte

Dans le domaine de l'imagerie numérique, l'optimisation des algorithmes de traitement d'image est cruciale pour répondre aux exigences croissantes en termes de performance et de qualité. Avec l'explosion des données et la nécessité de traitements en temps réel, la parallélisation des algorithmes sur des architectures matériels spécialisées comme les GPU est devenue une voie privilégiée pour obtenir des gains de performance significatifs. Notre projet se focalise sur la parallélisation d'un algorithme de débruitage d'image basé sur la Transformée Cosinus Discrète 8x8 (DCT8x8) en utilisant la librairie CUDA de NVIDIA.

## 1.2 Objectif

L'objectif principal de ce projet est de maximiser les performances de l'algorithme de débruitage d'image en tirant parti des capacités de calcul massivement parallèles des GPU. Nous visons à obtenir un gain de vitesse substantiel tout en préservant la qualité du débruitage. Par ailleurs, nous cherchons à étendre la portée de cet algorithme en le généralisant pour traiter des cubes d'images en trois dimensions, ouvrant ainsi de nouvelles perspectives pour des applications avancées en imagerie médicale et reconstruction 3D.

# 2 Revue de littérature

Depuis l'avènement et la démocratisation des GPU, la parallélisation des algorithmes a suscité un intérêt considérable dans la littérature scientifique. Un des domaines ayant été particulièrement touché par cette avancée est le domaine du traitement d'image. Que ce soit en imagerie médicale, en animation ou encore en jeu vidéo, le domaine de l'infographie a indéniablement bénéficié de ces concepts. C'est donc tout naturellement que le sujet du débruitage (et par extension de la compression) d'images a vu naître un bon nombre d'articles. Parmi ces articles, on peut en retrouver certains décrivant la technique que nous allons utiliser dans ce projet.

— En 2008, NVIDIA détaille dans une publication l'implémentation de la DCT8x8 avec des noyaux CUDA. Cet article [1] démontre les avantages substantiels de l'utilisation de CUDA pour accélérer les opéra-

tions de transformation d'image. Cet article nous aura été d'une aide précieuse dans la réalisation des noyaux CUDA nécessaires au débruitage.

— De plus, nous nous sommes également appuyé sur la littérature derrière le format JPEG pour comprendre les principes fondamentaux de la compression par DCT8x8 (procédé presque identique à notre débruitage).

# 3 Concepts théoriques

La Transformée Cosinus Discrète (DCT) est une fonction inversible définie formellement comme une fonction $F : \mathbb{R}^N \to \mathbb{R}^N$ ou, de manière équivalente, comme une matrice carrée $N \times N$. La définition formelle pour la DCT unidimensionnelle d'une séquence de longueur $N$ est donnée par la formule suivante :

$$\mathrm{DCT}(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) \cos\left[\frac{(2x+1)u\pi}{2N}\right]$$

où $f(x)$ est la valeur de l'échantillon à la position $x$, et $N$ est la taille de l'échantillon. La transformation inverse est définie comme :

$$f(x) = \sum_{u=0}^{N-1} \alpha(u)\mathrm{DCT}(u) \cos\left[\frac{(2x+1)u\pi}{2N}\right]$$

La DCT bidimensionnelle pour un échantillon de taille $N \times N$ est donc définie comme :

$$\mathrm{DCT}(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y) \cos\left[\frac{(2x+1)u\pi}{2N}\right] \cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

La transformation inverse de la DCT bidimensionnelle est :

$$f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} \alpha(u)\alpha(v)\mathrm{DCT}(u,v) \cos\left[\frac{(2x+1)u\pi}{2N}\right] \cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

La DCT bidimensionnelle bénéficie de la propriété de séparabilité, qui permet d'exprimer la transformation de manière plus simple :

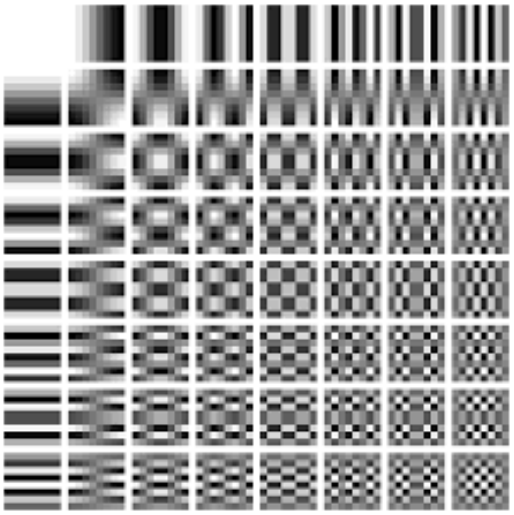


FIGURE 1 – blocs 8x8 en fonction des coefficients de la DCT8x8

$$\mathrm{DCT}(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \cos\left[\frac{(2x+1)u\pi}{2N}\right] \left\{ \sum_{y=0}^{N-1} f(x,y) \cos\left[\frac{(2y+1)v\pi}{2N}\right] \right\}$$

Cette propriété de séparabilité permet de calculer les fonctions de base de la transformée 2D en multipliant les fonctions de base 1D orientées verticalement avec leurs représentations horizontales. La visualisation de ces fonctions est illustrée ci dessous. Comme on peut le voir sur le graphique, les fonctions de base montrent une augmentation progressive de la fréquence dans les directions verticales et horizontales.

Pour effectuer efficacement la DCT de longueur $N$, les valeurs des cosinus sont généralement précalculées hors ligne. Une DCT 1D de taille $N$ nécessitera $N$ vecteurs de $N$ éléments pour stocker les valeurs de cosinus (matrice $A$). La transformation cosine 1D peut alors être représentée comme une séquence de produits scalaires entre l'échantillon de signal (vecteur $x$) et les vecteurs de valeurs de cosinus (matrice $A$), résultant en un vecteur transformé $A \times x^T$.

L'approche bidimensionnelle applique la DCT à un échantillon d'entrée $X$ en appliquant successivement la DCT aux lignes et aux colonnes du signal d'entrée, en utilisant la propriété de séparabilité de la transformée. En notation matricielle, cela peut être exprimé par la formule suivante :

$$C(u,v) = A^T X A$$

où $A$ est la matrice $N \times N$ des valeurs pré calculés (pour une question de performance) des cosinus et $X$ est le vecteur passé en entrée.

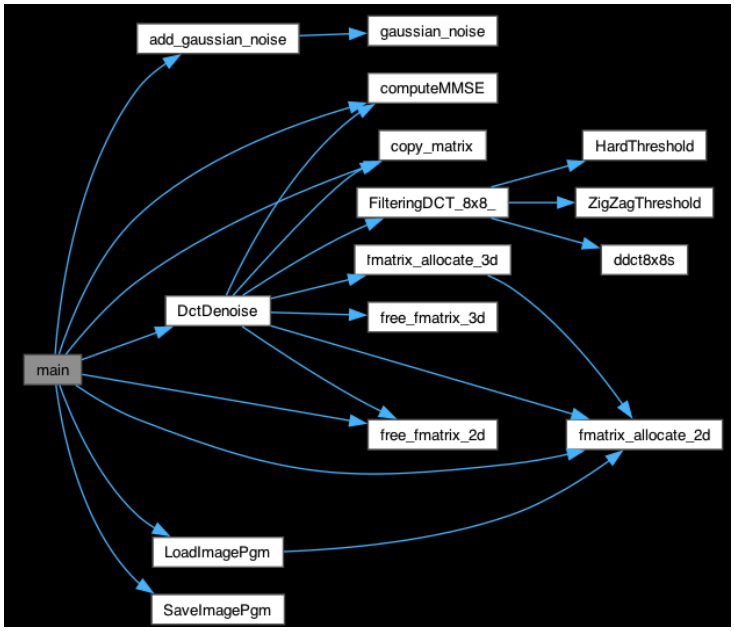
# 4 Méthodologie

## 4.1 Description de l'algorithme original

L'algorithme original de débruitage DCT 8x8 est conçu pour filtrer les images en utilisant la Transformée Cosinus Discrète (DCT) appliquée sur des fenêtres de taille 8x8 de l'image. Initialement, pour chaque pixel de l'image d'entrée, une fenêtre 8x8 centrée autour du pixel est extraite. Des mécanismes de gestion des bords sont mis en place pour assurer la périodicité de l'image lors de la sélection des pixels. Une fois la fenêtre extraite,on apllique la DCT 8x8, transformant les données spatiales de la fenêtre en coefficients fréquentiels. Ces coefficients subissent ensuite un éventuel seuillage dur, où ceux dont la valeur absolue est inférieure à un seuil prédéfini (sigma) sont éliminés. Cette étape de seuillage est la véritable raison derrière le débruitage, les coéfficients de plus haute fréquence étant généralement les plus sensible à disparaitre à cette étape. Après le seuillage, une transformée DCT inverse 8x8 est appliquée pour revenir à l'espace spatial. Les coefficients DCT inversés sont ensuite réintégrés dans une image 3D pour la reconstruction. Enfin, pour améliorer la qualité de l'image, un processus de moyennage est effectué. Il calcule la moyenne des valeurs positives pour chaque pixel dans l'image 3D et met à jour l'image d'entrée avec ces nouvelles valeurs moyennées, fournissant ainsi une image débruitée.

## 4.2 Étude de la codebase

Pour paralléliser cet algorithme sur GPU, il nous fallut d'abord comprendre sa structure. Pour cela nous avons effectué une analyse détaillée du programme en générant une documentation complète grâce à l'outil Doxygen. Cet documentation nous a par la suite permis d'identifier les parties parallélisables ainsi que de mieux comprendre la pipeline de débruitage.




Sur ce diagramme d'appel on peut clairement voir que le coeur du projet réside dans la fonction DctDenoise qui appelle la fonction de débruitage FilteringDCT_8x8_.

En poussant un peu plus l'inspection, on se rend vite compte que la plus-part des fonctions responsables du traitement de l'image (c'est à quasiment toutes les fonctions, sans compter les fonctions utilitaires) passent au travers de doubles voir triples boucles imbriquées. Ces boucles sont donc à coup sûr la source de la grande complexité de ce programme (ces fonctions ayant des complexités asymptotiques en $O(n^2)$ ou bien en $O(n^3)$). Notre but sera donc de nous servir au maximum des techniques de parallélisation sur GPU pour dé-construire ces boucles et essayer de simplifier au maximum le programme.

Au cours de nos recherches (et toujours guidés par le papier de Nvidia), nous avons pu déterminer qu'il nous faudrait écrire des noyaux CUDA permettant de réaliser la DCT8x8 ainsi que sa transformation inverse mais également un noyau de quantification (noyau qui sera chargé du véritable débruitage).

# 5 Implémentation

## 5.1 Parallélisation

notre implémentation de l'algorithme de débruitage DCT 8x8 parallélisé pour GPU utilise plusieurs techniques clés pour optimiser les performances et l'efficacité du traitement :

— Premièrement, nous avons fait le choix de n'utiliser que des tableaux de floats dans notre implémentation. Bien que CUDA offre la possibilité d'utiliser des textures, cela rajouterais une surcouche à notre programme qui n'en deviendrais que plus complexe à comprendre. De plus, le programme ne traitant que des images en niveaux de gris, l'utilité de telles structures est à débattre.

— Ensuite, la parallélisation de la Transformée Cosinus Discrète (DCT) 8x8 et de son inverse (IDCT) est réalisée en divisant l'image en blocs de 8x8 pixels, avec un thread par pixel (soit 64 threads au total). Cette division permet d'exécuter simultanément le calcul de la DCT pour chaque bloc (et par extension, chaque pixel), maximisant ainsi l'utilisation des ressources du GPU. La DCT8x8 en elle même est implémentée en utilisant sa forme matricielle, comme vu plus haut.

— En ce qui concerne l'implémentation à proprement parler des noyaux de DCT8x8 et IDCT8x8, nous avons suivi le papier de Nvidia en implémentant le premier noyau présenté (avec cependant quelques modifications pour que ce dernier colle a la structure de notre programme).

— Nous utilisons deux blocks de 8x8 floats stockés dans la mémoire partagée pour stocker temporairement les données en entrée et en sortie lors du calcul de la DCT. Cela permet de réduire les accès à la mémoire globale et ainsi améliorer les performances tout en simplifiant le calculs. le premier bloc permet de stocker la multiplication $A^T X$ tandis que le deuxième, utilisé juste après, stocke le résultat de l'opération $(A^T X)A$.

— Un noyau permettant de réaliser un décalage toroïdal sur l'entièreté de l'image en simultané permet de simplifier le processus de moyennage de l'image et d'y sauver du temps de calcul (même si le moyennage final des images décalées est quand même réaliser dans une triple boucle imbriquée). Cette approche de décalage assure que les opérations ef-

fectuées sur les pixels à proximité des bords de l'image prennent en compte les pixels opposés de l'image, réduisant ainsi les artefacts de bord.

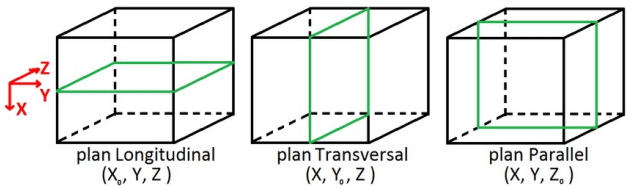
— certaines petites optimisations ont aussi été rajoutées comme par exemple l'optimisation du code avec le déroulement des boucles (#pragma unroll) permet d'accélérer l'exécution en réduisant le surcoût lié à la gestion de la boucle (on parle ici d'une optimisation au niveau de code machine, après compilation) cette petite optimisation particulièrement importante dans les boucles intensives de la DCT et de l'IDCT.

— Enfin, une fois le débruitage effectué, un mécanisme de moyennage est appliqué pour améliorer la qualité de l'image débruitée. Ce processus calcule la moyenne des valeurs positives des coefficients DCT pour chaque pixel, fournissant ainsi une image finale plus lisse.

En combinant ces techniques de parallélisation et d'optimisation du code, notre algorithme DCT 8x8 offre une solution robuste et efficace pour le débruitage d'images sur les architectures GPU.

## 5.2 Généralisation en 3D

Le dernier objectif de ce projet était la généralisation de cet algorithme à des cubes d'images en 3 dimensions. L'approche visée était d'allouer une array de float en 3 dimensions (float***), d'y stocker les valeurs de gris de chaque pixel puis de filtrer l'image tranche pas tranche le long de l'axe X, puis de l'axe Y et enfin le long de l'axe Z.




Pour cela, une fonction permettant de récupérer une "tranche" de la matrice 3D à un certain indice selon une dimention donnée a été implémentée, ainsi qu'un mode d'exécution du programme pouvant être lancer avec le flag "-3d". Cependant, ce mode n'est pas encore fonctionnel dans l'état actuel du projet.

## 5.3 Plateforme et outils

L'implémentation de ce logiciel a été réalisée sur une plateforme équipée d'un GPU NVIDIA RTX 3080, compatible avec CUDA. Le développement du code a été effectué en utilisant le langage CUDA C/C++ version 12 sous UNIX.

## 5.4 Utilisation

Un makefilea été élaboré pour automatiser le processus de compilation et d'exécution. Le projet peut donc être compilé en naviguant dans le dossier build puis en exécutant la commande make.

le programme est exécutable en 2 modes différents (débruitage 2D et 3D) et prend en charge des images différentes. il est possible de spécifier quelle image débruiter en les passant en paramètre du programme.

# 6 Résultats

Suite à la parallélisation de l'algorithme et son exécution sur GPU, une amélioration significative des performances a été observée par rapport à la version exécutée sur CPU. Le temps d'exécution passe de 230ms en moyenne lors de l'exécution du programme principal sur CPU à 90ms avec la version actuelle. Même si certaines optimisations pourraient certes être faites afin d'améliorer améliorer encore les performances, l'état actuel du logiciel apporte une preuve solide de l'intérêt de la méthode.

Le résultat dans l'état actuel de notre algorithme de débruitage exécuté sur l'image de référence "Lena" et le suivant :

FIGURE 2 – image dégradée avec un bruit gaussien blanc

(a) CPU

(b) GPU (CUDA)

FIGURE 3 – comparaison entre les versions CPU et GPU

# 7 Discussion

Les résultats obtenus confirment l'efficacité et la pertinence de la parallélisation sur GPU pour accélérer les algorithmes de traitement d'image. L'approche adoptée dans ce projet a non seulement optimisé les performances, mais a également permis de bâtir une base solide pour de futures améliorations. Cependant, même si le bilan semble positif, certains problèmes subsistent.

En effet, malgré une quantité non négligeable d'effort, la réalisation du calcul de la DCT8x8 fut un échec. Bien que les noyaux soient fonctionnels et mettent correctement en oeuvre la parallélisation, une erreur dans le calcul matriciel de la DCT empêche d'obtenir un débruitage correct.

De plus, bien que les bases soit implémentées, la généralisation 3D ne fonctionne pas à l'heure actuelle.

# 8 Conclusion

Ce projet a abouti à une amélioration significative des performances d'un algorithme de débruitage d'image en exploitant efficacement la puissance de calcul des GPU grâce à la librairie CUDA. Bien que certains des ses objectif n'ai pas été atteints, les résultats obtenus ouvrent de nouvelles perspectives pour des applications avancées en imagerie numérique 3D et permettent de confirmer l'importance de la parallélisation sur GPU dans le domaine du traitement d'image.

Retrouvez le répertoire github du projet ici

# Références

[1] NVIDIA Corporation (2008). "Discrete Cosine Transform for 8x8 Blocks with CUDA". Anton Obukhov et Alexander Kharlamov. `https://developer.download.nvidia.com/assets/cuda/files/dct8x8.pdf`