Rapport de Projet

JOBE Ophélie
ADAM Baptiste

Sommaire

JOBE Ophélie
ADAM Baptiste

Introduction

Le projet final de cet unité a pour but de restituer toutes nos connaissances vu jusque là dans un exemple de traitement d'image avec filtre Sobel et Médian via une caméra fourni.

Il nous a été fournit une version de base du programme, que nous appellerons V1 par la suite, qui utilise les fonctions d'OpenCV afin d'optimiser au mieux les fonctions de filtre Sobel et Médian pour avoir un rendu d'image fluide. L'objectif principal de ce projet est de remplacer ces deux fonctions par celle que nous créerons nous même et de les optimiser le plus possible dans le but d'égaler ou éventuellement dépasser les versions d'OpenCV. Ce projet se déroulera donc de la manière suivante :

- Remplacement des fonctions Médian et Sobel par nos propre fonctions non optimisé et dites "naïves"
- Optimisation du filtre Médian grâce à la méthode des histogrammes
- Optimisation du filtre Sobel
- Optimisation par méthodes dites "classique" (réagencement des lignes, déroulage de boucle, etc.)

JOBE Ophélie
ADAM Baptiste

V2 - Utilisation de nos fonctions “naïves”

Nous avons commencé par remplacer les deux fonctions d’OpenCV par deux versions “naïves”.

Le filtre Médian collecte les valeurs du voisinage et les stocke dans un tableau qui est ensuite trié et dont la valeur retournée est donc la valeur médiane du tableau.

Cette version est très naïve et très peu pratique puisque qu’il n’est même pas possible de changer le degré de voisinage que l’on regarde via une variable k. en effet, la taille du tableau et la façon d’aller chercher les valeurs du voisinage ne sont pas modulable.

Le filtre Sobel est un filtre sobel comme il nous a été présenté dans les TP précédents. Il fait le gradient horizontal et vertical pour finalement retourner la racine carrée des carrés de ces deux gradients.

Nous avons néanmoins directement “déroulé” la boucle de parcours du voisinage pour ne prendre que les valeurs qui nous intéresse (celles qui ne sont pas multipliées par 0).

Comparaison avec la version précédente

	temps d'exécution du filtre Médian (avec k=3, v2 ne pouvant pas se moduler)
V1	2 042 microsecondes
V2	55 211 microsecondes

Cette version est 27 fois plus lentes que la fonction d’OpenCV. C’est “normal” puisqu’elle n’est absolument pas optimisée, que ce soit en ayant appliqué les méthodes d’optimisation classique ou même algorithmiquement (nous rappelons que ce méthode trie le tableau, et les tris, c’est long)

JOBE Ophélie
ADAM Baptiste

V3 - Filtre Median : Méthode des histogramme

Méthode des histogrammes

Pour éviter de faire un tri de tableau qui se fait en temps n*log(n), la méthode des histogrammes a été développée. Le principe est le suivant :

- Un tableau est créé qui possède autant de cases que de valeur possible pour notre image (les indices des cases correspondent donc aux valeurs possibles) et la valeur dans ce tableau correspond au nombre d'occurrence des valeurs dans l'image.

- Nous parcourons le voisinage et incrémentons chaque case du tableau pour chaque occurrence des valeurs de l'image.

- Nous cherchons la médiane, c'est à dire, la valeur centrale avec autant de valeurs avant et après elle. Il va donc falloir compter le nombre d'occurrence en partant du début du tableau jusqu'à arriver à un somme plus grande ou égale à la moitiée du nombre totale de valeurs regardées. En l'occurrence, cette valeur seuil est égale à $(k^2-1)/2$.

Il n'y a donc pas eu besoin de trier quoi que ce soit puisque les valeurs sont naturellement triée dans le tableau.

Comparaison avec la version précédente

	temps d'exécution du filtre Médian (avec k=3, v2 ne pouvant pas se moduler)
V2	55 211 microsecondes
V3	88 235 microsecondes

Nous constatons que le temps d'exécution est plus long avec la nouvelle méthode. Néanmoins, cette méthode apporte un modularité indispensable car regarder un voisinage a k=3 (voisinage direct) est finalement peu intéressant.

JOBE Ophélie
ADAM Baptiste

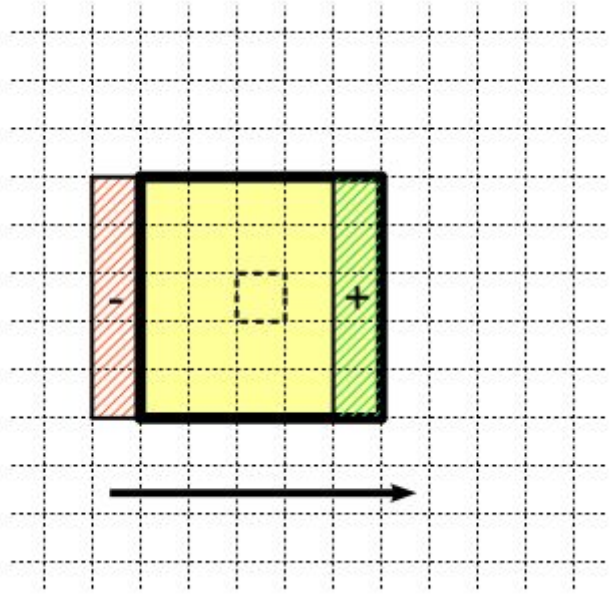
V4 - Optimisation algorithmique

Amélioration de la Méthode des histogrammes

Actuellement, on re-construit l'histogramme de zéro pour chaque pixel de l'image.

Une façon d'optimiser serait de ne pas reconstruire l'histogramme a chaque fois. On remarque que la majorité des voisinages est en commun entre deux pixels adjacents (en jaune sur l'image). Les deux seules différences sont :

- la première colonne que nous ne voulons plus (en rouge)
- la dernière colonne que nous voulons (en vert)



Cette opération est répétée pour toute la ligne. Lorsque l'on change de ligne, on re-construit l'histogramme de zéro. et on recommence cette opération.

Amélioration du filtre Sobel

En ce qui concerne le Sobel, nous y avons aussi apporté une amélioration en enlevant la racine carré qui prend du temps à exécuter puisqu'il faut, à chaque fois qu'elle est appelé, aller chercher l'information de cette opération dans un autre fichier, ce dernier ayant une succession d'opérations à faire puisque la racine carré n'est pas une opération basique.

Nous avons donc remplacé cette racine carré par une somme normalisé : la somme des valeurs absolues divisées par 2.

JOBE Ophélie
ADAM Baptiste

Comparaison avec la version précédente

Filtre Médian

	temps d'exécution du filtre Médian	
	k=3	k=9
V3	97 611 microsecondes	186 196 microsecondes
V4	87 430 microsecondes	103 123 microsecondes

Pour garder le même ordre de grandeur que précédemment, pour k=3, le temps d'exécution est 10% plus rapide. Tandis que dans un cas plus pratique, pour k=9 par exemple, le temps d'exécution est 45% plus rapide.

Lorsque k augmente linéairement, le nombre de valeur à regarder dans le voisinage augmente du carré. De ce fait, l'optimisation faite est d'autant plus importante que k est grand.

Filtre Sobel

	Temps d'exécution du filtre Sobel
V2/V3	14 145 microsecondes
V4	11 806 microsecondes

Comparé au filtre Sobel initialement implémenté, le fait d'approximer la racine carrée par un calcul plus simple possible avec les opérations basique nous donne un gain tout à fait notable. Le temps d'exécution est 18% plus rapide que notre version précédente.

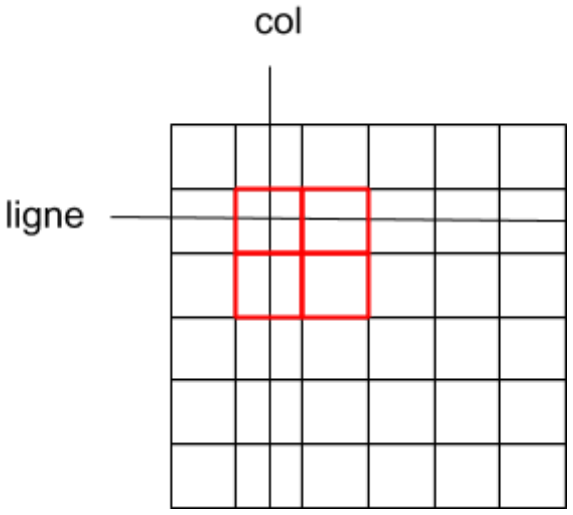
JOBE Ophélie
ADAM Baptiste

V5 - Optimisations “classiques”

Déroulage de boucle

L’une des méthodes “classique” d’optimisation vu en cours est celle du déroulage de boucle. Cela consiste à paralléliser les opérations contenues dans une boucle, quand il n'y a pas de dépendance entre les itérations. Il faut donc travailler sur plusieurs itérations en même temps en dupliquant le traitement de la boucle un certain nombre de fois selon l’amélioration voulu.

Pour le filtre Médian, nous avons “parallélisé” le traitement de deux lignes à la fois. Pour le filtre Sobel, nous avons “parallélisé” le traitement de 4 cases en carré. La boucle des lignes et la boucle des colonnes sont donc incrémentée de 2 en 2.



Réagencement des lignes

Une autre méthode dite “classique” est le réagencement des lignes. Il a pour but d'éviter les aléa de dépendances entre des lignes successives. Nous avons réagencés les lignes pour que chaque ligne soit toujours indépendante avec au moins les lignes directement avant et après.

Comparaison avec la version précédente

Filtre Médian

	temps d'exécution du filtre Médian	
	k=3	k=9
V4	87 430 microsecondes	103 123 microsecondes
V5	87 085 microsecondes	100 128 microsecondes

Encore une fois pour garder le même ordre de grandeur que dans les autres parties, lorsque k=3, le gain est de 0.4%. Pour k=9, le gain est de 2.9%. Dans les deux cas, le gain est assez minime, mais encore une fois, plus k augmente, plus l'optimisation faite est importante.

Filtre Sobel

	Temps d'exécution du filtre Sobel
V4	11 806 microsecondes
V5	11 018 microsecondes

En ce qui concerne le filtre Sobel, le gain est de 6.7%.

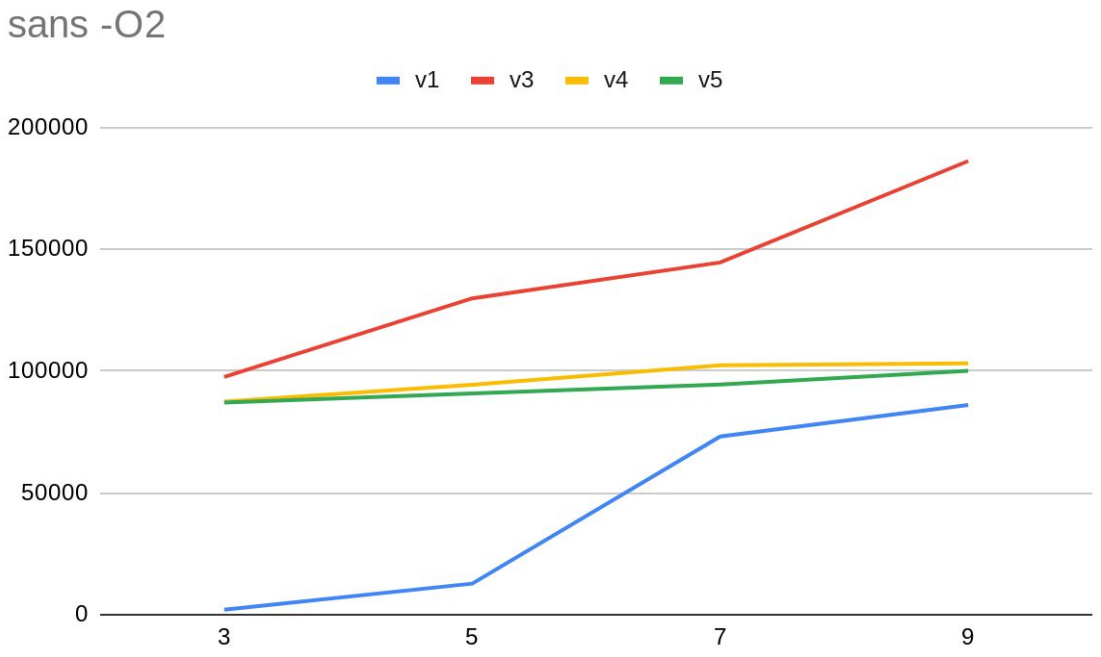
JOBE Ophélie
ADAM Baptiste

Comparaisons OpenCV et Produit final

Sans le paramètre -O2 de gcc

Voici le tableau récapitulatif ainsi que le graphique associé des temps d'exécutions en microsecondes du filtre Médian de toutes les versions avec un k qui varie entre 3 et 9.

k=	3	5	7	9
V1	2042	12749	73160	86113
V3	97611	129839	144583	186196
V4	87430	94327	102328	103123
V5	87085	90805	94416	100128



Nous voyons bien que malgré tous nos efforts, OpenCV (en bleu) nous reste bien supérieur. Malgré tout, nous remarquons aussi que nos optimisations n'ont pas été vaine puisque nous somme de plus en plus proche du temps d'exécution d'OpenCV. au début (en rouge), nous étions 116% plus lent alors que notre dernière version (en vert) ne l'est que de 16%, pour k=9 en tout cas.

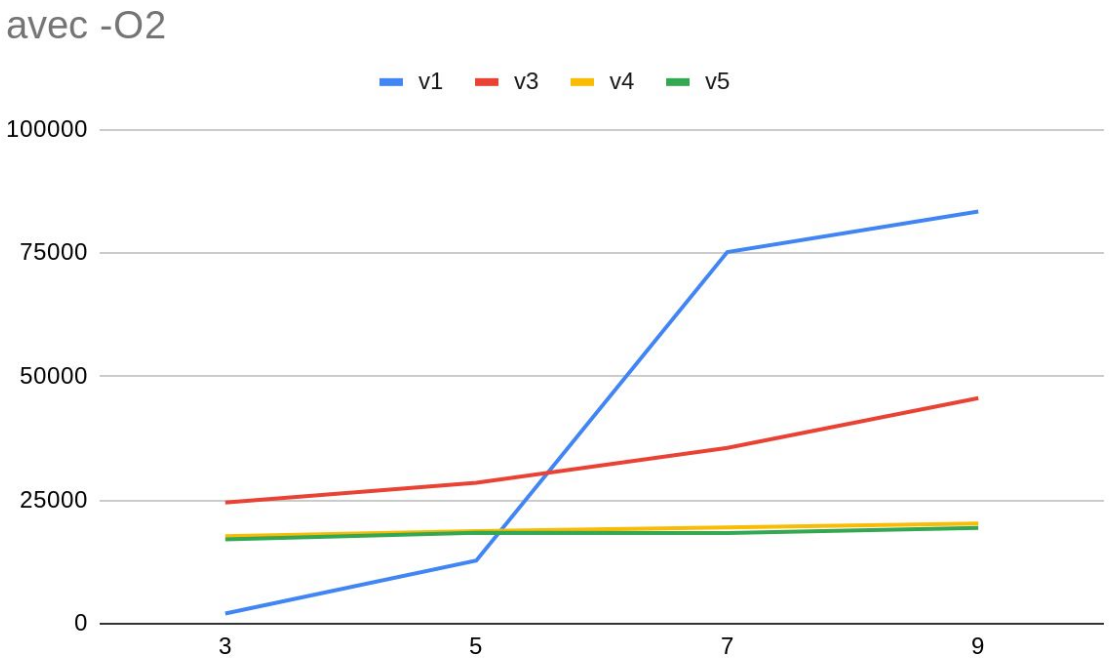
Avec le paramètre -O2 de gcc

Ne voyant plus quelles optimisations nous pouvions ajouter, nous nous somme penché sur les options de compilations de gcc. en ajoutant le paramètre -O2, nous avons remarqué plusieurs choses :

- les temps d'OpenCV sont similaire, impliquant que leurs fonctions sont déjà optimisées au maximum.
- nos temps sont nettement amélioré, rivalisant, voir dépassant, les temps d'OpenCV.

Voici donc le tableau récapitulatif ainsi que le graphique associé des temps d'exécutions en microsecondes du filtre Médian de toutes les versions avec un k qui varie entre 3 et 9 compilées avec le paramètre -O2.

k=	3	5	7	9
V1	2056	12772	75228	83421
V3	24537	28514	35582	45644
V4	17709	18765	19470	20277
V5	17062	18412	18333	19365



Cette fois ci, nous voyons qu nos versions dépassent OpenCV pour un k supérieur ou égal à 7, ce qui correspond à des utilisations pratique du filtre. Pour k=9, notre version finale (en vert) est 77% plus rapide que la version d'OpenCV (en bleu).

Tests sur la carte

Nous avons aussi fait des tests sur la carte mis à notre disposition. Nous avons testé la V1 du programme (avec les fonctions d'OpenCV) ainsi que notre V5, notre version la plus aboutie avec et sans le paramètre -O2 de gcc.

Filtre Médian (k=9)

	V1	V5
sans -O2	256 073	1 065 429
avec -O2	254 917	201 690

Nous remarquons tout d'abord que les tendances observées précédemment se confirment :

- les temps des fonctions d'OpenCV sont les mêmes avec ou sans le paramètre -O2.
- OpenCV est bien plus rapide, environ 75%, que notre version sans le paramètre -O2.
- Notre version est plus rapide, environ 20%, que OpenCV avec le paramètre -O2.

Nous remarquons aussi que les temps d'exécutions ont été décuplés par rapports aux tests effectués sur le PC :

- pour la V1 (OpenCV), les temps sont environ 200% plus long.
- pour la V5, les temps sont environ 950% plus long.

Concrètement, les fonctions d'OpenCV sont moins sujettes aux fluctuations de la puissance de la machine sur laquelle elles tournent mais fluctuent plus violemment avec l'augmentation de la taille du voisinage à regarder (le k) contrairement à notre version qui a un comportement inverse. OpenCV augmente ses temps d'exécutions de 3957% lorsque l'on passe de k=3 à k=9 alors que notre version n'augmente ses temps que de 13.5%.

Filtre Sobel

	V1	V5
sans -O2	40 202	180 112
avec -O2	36 447	13 382

Les mêmes observations peuvent être faites pour le filtre Sobel.

Conclusion

Nous avons donc vu durant ce projet comment optimiser du code en re-définissant des fonctions déjà existantes avec les méthodes d'optimisation vu en cours pour que cela fonctionne plus rapidement sur notre ordinateur mais surtout sur les cartes mises à disposition qui ont la puissance d'un smartphone.

Le premier changement effectué par rapport à la version donné utilisant les formules d'OpenCV fut de créer des fonctions dites "naives" qui feraient l'équivalent des fonctions d'OpenCV mais sans prendre en compte le temps d'exécution qui seraient forcément plus lent vu le temps d'exécution.

Notre première amélioration s'est porté sur la re-définition de la méthode de median afin que celle-ci soit moins coûteuse en données pour le programme grâce à un stockage de ces dernières dans des histogrammes se triant au fur et à mesure

Lors de notre deuxième amélioration nous avons réorganisé le programme pour faire en sorte que le surplus de données enregistré ne soit plus là, notamment dans la médiane qui stocker des données qu'elle pouvait déjà avoir en réserve lors de l'itération précédente de boucle. Ainsi que la complexité de certain calculs comme celui de la racine carré du sobel qu'il a fallu simplifier.

Et enfin, durant notre dernière optimisation via des méthodes "classiques" nous avons vu que le déroulage de boucle et le réagencement de lignes, même s'il n'est pas forcément visible sur notre ordinateur, font une amélioration visible sur la carte ayant une puissance de smartphone.

De ce TP, nous retenons que l'ordre des instructions dans un programme ainsi que la manière de penser l'algorithme général d'une fonction d'un programme peut impacter sur son temps d'exécution, encore plus quand on augmente le voisinage dans lequel on regarde nos données dans le cas de notre projet