IF112 - Projet d'Informatique

2023 - 2024

Yannick Bornat - yannick.bornat@enseirb-matmeca.fr
Guillaume Bourmaud - guillaume.bourmaud@enseirb-matmeca.fr
Clémence Gillet - clemence.gillet@enseirb-matmeca.fr
Rémi Giraud - remi.giraud@enseirb-matmeca.fr
Liam Moroy - liam.moroy@onera.fr

Projet : Manipulation d'images en ligne de commande

1 Introduction

Le projet de cette année concerne la mise au point d'une bibliothèque de programmes pour manipuler les images depuis la ligne de commande. Cette bibliothèque doit permettre l'application de filtres simples sur des images. Techniquement, une telle bibliothèque existe déjà, il s'agit de ImageMagick (`https://imagemagick.org/index.php`), mais il faut bien un prétexte.

Le résultat final est présenté comme une bibliothèque mais il n'y a qu'un seul programme à écrire. Ce dernier se comportera différemment selon les arguments qui lui seront donnés.

1.1 Évaluation

L'évaluation finale aura trois composantes :
- Du contrôle continu
- La qualité technique de votre projet
- La qualité de votre rapport

Le rapport sera à déposer sur git dans les jours qui suivent la dernière séance de projet. Pensez à anticiper son écriture si vous en avez l'opportunité. Votre rapport doit expliquer l'ensemble des démarches mises en œuvre pour le projet. Il doit au moins contenir (liste non exhaustive) :
- Vos choix d'implémentation
- Les algorithmes (principal, ainsi que pour les sous-fonctions les plus complexes ou intéressantes)
- Représentation des données (si nécessaire)
- Technique de validation des fonctions
  - À ce propos, il existe une grosse différence entre une fonction qui a pu produire un bon résultat une ou deux fois, et une fonction qui produit un bon résultat tout le temps.
- Analyse des performances
  - Estimation de la complexité
  - Explicitation des étapes les plus lourdes/limitantes
  - Discours sur le caractère optimal et/ou sur les pistes d'amélioration
- Propositions d'amélioration de votre travail (si le temps vous l'avait permis)

1.2 Consignes générales

Nous sommes dans l'esprit d'un projet, vous avez donc toute liberté quant à la méthode à utiliser pour produire le résultat attendu, moyennant les restrictions suivantes :
- Le résultat que vous présenterez doit être le fruit d'un travail personnel (à toutes fins utiles, précisons que l'intérêt du projet n'est pas tant le résultat final que les compétences que vous aurez mises en œuvre pour y parvenir).
- Gérez votre espace disque : veillez à ne pas garder trop de versions haute résolution de vos images de test.

Quelques conseils :
- Aidez-vous du débogueur *GDB* et de l'outil d'analyse mémoire *Valgrind* lors du développement.
- Documentez (commentaires, fichier readme, ...) l'utilisation de votre programme.
- Placez vos fonctions et organisez vos fichiers de manière pertinente (.c, .h correspondants).
- Testez votre programme, pour vous assurer qu'il soit robuste (mauvaises entrées, tailles d'images non équivalentes, etc).
- Le résultat demandé est attendu en fin de projet. Il vous appartient de fixer des objectifs intermédiaires, voire indépendants pour travailler efficacement en binôme et pour aider le débogage pendant le développement.
- Vos encadrants AUSSI, savent utiliser une connexion internet et un moteur de recherche...

2 Programme global

Un gros travail consistera à rendre votre programme, capable d'interpréter les différents arguments qui lui seront donnés en ligne de commande, et robuste aux erreurs. Comme dans de nombreux programme, la paramétrisation de votre commande se fera sous la forme de couple d'arguments sous la forme `-<type> <arg>`. Par exemple, pour donner le nom de l'image d'entrée on peut choisir l'option `-i <name_input>`, pour l'image de sortie `-o <name_output>`, etc. Une ligne de commande d'appel de votre programme pourra donc ressembler à :

```
./prog -i input.ppm -t blur -s 3 -o output.ppm
```

Pour rappel, l'ensemble des arguments reçus en ligne de commande est accessible par l'intermédiaire du tableau `argv[]`.

Pourquoi un tel fonctionnement ? On pourrait tout aussi bien déterminer la nature de l'argument donné selon sa position. Cependant, selon les méthodes appliquées, vous pourrez avoir besoin de certains arguments supplémentaires. Par exemple pour une conversion en niveaux de gris (méthode `gray`), on n'a pas besoin de spécifier d'autres arguments que l'image d'entrée et éventuellement de sortie, mais pour un floutage (méthode `blur`), on peut spécifier une taille de filtre avec l'option `-s`. La forme `-<type> <arg>` permet également de prévoir des valeurs par défaut. Par exemple si il n'est pas spécifié, le nom de l'image de sortie sera *output.ppm*. Enfin, notons que pour que le programme s'exécute, au moins deux arguments sont nécessaires : `-t <type>` et `-i <name_input>`, le type de filtrage à effectuer et le nom de l'image à traiter.

La seule contrainte est le fonctionnement par balise `-<type> <arg>` et le fait que l'ordre n'ait pas d'impact. La commande suivante donnera donc le même résultat que la commande précédente :

```
./prog -o output.ppm -s 3 -i input.ppm -t blur
```

L'outil privilégié pour récupérer facilement ces arguments est la fonction `getopt` (`man 3 getopt`). Mais libre à vous de proposer (et d'expliquer dans le rapport) une solution alternative et votre façon de gérer les arguments et les valeurs par défaut. Prenez soin de rendre votre programme robuste aux mauvaises entrées. Si les arguments sont incorrects, le programme ne peut (doit) pas se lancer. Idéalement, on souhaite même afficher l'erreur effectuée par l'utilisateur.

Il sera important de documenter le fonctionnement de votre programme pour indiquer à l'utilisateur comment l'utiliser. Inspirez-vous par exemple des messages d'aide des commandes UNIX...

Bonus : La compilation de votre programme pourra se faire via un fichier Makefile.

3 Fonctionnalités

Les fonctions suivantes seront ajoutées au fur et à mesure du projet pour venir étoffer les possibilités du programme. Il est possible de développer ces fonctionnalités en parallèle de la gestion des arguments.

3.1 Lecture/Écriture

Deux formats de fichiers sont possibles, le format PPM que vous avez utilisé jusqu'ici dans les TPs, et le format JPG, plus complexe, mais beaucoup plus répandu. Votre programme pourra aisément déterminer quel format utiliser en fonction du nom des fichiers (surtout de leur extension). Pour gérer le format JPG, une bibliothèque vous est fournie, à vous d'en faire bon usage. Cette dernière fait appel à `<jpeglib.h>`, qui a besoin de l'option `-ljpeg` à la compilation.

Vous pouvez réaliser une première fonction (`-t copy`) qui lit une image dans un format et la recopie sans modification dans un autre fichier avec un format potentiellement différent. À noter que si les formats sont les mêmes et seul le nom change, sans doute y a-t-il une méthode très simple de faire...

Vous pouvez bien sûr tester cette fonction et les suivantes, indépendamment du programme complet qui fait l'analyse des arguments. À terme, la ligne de commande pourra ressembler à :

```
./prog -t copy -i input.ppm -o output.jpg
```

3.2 Conversion niveaux de gris

Un filtre assez simple qui peut être réalisé est la conversion en niveaux de gris (`-t gray`). Vous découvrirez ce que cela implique plus en détails au Semestre 8. On se souvient déjà qu'une image couleur est composée de 3 canaux (R,G,B). La manière dont sont ordonnés les pixels vous est illustrée dans le premier TP.

Une image visualisée en niveaux de gris (monochrome), est une image pour laquelle en chaque pixel, les valeurs des 3 canaux sont identiques (aucune nuance de couleur). On peut par exemple calculer cette valeur en faisant une moyenne des trois canaux, ce qui peut correspondre à une définition de l'intensité lumineuse. Autrement dit en chaque pixel (i,j), on a L(i,j) = (R(i,j)+G(i,j)+B(i,j))/3. Cette valeur d'intensité lumineuse est alors recopiée sur les trois composantes de l'image de destination pour obtenir des variations de gris.

3.3 Floutage

Le terme *filtrage*, renvoie souvent au fait d'appliquer une *convolution* (cf. cours de traitement du signal). En 2D, dans le domaine image, c'est la même chose, on va convoluer une image avec un filtre. En pratique, dans l'image filtrée I', la valeur d'un pixel dépend de celles des pixels de son voisinage dans l'image à traiter I et des coefficients du filtre (combinaison linéaire). On peut se représenter l'opération comme le passage du filtre H de taille $(l \times l)$ en fenêtre glissante sur toute l'image, comme illustré dans la figure suivante.

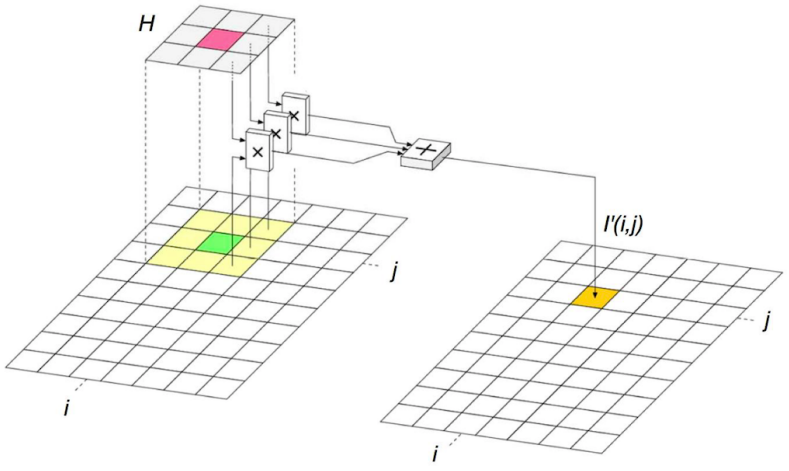

Figure 1: Illustration du principe de convolution pour un filtre 3×3.

L'équation générale de convolution s'écrit comme suit :

$$I'(i,j) = \sum_{(u,v)\in H} I(i-u, j-v)H(u,v). \quad (1)$$

Par exemple pour un filtre 3x3 :

$$I'(i,j) = \sum_{u=-1}^{1}\sum_{v=-1}^{1} I(i-u, j-v)H(u,v) \quad (2)$$

Dans ce cas si les coefficients du filtre sont tous à valeur 1/9, on obtient un filtre moyenneur, qui aura pour effet de flouter l'image.

Implémenter la fonctionnalité floutage se révèle donc beaucoup plus simple que d'implémenter explicitement la convolution avec un filtre à coefficients non égaux, car elle revient à calculer pour chaque pixel la moyenne des pixel dans un voisinage carré de taille $l \times l$. La principale difficulté vient de la gestion des bords, où le voisinage dépasse des bornes de l'image. Plusieurs solutions existent mais la plus simple dans notre contexte peut consister à réduire la taille de la fenêtre et adapter le calcul de la moyenne.

Ce traitement s'effectue de manière indépendante pour chaque canal de couleur.

Côté commande, le floutage correspondra à l'option `-t blur`, et on pourra éventuellement spécifier la demi-taille du voisinage par l'option `-s`, tel que $l = 2 * s + 1$, sinon $s = 1$ par défaut.

3.4 Filtrage médian

Cette fonctionnalité s'inspire fortement du filtrage effectué dans la partie précédente. Cette fois au lieu faire la moyenne des pixels dans un voisinage, on va récupérer la valeur médiane des pixels. En reprenant les codes de tri de tableau d'entiers vus au premier semestre, adaptez la fonctionnalité précédente pour effectuer ce filtrage médian (option `-t medfilt`).

Vous pouvez tester la fonction sur l'image *noise_sp.jpg* et comparer à l'effet du floutage précédent.

3.5 Masque

Votre bibliothèque permettra également d'effectuer les traitements sur un masque binaire en entrée (`-m <mask_name>`). Ce masque sera donc une image de la taille de votre image d'entrée (attention aux dimensions !), qui vaudra 255 partout où le traitement devra s'effectuer, 0 ailleurs.

Modifiez vos fonctions pour prendre en compte ce masque. Par défaut, les traitements se feront sur toute l'image. Vous pouvez par exemple appliquer un floutage sur l'image *person.jpg* avec masque *mask_person.ppm*.

3.6 Autres

Vous êtes libres d'ajouter des fonctionnalités à votre bibliothèque. À vous de juger dans quel fichier (.c, .h) les ajouter. Ci-dessous quelques idées de fonctions classiques de traitement d'images :

- Effet négatif
- Miroir (vertical/horizontal)
- Tracé d'images vectorielles (cf. TP 3)
- Floutage Gaussien
- Mise à 0 de canaux couleurs
- Mesure du temps d'exécution
- Crop
- Redimension
- ...

Toute remarque de votre encadrant est prioritaire sur ce sujet