Rapport de projet de programmation impérative

UE PRIM11 encadrée par M. Guillaume BUREL

Décembre 2024 - Janvier 2025

Mehdi SAHTALI

Table des matières

1 Court manuel d'utilisation et conventions

1.1 *Lenna_input* et *Lenna_output*

J'ai pris une petite liberté avec le sujet en créant deux dossiers `Lenna_input` et `Lenna_output` de manière à séparer les entrées et les sorties du programme. Cela permet de faire un simple `rm -rf Lenna_output/*`, contenu dans la cible `clean` du `Makefile` fourni dans mon archive, pour supprimer toutes les images produites par le programme, et ainsi tester plus rapidement et avec moins de risque d'erreur. Avant de prendre cette initiative, je supprimais à la main les fichiers de sortie en prenant garde à ne pas supprimer les entrées `Lenna_grey.pgm`, `Lenna_color.ppm`, `Lenna_BW.pgm`. J'ai fini par trouver cela fastidieux au bout de 3-4 fois, d'où ma décision.
Cependant, pour rester proche de l'énoncé, le programme ne suppose pas l'existence de `Lenna_input`, ni celle de `Lenna_output`. On pourra supposer que la source des fichiers est quelconque, et quant au résultat, le dossier `Lenna_output` est produit automatiquement s'il n'existe pas déjà.

1.2 Usage

Le programme prend 1 argument ou plus. Le troisième argument, s'il existe (ce qui fixe la valeur de THIRD_ARGUMENT_FLAG), sert de masque. Si l'on suppose que les fichiers à traiter sont stockés dans un dossier `input`, la commande

`./prog input/Lenna_gray.pgm input/Lenna_color.ppm input/Lenna_BW.pgm input/commented.ppm`
génère le dossier `Lenna_output` et les fichiers suivants:

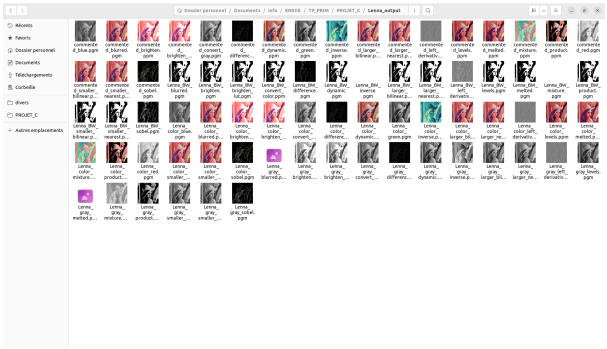

Figure 1: Les fichiers générés par la commande

NOTE IMPORTANTE: A cause du fonctionnement de `filename.c`, fichier fourni, pour utiliser des fichiers du répertoire courant il faut écrire `./prog ./arg1 ./arg2 ... ./argn` et pas seulement `prog arg1 arg2 ... argn`.

2 Organisation du programme - fonctions des fichiers

2.1 Informations générales

Le programme se compose de x fichiers, dont y sont demandés par l'énoncé et 2 sont des ajouts personnels.

2.2 main.c

Le fichier `main.c` contient la fonction principale `main`. On y effectue la création du dossier de sortie `Lenna_output` si besoin, et l'appel aux fonctions de filtrage via celles de `wrappers.c`

2.3 pictures.[h|c]

Le fichier `pictures.c` contient l'essentiel des fonctions qui étaient à écrire. C'est également le plus long, avec près de 1050 lignes dont un nombre non négligeable sont des commentaires et des lignes vides pour plus de lisibilité.

2.4 pixels.[h|c]

Le fichier `pixels.c` contient les fonctions utilitaires de lecture et d'écriture de pixels dont l'implémentation est suggérée dans l'énoncé. J'ai créé deux fonctions que je n'ai pas utilisé par la suite, pour modifier par effet des variables de type byte (`read_pixel_rgb` et `read_pixel_bw`). On pourrait les supprimer dans une version *release* destinée à être publiée , tout comme les vérifications faites avec `assert` qui ralentissent probablement un peu le code mais ont permis un débogage rapide dans les premiers temps du développement du projet.

2.5 picture_rw.[h|c]

J'ai écrit dans ce fichier les fonctions demandées `read_picture` et `write_picture`, ainsi que quelques fonctions auxiliaires dont j'ai pris l'initiative et qui permettent de factoriser et alléger le code. Notamment, la fonction
`bool read_correctly_block(bool *rc, char *buf, FILE *f, picture *res, int *lc_p);`
est utilisée à deux reprises afin de gérer les commentaires. On pourra tester le programme sur le fichier `commented.ppm` de la manière suivante:
`./prog Lenna_input/commented.ppm`

2.6 wrappers.[h|c]

J'ai créé ce fichier afin d'encapsuler la gestion des ressources nécessaires au fonctionnement de chaque fonction et à l'écriture de l'image correspondante. Cela permet d'alléger la fonction `main`, et facilite le débogage au fur et à mesure. En particulier, on peut se contenter de commenter exactement une ligne et une seule dans le main pour chaque fonction si l'on ne veut pas que le fichier correspondant soit écrit.

3 Choix effectués

3.1 Fichiers créés: énoncé et ajouts personnels

3.2 Fichiers modifiés

3.2.1 filename.[h|c]

J'ai pris la liberté de modifier très légèrement filename.[h|c] pour ajouter un prototype manquant ainsi que le mot-clef const devant certains arguments de type `char*`.

3.3 Implémentation de la fonction *mix_picture*

3.3.1 Combinatoire des entrées

A priori, $2^3 = 8$ cas possibles en entrée. Cependant, certains n'ont pas vraiment de sens. En effet, tous les cas où les deux premières images sont de type différent posent problème conceptuellement: avec quelle composante de l'image en niveaux de gris doit-on effectuer le barycentre entre deux composantes, si l'autre image est en couleurs ? Pour y remédier, on voit deux solutions.

3.3.2 Premier choix possible: conversion en image RGB

C'est le choix retenu, qui m'a paru le plus simple à implémenter: on convertit toutes les images d'entrée en images couleurs grâce à `convert_to_color_picture`

3.3.3 Second choix possible: Séparation/Fusion

On aurait également pu faire le choix inverse, en implémentant d'abord la fonction pour des images en niveaux de gris, puis en utilisant `split` et `merge` pour recombiner les images.

3.3.4 Considérations sur l'efficacité

3.4 Instructions conditionnelles dépendant d'une constante dans une boucle

A plusieurs reprises, j'effectue un test dans une boucle imbriquée sur tous les pixels, qui ne dépend pas des indices du pixel. Cela afin d'avoir à éviter de dupliquer la boucle elle-même dans différente branches d'une instruction conditionnelle, ce qui est autrement plus lourd à la lecture (je l'ai fait une fois pour le principe). De plus, le compilateur, en particulier avec l'option -O2, détecte ce genre de micro-optimisation (la complexité asymptotique est inchangée dans tous les cas) et l'effectue seul.

3.4.1 Discussion de l'efficacité

A cause notamment du système de wrappers, l'utilisation mémoire totale au cours du temps n'est pas optimale. On le voit avec `valgrind` qui signale plus de 300 allocations et libérations de mémoire. Les algorithmes me semblent difficilement améliorables en termes de complexité asymptotique, mais certains choix pourraient probablement être revus pour une meilleure efficacité pratique. En particulier, les `assert` pourraient être enlevés

3.5 Ajouts personnels

3.5.1 Filtre de Sobel-Feldman

La notion de dérivée d'une image m'intriguait, j'ai donc décidé de me renseigner sur la page Wikipédia. J'ai ainsi implémenté le filtre de Sobel.

4 Difficultés rencontrées et comportements inattendus

4.1 La fonction *mix_picture*

La fonction `mix_picture` m'a causé des difficultés par la gestion délicate de la mémoire qu'elle a requis (avec le comportement récursif que j'avais alors défini). J'ai finalement résolu le problème de double `free()` en passant par des copies et une fonction auxiliaire `mix_reformat`.

4.2 Outils de débogage

Afin de détecter au plus vite des bugs dans mon programme, j'ai eu recours à quelques outils supplémentaires.

4.2.1 Adress Sanitizer

Cet outil initialement développé par Google peut s'utiliser avec l'option de compilation -fsanitize=address. Il permet de détecter les buffer overflows, heap overflows et certaines fuites mémoire. Je l'ai désactivé dans le makefile de la version rendue car il peut écrire "AddressSanitizer: DEADLYSIGNAL" et arrêter le programme lors de certaines exécutions et je ne voulais pas que cela puisse passer pour un bug de mon programme.

4.2.2 Options de compilation

J'ai ajouté de nombreuses options de compilation, dont: -Wpedantic -pedantic-errors -Werror -Wall -Wextra -Wbad-function-cast -Wcast-align -Wcast-qual -Wdisabled-optimization -Wfloat-equal -Wformat=2 -Wlogical-op -Wmissing-declarations -Wmissing-include-dirs -Wmissing-prototypes -Wnested-externs -Wpointer-arith -Wredundant-decls -Wsequence-point -Wshadow -Wstrict-prototypes -Wswitch -Wundef -Wunreachable-code -Wunused-but-set-parameter -Wwrite-strings -Wformat=2, et ce afin de favoriser le code le plus strict possible pour avoir le moins de bugs à corriger par la suite.

4.2.3 Valgrind

En fin de projet, j'ai utilisé l'outil Valgrind pour détecter les fuites mémoire que AddressSanitizer aurait pu manquer. Il s'avère qu'il n'y en avait pas, mais certaines choses sont améliorables d'après le rapport de Valgrind:

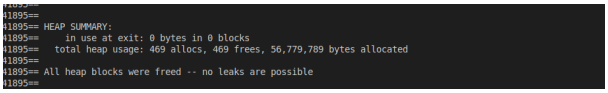

Figure 2: Programme lancé sur 4 arguments

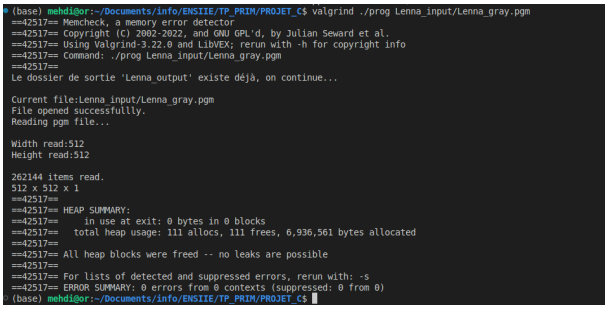

Figure 3: Programme lancé sur 1 argument

4.3 Non-détection remarquable d'une fuite mémoire par ASan

L'utilisation de Valgrind en plus de AddressSanitizer a été motivée par un bug particulier: ce dernier ne détectait une fuite mémoire que si j'en créeais une autre...

5 Ouverture: améliorations potentielles

5.1 Support plus abouti des formats PGM/PPM/PBM

Après renseignements ultérieurs, il semblerait que le format demandé soit une version stricte (un "subset") des formats PGM/PPM. On pourrait, en guise d'amélioration, gérer de manière plus souple les commentaires, les retours à la ligne, etc. comme le précise le standard. On pourrait également gérer la valeur maximale.

5.2 Support d'autres formats

De même, on pourrait envisager de supporter d'autres formats.

5.3 Matrices de transformation

Il serait intéressant de supporter des matrices de transformation pour généraliser les fonctions de redimensionnement, permettre des rotations et symétries,etc.

5.4 Parallélisme: threads, GPU...

Un logiciel de manipulation d'image digne de ce nom devrait utiliser une forme de parallélisme, au moins sur de grandes images, pour accélerer les traitements. Par nature, les calculs sont largement indépendants et il serait donc aisé d'utiliser des `pthreads` pour gérer cela à partir d'une taille d'image seuil. Une autre piste est l'utilisation de cartes graphiques, via OpenGL par exemple.

6 Annexe

6.1 Gestion du code source avec Git

J'ai utilisé le gestionnaire de code source Git pour réaliser mon projet. Cela m'a particulièrement aidé lorsque j'ai écrasé par mégarde le contenu d'un fichier de code avec les données d'une image produite. De manière plus générale, cela m'a permis d'avancer avec plus de confiance dans l'implémentation de nouvelles fonctionnalités, sachant que je pourrais toujour revenir à une version qui fonctionnait en cas de problème.

Le dépôt local sur mon ordinateur est attaché à un dépôt distant privé. Après avoir amélioré le programme, l'avoir rendu plus clair et utile, et avec l'autorisation de M.BUREL, que je compte demander, j'envisage de rendre le dépôt public.

6.2 Filtrage de Sobel: Une ou deux images.

J'ai mis du temps à comprendre qu'il fallait appliquer un seuil pour obtenir les contours après la convolution de l'image avec l'opérateur de Sobel-Feldman...

Figure 4: Noyau: opérateur de Sobel-Feldman, puis seuil déterminé empiriquement.