Institut de Technologie
Enseignement de Promotion sociale
iTech

avril 2021

Programmation procédurale

Dossier 2

Transformations d'images bitmap

Etudiante : Laura BINACCHI
Professeur : Eric BOSLY
Cours : Programmation procédurale 2020-2021

Table des matières

1 Cahier des charges

On demande d'écrire un programme en C qui permet de générer une série de fichiers image au format .bmp 24 bits. Ces images seront obtenues par application de filtres décrits ci-dessous à des images existantes. Le programme fonctionnera en 2D, le but de l'exercice n'est pas artistique mais bien de concevoir et implémenter des effets graphiques faisant appel à plusieurs images. Le programme sera décomposé en blocs fonctionnels cohérents,fera appel à l'allocation dynamique des matrices utilisées et sera écrit de manière lisible et bien documentée. Il utilisera au maximum les notions de programmation vues au cours, spécialement les structures, les fichiers binaires,les boucles imbriquées complexes à indices multiples et variables.

Ce programme sera purement «batch», aucune interactivité n'est demandée. Le programme principal appellera un ensemble de routines de tests générant les images demandées, mais aussi testant de manière systématique les fonctions primitives nécessaires.

1.1 Filtre 1 : pavage d'images

Une image est divisée en pavés carrés. Dans une première version, ces pavés seront de couleur constante, égale à la couleur moyenne de la portion d'image. Ensuite le pavé sera obtenu par redimensionnement de l'image et recolorisation afin de rendre l'aspect global de l'image.

1.2 Filtre 2 : images en donut

Une transformation de coordonnées entre des coordonnées cartésiennes et polaires permet de représenter l'image sous forme d'un donut.

Ces deux effets sont évidement combinables...

2 Description du programme

2.1 Fonctions utilitaires

Avant l'implémentation des filtres, il est nécessaire d'implémenter des fonctions utilitaires permettant de manipuler des fichiers bitmap. Ces fonctions sont implémentées dans le fichier *bmp_file.c*.

2.1.1 Chargement d'un fichier bitmap

La fonction `load_bmp3` permet de charger une image bitmap dans la mémoire RAM du programme.

Cette fonction doit charger en deux temps l'identifiant du fichier (les caractères `BM`) et le reste des données afin d'éviter un décalage dû au padding ajouté par le compilateur. En effet, le compilateur ajoute de l'espace pour que les champs soient alignées sur quatre bytes, ce qui ne pose de problème pour ce programme que dans le cas de la lecture des informations du fichier.

Le chargement de l'image dans la matrice de pixels tient également compte du padding du fichier bitmap : chaque ligne est composée d'un certain nombre de pixels de trois bytes mais la taille de la ligne entière est un multiple de quatre bytes.

Dès que l'image n'est plus utilisée, la fonction `free_bmp3` doit être appelée pour libérer la mémoire allouée à la matrice de pixels de l'image.

2.1.2 Affichage des métadonnées du fichier

La fonction `print_bmp3_info` permet d'afficher les métadonnées d'une image chargée en mémoire. Cette fonction permet de vérifier la conformité des données chargées avec l'image source. Elle permet également de vérifier l'intégrité des données chargées avec celles du fichier bitmap en ouvrant ce fichier dans un éditeur hexadécimal.

2.1.3 Écriture d'une image

La fonction `write_bmp3` écrit une image en mémoire dans un nouveau fichier (ou en écrasant le fichier existant s'il y en avait un). Comme la fonction de chargement, elle tient compte du padding des lignes du fichier bitmap.

2.1.4 Création d'une image

La fonction `create_bmp3` permet de créer une nouvelle image aux dimensions souhaitées. Par défaut, l'image est noire car sa matrice de pixels est initialisée à `0`. Il est possible de la remplir avec une autre couleur en appelant la fonction `fill_bmp3`.

2.2 Pavage d'images

Le pavage d'images est implémenté dans le fichier *mosaic_filter.c*. Ce filtre utilise la fonction utilitaire `reduce` qui permet, d'une part, de générer une image réduite contenant les couleurs moyennes des tuiles de l'image filtrée et, d'autre part, de générer l'image réduite qui sera insérée dans la tuile.

2.3 Images en donut

3 Tests

Les tests sont implémentés dans le dossier *tests* sous forme de routines appelant les différentes fonctions utilisées par le programme. Ils sont compilés en fichiers exécutables par un *Makefile* généré par *cmake*. La commande `cmake ..` [1] lancée depuis le répertoire *build* génère le *Makefile*. La commande `make` lancée depuis ce même répertoire génère les différents exécutables.

3.1 Chargement d'une image

Le chargement d'une image en mémoire RAM est testé dans le fichier *test_ load.c*. Il teste le chargement en mémoire RAM des images de test placées dans le répertoire *images* et affiche les données du header pour vérifier le chargement correct de celui-ci. Ces informations permettent notamment de vérifier que les hauteurs et largeurs en pixels des images et les tailles en bytes correspondent bien aux données des images testées.

3.2 Copie d'une image

Le test de copie d'image implémenté dans le fichier *test_ copy.c* permet de tester la fonction d'écriture d'une image bitmap en mémoire dans un fichier bitmap. Il permet également de vérifier l'intégrité des données chargées dans la matrice de pixels. Les images crées sont comparées aux images d'origine avec la commande `diff`.

3.3 Création d'une image

La création d'une nouvelle image est testée par le fichier *test_ create.c*. Ce test vérifie que la fonction de création crée bien une image noire (la matrice de pixels est initialisée à `0`) aux dimensions demandées. Il crée des images de largeur variables dont le modulo `4` est égal à `0`, `1`, `2` et `3` pour vérifier que l'écriture de l'image gère correctement le padding des lignes. Il permet également de tester la fonction de remplissage d'une matrice de pixels avec une couleur donnée.

3.4 Réduction d'une image

La fonction utilitaire de réduction d'une image utilisée pour le filtre de pavage est testée dans les fichiers *test_ reduction.c* et *test_ tile.c* qui testent les deux manières dont la fonction sera utilisée par le filtre. Le fichier *test_ reduction.c* teste égalemetn les cas d'erreurs de la fonction de réduction.

Le test de réduction génère une image réduite d'un facteur donné. Chacun de ses pixels est initialisé avec la couleur moyenne de l'ensemble des pixels de l'image source qui sont réduits. Les réductions crées à partir des moyennes de couleurs sont plus lisible (i.e. plus proche de l'image d'origine) que les réductions ne reprenant qu'un pixel sur `x` de l'image d'origine. Cet effet est particulièrement visible pour le fichier *landscape1.bmp* où les bleus sont atténués mais ne sont pas perdus.

Le test de création d'une tuile génère une tuile de taille de donnée et non plus de facteur donné. La tuile à insérer dasn l'image pavée sera toujours carrée. Ce test permet de vérifier le bon fonctionnement des offsets : les tuiles sont bien générées à partir du centre de l'image et pas des bords.

1. Pour ajouter des informations sur le code source lors du développement, j'utilise la commande `cmake -DCMAKE_C_FLAGS=-g ..` qui permet de remonter les erreurs plus facilement avec *gdb*. Pour la version finale, j'utilise la commande `cmake -DCMAKE_BUILD_TYPE=Release ..`

3.5 Valgrind

Tous les exécutables ont été testés avec *valgrind* pour vérifier que tous les espaces de mémoire alloués lors de l'exécution du programme sont bien libérés. Ces tests sont effectuées avec la commande `valgrind -leak-check=full -s ./test` [2] où `test` est le nom de l'exécutable à tester.

Ces tests m'ont permis de rester attentive à toujours libérer les espaces mémoire alloués aux matrices de pixels.

2. `leak-check=full` pour avoir des détails sur les fuites mémoires et `-s` pour avoir des détails sur les erreurs

4 Problèmes rencontrés