INF-155 Travail Pratique #3 (partie 2)

La reconnaissance des régions les plus lumineuses d'une image BMP

(Partie 2 : rassembler et trier les spectres, reconstruction et rapport)

Travail en Équipe Remise en fin de session

1. Rappel des faits (partie 1) :

Dans la partie 1, vous avez développé le module « ***tuiles*** » et vous en mesure, notamment :

- de créer des tuiles et de calculer les spectres gris de chaque tuile. À partir de ce spectre, vous avez été en mesure de calculer les intégrales de luminosité (avec et sans seuils) ;
- de créer de nouveaux bitmaps à partir de votre image originale.

Cette seconde partie complète le travail. Le module développé dans la partie 1 nous servira pour identifier les zones lumineuses d'une image et générer une nouvelle image ne conservant que ces zones. Pour cela, vous devez compléter votre module ***tuiles*** et développer un nouveau module ***recouvrement***.

2. Ajouter trois fonctions au module « tuiles »

La première fonction assure un meilleur ajustement de la tuile dans l'image originale, on va minimiser la partie non-traitée de l'image (les pixels restants au bas et à la droite de l'image) :

```
void calibrer_taille_tuile(BMP *original, int *nb_col, int *nb_lig);
```

Les références « nb_col » et « nb_lig » contiennent déjà les tailles prédéfinies – par exemple : 50 pixels en colonnes et 30 pixels en lignes. Supposons, maintenant, que notre image fasse 1438 pixels en largeur et 1300 pixels en hauteur. Avec une largeur de tuile de 50, nous pouvons avoir 28 tuiles sur la largeur de l'image. Il restera une largeur de 38 pixels (à la droite de l'image) qui ne seront pas incluses dans une tuile. Or, si on répartit 28 des 38 pixels restants pour agrandir nos tuiles de un pixel, il ne restera plus que 10 pixels qui ne seront pas traités. Une largeur de tuile de 51 serait donc plus optimale, car elle minimise la partie qui ne sera pas incluse dans une tuile (et donc non traitée). On applique le même principe pour la hauteur de la tuile et on modifie les valeurs des 2 références.

La seconde fonction indique si deux tuiles sont voisines, c'est un prédicat qui retourne 0 ou 1 :

```
int tuiles_voisines(const t_tuile *tuile1, const t_tuile *tuile2);
```

Vérifier si deux tuiles se touchent (sur tout un côté ou juste diagonalement en un point). Une tuile qui n'est pas au bord d'une image a 8 tuiles voisines et celles d'un bord ont 5 ou 3 voisines (un coin).

La troisième fonction permet de copier le contenu d'un bitmap-tuile dans une image-résultat pixel par pixel selon la taille et les décalages de la tuile utilisée :

```
void copier_tuile_ds_image(BMP *image_res, BMP *image_tu, const t_tuile *tuile);
```

3. Créer le module « recouvrement »

Le module ***recouvrement*** sert à stocker l'intégralité de la partition : il contient tous les spectres des tuiles de l'image (qu'on est capables d'obtenir depuis la partie 1) ainsi que des informations utilitaires tels que le nombre total de tuiles ainsi que la taille des tuiles. Les spectres seront entreposés dans un tableau dynamique de pointeurs à des spectres (type « `t_spectre_gris **` »).

Chaque spectre sera générée dynamiquement avec votre fonction « `get_spectre_tuile();` » (partie 1). Cette fonction retourne l'adresse d'un nouveau spectre (type « `t_spectre_gris *` ») et ce sont ces adresses qui sont entreposées dans le tableau de spectres.

```
typedef struct {
   t_spectre_gris **tab_spectres;  //Tableau dynamique des spectres, chaque case
                                   //contient l'adresse d'un spectre.
   int taille_tab_spectres;        //Taille du tableau dynamique tab_spectre
   int largeur_tuile;              //La largeur et la hauteur de chacune des tuiles
   int hauteur_tuile;              //contenues dans le tableau des spectres
 } t_recouvrement;
```

Ce module contient les huit fonctions essentielles au nouveau type dont :

Un constructeur qui se charge d'allouer un nouveau « t_recouvrement », incluant le tableau de spectres, et d'initialiser les entiers. La taille du tableau de spectres « nb_tuiles » doit avoir été déduite au préalable à partir de la taille de tuile calibrée :

```
t_recouvrement * init_recouvrement(int nb_tuiles, int nb_col_tuile, int nb_lig_tuile);
```

Une fonction qui se charge de libérer toute la mémoire dynamique occupée par les spectres contenus dans le tableau du recouvrement, tous les membres entiers sont conservés intacts, le tableau lui-même n'est pas libéré et le recouvrement pourra donc toujours resservir :

```
void detruire_spectres(t_recouvrement *rec);
```

Un destructeur qui se charge de libérer le tableau dynamique du recouvrement, tous les membres entiers sont mis à 0 et le recouvrement ne pourra plus servir :

```
void detruire_recouvrement(t_recouvrement *rec);
```

Ajouter le spectre reçu en paramètre à sa position d'énumération dans le tableau. La taille de la tuile et son numéro unique dans l'énumération sont validés. Retour de 1 si le spectre est ajouté et 0 autrement :

```
int ajouter_spectre_rec(t_spectre_gris *ptr_sp, t_recouvrement *rec);
```

Obtenir l'adresse d'un spectre à partir d'un entier « k » représentant son ".id_enum" de tuile dans le tableau des spectres. Retour automatique de NULL si la valeur de k est aberrante :

```
t_spectre_gris * get_kieme_ptr_sp(const t_recouvrement *rec, int k);
```

S'assurer que toutes les intégrales (avec seuil) des spectres ont été calculées au ***seuil*** donné en paramètre (le champ *.seuil_limun* d'un spectre contient la valeur du seuil utilisée pour calculer l'intégrale). Les spectres dont l'intégrale a été calculée avec un seuil différent sont recalculés avec ce seuil :

```
void calcul_integrales_seuil(t_recouvrement *rec, double seuil);
```

Trier le tableau des spectres en ordre décroissant selon les valeurs de l'intégrale de luminosité calculées selon le seuil fourni en paramètre (utilisez l'algorithme de tri de votre choix). Si le seuil reçu en deuxième paramètre est différent de « 0.0 » on fera le tri selon l'intégrale « `.integrale_lumin_seuil` » et sinon, le tri utilisera les valeurs de « `.integrale_lumin` » (l'intégrale non-seuillée) :

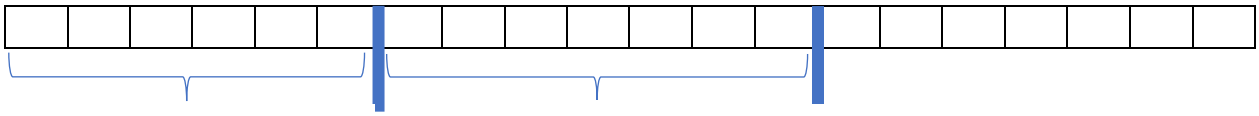
```
void trier_spectres(t_recouvrement *rec, double seuil);
```

La dernière fonction permet de générer l'image résultante à partir d'un recouvrement trié. En plus de l'image originale et le recouvrement trié, la fonction reçoit également deux proportions (entre [0.0, 1.0]) qui indiquent jusqu'à où dans la liste des spectres nous allons limiter notre choix des tuiles retenues. La fonction reçoit aussi le nom du fichier de données à générer (le « *log file* ») durant la reconstruction. (L'utilisation de sous-fonctions privées ici est fortement recommandé, autrement la fonction sera très grosse) :

```
BMP * reconstruire_image(BMP * original, const t_recouvrement *rec,
                         double prop_garde, double prop_min,
                         char * fichier_log);
```

L'algorithme de reconstruction consiste à :

- garder en mémoire la valeur de l'intégrale de luminosité de la tuile la plus lumineuse (qu'on note i_0),
- générer une nouvelle image aux mêmes dimensions,
- copier dans la nouvelle image toutes les tuiles dont le ratio R = (i_x / i_0) de l'intégrale de luminosité (noté i_x) versus i_0 est supérieure à « `prop_garde` » : la tuile est copiée si R > `prop_garde`,
- copier dans la nouvelle image toutes les tuiles avec le ratio R tel que (`prop_min` ≤ R ≤ `prop_garde`) et qui sont voisines d'une des tuiles principales copiées à l'étape précédente.



Tuiles dont le rapport
R > `prop_garde`

→ Copiées automatiquement dans la nouvelle image

Tuiles dont le rapport `prop_min` ≤ R ≤ `prop_garde`

→ Copiées dans la nouvelle image seulement si elles sont voisines d'une des tuiles de la partie gauche

De plus, votre fonction doit générer un fichier de log au format texte (dont le nom de fichier « fichier_log », qui a été généré au préalable, est fourni en paramètre).

Le fichier de log doit spécifier les valeurs de calcul suivantes sur la première ligne :

- le seuil de luminosité,
- la taille des tuiles,
- les valeurs « prop_garde » et « prop_min ».

Ensuite, dans les lignes subséquentes du fichier, un descriptif de toutes les tuiles qui ont été copiées dans l'image résultante (une par ligne). Pour chaque tuile, vous devez spécifier les informations suivantes séparées par des virgules :

- le numéro de la tuile (id_enum),
- son décalage (en x et en y) sur l'image,
- la valeur de son intégrale de luminosité,
- la valeur de son rapport R à la luminosité maximale (le R de la tuile la plus lumineuse sera = 1.0).

4. Votre rapport et tests avec un fichier de configurations

Vous détenez maintenant un programme en mesure d'identifier les zones les plus lumineuses d'une image, et de ne conserver que celles-ci dans une image résultante. Cependant, la performance votre programme pour bien détecter les zones lumineuses dépend des choix initiaux que vous allez effectuer (ex. la taille désirée de la tuile, le seuil de luminosité, *prop_garde* et *prop_min*). De plus, l'approche qui vous est proposée n'est pas infaillible et vous pourriez introduire de petites améliorations aux méthodes de calcul pour optimiser le résultat.

Les cinq valeurs initiales (ainsi que le nom de fichier de l'image originale) doivent être saisies à partir d'un fichier texte de configurations que vous devez lire. La première ligne de ce fichier contient le nombre « nb_essais » qui indique le nombre de lignes de données à lire dans ce fichier. Le format de chaque ligne de données pour faire une exécution spécifique du programme est le suivant :

<nom fichier bitmap> <le seuil> <nb. col. tuile> <nb. lig. tuile> <prop. garde> <prop. min>

Exemple d'un fichier de configuration avec deux exécutions :

```
4
image.bmp  0.4   50  60  0.8  0.5
image.bmp  0.7   80  70  0.7  0.3
```

} 2 lignes de données

Vous aurez donc à écrire les deux fonctions suivantes (dans le fichier principal du projet) :

Ouvrir le fichier de données et retourner son *handle* FILE* (NULL si non-ouvert). On renvoie aussi son nombre de lignes dans la référence « *nb_tests » (la première valeur dans le fichier) :

```
FILE * ouvrir_fich_data(const char * nom_fich, int * nb_tests);
```

Fonction pour lire la prochaine ligne de données dans le fichier de données déjà ouvert. Retour de 1 si succès, 0 sinon. On garde le fichier ouvert en tout temps. On ne validera pas le format du fichier.

```
int lire_donnees_fich(FILE * fich, char * nom_image, double *seuil,
                      int * nb_col_tu, int * nb_lig_tu,
                      double * prop_garde, double * prop_min);
```

Une bonne extraction des zones lumineuses en est une qui :

- réduit les faux positifs : des zones considérées comme lumineuses alors qu'elles ne le devraient pas,
- réduit les faux négatifs : des zones ignorées alors qu'elles auraient dû faire partie de l'image finale.

De plus, les arguments peuvent être adaptés pour un type d'image, tandis que d'autres arguments vous permettent d'obtenir de meilleurs résultats avec une autre image.

À la fin de votre travail, vous devez remettre un rapport (format WORD) qui fait état de :

- vos différents tests que vous considérez significatifs. Le rapport doit inclure les paramètres de calcul que vous avez utilisés, les fichiers logs générés et l'image résultante pour chacun de vos tests jugés significatifs. Ne perdez pas de temps avec la mise en page!
- vos conclusions : Qu'est-ce qui fonctionne mieux? Dans quelles circonstances?

5. Le programme principal

Voici donc l'ordre des étapes à exécuter dans votre « main » :

- Ouvrir le fichier de configurations et récupérer son nombre de lignes.
- Pour chaque configuration du fichier,
  - Lire les données de la ligne suivante du fichier et ouvrir l'image bitmap demandée,
  - Calibrer la taille des tuiles selon l'image et initialiser une tuile avec cette nouvelle taille,
  - Obtenir le nombre total de tuiles et initialiser une variable de recouvrement avec ce nombre,
  - Pour toutes les tuiles de ce recouvrement,
    - Obtenir la « k » ième tuile (selon l'itérateur « k ») et générer son spectre selon l'image,
    - Ajouter ce nouveau spectre au tableau de spectres du recouvrement.
  - Calculer les intégrales de tous les spectres avec le seuil obtenu du fichier et trier les spectres,
  - Composer un nouveau nom de fichier pour le « log file » (dans une *string*). Par exemple :
    « LOG_image.bmp_0.4_(50, 61)_0.8_0.5.txt » (selon la première ligne de configuration)
  - Reconstruire l'image finale avec les données du recouvrement trié,
  - Sauvegarder l'image finale sur disque avec le nom suivant « image_fin_*i*.bmp » (pour le test « *i* »),
  - Libérer les pointeurs d'images et le tableau dynamique du recouvrement et recommencer.

6. Remise finale et contraintes de l'enseignant

- Vous devez vous conformer aux mêmes normes de qualité spécifiées lors des précédents travaux ;
- Vous devez effectuer votre remise sur Moodle (avant la date limite indiquée dans Moodle).
- Votre remise comprend votre (1) votre programme complet (fichiers .c et .h) ET (2) votre rapport au format WORD. BON TRAVAIL !