INF-155 Travail Pratique #3 (partie 1)

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

(Partie 1 : Implémentation d'un premier module et tests à réaliser)

Travail en Équipe Remise en fin de session

1 Objectifs du travail (parties 1 et 2)

1 Développer pour la première fois un outil logiciel dont vous n'avez à priori aucune certitude de résultats. Réaliser qu'alors vous devez traiter tous les résultats obtenus avec une attitude responsable, ce qui va des tests unitaires jusqu'aux expériences finales.

2 Continuer à privilégier le développement modulaire déjà fortement utilisé cette session dans le premier et second TP.

3 Développer votre expérience en programmation scientifique et technologique :

- Réaliser l'implémentation de fonctions correctes selon toutes leurs spécifications,
- Réaliser les tests de validité de chaque fonction (tests unitaires et d'intégration, voir le texte d'Anis Boubaker),
- Réaliser l'importance des « Conditions Initiales » pour juger la qualité des résultats obtenus du logiciel de recherche (partie 2),
- Donner un court rapport final des expériences réalisées et de la performance du logiciel, une analyse comparée des résultats obtenus de chaque image est exigée (partie 2).

4 Respecter plus que jamais toutes les exigences de programmation : l'indentation du code, l'utilisation de constantes, l'ajout de commentaires homogènes et standardisés dans tous vos fichiers (autant *.h que *.c).

2 Les régions les plus lumineuses d'un bitmap

D'abord un peu de mathématiques :

Partitionner un ensemble E*, c'est énumérer des sous-ensembles* $P(i)$ *de* E *mutuellement exclusifs (*$P(i) \cap P(j) = \varnothing$ *) et dont l'union de ces sous-ensembles égale E* $(\cup P(i) = E)$*.*

Pour parvenir à identifier les régions les plus lumineuses d'un fichier bitmap, une méthode algorithmique simple semble évidente :

0 Développer une mesure quantitative de l'intensité lumineuse d'une région rectangulaire quelconque d'un bitmap (ces régions s'appellent des « tuiles »).

1 Partitionner l'image originale en régions de surface égales *(par analogie, on parle alors de pavage, de recouvrement, d'une tuile et de sa surface).*

2 Pour chaque tuile de la partition. Obtenir et conserver la mesure (un nombre réel) de son intensité lumineuse.

3 Ordonner l'ensemble des tuiles en ordre croissant en fonction de la mesure de luminosité obtenue au point précédent (partie 2).

4 Ne conserver que les tuiles les plus lumineuses selon un « seuil » de sélection et reconstruire les images résultantes (partie 2).

Avant même de penser à réaliser cet algorithme, votre première obligation est de savoir utiliser intelligemment les outils fonctionnels élémentaires fournis par le module d'accès aux images bitmap « QDBMP » (*Quick n' Dirty BitMaP*) de Chai Braudo. Cette acclimatation est essentielle à la réussite du projet. On parle ici de tests véritables et réussis en exécution avec des vraies images. Un exemple d'utilisation de la librairie « qdbmp » vous est fourni dans Moodle.

3 L'outil de mesure : le spectre

En physique, le mot « spectre » a une histoire illustre (Newton en a déjà parlé) mais qui garde toujours une définition singulièrement simple : la décomposition d'une lumière en ses composantes élémentaires.

Dans notre monde numérique, le spectre de luminosité se résume au tableau des fréquences des valeurs observées lors de la traversée de l'ensemble des pixels de toute ou d'une partie d'une image RGB. On peut calculer un spectre rouge, vert, bleu ou en différentes teintes de gris.

Notre implémentation du spectre va donc posséder sans surprise un tableau d'entiers.

En imagerie numérique, il existe 256 intensités lumineuses de teintes de gris possibles (sur 8 bits) obtenues d'une combinaison linéaire des trois valeurs R (rouge), G (vert) et B (bleu) d'un pixel d'une image bitmap. L'équation est donnée par : `teinte_gris = (0.299*R + 0.587*G + 0.114*B).`

Dans notre TP, le spectre utilisé va se limiter aux intensités en teintes de gris. Vous savez maintenant ouvrir un fichier bitmap, le traverser au complet ou sur une de ses parties et obtenir les composantes RGB d'un pixel, donc vous pouvez obtenir le spectre d'une tuile.

L'intégrale élémentaire de l'intensité lumineuse d'un spectre restera simple et analogue à l'intégrale de Riemann, soit la somme des produits de la fréquence d'une teinte « `i` » multiplié par l'intensité relative de cette teinte :

`integrale_lumin` = $\sum_{i=0}^{255} \frac{spectre[i]*i}{255}$ `(ou : Somme de tous les (spectre[i] * i/255.0) ).`

4 Description du module attendu

Ce module, vous en avez reçu l'interface « `tuile.h` » qui contient la définition des deux types `t_tuile` et `t_spectre_gris`, vous aurez à y ajouter sept fonctions dédiées au bon usage de ces deux types. Nous nous servons naturellement des types déjà obtenus de « `qdbmp.h` » puisqu'ils sont essentiels au déploiement des fonctions décrites ici. Donc toutes les images bitmap doivent être reçues en paramètre en utilisant le type « `BMP *` » de ce module.

1 Une fonction qui reçoit trois paramètres : un BMP * et les nombres de pixels en colonne et en lignes de chaque tuile. Retourne le nombre total de tuiles de cette taille que l'on peut obtenir dans ce bitmap. Vous considérez que le fichier BMP* aura déjà été ouvert avec `BMP_ReadFile.`

```
int get_nb_tuiles(BMP *original, int nb_col, int nb_lig);
```

2 Une fonction qui initialise une tuile avec la taille reçue (en colonne et en ligne), et qui initialise les autres champs à -1.

```
void init_tuile(int nbcol_tuile, int nblig_tuile, t_tuile * tuile);
```

3 Une fonction qui reçoit trois paramètres : un BMP *, un numéro d'ordre d'une tuile (en supposant que les tuiles sont numérotées linéairement, la tuile 0 est celle du coin supérieur gauche) et une référence sur une tuile dont le nombre de colonnes et de lignes a déjà été fixé par la fonction précédente. Si ce numéro « `k` » n'est pas valide, la fonction retourne 0 sinon elle calcule les deux offsets en pixels de cette tuile qu'elle place dans la référence ***tuile*** et retourne 1.

```
int get_kieme_tuile(BMP *original, int k, t_tuile * tuile);
```

4 Une fonction qui reçoit deux paramètres : un BMP * et une tuile obtenue de la fonction précédente (donc dont tous les champs ont reçus une valeur). La fonction va faire l'allocation dynamique d'un `t_spectre_gris` puis calcule le spectre de la tuile donnée. Elle complète le contenu du nouveau spectre en faisant le calcul de l'intégrale de l'intensité du spectre et initialise les autres champs à la valeur **-1**. La fonction retourne l'adresse du nouveau spectre.

```
t_spectre_gris * get_spectre_tuile(BMP *original, const t_tuile * tuile);
```

Notes :
- Rappelez-vous que le spectre est un tableau de fréquences. Calculer le spectre revient à déterminer, dans la tuile combien de pixels ont chaque teinte de gris. Donc pour chaque pixel, il faut calculer sa teinte de gris - on obtient une valeur entre 0 et 255, par exemple 123. Il faut alors spécifier dans le tableau du spectre que nous avons un nouveau pixel dont la teinte de gris est 123. Une fois qu'on aura traité tous les pixels, nous saurons combien de pixels ont une teinte de gris qui vaut 0, combien de pixels ont une teinte de gris valant 1, …. , combien ont une teinte de gris de 255.

- Une fois le spectre calculé, il suffit de calculer l'intégrale lumineuse selon la formule de la page précédente.

5 Une fonction qui reçoit deux paramètres : l'adresse d'un spectre et un seuil réel de luminosité. Elle calcule l'intégrale du spectre au-dessus de ce seuil : on fait le même calcul que le calcul de l'intégrale complète, sauf que tous les `i` tels que `((i / 255.0) < seuil_lum)` ne sont PAS considérés dans la sommation. Finalement elle fixe les deux champs `seuil_lumin` et `integrale_lumin_seuil` dans le spectre et retourne la valeur calculée de l'intégrale.

Validation : si la valeur du seuil reçu en paramètre n'est pas entre [0.0, 1.0] la fonction retourne **-1**.

Note : avec la valeur de seuil « 0.0 » la fonction calcule l'intégrale complète de l'intensité du spectre.

```
double integrale_seuil_lum(t_spectre_gris * ptr_sp, double seuil_lum);
```

Les deux dernières fonctions sont spéciales puisqu'elles retournent un nouveau « BMP * » obtenu de `BMP_Create()` qui pourra être sauvegardé en mémoire avec la fonction `BMP_WriteFile()`.

6 Une fonction qui reçoit deux paramètres : un BMP * (l'image originale) et une tuile. Elle obtient un nouveau BMP * de `BMP_Create()` de la même taille que la tuile reçue et copie dans le nouveau BMP tous les pixels de cette tuile à partir l'image originale, sans aucune transformation.

Note : le nouveau bitmap doit absolument être créée avec le même nombre de bits par pixels que l'image originale (obtenu avec « `BMP_GetDepth()` »)

```
BMP * get_bitmap_tuile(BMP *original, const t_tuile * tuile);
```

7 Proche de la précédente fonction sauf que l'image obtenue sera en teintes de gris (en utilisant la règle de calcul de teinte de gris à partir de RGB) et qu'on applique un seuil « `seuil_lum` » lors de la copie.

Par exemple avec un seuil_lum = 0, les pixels originaux sont tous conservés dans leurs tons de gris, avec un seuil_lum = 0.5, tous les pixels à luminosité inférieure à 128 sont noirs et avec seuil_lum = 1 tout est noir.

```
BMP * get_bitmap_gris_tuile(BMP *original, const t_tuile * tuile,
                            double seuil_lum);
```

Note : Pour créer une image à teintes de gris en 24 bits (3 couleurs), vous devez calculer la teinte de gris et utiliser cette valeur pour les composantes R, G et B.

En plus, vous devez aussi implémenter les 4 petites fonctions informatrices suivantes :

- Retourne l'intégrale calculée en incluant tous les i (intégrale sans seuil)

```
double get_integrale_sans_seuil(const t_spectre_gris * ptr_sp);
```

- Retourne le seuil utilisé dans le calcule de l'intégrale avec seuil (-1 par défaut)

```
double get_seuil_luminosite(const t_spectre_gris * ptr_sp);
```

- Retourne l'intégrale optionnelle calculée (-1 si non calculé)

```
double get_integrale_avec_seuil(const t_spectre_gris * ptr_sp);
```

- Place dans la référence la copie de la tuile originale de ce spectre

```
void get_tuile(const t_spectre_gris * ptr_sp, t_tuile * tuile);
```

On s'attend évidemment à ce que vous pensiez à implémenter aussi un « `afficher_spectre()` ».

4 MANDAT 1 : réussir l'implémentation de l'ensemble de la première partie

Dans un simple « main » vous avez à tester l'ensemble de ces fonctions, il suffit à priori de ;

- Lire avec « `BMP_ReadFile()` » une seule image couleur originale (on parle ici d'une image importante de plus de 10Mo, rien de minuscule, c'est clair).
- Définir trois tailles de tuiles nettement distinctes mais toutes trois de taille crédible (encore, rien de minuscule), obtenir et afficher le nombre de tuiles de chacun des trois recouvrements.
- Choisir une tuile au hasard dans chacun des trois recouvrements et calculer et afficher leur spectre.
- Terminer en obtenant deux bitmaps (en couleur ET en gris avec un seuil de 0.0) de chacune des trois tuiles obtenues au point précédent. Chacune des 6 images est sauvegardée en mémoire avec « `BMP_WriteFile()` ».
- Réessayez ensuite avec différentes valeurs de seuils pour comparer et comprendre leurs résultats.

NOTE : Ce programme doit s'exécuter correctement avant même de songer à s'attaquer à la partie 2.

BON TRAVAIL !