Traitement d'image (2)

Dans ce TP vous allez programmer des fonctions pour traiter des images couleur : conversion en niveaux de gris, changement de la luminosité, etc. Pour manipuler les pixels, vous allez définir deux types :

- le type `point` qui représente une position (x, y) dans l'image, avec x et y des entiers positifs ;
- le type `couleur` qui représente un triplet (r, v, b) où chaque composante rouge, verte ou bleue est un entier entre 0 et 255.

Le fichier `couleur.ppm` contient l'image sur laquelle nous allons travailler. Le fichier `traitement-nuplets.ml` est à compléter pour répondre aux questions. Le fichier `traitement-nuplets-aux.ml` contient les fonctions fournies (chargement d'une image, affichage, application des traitements, etc.). Les fonctions fournies qui nous importent sont :

- `traiter` qui prend en paramètre une fonction de traitement (`couleur -> couleur`).
- `traiter_local` qui prend en paramètres une fonction de sélection (`point -> bool`) et une fonction de traitement (`couleur -> couleur`).

Ces deux fonctions appliquent le traitement à chaque pixel puis elles affichent l'image. Elles ne renvoient rien à proprement parler : leur type de retour est `unit`. Attention à ne jamais fermer la fenêtre graphique, sinon cela ferme `ocaml`.

Exercice 1 les points et les couleurs

1. Définissez un type `couleur` comme un triplet d'entiers.
2. Définissez un constructeur `creer_couleur`. Utilisez la fonction `tronque` fournie pour assurer un résultat valide (composantes dans $[0, 255]$).
3. Définissez des accesseurs `acces_rouge`, `acces_vert` et `acces_bleu`.
4. Testez ces fonctions.
5. Définissez un type `point` comme un couple d'entiers.
6. Définissez un constructeur `creer_point` et des accesseurs `acces_x` et `acces_y`.
7. Testez ces fonctions.
8. Concevez et programmez une fonction `distance` de prototype `point -> point -> int` qui calcule la distance euclidienne entre deux points (on utilisera des fonctions de conversion de types). Testez votre fonction.

Exercice 2 traitement global de l'image

Nous allons ici traiter l'image globalement (c'est à dire tous les pixels de la même manière).

1. La fonction fournie `calculer_gris` a pour prototype `couleur -> couleur` et transforme une couleur en gris. Observez que la commande `traiter calculer_gris` transforme tous les pixels en gris.
2. Concevez et programmez une fonction `change_lum` de prototype `int -> couleur -> couleur` qui prend en paramètres un décalage l (positif ou négatif) et une couleur (r, v, b) , et qui renvoie la couleur $(r + l, v + l, b + l)$.

3. Pourquoi est-il inutile de tronquer le résultat ?
4. Quel est le prototype de l'application partielle (`change_lum (-70)`) ? Que fait-elle ?
5. Déduisez-en une syntaxe pour appliquer la fonction `change_lum` sur toute l'image.
6. Concevez et programmez une fonction `contraste` de prototype `float -> couleur -> couleur` qui prend en paramètres un facteur d'amplification m (flottant positif) et une couleur, et qui transforme chaque composante t (rouge, verte ou bleue) en $128 + m(t - 128)$. Comme m est flottant, le résultat du calcul doit être arrondi et converti en entier (fonction `int_of_float`).
7. Testez votre fonction.

Exercice 3 les rectangles et les disques

Nous allons définir quelques formes géométriques simples pour délimiter des zones dans l'image.

1. Définissez un type `rectangle` comme un triplet contenant un point (le coin supérieur gauche) et deux entiers (la hauteur et la largeur).
2. Définissez un constructeur `creer_rectangle` et des accesseurs `acces_origine`, `acces_hauteur` et `acces_largeur`.
3. Testez ces fonctions.
4. Définissez un type `disque` comme un couple contenant un point (le centre) et un entier (le rayon).
5. Définissez un constructeur `creer_disque` et des accesseurs `acces_centre` et `acces_rayon`.
6. Testez ces fonctions.

Exercice 4 traitement local de l'image

Pour traiter l'image localement, nous allons utiliser des fonctions permettant de tester si un point est dans (ou hors d') une zone (rectangle ou disque).

1. Concevez et programmez une fonction `est_dans_rectangle` de prototype `rectangle -> point -> bool` qui prend en paramètre un rectangle r et un point p , et qui renvoie `true` si et seulement si p est dans r .
2. Concevez et programmez une fonction `est_hors_rectangle` complémentaire de la précédente.
3. Testez votre fonction. La fonction `traiter_local` prend comme premier paramètre une fonction de prototype `point -> bool` qui indique si un point doit être traité (et comme second paramètre une fonction de traitement). Un exemple de syntaxe est fourni dans le fichier.
4. Concevez et programmez une fonction `est_dans_disque` de prototype `disque -> point -> bool` qui prend en paramètre un disque d et un point p , et qui renvoie `true` si et seulement si p est dans d .
5. Concevez et programmez une fonction `est_hors_disque` complémentaire de la précédente.
6. Testez votre fonction.