Manipulation basique d'images

Marin Bougeret

13 septembre 2010

Résumé

Le but ici est de manipuler un format d'image simple, et d'écrire au passage quelques algorithmes classiques de traitement d'image.

1 Objectif

Vous devez écrire un module d'entrée/sortie minimal de pgm (le pgm est une restriction du ppm en niveaux de gris) en Ocaml, c'est à dire de quoi convertir une image en un format interne qui vous plaît, et de quoi convertir ce format interne en une image. On peut alors en profiter un peu, en implémentant quelques fonctions classiques des logiciels de traitement d'image :

- l'inversion noir et blanc
- le "floutage" (blur)
- la détection de contour
- ...

2 Un format d'image simple : ppm

Je vous renvoie à la section 1.1 du TP "seam_carving.pdf" de 2007 (et peut être de 2010 !) qui explique bien le minimum à savoir sur ppm.

3 Quelques fonctions d'E/S

Voici des primitives simples d'entrée/sortie qui suffisent pour ce TP (libre à vous d'en utiliser d'autres si vous préférez).

3.1 Lecture

La fonction **open_in nomfichier** prend une chaine *cheminfichier* en argument et retourne un "in_channel" (un canal) positionné au début du fichier correspondant. Une fois le canal ouvert, on peut lire dedans avec les fonctions suivantes (voir l'exemple plus bas) :

- **input_line x** qui prend un in_channel x, avance dans x de la "position courante" jusqu'au prochain retour chariot (code 0a en hexadécimal), et retourne la chaîne correspondant à la concaténation de tous les caractères parcourus (sans le retour chariot)
- **input_char x** qui prend un in_channel x, avance d'une position dans x, et retourne le caractère lu

Ces deux fonctions retournent une exception lorsque qu'elles rencontrent la fin du canal. Enfin, on referme un tel canal en appelant **close_in x**. Par exemple, si l'on exécute sur l'image d'exemple de la Section 1.1 du document "seam_carving.pdf" :

```
let ic = open_in "mon_image.pgm";;

let s1 = input_line ic;; (* s1 vaut "P5" *)
let c1 = input_char ic;; (* c1 vaut '1' *)
let s2 = input_line ic;; (* s2 vaut "60 221" *)
let s3 = input_line ic;; (* s3 vaut "255" *)

let c2 = input_char ic;; (* c2 est le caractère de code ASCII c1 (en hexa) *)
let c3 = input_char ic;; (* c3 est le caractère de code ASCII c1 (en hexa) *)
let c4 = input_char ic;; (* c4 est le caractère de code ASCII c1 (en hexa) *)
let c5 = input_char ic;; (* c5 est le caractère de code ASCII c0 (en hexa) *)

close_in ic;;
```

3.2 Ecriture

La fonction **open_out nomfichier** prend une chaîne *nomfichier* en argument, créer un fichier "nomfichier" (ou l'écrase si il existe déjà) et retourne le "out_channel" correspondant. Une fois le canal ouvert, on peut écrire dedans avec la fonction **output x ch deb long** qui prend un in_channel x, une chaîne *ch*, et écrit dans x les *long* caractères situés à partir de la position *deb*. Enfin, on referme un tel canal en appelant **close_out x**. Les modifications faites via output ne seront effectives que lorsque le canal sera fermé. Un exemple :

```
let oc = open_out "res.ppm";;
output oc "bonjou" 2 6;; (* ici si on regarde dans fichier on ne voit rien *)
close_out oc;; (* ici on voit *)
```

4 Quelques fonctions diverses utiles

Voici une liste de primitives utiles (je ne donne pas de commentaire lorsque l'effet est évident !) :

- **char_of_int i** transforme l'entier i en caractère
- **int_of_char c**

- **int_of_string s**
- **string_of_int i**
- **String.make l c** crée une chaine de taille l contenant partout le caractère c
- **String.length s**
- **Array.make l x** crée un tableau de l cases contenant partout l'élément x (attention si x n'est pas un élément "de base", il y aura "partage")
- **Array.length t**