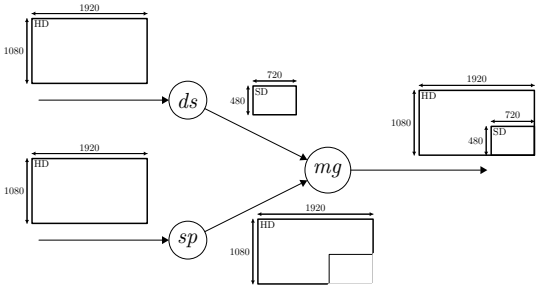
Picture in Picture [1]



Dans ce TP, nous allons programmer des fonctions de manipulation d'image en nous basant sur l'écriture de fonctions récursives sur les listes. Nous allons en particulier réaliser l'incrustation d'une image dans une autre comme montré dans la figure ci-dessus.

Récupérer les fichiers `image.ml` et `img.ppm` à l'adresse `http://www.di.ens.fr/~pouzet/cours/systeme/projet/pip`. Compiler le fichier `image.ml` avec la ligne de commande suivante :

```
ocamlc -o image graphics.cma image.ml
```

Le fichier `image.ml` définit les types `pixel`, `line` et `image` suivants :

```
type pixel = int * int * int
type line = pixel list
type image =
    { width: int;
      height: int;
      matrix: line list; }
```

Le type `pixel` est un triplet d'entiers qui représente la couleur d'un pixel par ses trois composantes rouge, verte et bleue (Red, Green, Blue). Le type `line` représente une ligne de pixels triés de la gauche à la droite. Enfin, le type `image` est un enregistrement contenant les dimensions de l'image (largeur = `width` et hauteur = `height`) et la liste des lignes de l'image (`matrix`) triée de haut en bas.

Le fichier contient aussi une fonction `read_ppm: string -> image` qui prend en paramètre un nom de fichier contenant une image au format PPM et retourne une valeur correspondant de type `image`.

Enfin, le fichier contient une fonction `draw_image: image -> unit` qui affiche une image à l'écran.

Question 1
Écrire un programme qui affiche à l'écran l'image au format PPM donnée en argument sur la ligne de commande. La chaîne de caractères qui correspond au premier argument de la ligne de commande est obtenue par l'expression OCaml `Sys.argv.(1)`

Pour construire le miroire d'une image, il suffit d'en inverser l'ordre des lignes.

Question 2
Écrire une fonction `reverse: 'a list -> 'a list` *qui inverse l'ordre des éléments d'une liste. Puis, écrire une fonction* `mirror: image -> image` *qui inverse toutes les lignes d'une image. Tester cette fonction sur l'image* `img.ppm`.

1. Ce sujet a été conçu par Louis Mandel et Florence Plateau à partir d'un exemple réel venant de NXP.

Pour calculer le niveau de gris correspondant à une couleur (r, g, b), on peut appliquer la formule suivante :

$$n = 0.299 \times r + 0.587 \times g + 0.114 \times b$$

Question 3
Écrire une fonction `grey_pixel: pixel -> pixel` *qui transforme la couleur d'un pixel en niveau de gris. Les trois composantes de la couleur de retour doivent avoir la même valeur.*

Question 4
Écrire une fonction récursive `grey_line: line -> line`*, qui applique la fonction* `grey_pixel` *à tous les pixels d'une ligne. Définir ensuite une fonction* `grey_matrix: line list -> line list` *qui applique la fonction* `grey_line` *sur une liste de lignes. Enfin, définir une fonction* `grey_image: image -> image` *qui transforme une image couleur en noir et blanc. Tester cette fonction sur l'image* `img.ppm`.

Pour réduire la hauteur d'une image, on peut supprimer des lignes de celle-ci. Pour cela, nous allons définir un filtre vertical qui supprime une ligne sur deux.

Question 5
Écrire une fonction récursive `half_sampler: 'a list -> 'a list` *qui supprime un élément sur deux dans une liste. Si la liste est de longueur impaire, on gardera le dernier élément. Écrire une fonction* `vf: image -> image` *qui réduit d'un facteur deux la hauteur d'une image.*

Si l'on veut réduire la largeur de l'image, il faut supprimer un pixel sur deux sur chaque ligne.

Question 6
Écrire une fonction `hf: image -> image` *qui réduit d'un facteur deux la largeur d'une image. À partir des fonctions* `vf` *et* `hf`*, écrire une fonction* `downscaler: image -> image` *qui réduit la taille d'une image d'un facteur deux.*

Nous voulons maintenant incruster une petite image dans une grande. Pour cela, on va transformer la matrice des pixels en une liste de pixels, puis on enlève des pixels de la grande image avant de mettre à la place les pixels de la petite image. Enfin, on retransforme la liste des pixels en matrice.

Dans le fichier `image.ml` les fonction suivantes sont fournies :

— `vector_of_matrix: ligne list -> pixel list` : transforme une matrice de pixels en une liste ;
— `matrix_of_vector: pixel list -> int * int -> ligne list` : fait la transformation inverse : à partir d'une liste de pixels et des dimensions de la matrice que l'on souhaite obtenir reconstruit une matrice de pixels ;
— `build_mask: int * int -> int * int -> bool list` : retourne une liste de booléens qui représente le « masque » des pixels à garder à partir de la taille de la grande image et de la petite image.

Question 7
Écrire une fonction `sampler l mask` *qui à partir de deux listes de même taille retourne les éléments de* `l` *filtrés par le motif* `mask`*. Le i-ème élément de la liste* `l` *se retrouve dans la liste résultat uniquement si le i-ème élément de la liste* `mask` *est la valeur* `true`*. Par exemple, l'exécution de*

```
sampler [1; 2; 3; 4; 5] [true; false; true; false; true]
```

doit retourner la liste `[1; 3; 5]`*. Si les deux listes ne sont pas de même longueur, la fonction doit terminer sur une erreur. La fonction* `sampler` *a le type suivant :*

```
val sampler: 'a list -> bool list -> 'a list
```

Pour fusionner la grande image (avec un trou) et la petite image, nous allons utiliser une fonction `merge` qui à partir du masque, de la liste des pixels de la grande image trouée et la liste des pixels de la petite image retourne la liste des pixels de l'image reconstruite.

Question 8
Écrire la fonction `merge mask l1 l2` *qui fusionne les éléments des listes* `l1` *et* `l2` *selon les valeurs de la liste* `mask`. *La longueur de la liste* `l1` *doit être égale on nombre de valeurs* `true` *dans la liste* `mask` *et la longueur de la liste* `l2` *doit être égale on nombre de valeurs* `false` *dans la liste* `mask`. *Sinon, le fonction doit se terminer sur une erreur. Par exemple, l'exécution de* `merge [true; false; true; false; true] [1; 3; 5] [2; 4]` *doit retourner* `[1; 2; 3; 4; 5]`. *La fonction* `merge` *est de type :*

```
val merge: bool list -> 'a list -> 'a list -> 'a list
```

Question 9
Écrire une fonction `pip: image -> image -> image` *qui prend en argument une grande image et une petite et qui retourne une image où la petite a été incrustée dans la grande. Tester votre fonction en incrustant l'image* `img.ppm` *dans elle même (après réduction). Puis, essayer d'incruster* `img.ppm` *dans l'image* `bigimg.ppm`.

Question 10
Réécrire les fonctions `sampler`, `merge` *et* `pip` *de façon récursive terminal. Tester votre nouvelle fonction* `pip` *sur les images* `img.ppm` *et* `bigimg.ppm`.

Question 11
Écrire une fonction `pip_n: int -> image -> image` *qui incruste* n *fois une image dans elle même.*