Compléments d'informatique

Devoir : Seam Carving

11 octobre 2020

Dans ce devoir, on vous propose d'implémenter un algorithme de recadrage "intelligent" d'images appelé "Seam Carving"[1]. L'objectif pédagogique est de vous faire écrire des algorithmes simples pour la manipulation de tableaux dans le cadre d'une application réelle. Vous apprendrez également à utiliser une librairie externe et des pointeurs de fonction.

Ce devoir est à réaliser **seul**. La date limite de remise est précisée sur Ecampus et sur la plateforme de soumission.

1 Seam Carving

L'algorithme de Seam Carving permet de réduire les dimensions d'une image de manière intelligente, c'est-à-dire sans perte d'information et sans déformation des objets présents dans l'image. Un exemple classique est montré à la figure 1.

1. `https://fr.wikipedia.org/wiki/Seam_carving`

FIGURE 1 – Une image à gauche et sa réduction par Seam Carving à droite (Source : wikipédia)

L'algorithme en lui-même prend en entrée une image de dimension $h \times w$ (h pixels de hauteur et w pixels de large) et un paramètre k, tel que $k < w$, et fournit en sortie une nouvelle image de dimension réduite $h \times (w - k)$.

Pour réduire la largeur de l'image de k pixels, l'algorithme répète l'opération suivante k fois :

— On détermine dans l'image un chemin allant du haut au bas de l'image, appelé couture ("seam" en anglais), minimisant l'énergie des pixels traversés (voir plus bas pour une définition de l'énergie d'un pixel). Une couture est définie par une suite de h abscisses i_l^*, avec $0 \leq i_l^* < w$, $0 \leq l < h$, telles que $|i_l^* - i_{l+1}^*| \leq 1$ pour tout $0 \leq l < h-1$. Cette dernière contrainte exprime que les pixels d'une couture doivent être connectés d'une ligne à l'autre (directement ou en diagonale).

— On crée une nouvelle image de dimension $h \times (w-1)$ en retirant de l'image originale les pixels (l, i_l^*) pour tout $l \in \{0, \ldots, h-1\}$.

L'énergie d'un pixel peut être défini de plusieurs manières. Si on note $I_{i,j,c}$ la valeur de la composante [2] c (où $c \in \{R, G, B\}$) du pixel à la position (i, j) de l'image [3], on définira dans ce devoir l'énergie du pixel (i, j) de la manière suivante :

$$E_{i,j} = \sum_{c \in \{R,G,B\}} E_{i,j,c} \tag{1}$$

avec

$$E_{i,j,c} = |I_{i,j-1,c} - I_{i,j+1,c}| + |I_{i-1,j,c} - I_{i-1,j+1,c}| \tag{2}$$

Si les coordonnées (i, j) sont telles qu'un ou plusieurs pixels voisins $(i-1, j)$, $(i+1, j)$, $(i, j-1)$ ou $(i, j+1)$ n'existent pas, ils seront remplacés dans l'expression (2) par le point (i, j) lui-même. Intuitivement, un pixel a une grande énergie si les pixels à sa droite et à sa gauche et/ou au-dessus et en dessous sont très différents. Dans ce cas, supprimer ce pixel risque d'entraîner une perte d'information.

2 Implémentation

On vous demande d'implémenter l'algorithme de Seam Carving dans le fichier `SeamCarving.c` fourni (et son entête `SeamCarving.h`). Pour représenter une image, on utilisera le type `PNMImage` défini dans la libraire `PNM.c/.h`, fournie également, décrite dans la section 2.2.

2. Dans le format RGB utilisé dans ce projet, la couleur d'un pixel d'une image est encodée par trois valeurs indiquant respectivement ses composantes rouge, verte et bleue, typiquement sur une échelle allant de 0 à 255. Voir par exemple `https://fr.wikipedia.org/wiki/Rouge_vert_bleu`.

3. Les coordonnées de l'image sont telles que la coordonnée $(0, 0)$ correspond au coin supérieur gauche et $(h-1, w-1)$ au coin inférieur droit pour une image de dimension $h \times w$.

2.1 Seam carving (fichiers `SeamCarving.c/h`)

Le fichier `SeamCarving.c` devra contenir le code des 4 fonctions suivantes :

`int computePixelEnergyGradient(PNMImage *img, int i, int j)` : Cette fonction doit renvoyer l'énergie à la position (i, j) dans l'image `img`.

`int *computeMinEnergySeam(PNMImage *img, int (*computePixelEnergy)(PNMImage *, int, int))` : Cette fonction prend comme argument une image et une fonction permettant de calculer l'énergie d'un pixel et renvoie un tableau d'entiers de la taille de la hauteur de l'image `img` contenant les abscisses des pixels appartenant à la couture d'énergie minimale dans l'image.

`PNMImage *cutImage(PNMImage *img, int *seam)` : Cette fonction crée une nouvelle image obtenue en retirant de l'image `img` les pixels dont l'abscisse est donnée par les positions dans le tableau `seam` donné en argument (qui est supposé être renvoyé par la fonction `computeMinEnergySeam`). Plus précisément pour tout $i \in \{0, \ldots, h-1\}$, le pixel $(i, \texttt{seam}[i])$ doit être retiré de l'image. Cette fonction ne doit pas modifier l'image `img` passée en argument.

`PNMImage *shrinkImage(PNMImage *img, int k)` : Cette fonction renvoie une réduction de l'image `img` donnée en argument de `k` pixels en largeur en utilisant la procédure décrite dans la section 1. La fonction doit créer et renvoyer une nouvelle image et ne doit pas modifier l'image `img` passée en argument.

Un fichier `main.c` vous est fourni pour tester votre implémentation. Via le fichier `Makefile` fourni également, ce fichier `main.c` crée un exécutable `SeamCarving` qui peut être utilisé pour réduire une image en ligne de commande. La commande suivante :

```
./SeamCarving input.pnm output.pnm k
```

réduira la largeur de l'image du fichier `input.pnm` de `k` pixels et mettra le résultat dans le nouveau fichier `output.pnm`. Trois images vous sont fournies pour faire des tests (`chateau.pnm`, `bd1.pnm`, et `bd2.pnm`). Vous pouvez créer d'autres images dans le bon format et visualiser les images générées en utilisant un outil de conversion d'images [4].

4. Le site suivant offre par exemple ce genre de service : `https://convertio.co`, ainsi que des logiciels tels que `Gimp` ou `Fiji`.

2.2 Gestion des images (librairie `PNM.c/h`)

Pour la gestion des images, une libraire rudimentaire vous est fournie dans les fichiers `PNM.c/h` qui permet de charger et d'enregistrer une image au format `pnm`, qui est un format non compressé d'image facile à manipuler. La structure d'image (non opaque) utilisée par ce fichier est la suivante :

```
typedef struct {
    unsigned char red, green, blue;
} PNMPixel;

typedef struct {
    size_t width;
    size_t height;
    PNMPixel* data;     // Pixel (i, j) is at position i * width + j
} PNMImage;
```

Le champ `data` est un tableau à une dimension qui concatène les pixels de l'image de gauche à droite et de haut en bas. Chaque pixel est une structure de type `PNMPixel` encodant les valeurs RGB des pixels dans trois champs `red`, `green` et `blue`. Par exemple, pour accéder à la valeur rouge au pixel $(4, 5)$ d'une image de taille 100×50 pointée par la variable `img` (notée $I_{4,5,R}$ ci-dessus), on utilisera l'expression `img->data[4*50+5].red`. Dans votre code, vous n'aurez besoin d'utiliser que les fonctions `createPNM` et `freePNM` permettant respectivement de créer une nouvelle image et de libérer la mémoire prise par une image.

3 Conseil d'implémentation

Commencez par implémenter les fonctions `computePixelEnergyGradient` et `cutImage` qui ne devraient pas vous poser de problème particulier.

Implémentez ensuite la fonction `computeMinEnergySeam` qui est la plus compliquée. Cette fonction comprend deux étapes.

La première étape consiste à remplir une table `M` de la même taille que l'image dont chaque élément (i, j) contiendra l'énergie minimale d'une couture partant de la première ligne et s'arrêtant au pixel (i, j). En négligeant les bords de l'image, la valeur `M`$[i][j]$ peut se calculer comme suit :

$$\texttt{M}[i][j] = E_{i,j} + \min\{\texttt{M}[i-1][j-1], \texttt{M}[i-1][j], \texttt{M}[i-1][j+1]\}.$$

A vous de trouver une manière de remplir la table pour que cette équation soit satisfaite pour tous ses éléments (aux bords près).

Une fois cette table calculée, vous pouvez obtenir un tableau correspondant à la couture d'énergie minimale de la manière suivante :

— Le pixel à supprimer sur la dernière ligne de l'image est le pixel j qui minimise `M`$[h-1][j]$ (vu la définition du tableau `M`, c'est en effet le dernier pixel de la couture d'énergie minimale arrivant au bas de l'image).
— Le pixel à la ligne juste au dessus qui appartient à la couture d'énergie minimale est celui parmi $(h-2, j-1)$, $(h-2, j)$ et $(h-2, j+1)$ qui correspond à la valeur minimale dans `M`.
— Le même principe peut être invoqué pour retrouver les abscisses de la couture aux lignes supérieures de l'image

Implémentez enfin la fonction `shrinkImage` qui permet de réduire la taille d'une image en appliquant k fois successivement les fonctions `computeMinEnergySeam` et `cutImage`. Cette fonction ne devrait pas vous poser trop de problème. Faites cependant attention à bien libérer la mémoire utilisée pour stocker les images intermédiaires (sans effacer l'image de départ).

Pour information, notre implémentation de référence comprend environ 160 lignes de code (35, 25, 75, et 25 lignes respectivement pour les fonctions `computePixelEnergyGradient`, `cutImage`, `computeMinEnergySeam`, et `shrinkImage`)

4 Soumission

Une archive `devoir.zip` doit être soumise sur la plateforme qui contiendra l'unique fichier `SeamCarving.c`. Cela signifie que votre code doit fonctionner avec les fichiers `SeamCarving.h`, `PNM.c/h`, et `main.c` que nous vous fournissons, sans aucune modification. Vérifiez que c'est bien le cas avant soumission.

Bon travail !