Seamcarving

Étienne Miquey
etienne.miquey@ens-lyon.fr

L'objectif de ce TP est de vous assurer que vous savez manipuler allègrement les tableaux, et de voir un joli exemple de programmation dynamique. Il est en théorie volontairement plus simple que certains de ses prédécesseurs, vous devriez donc tous aller jusqu'au bout de l'algorithme. Et que les très motivés ne s'inquiètent guère, la fin devrait pouvoir les occuper quelques temps.

1 Préliminaires

Le Seamcarving est un récent [1] algorithme de réduction d'image, dont l'objectif est de garder les proportions de l'objet tout en supprimant des informations peu utiles de l'image. Exemple par la mise en situation : vous désirez envoyer à votre grand-mère une superbe photo de la plage où vous passez vos vacances, mais celle-ci est trop grande pour l'enveloppe. Et vous n'avez aucune envie de la réduire, ni de couper le superbe palmier que l'on voit à droite, encore moins vos amis au milieu, par contre, la plage entre est assez inutilement large. On pourrait d'ailleurs même sûrement en enlever une bande sans que l'on s'en rende compte. C'est le principe du Seamcarving.

2 Énergie

Une image est une matrice de triplets (r, g, b), chacun représentant un pixel. L'idée va être de se donner une fonction d'énergie qui pour chaque pixel mesure sa ressemblance avec ses voisins. À la louche, un pixel bleu au milieu de la mer va être d'énergie faible, tandis qu'un pixel rouge au centre d'une pelouse va quant à lui avoir une forte énergie. On va ensuite chercher dans la matrice un chemin de moindre énergie, et le supprimer.

Pour travailler sur une image, nous aurons donc besoin de sa matrice de pixels, mais aussi de celle d'énergie, qui sera souvent définie à partir d'une tierce matrice, celle de luminosité. Enfin, comme l'on va réduire nos images, afin de le faire en place, on restera dans une matrice un peu trop grande, en transmettant l'information des dimensions. Une image sera définie donc par le type suivant :

```
type image = {
   img : (int*int*int) vect vect;
   energ : float vect vect;
   lum : float vect vect;
   mutable x : int;
   mutable y : int };;
```

Question 1. Sachant que la luminosité d'un pixel est donnée par $lum(r, g, b) = 0.3r + 0.6g + 0.1b$, écrire une fonction qui calcule la luminosité d'une image.

```
calcul_lum : image -> unit
```

Question 2. De même, écrire une fonction qui prendra en entrée une image et une fonction de calcul de l'énergie de type `image -> int -> int -> float`, et qui effectuera le calcul de la matrice d'énergie.

```
calcul_energie : image -> (image -> int -> int -> float) -> unit
```

Comme vous l'aurez compris, nous allons avoir besoin de supprimer des chemins dans l'image. Un chemin sera une liste de pixels traversant l'image de proche en proche :

```
type chemin == (int*int) list;;
```

Question 3. Écrire une fonction `reducev` (resp. `reduceh`) qui supprime un chemin vertical (resp. horizontal) d'une image. On prendra bien soin de modifier les dimensions de l'image.

```
reducev : image -> chemin -> unit
```

3 Chemin de moindre énergie

Toutes les questions dans cette section porteront sur des chemins verticaux, charge à vous de faire les modifications nécessaires pour les chemins horizontaux.

1. *Vous pouvez consulter la page wikipedia à ce sujet, ou le papier originel :* Seam Carving for Content-Aware Image Resizing, Shai Avidan, Ariel Shamir, 2007.

On sait qu'un chemin vertical doit nécessairement traverser l'image de haut en bas, on choisira de les construire en partant du haut, par simplicité. On notera $e(i,j)$ l'énergie au point (i,j), $c(i,j)$ le coût du chemin de moindre énergie arrivant en (i,j), et l'on notera n,p les dimensions de la matrice.

Question 4. Exprimer le coût du chemin vertical de moindre énergie en fonction des $c(i,j)$, n et p.

Question 5. En supposant connaître tous les $c(k,j)$ pour $0 \leq k < i$, $0 \leq j < p$, exprimer $c(i,j)$. Donner également les conditions au bord.

Question 6. En déduire une fonction `cheminv` qui calculera par programmation dynamique le chemin vertical de moindre énergie d'une image.

```
cheminv : image -> chemin
```

Question 7. Écrire une fonction `seamcarving` qui prendra en argument une image, le nombre de pixel dont veut la réduire en largeur et une fonction de calcul de l'énergie, et qui effectuera la réduction attendue.

```
seamcarving : image -> int -> (image -> int -> int -> float) -> image
```

Je vous fournis des fonctions pour l'ouverture et l'écriture d'image (au format *.ppm), je vous laisse regarder comment celles-ci fonctionnent pour pouvoir visualiser le résultat de votre fonction `seamcarving`. Vous pouvez notamment vous amuser à colorier en rouge le chemin que vous allez retirer, ou pousser à l'extrême une réduction jusqu'à déformer l'image. Et, cela va de soi, modifier votre fonction pour qu'elle gère aussi les réductions en hauteur.

4 Amélioration

L'un des objectifs assez courant de ce type d'algorithme est de pouvoir s'exécuter en temps réel, ce qui pousse à réfléchir à un certain nombre d'optimisations. Il est clair que si nous voulions vraiment être efficace, nous ne ferions pas du Caml, interprété qui plus est. Mais pour le principe, il est toujours intéressant d'y réfléchir un petit peu.

Vous l'aurez peut-être remarqué, mais à chaque étape, on est contraint de mettre à jour la matrice de luminosité donc d'énergie. Jusque là, nous le faisions par un nouveau calcul de l'ensemble de la matrice, ce qui n'est pas complétement nécessaire.

Question 8. Créer une fonction `calcul_next_energie_v` qui prendra en argument une image, un chemin et une fonction de calcul de l'énergie, et qui fera le moins de calcul possible pour mettre à jour la matrice d'énergie ; puis modifier en conséquence la fonction `reducev`.

```
calcul_next_energie_v : image -> chemin -> (image -> int -> int -> float) -> unit
```

Question 9 (Dure)**.** On pourrait aussi imaginer se servir du SeamCarving dans l'autre sens : agrandir une image en rajoutant de l'information non-significative. Pour agrandir d'un pixel, c'est facile, il suffit de chercher le chemin de moindre énergie, et de le dupliquer. Est-ce aussi évident pour agrandir de n pixels ? Auriez-vous une heuristique pour contourner le problème ? Si oui, modifiez votre fonction `seamcarving` pour qu'elle puisse aussi agrandir.