

## TD N° 7 : NANO PROJET : SEAM CARVING

**Buts :** Travail en binôme, Allocation dynamique, tableau multidimensionnel dynamique

Ce TD est un mini projet qui se réalise en binôme. Il a pour objectif de réduire de manière intelligente une image. C'est une version simplifiée des algorithmes intégrés à Photoshop ou GIMP. La réduction de taille se fait habituellement par un changement d'échelle linéaire. Tous les objets sont donc réduits de la même manière, qu'ils soient importants pour la perception humaine ou non. Le SeamCarving apporte une solution dans certains cas. Les images ci dessous illustrent cette application.

Pour simplifier, notre objectif est donc de supprimer  $m$  colonnes de l'image initiale.



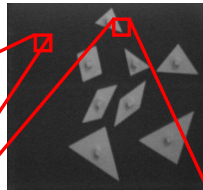
### 1. Quelques éléments sur l'image

Une image est une fonction  $I(x,y)$  à support fini (ses dimensions) et dont les valeurs sont ici des scalaires sur 8 bits : le noir correspond à une valeur nulle, le blanc à 255. En informatique, c'est simplement un **tableau d'octets non signés à 2 dimensions**.

Vous avez ci-contre deux tableaux représentant les valeurs de l'image du tangram sur un voisinage de  $6 \times 6$  pixels dans le fond et sur un sommet de triangle. Les valeurs de l'intensité ne sont pas uniformes et comportent un aléa dû au bruit d'acquisition.

50	49	52	52	46	46
54	51	58	54	49	42
45	48	44	46	48	48
44	54	42	49	58	47
48	50	48	53	50	42
45	48	46	54	58	53

132	147	114	74	62	45
119	129	125	87	58	48
77	93	106	88	57	55
55	61	70	73	61	52
50	51	49	56	55	56
55	50	54	59	53	55



### 2. Réduction linéaire de la dimension

Pour supprimer  $m$  colonnes d'une image de dimension  $n_l$  lignes et  $n_c$  colonnes, on réalise un zoom de cette image de rapport  $k=(n_c-m)/n_c$  sur les colonnes.

Pour calculer les valeurs de la nouvelle image  $im_{reduite}$  de dimension  $n_l$  lignes et  $n_c-m$  colonnes, on parcourt tous les pixels de cette image et on cherche leurs antécédents dans l'image d'origine  $im_{origine}$ . Le pixel de coordonnées  $i,j$  de l'image  $im_{reduite}$  est normalement le pixel de coordonnées  $ni=i, nj=1/k*(j-n_c/2)+n_c/2$ .

Malheureusement, la coordonnée  $nj$  est en général un réel et on ne peut accéder à la valeur de ce pixel dans l'image d'origine (tous les pixels ont des coordonnées entières).

La solution consiste à utiliser une interpolation : on calcule la valeur du pixel  $nj$  à partir de la valeur de ses voisins  $[nj]$ ,  $[nj+1]$ ,  $[nj]-1$  où  $[x]$  désigne la partie entière d'un réel. Plusieurs types d'interpolation existent dont :

- Interpolation d'ordre 0 ou plus proche voisin : le résultat est la valeur du pixel le plus proche de  $nj$

$$I_{reduite}(i,j) = I_{originale}(i, [nj+0.5]) \text{ avec } nj=1/k*(j-n_c/2)+n_c/2 \text{ et } k=(n_c-m)/n_c$$

- Interpolation d'ordre 1 ou linéaire : on considère que l'image est une fonction continue linéaire entre les points  $[nj]$ ,  $[nj+1]$  : la valeur de l'image en  $nj$  est alors :

$$I_{reduite}(i,j) = (1-nj+[nj])*I_{originale}(i, [nj]) + (nj-[nj])*I_{originale}(i, [nj]+1);$$

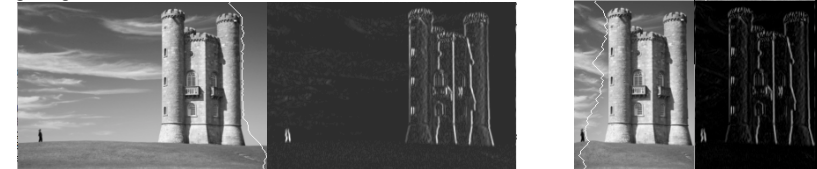
avec  $nj=1/k*(j-n_c/2)+n_c/2$  et  $k=(n_c-m)/n_c$

Attention : la notation  $[ ]$  indique ici les parties entières

### 3. SeamCarving

Le Seam Carving commence par rechercher le chemin le plus pertinent entre la première ligne de l'image et la dernière ligne de l'image. Un chemin ne comporte qu'un seul pixel par ligne et chaque pixel de ce chemin est voisin du précédent et du suivant. Autrement dit, si un pixel de coordonnées  $i,j$  appartient à un chemin, alors un et un seul des pixels de coordonnées  $(i-1,j-1)$ ,  $(i-1,j)$  ou  $(i-1,j+1)$  appartiendront à ce chemin. Cette pertinence est donnée sous forme d'un coût pour chaque pixel de l'image, et le coût est additionné le long des chemins. Quand le meilleur chemin est trouvé, on supprime chaque pixel appartenant au chemin de coordonnées  $i,j$  dans l'image en décalant la partie  $i, j+1$  de l'image d'un cran vers la gauche.

Voici deux exemples de chemin dont la suppression n'est pas importante du point de vue de la perception.



La pertinence ou coût est ici donnée par les contours verticaux de l'image.

#### 3.1. Coût ou énergie

De nombreuses fonctions de coût existent et peuvent être définies. La plus simple est la composante horizontale de la dérivée de l'image. Elle se calcule par différences finies :

$$\text{Energie}(i,j) = \text{abs}(I_m(i-1,j-1) - I_m(i-1,j+1)) + I_m(i,j-1) - I_m(i,j+1) + I_m(i+1,j-1) - I_m(i+1,j+1)) / 3$$

Sur les bords de l'image, cette fonction n'est pas définie, puisque certains pixels n'existent pas, comme le pixel  $-1,0$  par exemple. Les solutions sont :

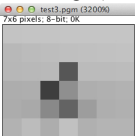
- soit prolonger l'image par continuité (les pixels de coordonnées  $-1,y$  sont identiques aux pixels de coordonnées  $0,y$ )
- soit périodiser l'image, ce qui consiste à répliquer l'image dans toutes les directions (droite, gauche, bas, haut, diagonales)

Il est plus facile d'utiliser ici la périodisation. Pour cela, il suffit d'utiliser la fonction modulo (symbole % en C) et de définir l'énergie par :

```
Energie(i,j) =
    abs(I_m((i-1+nl)%nl, (j-1+nc)%nc) - I_m((i-1+nl)%nl, (j+1)%nc)
    + I_m(i, (j-1+nc)%nc) - I_m(i, (j+1)%nc)
    + I_m((i+1)%nl, (j-1+nc)%nc) - I_m((i+1)%nl, (j+1)%nc))/3
```


Exemple :

Une image (test.pgm) et les valeurs numériques de cette image



195	196	196	197	197	197	197
184	187	189	191	192	193	194
183	184	185	86	187	187	188
173	174	62	143	174	176	178
172	169	137	99	157	174	176
178	188	173	193	176	180	180

Le gradient de cette image et les valeurs numériques de cette énergie



0	0	3	2	3	2	4
4	2	31	2	34	1	5
5	34	41	39	45	2	5
5	48	66	44	69	8	3
1	50	32	45	31	9	2
0	13	21	8	20	7	2

3.2. Calcul du chemin

3.2.1. Calcul du coût total de tous les chemins possibles

Le calcul du chemin se fait par programmation dynamique en remarquant que :

```
cout(i,j) = min( cout(i-1,j)+energie(i,j),
                cout(i-1,j+1)+energie(i,j),
                cout(i-1,j-1)+energie(i,j))
```

Par récurrence, on conçoit bien que partant de l'indice i=0 avec une valeur nulle, on peut calculer de ligne en ligne le coût final, chaque ligne ne dépendant que de la précédente. Le chemin pris pour atteindre le pixel i,j sera donc celui qui a donné le coût minimum parmi ses voisins de la ligne précédente (par exemple, celui qui vient du pixel i-1, j+1. Il faut alors stocker cette information dans un tableau pour pouvoir retrouver le chemin ensuite.

De ce fait, l'algorithme s'écrit de la manière suivante.

```
Initialiser le tableau cout avec une grande valeur (0xFFFFFFFF)
Initialiser la premiere ligne de cout avec l'energie de la premiere
ligne
Initialiser le tableau pere avec la valeur 0
Pour i=1 à nl-1 faire
    Pour j=0 à nc-1 faire
        Pour k=-1 à 1
            Si j+k>=0 && j+k<nc &&
                cout(i,j) > cout(i-1,j+k)+energie(i,j))
            alors
                cout(i,j) = cout(i-1,j+k)+energie(i,j)
                pere(i,j)=j+k;
            finsi
        finpour
    finpour
finpour
```

Exemple :

Les tableaux cout et pere pour l'image et l'énergie précédente

0	0	3	2	3	2	4
4	2	31	4	36	3	7
7	36	43	43	48	5	8
12	55	102	87	74	13	8
13	62	87	119	44	17	10
13	26	83	52	37	17	12

0	0	0	0	0	0	0
0	0	1	3	3	5	5
1	1	1	3	5	5	5
0	0	1	2	5	5	5
0	0	1	4	5	6	6
0	0	1	4	5	6	6

3.2.2. Calcul du meilleur chemin

Le tableau cout contient alors les coûts finaux, et en parcourant la dernière ligne de ce tableau, on connaît la valeur et le point final du chemin le moins coûteux.

Lorsque l'on connaît le point d'arrivée du chemin, il suffit de remonter à partir de ce point en suivant le tableau pere qui indique par quel pixel on est arrivé sur ce point.

L'algorithme pour retrouver le chemin est donc

```
k = indice de la colonne de valeur minimale de la ligne nl-1 de cout
chemin(nl-1)=k ;
Pour i=nl-2 à 0 faire
    chemin(i)= pere(i+1,chemin(i+1)) ;
finpour
```

Exemple :

Le tableau chemin pour les tableaux coût et pere précédents.

5	5	5	6	6	6
---	---	---	---	---	---

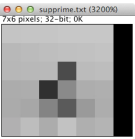
3.3. Suppression d'un chemin

Le tableau chemin contient alors les numéros de colonnes du chemin le moins coûteux entre la première ligne et la dernière ligne

Il reste maintenant à supprimer dans l'image les pixels de coordonnées i, chemin(i) en décalant les pixels à partir des coordonnées i, chemin(i)+1 d'une case vers la gauche. On met la dernière colonne à la valeur nulle (0).

Exemple :

L'image (test.pgm) après suppression du chemin précédent i.e. les pixels de coordonnées (5,6), (4,6), (3,6), (2,5), (1,5), (0,5);



195	196	196	197	197	197	0
184	187	189	191	192	194	0
183	184	185	86	187	188	0
173	174	62	143	174	176	0
172	169	137	99	157	174	0
178	188	173	193	176	180	0

3.4. Exemple complet

Voici 2 itérations des valeurs de du coût, des pères, du chemin et de l'image après suppression.

1ère itération = Image

195	196	196	197	197	197	197
184	187	189	191	192	193	194
183	184	185	86	187	187	188
173	174	62	143	174	176	178
172	169	137	99	157	174	176
178	188	173	193	176	180	180

Gradient

0	0	3	2	3	2	4
4	2	31	2	34	1	5
5	34	41	39	45	2	5
5	48	66	44	69	8	3
1	50	32	45	31	9	2
0	13	21	8	20	7	2

Coût						
0	0	3	2	3	2	4
4	2	31	4	36	3	7
7	36	43	43	48	5	8
12	55	102	87	74	13	8
13	62	87	119	44	17	10
13	26	83	52	37	17	12

0	0	0	0	0	0	0
0	0	1	3	3	5	5
1	1	1	3	5	5	5
0	0	1	2	5	5	5
0	0	1	4	5	6	6
0	0	1	4	5	6	6

Pères

Chemin

5	5	5	6	6	6
---	---	---	---	---	---

Image après suppression

195	196	196	197	197	197	0
184	187	189	191	192	194	0
183	184	185	86	187	188	0
173	174	62	143	174	176	0
172	169	137	99	157	174	0
178	188	173	193	176	180	0

2<sup>ème</sup> itération = Image

195	196	196	197	197	197
184	187	189	191	192	194
183	184	185	86	187	188
173	174	62	143	174	176
172	169	137	99	157	174
178	188	173	193	176	180

Gradient

0	0	3	2	3	2
4	2	31	2	35	4
4	34	41	39	46	4
3	48	66	44	70	3
0	50	32	45	31	5
0	13	21	8	20	5

Coût

0	0	3	2	3	2
4	2	31	4	37	6
6	36	43	43	50	10
9	54	102	87	80	13
9	59	86	125	44	18
9	22	80	52	38	23

Père

0	0	0	0	0	0
0	0	1	3	3	5
1	1	1	3	3	5
0	0	1	2	5	5
0	0	1	4	5	5
0	0	1	4	5	5

Chemin

Image

après

suppression

0	1	0	0	0	0
---	---	---	---	---	---

196	196	197	197	197	0
184	189	191	192	194	0
184	185	86	187	188	0
174	62	143	174	176	0
169	137	99	157	174	0
188	173	193	176	180	0

#### 4. Exemple de manipulation d'images en SDL

Toutes les fonctions qui effectuent les entrées sorties d'images dans un fichier ou à l'écran sont données. L'exemple lit et affiche une image, son inverse vidéo et sauve l'image dans un fichier.

```
#include <SDL_phelma.h>
#include <image.h>

int main(int a, char** b) { int i,j,nbligne,nbcol;
    unsigned char** im=NULL; // Le tableau image
    SDL_Surface* fenetre=NULL;

    /* Creation d'une fenetre 640 x 480, couleurs sur 32 bits */
    fenetre=newfenetregraphiqe (640, 480);

    /* Lecture du fichier image pgm, en niveau de gris, sur 8 bits */
```

```
im = lectureimage8("lena2.pgm",&nbligne,&nbcol);

/* On place cette image dans la fenetre, en position 10 20
Attention : la fenetre doit pouvoir contenir l'image */
afficheim8SDL(fenetre,im,nbligne,nbcol,10,20);

/* On parcourt toute l'image et on inverse chaque pixel */
for (i=0; i<nbligne; i++) for (j=0; j<nbcol; j++) im[i][j]=255-im[i][j];

/* On affiche cette nouvelle image sur l'ecran, en position 350 150 */
afficheim8SDL(fenetre,im,nbligne,nbcol,350,150);
puts("Taper pour continuer"); getchar();

/*On sauve l'image dans un fichier nommé resultat.pgm
ecritureimagepgm("resultat.pgm",im,nbligne,nbcol) ;
}
```

Si votre programme est écrit avec les fichiers p.c, fonction1.c fonction2.c, le fichier Makefile est le suivant :

```
DIRSDL=/users/progla/C/librairie/2011
CFLAGS=-g -c -I$(DIRSDL)/include
LDFLAGS= -L$(DIRSDL)/lib -lSDLmain -lSDL -lSDL_image -lSDL_phelma
OBJETS= p.o fonction1.o fonction2.o
p: $(OBJETS)
    gcc -o $@ $^ $(LDFLAGS)
%.o: %.c
    gcc $(CFLAGS) $<
```

#### 5. Travail à réaliser

En suivant une démarche d'analyse descendante, on décompose le problème en plusieurs fonctions simples à coder (5 à 10 lignes de code C). Ce projet se fait sur 2 séances avec un peu de travail en dehors des séances encadrées. En particulier, il faut avoir bien réfléchi et écrit le code sur machine avant les séances. Compiler et tester les fonctions les unes après les autres (développement incrémental).

**Remarque importante** : les fonctions font appel au voisinage d'un point de coordonnées i,j. Il faut respecter les limites des tableaux, i.e. indices **positifs** et **strictement plus petits** que nl ou nc.

1. Faire les fonctions de création et de libération d'images et de tableau de coût et pere sous forme de tableau à deux dimensions, alloués dynamiquement, d'octets ou d'entiers. La zone de données sera contiguë. Les entêtes des fonctions sont

```
unsigned char ** alloue_image_char(int nl, int nc);
unsigned int ** alloue_image_int (int nl, int nc) ;
void libere_image(unsigned int ** im) ;
```

Ces fonctions seront écrites dans le fichier allocation.c et testées avec le programme allocation\_test.exe du fichier allocation\_test.c. Il sera obtenu par la commande make allocation\_test.exe
2. Faire une fonction qui calcule l'énergie de l'image. Cette fonction calcule l'énergie de l'image im. Si le paramètre energ est NULL, la fonction réalise une allocation dynamique. La fonction retourne ce tableau 2D. L'entête de la fonction sera:

```
unsigned char ** gradienty(unsigned char ** energ, unsigned char ** im, int nl, int nc).
```

Cette fonction sera écrite dans le fichier energie.c et testée avec le programme energie\_test.exe du fichier energie\_test.c. Il sera obtenu par la commande make energie\_test.exe

3. Faire une fonction (fichier `calcul_cout.c`) qui calcule le coût final (tableau 1D : coût entre la première ligne et la dernière ligne pour chaque colonne) et les pères (tableau 2D). Elle utilise un tableau intermédiaire 2D permettant de calculer le coût par récurrence (algorithme § 3.2.1). L'entête de la fonction sera `void calcul_cout(unsigned char** energie, unsigned int** pere, unsigned int* cout_final, int nl, int nc);`
4. Faire une fonction (fichier `trouve_chemin.c`) qui calcule le chemin à partir des tableaux `coutfinal` et `pere`. Elle trouve l'indice de la colonne la moins coûteuse de la dernière ligne. Elle calcule le chemin (algorithme § 3.2.2) et retourne l'indice de la colonne de coût minimum sur la dernière ligne. L'entête de la fonction sera `int trouve_chemin(unsigned int* chemin, unsigned int** pere, unsigned int* coutfinal, int nl, int nc)`
5. Faire une fonction (fichier `supprime_colonne.c`) qui supprime un chemin dans l'image. Le nombre de colonne est décrémenté. L'entête de la fonction sera `void supprime_colonne(unsigned char** im, unsigned int* chemin, int nl, int* pnc)`
6. Faire une fonction (fichier `seam_carving.c`) qui réalise le SeamCarving d'une image. Cette fonction alloue les tableaux 1D (`coutfinal`, `chemin`) et 2D (`pere`, `energie`) utiles. Elle calcule l'énergie de l'image, puis le coût et les pères. Elle détermine le meilleur chemin, et supprime ce chemin de l'image. Cette opération est réalisée `nbcol` fois. L'image est modifiée à chaque itération. L'entête de la fonction sera `unsigned char** seam_carving(unsigned char** im, int nbcol, int nl, int nc)`
7. Faire une fonction (fichier `zoomx.c`) qui réalise le zoom d'une image. L'entête de la fonction sera `unsigned char** zoomx(unsigned char** im, int nbcol, int nl, int nc)`
8. Faire le programme principal (fichier `prog.c`) qui réalise les actions suivantes :
  - a. lit l'image
  - b. réalise le zoom linéaire
  - c. réalise la réduction par SeamCarving
  - d. Affiche les 3 images dans une fenetre

Une version de chaque fonction est disponible dans un fichier objet correspondant. Vous pouvez donc construire le programme final à l'aide de nos fonctions même si toutes vos fonctions ne sont pas terminées. Il faut simplement télécharger les fichiers objet et réaliser l'édition des liens à l'aide de la commande unix `make`. L'objectif est bien sur de remplacer ces fichiers objets par vos versions, obtenue en compilant vos propres fonctions écrites en C.

Nous proposons aussi une version `demo_seam` qui affiche à chaque itération le chemin à supprimer. Elle utilise une fonction `seam_carving_graphique` qui effectue exactement le même travail que la fonction `seam_carving` utilisant les fonctions 1 à 5, mais qui construit et affiche en plus une image contenant le chemin en blanc (fonction `trace_chemin`), ainsi que l'énergie. Ce programme est obtenu à l'aide de la commande `make demo_seam`.