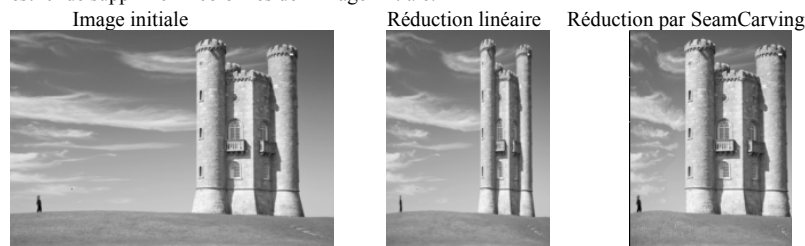


NANO PROJET : SEAM CARVING

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 la taille d'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. Il supprime non pas une colonne, mais un chemin prenant en compte les informations de l'image. Par exemple, sur l'exemple ci dessous, la partie à droite de la tour ne contient presque aucune information et cette partie sera supprimée dans les premières itérations du SeamCarving. Pour simplifier, notre objectif est ici de supprimer m colonnes de l'image initiale.



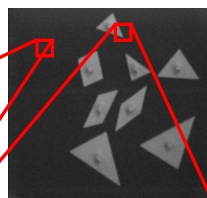
1. Quelques éléments sur l'image

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 6x6 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

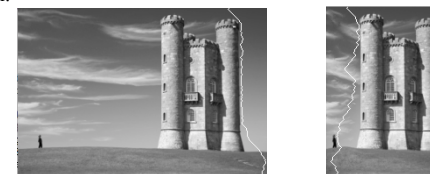


2. SeamCarving

Le Seam Carving commence par rechercher le chemin le plus pertinent à supprimer i.e. le moins coûteux pour aller d'un pixel de la première ligne de l'image jusqu'à un pixel de la dernière ligne de

l'image. Un chemin est une suite de pixels voisins les uns des autres et menant de la première ligne à la dernière ligne. Un chemin comporte un et un seul pixel par ligne et chaque pixel de ce chemin est voisin du précédent et du suivant : 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. La pertinence ou coût d'un chemin est la somme des coûts le long de ce chemin. Le coût d'un pixel est ici donné par son gradient horizontal, qui est la dérivée première de la l'intensité de l'image et indique la présence de contours. Le chemin recherché est celui qui présente la plus faible de variation d'intensité. Quand le meilleur chemin (le plus pertinent i.e. le moins coûteux) est trouvé, on supprime chaque pixel de coordonnées i, j appartenant au chemin en décalant la partie $i, j+1$ de l'image d'un cran vers la gauche.

Voici deux exemples de chemin (tracés en blanc) dont la suppression n'est pas importante du point de vue de la perception.



Remarque : dans toute la suite de l'énoncé, on considère que l'image est un tableau 2D d'octets (unsigned char) de taille n_l lignes et n_c colonnes.

2.1. Cout ou énergie

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

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

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. Une solution consiste à périodiser l'image : il faut dupliquer (virtuellement) l'image dans toutes les directions (droite, gauche, bas, haut, diagonales). Par exemple, les pixels dupliqués à droite en i, n_l sont les pixels du début de la ligne en $i, 0$.

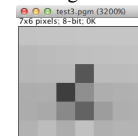
Pour cela, il suffit d'utiliser la fonction modulo (symbole % en C) et de définir l'énergie par :

$$\text{Energie}(i, j) = \text{abs}(I_m(i, (j-1+n_c)\%n_c) - I_m(i, (j+1)\%n_c)) ;$$

Energie est un tableau 2D d'octets de n_l lignes et n_c colonnes

Exemple :

L'image test.pgm fournie sur le site 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



1	1	1	1	0	0	2
7	5	4	3	2	2	9
4	2	98	2	101	1	4
4	111	31	112	33	4	3
7	35	70	20	75	19	2
8	5	5	3	13	4	2

2.2. Calcul du chemin

2.2.1. Calcul du coût total des chemins possibles entre la première et la dernière ligne.

Le calcul du meilleur chemin entre un des pixels de la première ligne et un des pixels de la dernière ligne se fait par programmation dynamique en remarquant que :

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

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 la valeur 0
Initialiser le tableau pere avec la valeur 0
Initialiser la premiere ligne de pere avec la valeur -1
  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-1,j+k)
        alors
          cout(i,j) = cout(i-1,j+k)+energie(i-1,j+k)
          pere(i,j)=j+k;
        finsi
      finpour
    finpour
  finpour
```

Le tableau `cout` est un tableau 2D d'entiers de nl lignes et nc colonnes et le tableau `pere` est un tableau 2D d'entiers de nl lignes et nc colonnes.

Exemple :

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

Tableau cout

Tableau pere							
0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0
6	5	3	2	2	2	2	2
7	7	4	4	3	3	3	3
11	11	35	35	7	6	6	6
18	18	46	55	25	8	8	8

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

2.2.2. Calcul du meilleur chemin

La dernière ligne du tableau `cout` contient alors les coûts finaux, et en parcourant cette dernière ligne de ce tableau, on trouve la valeur et le point d'arrivée du chemin le moins coûteux entre la première ligne et la dernière ligne.

Lorsque l'on connaît le point d'arrivée du chemin, pour construire le chemin recherché, 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.

Le tableau `chemin` est un tableau 1D d'entiers de dimension nl .

L'algorithme pour retrouver le chemin est donc

```
k = indice de la colonne de valeur minimale de la ligne nl-1 de du
tableau cout : c'est l'arrivée du plus court chemin
chemin(nl-1)=k ;
Pour i=nl-2 à 0 faire
  chemin(i)= pere(i+1,chemin(i+1)) ;
finpour
```

4
4
5
6
6
5

Exemple : Le tableau `chemin` pour les tableaux coût et père précédents.

2.3. 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). Pour simplifier, on ne changera pas la taille de la variable `C` contenant l'image, mais on ne considérera plus ces pixels pour la suite des traitements (suppression éventuelle des autres chemins).

Exemple :

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



2.4. Exemple complet

Voici 2 itérations des valeurs de l'image, de l'énergie, du coût, des pères, du chemin et de l'image après suppression. Vous pouvez les retrouver en examinant les fichiers `gradient.txt`, `pere.txt` ou `cout.txt` de la commande `demo_seam` (voir fin du sujet)

1^{ère} 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

1	1	1	1	0	0	2
7	5	4	3	2	2	9
4	2	98	2	101	1	4
4	111	31	112	33	4	3
7	35	70	20	75	19	2
8	5	5	3	13	4	2

Coût

0	0	0	0	0	0	0
1	1	1	0	0	0	0
6	5	3	2	2	2	2
7	7	4	4	3	3	3
11	11	35	35	7	6	6
18	18	46	55	25	8	8

Pères

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

Chemin

4
4
5
6
6
5

Image après suppression

195	196	196	197	197	197	0
184	187	189	191	193	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^{ème} itération = Image

195	196	196	197	197	197
184	187	189	191	193	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

Coût

0	0	0	0	0	0
1	1	1	0	0	0
6	5	4	3	3	3
7	7	5	5	5	7
9	9	36	36	8	8
14	14	44	56	23	23

Chemin

1
2
1
0
0
0

Gradient

1	1	1	1	0	2
7	5	4	4	3	9
4	2	98	2	102	4
2	111	31	112	33	1
5	35	70	20	75	15
8	5	5	3	13	2

Père

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

Image après suppression

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

3. Exemple de manipulation d'images et bibliothèque SDL

Toutes les fonctions qui effectuent les entrées sorties d'images dans un fichier ou à l'écran sont données. L'exemple ci dessous lit et affiche une image, son inverse vidéo et sauve l'image dans un fichier. Les images sont donc simplement des tableaux 2D alloués dynamiquement et déclarés sous la forme de unsigned char** im.

```
#include <SDL_phelma.h>

int main(int a, char** b) { // a : nombre de mots sur la ligne de commande
    // b : tableau des mots de la ligne de commande

    int i,j,nbligne,nbcol;
    unsigned char** im=NULL; // Le tableau image
    SDL_Surface* fenetre=NULL;

    /* Creation d'une fenetre couleur 480 lignes x 640 colonnes */
    fenetre=newfenetregraphique (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 ligne 10, colonne 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, en ligne 350, colonne 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$(DIRSDL)/include -I$(DIRSDL)/include/SDL
LDLFLAGS= -L$(DIRSDL)/lib -lSDLmain -lSDL -lSDL_image -lSDL_phelma
OBJETS= p.o fonction1.o fonction2.o
p: $(OBJETS)
    gcc -o $@ $^ $(LDLFLAGS)
%.o: %.c
    gcc $(CFLAGS) $<
```

4. 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). Répartissez dès le début du projet le travail entre membres du binôme pour plus d'efficacité.

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 et stocke le résultat dans le paramètre energ. Si ce paramètre energ est NULL, la fonction réalise une allocation dynamique. La fonction retourne l'adresse du tableau 2D energ. 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 cout_final: coût entre la première ligne et la dernière ligne pour chaque colonne) et les pères (tableau 2D pere). Elle utilise un tableau intermédiaire 2D cout alloué dynamiquement permettant de calculer le coût par récurrence (algorithme §2.2.1). L'entête de la fonction sera void calcul_cout(unsigned int* cout_final, unsigned char** energie, unsigned int** pere, 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 d'abord l'indice de la colonne la moins coûteuse de la dernière ligne. Elle retrouve le chemin (algorithme §2.2.2) entre ce pixel

de la dernière ligne et la première ligne puis le stocke dans le tableau 1D `chemin`. L'entête de la fonction sera `void trouve_chemin(unsigned int* chemin, unsigned int** pere, unsigned int* coutfinal, int nl, int nc)`.

Pour visualiser le chemin si vous en avez besoin, nous fournissons (fichier `trace_chemin.c`) la fonction `trace_chemin` qui dessine avec la couleur `cl` le chemin indiqué par `ch` dans l'image `im`. Cette image est créée dynamiquement si le paramètre est `NULL` à l'appel, et cette image est retournée par la fonction. L'entête de la fonction est `unsigned char** trace_chemin(unsigned char** im, unsigned int* ch, int cl, int nl, int nc)`

5. Faire une fonction (fichier `supprime_colonne.c`) qui supprime un chemin dans l'image. L'entête de la fonction sera `void supprime_colonne(unsigned char** im, unsigned int* chemin, int nl, int nc)`
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 `void seam_carving(unsigned char** im, int nbcol, int nl, int nc)`
7. Télécharger le programme principal (fichier `prog.c`) et le zoom (fichier `zoomx.c`) qui réalise les actions suivantes (télécharger le fichier `Makefile` et compiler avec `make prog`)
 - a. lit l'image
 - b. réalise la réduction par zoom linéaire
 - c. réalise la réduction par SeamCarving
 - d. sauvegarde le résultat du SeamCarving dans un fichier au format `pgm`
 - e. Affiche l'image initiale et les images obtenues par réduction linéaire et par SeamCarving dans une fenêtre si on souhaite le mode graphique.

5. Code fourni

Nous fournissons 2 programmes exécutables `demo_seam` et `prog`, les fichiers sources `trace_chemin.c`, `zoomx.c` et `prog.c` ainsi que le fichier `Makefile`.

Le fichier `zoomx.c` contient la fonction `zoomx_image` qui réalise la réduction linéaire de l'image `im`. Cette fonction est utilisée par le programme `prog`.

Le fichier `prog.c` contient le programme principal utilisant les fonctions `zoomx_image` et `seam_carving` et réalisant les actions décrites au dessus en §4.7.

Le fichier `trace_chemin.c` contient la fonction `trace_chemin` qui dessine avec la couleur `cl` le chemin indiqué par le tableau 1D `ch` dans l'image `im`.

Le fichier `Makefile` sert à compiler le fichier `prog.c` pour obtenir les programmes exécutables :

1. `allocation_test.exe` avec la commande `make allocation_test.exe`
2. `energie_test.exe` avec la commande `make energie_test.exe`
3. `prog` avec la commande `make prog`

2 fichiers exécutables sont déjà disponibles sous forme de commandes unix : `demo_seam` et `prog_seam`

Nous proposons une version `demo_seam` qui affiche à chaque itération le chemin à supprimer en blanc ainsi que l'énergie comme sur la figure ci contre. Il s'utilise sous la forme :

```
/users/progla/C/librairie/projetS12017/demo_seam
image_initiale.pgm          image_reduite.pgm
nb_de_colonnes_a_supprimer.
```




Il affiche une fenêtre contenant l'image initiale avec le chemin à supprimer en blanc ainsi que l'image d'énergie.

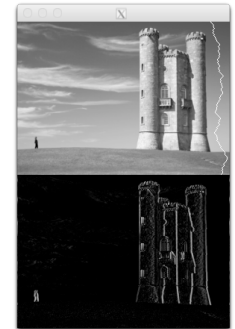
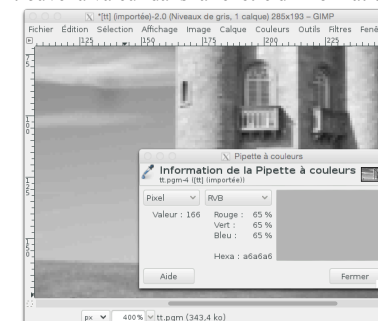
Il crée aussi les fichiers texte et image `gradient.txt`, `gradient.pgm`, `cout.txt`, `cout.pgm`, `pere.txt`, `pere.pgm` `chemin.txt` correspondant à la dernière suppression de colonnes.

Nous proposons une version `prog_seam` qui affiche l'image réduite linéairement, l'image réduite par seamcarving et l'image initiale.

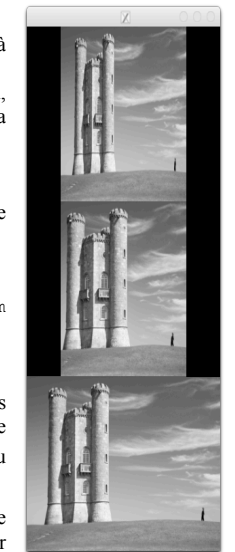
Il s'utilise sous la forme :

```
/users/progla/C/librairie/projetS12017/prog_seam
image_initiale.pgm          image_reduite.pgm
nb_de_colonnes_a_supprimer.
```

Pour visualiser la valeur des pixels d'une image, vous pouvez utiliser le logiciel `gimp` et l'outil pipette . Ouvrir le fichier image, sélectionner l'outil pipette  soit dans le menu Outils/pipette à couleur, soit dans le panneau boîte à outils, icône  et cocher 'Utiliser la fenêtre d'information' dans ce panneau. Le pointeur de souris devient une icône représentant la pipette pour désigner un pixel dont on trouve la valeur dans la fenêtre d'information.



Programme `demo_seam`



Programme `prog_seam`