TD N° 7 : NANO PROJET : INPAINTING

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 restaurer les trous dans une image dégradée. C'est une version simplifiée des algorithmes intégrés à Photoshop. Les images ci dessous illustrent cette application.

Image dégradée

Images des trous

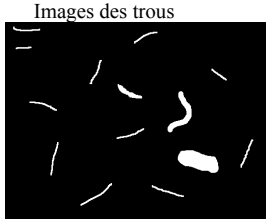
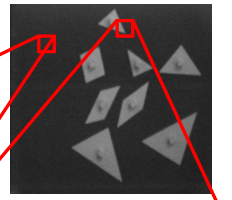


Image restaurée

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 u**n 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. Inpainting

On travaille ici avec une image dégradée comportant des zones dégradées, et une image de trous indiquant si le pixel est une zone dégradée ou non. Il existe de nombreuses méthodes pour extrapoler les zones manquantes. Nous utiliserons les méthodes à base de patchs. Un patch est simplement une partie de l'image de petites dimensions. Nous utiliserons des patchs carrés. Le patch de dimension n, situé en x,y est la partie de l'image dont les coordonnées sont [x-n,x+n] x [y-n,y+n]. Il sera noté par la suite P(x,y).

L'idée des méthodes par patchs pour l'inpainting est de combler les trous en partant du bord du trou. On recherche pour chaque pixel (x,y) sur le bord du trou, le patch P(xopt,yopt) de l'image qui ressemble le plus au patch P(x,y), en excluant bien sûr les pixels manquants de la comparaison.

Quand on a trouvé le patch le plus ressemblant, on recopie en x,y la valeur centrale I(xopt,yopt) de ce patch P(xopt,yopt). On recommence tant qu'il reste des pixels dans les trous.

Les problèmes soulevés par cette méthode sont nombreux, dont les suivants sont les plus difficiles :

- La ressemblance entre 2 patchs doit refléter la perception que nous avons de la « ressemblance » de 2 images
- le coût de calcul du meilleur patch : il faudrait passer en revue TOUS les patchs possibles. Pour une image NxM pixels, avec 10% de trous, cela représente 0.9xNxMxNxM. Ce qui est rapidement impossible.
- L'œil n'aime pas les variations brusques de l'intensité et il ne faut pas introduire des variations de ce type lorsque l'on copie le patch.

3. Distance

La distance entre le patch P(x,y) et le patch P(u,v) est simplement l'écart quadratique moyen ie

$$soit\ V(x,y,u,v) = \{(dx,dy)\,/\,dx \in [-n,n]\ et\ dy \in [-n,n]\ et$$

$$P(x+dx,y+dy) \notin trous\ et\ P(u+dx,v+dy) \notin trous$$

$$d^2(P(x,y),P(u,v)) = \frac{\sum_{(dx,dy)\in V(x,y,u,v)} \left(P(x+dx,y+dy) - P(u+dx,v+dy)\right)^2}{card(V(x,y,u,v))}$$

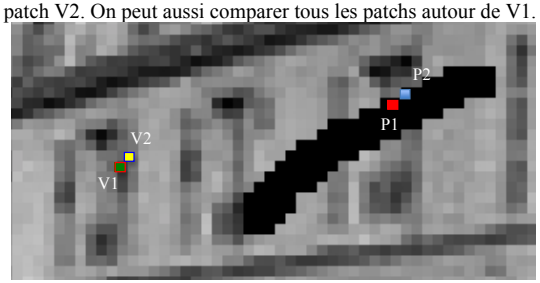
Remarque : Card(V(x,y,u,v)) est égal à$(2n+1)^2$, nombre de pixels du patch – le nombre de pixels appartenant aux trous d'un des 2 patchs P(x,y) ou P(u,v).

4. Trouver le meilleur patch et optimiser le tableau PPV

Pour trouver puis utiliser le patch P(u,v) le plus ressemblant au patch P(x,y), il faut conserver sa position (u,v) et sa distance d au patch P(x,y). On utilise donc un tableau PPV(x,y) qui contient deux entiers u,v représentant la position du meilleur patch actuel, et un réel d qui est la distance entre le patch P(u,v) et le patch P(x,y). C'est le calcul de ce tableau qui est très coûteux.

Plutôt que de chercher dans toute l'image la bonne position u,v, on utilise la méthode suivante , qui met à jour le tableau PPV(x,y) en 2 étapes:

1. On teste au hasard H (ici, H=100) patchs P(u,v) dans l'image. Les coordonnées (u,v) et la distance d du meilleur patch sont stockés dans PPV(x,y).
2. Pour augmenter les chances de trouver un patch ressemblant, on utilise le fait qu'une image est une représentation redondante du monde. Pour 2 points voisins, il y a de fortes chances pour que les patchs leur ressemblant le plus soient aussi voisins. Dans l'exemple ci dessous, si V1 est le patch le plus ressemblant de P1, quand on cherche le plus ressemblant à P2, on regarde le patch V2. On peut aussi comparer tous les patchs autour de V1.



Formellement, lorsque l'on cherche le patch le plus ressemblant à P(x,y), si P(u,v) est le patch ressemblant le plus à P(x+dx,y+dy) avec –1<=dx<=1, -1<=dy<=1, on regarde si P(u-dx,v-dy) est ressemblant à P(x,y). Si la distance d(P(u-dx,v-dy), P(x,y)) est plus petite que

la meilleure actuelle (PPV(x,y).d), alors on stocke les coordonnées (u-dx,v-dy) et la distance dans PPV(x,y).

Ces 2 étapes peuvent être répétées nbiter fois, où nbiter est un paramètre fixé par l'utilisateur, afin d'augmenter la probabilité de trouver un bon patch.

5. Initialiser le tableau PPV

L'initialisation de ce tableau est faite au hasard, car aucun patch n'a encore de meilleur ressemblant. Tous les points I(x,y) qui ne sont pas des trous sont eux mêmes leur plus ressemblant. Pour ces points, le tableau PPV(x,y) est initialisé avec leurs propres coordonnées et une distance nulle. Pour les points à l'intérieur des trous, le tableau PPV(x,y) est initialisé avec leurs propres coordonnées et une distance DMAX très grande.

6. Bords d'un trou

Le bord d'un trou est simplement un point de coordonnées x,y qui est un trou et dont au moins un des voisins de coordonnées x+/-1, y+/-1 n'est pas un trou.

7. Mise à jour de l'image & Inpainting complet de l'image

La mise à jour de l'image consiste à simplement affecter à chaque pixel (x,y) du trou la valeur I(u,v) du centre du patch P(u,v) qui ressemble le plus au patch P(x,y). Ce pixel (u,v) est donné par le tableau PPV(x,y). On commence par traiter les pixels du bord du trou. Et on répète l'opération tant qu'il reste que les trous ne sont pas bouchés.

L'image dégradée est donnée par I(x,y), les trous sont donnés par l'image trous(x,y), la taille des patch est un paramètre n. Les 2 images sont des matrices d'octets de dimensions nl lignes, nc colonnes.
L'inpainting de l'image réalise les actions suivantes :

- Création du tableau PPV de dimensions nl, nc
- Création d'une image résultat de dimensions nl, nc
- Initialiser les distances et les coordonnées dans le tableau PPV, en tirant au hasard les coordonnées des meilleurs patchs pour tous les points dans l'image trous, les autres ayant une distance nulle.
- Tant que l'image trous n'est pas vide faire
  - Trouver les bords des trous
  - Répéter nbiter fois
    - Trouver, pour les bords des trous, les meilleurs patchs en optimisant le tableau PPV
  - Remplacer les pixels du bord des trous par la valeur de leur PPV
  - Mettre à jour l'image des trous, car les pixels que l'on vient de remplacer ne sont plus des trous

8. 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/SDL.h>
#include <SDL/SDL_image.h>
#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 inpaint.c, fonction1.c fonction2.c, le fichier Makefile est le suivant :

```
DIRSDL=/users/prog1a/C/librairie/2011
CFLAGS=-g -c -O2 -I$(DIRSDL)/include -Iinclude
LDFLAGS= -L$(DIRSDL)/lib -L/usr/local/lib -lSDLmain -lSDL -lSDL_image -
lSDL_phelma
OBJETS= inpaint.o fonction1.o fonction2.o
inpaint: $(OBJETS)
      gcc -o $@ $^ $(LDFLAGS)
%.o: %.c
      gcc $(CFLAGS) $<
```

9. Travail à réaliser

En suivant une démarche d'analyse descendante, on décompose le problème en plusieurs fonctions simples à coder (5 à 20 lignes de code C au maximum). Ce nano projet se fait sur 2 séances. Il demande 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. Pensez à compiler et tester les fonctions les unes après les autres (développement incrémental).

1. Faire les fonctions de création d'images sous forme d'un tableau à deux dimensions, alloué dynamiquement. La zone de données sera contiguë ainsi que celle de création de PPV. Les entêtes des fonctions sont
   `unsigned char ** alloue_image(int nl, int nc);`
   `PPV ** alloue_champPPV(int nl, int nc)`
2. Faire une fonction qui calcule la distance entre le patch en position `is, js` et celui en position `it,jt` de taille `taillepatch`. Cette fonction retourne une constante très grande `DMAX` si le pixel `it,jt` est un trou ou si Card(V(x,y,u,v)) est trop petit. L'entête de la fonction sera `unsigned int distance( PIXEL** im, PIXEL** trous, int is, int js, int it, int jt, int taillepatch, int nl, int nc)`
3. Faire une fonction initialise le tableau PPV (cf §5). L'entête de la fonction sera `void initialisePPV(PPV** champ, PIXEL** im, PIXEL** trous, int taille, int nl, int nc);`
4. Faire une fonction qui calcule le patch le plus ressemblant à celui en position `is,js` selon le §4. L'entête de la fonction sera `void trouvemeilleurPPV(PPV** champ, PIXEL** im, PIXEL** trous, int is, int js, int taillepatch, int nl, int nc)`

5. Faire une fonction qui optimise tout le tableau PPV pour tous les pixels de l'image quand c'est possible (ie la meilleure distance n'est pas nulle). L'entête de la fonction sera `void optimisePPV(PPV** champ, PIXEL** im, PIXEL** zone, int nb, int taillepatch, int nl, int nc)`
6. Faire une fonction qui réalise l'inpainting: elle réalise les opérations du §6. L'entête de la fonction sera `unsigned char** inpaint(unsigned char** im, unsigned char** trous, int taillepatch, int nbiter, int nl, int nc)`
7. Faire le programme principal qui réalise les actions suivantes :
    a. lit l'image dégradée et l'image des trous
    b. réalise l'inpainting de l'image dégradée
    c. sauve la nouvelle image dans un fichier

Nous vous proposons plusieurs séries d'images.
Les fichiers 'trousxxxx.pgm' sont des fichiers indiquant où sont les trous dans l'image. Un pixel est à 0 (zéro) s'il n'est pas un trou, à 255 sinon
Les fichiers 'trousetxxx.pgm' sont les images dégradées
Les images xxxx.pgm sont les images non dégradées lorsqu'elles sont disponibles.