Projet de traitement d'images en C

Encadrant: Vincent Rodin (vincent.rodin@univ-brest.fr)

Un peu de traitement d'images très simple

Objectifs du projet :

- Initiation au traitement d'images par ordinateur
- Manipulation de tableaux bi-dimensionnels
- Passage en parametre de tableaux bi-dimensionnels
- En bref... du C!

1 Du Traitement d'Images très simple

Une image en niveaux de gris est un tableau bi-dimensionnel où chaque case contient une valeur comprise entre `0` et `255`. La valeur `0` représente le noir et la valeur `255` représente le blanc. Les valeurs intermédiaires correspondent à différentes nuances de gris.

Figure 1: Exemple d'image en niveaux de gris : Lena

Nous nous proposons ici de programmer quelques algorithmes très simples de Traitement d'Images comme :

- Calcul de convolution (Application de masques sur une image)

  $\Longrightarrow$ Image des gradients (Mise en évidence des contours présents dans une image)
- Calcul de l'histogramme d'une image

  $\Longrightarrow$ Pour chaque valeur de gris (de `0` à `255`) quelle est l'occurrence de cette valeur dans l'image?
- Binarisation (Seuillage de l'image des gradients)

  $\Longrightarrow$ Séparation du fond et des contours.
- Erosion et Dilatation pour supprimer les points de contours isolés ou combler les trous dans un contour.
- Amincissement des contours

  $\Longrightarrow$ Contours d'épaisseur 1.
- Et bien d'autres choses encore comme la segmentation en régions, ...

2 Exemples de résultats

Figure 2: Lena après un Sobel de taille 3 (image de gradients)

Figure 3: Lena après un seuillage (seuil=30) de l'image de gradients

Figure 4: Lena après une fermeture (dilatation puis érosion) de l'image seuillée

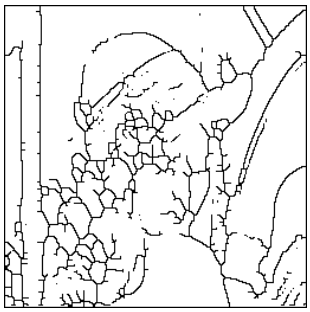


Figure 5: Lena après amincissement des contours

3 Exemples d'algorithmes à programmer

Une image est donc représentée par un tableau de pixels à deux dimensions, `NBROW` lignes et `NBCOL` colonnes.
Chaque élément du tableau représente l'intensité du pixel correspondant. Typiquement, les images en niveaux de gris sont codées sur un octet, la plage de valeur est comprise entre 0 et 255.

3.1 Calcul du gradient et seuillage

Un des traitements d'images parmi les plus simples consiste à appliquer un masque sur l'image afin de faire apparaître les endroits de l'image où les différences (gradients) entre des pixels voisins sont les plus fortes. Ces endroits correspondent normalement aux contours présents dans l'image.
Appliquer un masque sur une image correspond à déplacer sur l'image une fenêtre carrée de côté `sizeMask`, `sizeMask` étant impair et supérieur ou égal à 3, centrée à chaque fois sur un pixel (`i`,`j`) dont il faut évaluer le gradient. Il sera nécessaire de programmer des fonctions suffisamment génériques de façon à changer très simplement et rapidement la taille du masque ainsi que ses cofficients.

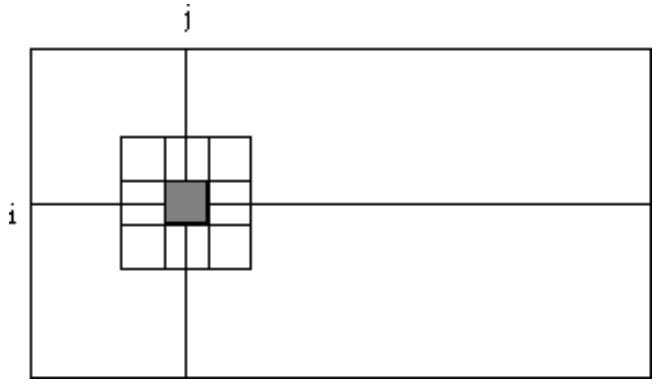


Figure 6: Application d'un masque de taille 3 sur une image

A chaque étape (et donc pour chaque pixel de l'image), l'application d'un masque centré sur le pixel (`i`,`j`) consiste à réaliser une somme pondérée des valeurs des pixels voisins de ce pixel. Les coefficients de pondération sont stockés dans le masque.

MaskV

−1	0	1
−2	0	2
−1	0	1

(a)

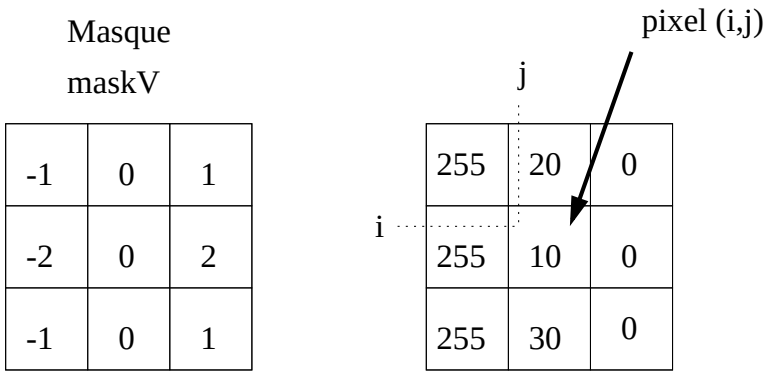
MaskH

1	2	1
0	0	0
−1	−2	−1

(b)

Figure 7: Exemples de masques. Le masque (a) permet de trouver les différences verticales et le masque (b) permet de trouver les différences horizontales

Exemple de calcul de somme pondérée :



$$Dv = -1*255 + 0*20 + 1*0 + \\ -2*255 + 0*10 + 2*0 + \\ -1*255 + 0*30 + 1*0$$

Afin de normaliser cette valeur (obtenir une valeur entre 0 et 255),

$$\rightarrow \quad Dv = \frac{Dv}{\text{Somme\_des\_coefficients\_positifs (poids)}}$$

Finalement, afin de calculer le gradient en un pixel (`i`,`j`), il faut à la fois calculer `Dv` (gradient vertical (application de `MaskV`)) et `Dh` (gradient horizontal (application de `MaskH`).

Le résultat du calcul du gradient pour le pixel (`i`,`j`) est alors :

$$\sqrt{Dv^2 + Dh^2}$$

La valeur calculée ainsi est ensuite placée dans le pixel (`i`,`j`) d'une image résultat appelée image des gradients.

Tout ce traitement revient donc à évaluer pour chaque point de l'image une valeur correspondant au gradient en ce point (en fait la norme des gradients verticaux et horizontaux). Plus ce gradient est fort, plus il est probable que ce point appartienne à un contour.

Un moyen simple pour trouver les contours à partir de l'image des gradients consiste à réaliser un seuillage de cette image. Tous les points dont la valeur est supérieure à un seuil peuvent être considérés comme des contours (Valeur `0` (`C`) : noir). Les autres points ne correspondent pas à des contours (Valeur `255` (`NC`) : blanc).

3.2 Erosion, Dilatation

3.2.1 Principes

Après avoir effectué un seuillage de l'image des gradients, il peut être intéressant de supprimer les points de contours isolés ou de combler les trous dans un contour.

Pour cela, il existe deux opérateurs morphologiques :

- **L'érosion:**

  L'érosion consiste à supprimer les points de Contour (valeur 0 (`C`): noir) ayant au moins un point voisin de type Non Contour (valeur 255 (`NC`) : blanc).

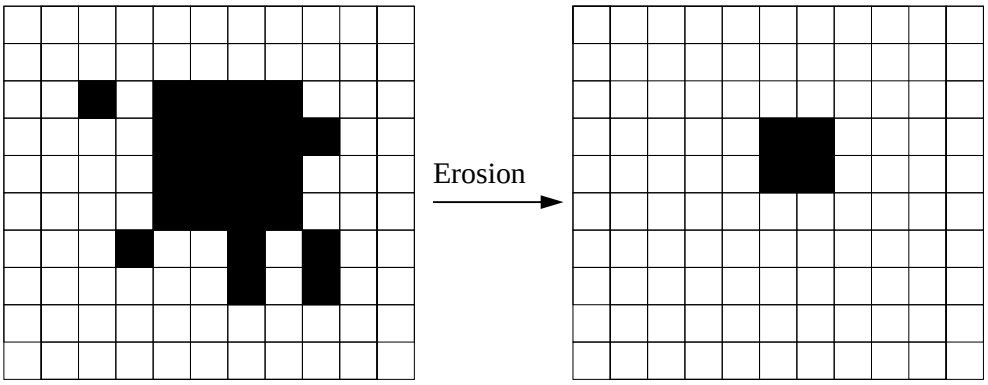


Figure 8: Erosion

- **La dilatation:**

  La dilatation consiste à transformer les points de type Non Contour (valeur 255 (`NC`): blanc) ayant au moins un point voisin de type Contour (valeur 0 (`C`): noir). Les points transformés deviennent alors des points Contour.

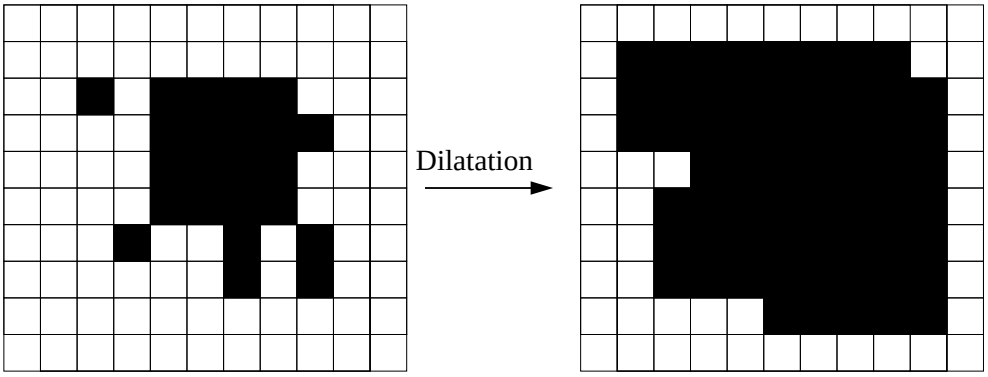


Figure 9: Dilatation

A partir de l'érosion et de la dilatation, il est possible de définir deux opérations : l'ouverture et la fermeture.

- **L'ouverture:**

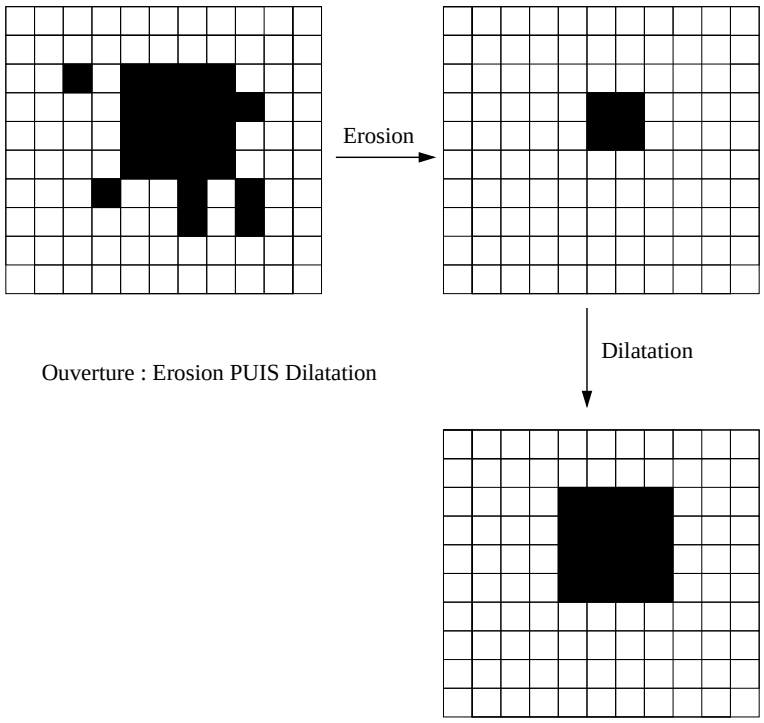


Figure 10: Ouverture

L'ouverture permet de supprimer les petits objets.

- **La fermeture:**

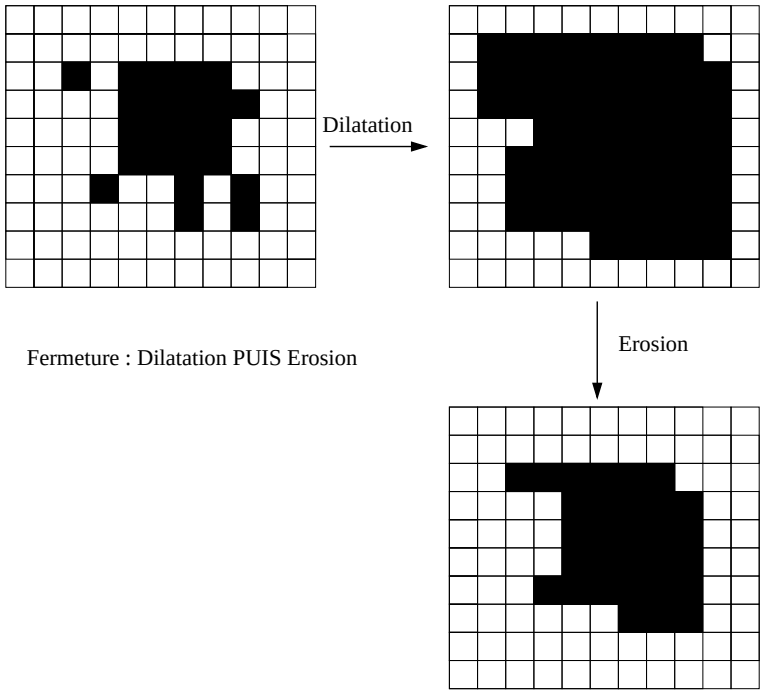


Figure 11: Fermeture

La fermeture permet de combler les trous, les chenaux étroits et de connecter les contours proches.

3.3 Amincissement

L'amincissement permet d'obtenir des contours d'épaisseur 1 (le *squelette*).

Le principe de l'amincissement est simple:

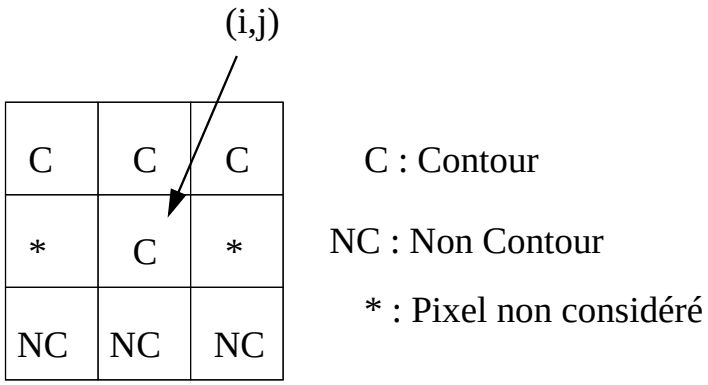


Figure 12: Une configuration d'amincissement

Le point de Contour (`i`,`j`) est supprimé si la configuration décrite sur figure 12 est respectée.

L'amincissement est itéré dans toutes les directions (rotation de $\pi/4$ de la figure 12) jusqu'à stabilisation (plus aucun changement).

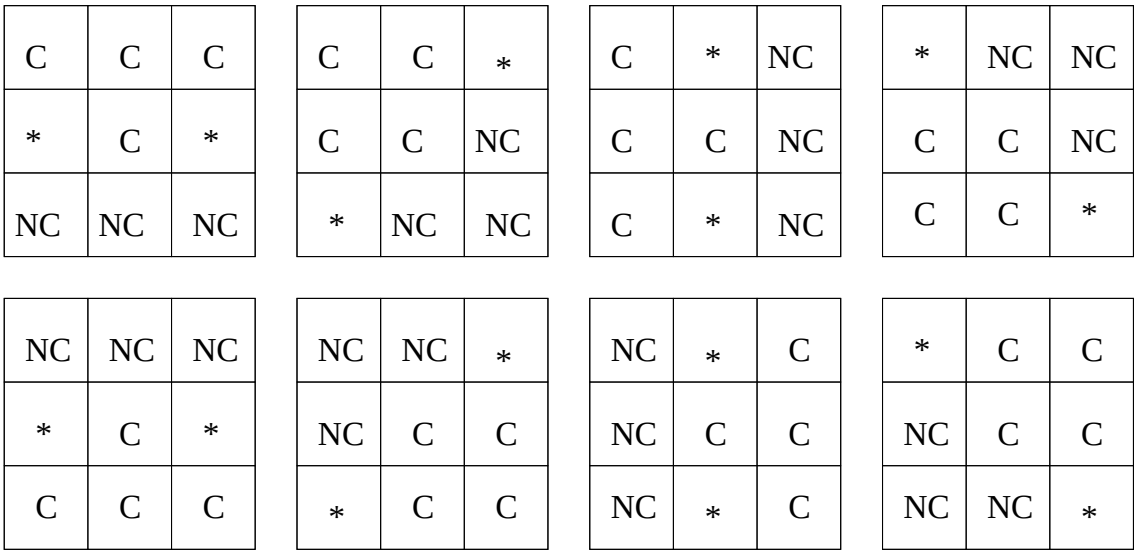


Figure 13: Ensemble des configurations d'amincissement

4 Structure de données `Image` et format de fichiers

Une image en niveaux de gris sera représentée à l'aide d'une structure. Cette structure contient une zone mémoire allouée dynamiquement. Le pointeur `ptrZone` permet d'y accéder "linéairement". Afin de retrouver l'aspect bi-dimensionnel d'une image, un second pointeur est disponible : `zone`.

```
typedef unsigned char octet;                /* Une partie du fichier image.h */

#define VR_MAGIC 0x59a66a90

struct _Image {

   int   magicNumber; /* Pour verifier que on a (pas) alloue de la memoire */

   int   nbRow;
   int   nbCol;

   octet *ptrZone; /* Le "vrai" pointeur sur la zone image */
   octet **zone;   /* Un pointeur qui permet : zone[ligne][colonne] */
};

typedef struct _Image Image;
```

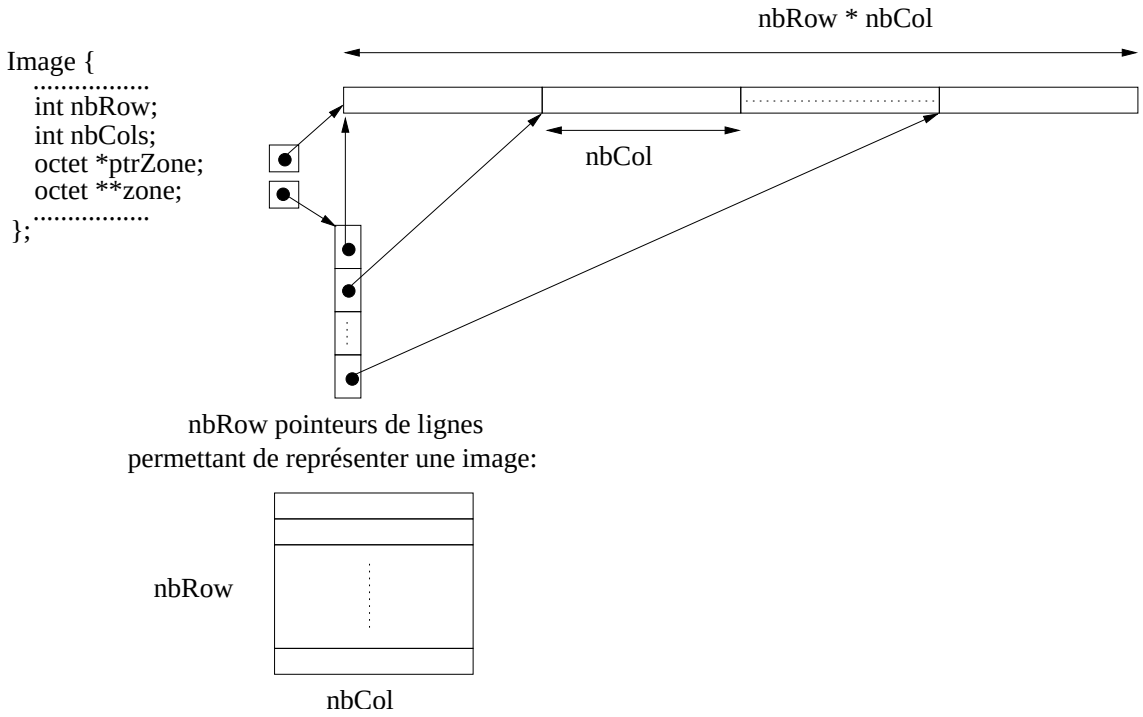


Figure 14: Structure de données permettant de retrouver l'aspect bi-dimensionnel d'une image.

Dans le fichier `image.h` il y a également la déclaration de 6 fonctions :

```
                                        /* Une autre partie du fichier image.h */

void mallocImage(Image *out, int nbRow, int nbCol);
void freeImage(Image *in);

void loadImage(Image *out, const char *nomFichier);
void saveImage(const Image *in, const char *nomFichier);

void displayImage(const char *nomFichier);

void setImage(Image *out, octet val);
void copyImage(const Image *in, Image *out);
```

Descriptions rapides de ces fonctions :

- `mallocImage` et `freeImage` gèrent l'allocation et la libération d'une image (pointeurs `ptrZone` et `zone`).
  - ◇ `mallocImage` : si necessaire (i.e. pas alloue ou pas de la bonne taile) $\Longrightarrow$ freeImage et malloc.
  - ◇ `freeImage` : si necessaire (i.e. alloue) $\Longrightarrow$ free de ptrZone et free de zone.
- `loadImage` permet le chargement d'une image, les formats en lecture sont *SUN rasterfile*, *ppm* et *pgm*. La distinction entre les formats est effectuée grâce aux extensions `.ras`, `.ppm` et `.pgm`.

  Remarque : un `mallocImage` est effectué avant le chargement effectif de l'image.
- `saveImage` permet de sauvegarder dans un fichier le contenu d'une image. Les formats en écriture sont *SUN rasterfile*, *ppm* (en ASCII), *ppm* (en binaire), *pgm* (en ASCII) et *pgm* (en binaire). La distinction entre les formats est effectuée grâce aux extensions `.ras`, `.ppm`, `.bin.ppm`, `.pgm` et `.bin.pgm`.
- `displayImage` réalise l'affichage d'une image à partir d'un fichier image de type *sun rasterfile*, *ppm* ou *pgm*.

  L'affichage est réalisé à l'aide d'un utilitaire bien pratique (`xv`).

  $\Longrightarrow$ `xv` doit être dans le `PATH`...

  $\Longrightarrow$ On peut changer de visualiseur !

  Valeur par défaut : `static char visualiseur[]="xv";` (voir fichier `image.c`).
- `setImage` initialise tous les pixels d'un image avec la même valeur `val`.
- `setImage` effectue la copie d'une image dans une autre. Attention l'image source et l'image destination doivent avoir la même taille.

Le fichier `prog.c` ci-après illustre un exemple de traitement très simple : chargement d'une image, binarisation, sauvegarde et affichage.

```
                        /* prog.c */
#include <stdio.h>
#include "image.h"

#define  C  0
#define NC 255

void seuillage(const Image * in, Image * out, octet seuil)
{
 int l = 0, c = 0;
 int nbRowI = in->nbRow, nbColI = in->nbCol;

 mallocImage(out,nbRowI,nbColI); /* Si necessaire */

 for(l=0;l<nbRowI;l++)
 {
  for(c=0;c<nbColI;c++)
  {
   out->zone[l][c] = (in->zone[l][c]>seuil) ? C : NC;
  }
 }
}

int main(void)
{
 Image  i0, i1;

 loadImage(&i0,"Images/cornouaille.ras");

 seuillage(&i0,&i1,150);

 displayImage("Images/cornouaille.ras");

 saveImage(&i1,"Resultats/visu1.ras");
 displayImage("Resultats/visu1.ras");

 freeImage(&i0);
 freeImage(&i1);

 return 0;
}
```