



R. Malgouyres, R. Zrour et F. Feschet
Initiation   l'algorithme et   la programmation en C,
Cours avec 129 exercices corrig s,
DUNOD, Collection Sciences Sup, 2011, 2^e  dition

Algorithmique et programmation en C

TP n  8

D tection de contours et images PGM

Objectifs :

Le but du TP est de pratiquer les tableaux   double entr e sous forme d'images, et d' tudier le format d'images *PGM*. On verra aussi comment ajouter des param tres   la fonction `main`.

1 Repr sentation en m moire d'une image en niveaux de gris

Dans une image noir et blanc en 256 niveaux de gris, la couleur de chaque pixel est donn e par un nombre entre 0 et 255. Plus la couleur est sombre, plus le niveau de gris est proche de 0. Pour repr senter un niveau de gris en langage C, on utilise un `unsigned char`.

Une image est donn e par la couleur de ses pixels. Chaque pixel est rep r e par ses coordonn es (i, j) , qui sont des nombres entiers. Pour repr senter une image, on utilise un tableau de dimension 2 de `unsigned char`. La largeur et la hauteur de l'image sont des nombres de pixels.

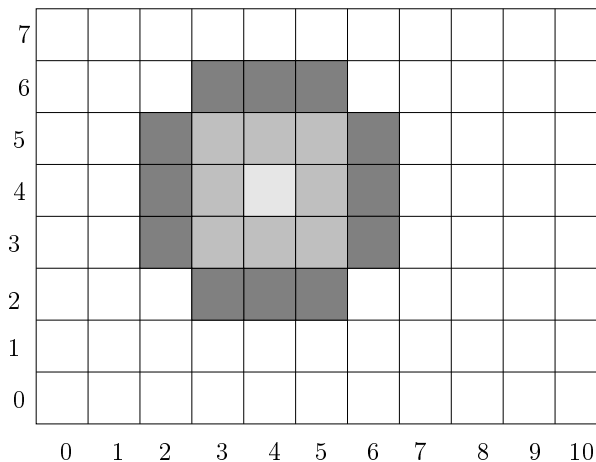


FIGURE 1: Un objet avec dans une image en niveaux de gris de largeur 11

On propose la structure de données suivante pour représenter une image :

```
typedef struct
{
    int haut, larg; /* largeur et hauteur de l'image */
    unsigned char **img; /* tableau de dimension 2 d'octets */
}Image;
```

2 Format de fichier *PGM*

Un fichier *PGM* est un fichier texte qui code les données d'une image dans le format suivant :

- La première ligne du fichier contient le mot clé P5.
- La deuxième ligne du fichier contient la largeur (nombre de colonnes), puis la hauteur (nombre de lignes) de l'image, séparées par un espace.
- La troisième ligne du fichier contient les caractères 255 (cela représente le niveau de gris le plus élevé).
- Suivent ensuite les caractères qui représentent les niveaux de gris. Chaque niveau de gris est codé en binaire par 1 caractère. Les caractères sont rangés par lignes, les lignes étant les unes à la suite des autres dans le fichier sans séparation.

3 Routines d'imagerie

On mettra les fonctions de cette partie dans un fichier `pgm.c` et le programme principal dans un fichier `main.c`.

Exercice 1 Faire une fonction d'allocation qui prend en parallèle une largeur et une hauteur, et qui retourne une image avec ses champs initialisés et la mémoire allouée.

Exercice 2 Faire une fonction de libération de mémoire qui prend en paramètre une image et libère la mémoire.

Exercice 3 En utilisant la fonction d'allocation, faire une fonction de chargement d'une image au format *PGM*. La fonction de chargement retournera l'image.

Exercice 4 Faire une fonction de sauvegarde d'une image au format *PGM*.

Exercice 5 Faire une fonction de test qui charge une image *PGM*, et la sauvegarde dans un fichier avec un autre nom.

On pourra utiliser les fichiers *PGM* du répertoire `/home/prof/malgouyr/pgm/`.

```
$ cp /home/prof/malgouyr/pgm/*.pgm .
```

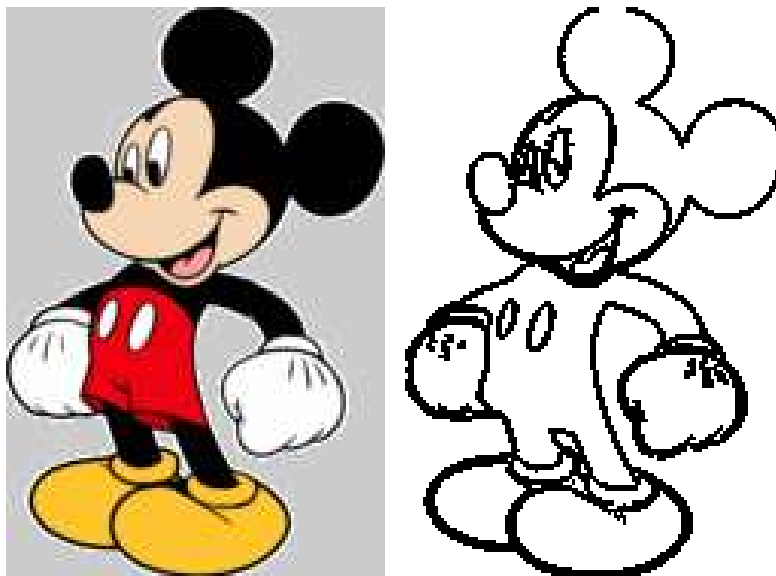
Pour visualiser le résultat, on pourra utiliser la commande linux :

```
$ display nomfichier.pgm &
```

4 Une application de traitement d'images : la détection de contours

On mettra les fonctions de cette partie dans un fichier `contour.c`.

Une image présente fréquemment des zones homogènes séparées par des zones de fort contraste appelées *contour* (voir la figure 2). Un algorithme de détection de contours permet d'afficher les contours en noir (niveau de gris 0), et les zones homogènes en blanc (niveau de gris 255).



(a) Une image *PGM* en niveaux de gris (b) L'image après détection de contours

FIGURE 2: Exemple d'application de la détection de contours

Exercice 6 Faire une fonction qui prend en paramètre une image *I1*, et qui réalise les traitements suivants :

1. Créer une image *I2* aux mêmes dimensions que l'image *I1*.
2. Initialiser la couleur de tous les pixels de *I2* à blanc (255).
3. On introduit un seuil *SEUIL* qui est une constante (par exemple égale à 60).

Un pixel est un pixel de contour s'il est très différent de ses voisins, par exemple si la différence des niveaux de gris (convertis en `int`) avec ses voisins est supérieure (en valeur absolue) au seuil. On a donc l'algorithme suivant :

Pour chaque pixel (i, j) de *I1* tel que $1 \leq i \leq I.haut - 2$ et $1 \leq j \leq I.larg - 2$, si la valeur absolue de la différence avec ses voisins du niveau de gris de (i, j) dans *I1* est supérieure à *SEUIL*, on met le niveau de gris de *I2* à 0. Les voisins de (i, j) sont $(i - 1, j)$, $(i + 1, j)$, $(i, j - 1)$ et $(i, j + 1)$.

4. Retourner *I2*.

Exercice 7 Faire une fonction `GenereContour` qui prend en paramètre un nom de fichier *PGM* et qui génère (en utilisant la fonction de sauvegarde) une image `output.pgm` qui est le résultat de la détection de contours. Visualiser avec la commande `display`.

5 Création d'une commande système

Le but de cette partie est de créer un exécutable appelé `detectcontour` qui fonctionne comme une commande système. On l'appelle en tapant dans la console :

```
$ ./detectcontour nomfichier.pgm
```

Le programme affiche alors le résultat de la détection de contours dans une fenêtre graphique.

Exercice 8 Automatiser l'affichage du résultat `output.pgm` du programme précédent par un appel à la fonction `C system` de la bibliothèque `stdlib.h`. Cette fonction `C` permet de faire un appel système, c'est à dire qu'elle permet d'exécuter une commande du système linux à partir d'un programme `C`. On ajoutera donc la ligne de code suivante après l'appel de la fonction `GenereContour` dans le `main` :

```
system("display output.pgm");
```

Exercice 9 Le but de cet exercice est de mettre le nom du fichier en argument du programme. Ceci est fait au moyen de deux paramètres `argc` et `argv`, ajoutés à la fonction `main`. Le prototype de la fonction `main` est alors :

```
int main(int argc, char**argv);
```

Le nombre `argc` est le nombre d'arguments de la commande donnés par l'utilisateur (**en comptant le nom de l'exécutable**). Pour un programme prenant en paramètre un nom de fichier, on doit donc vérifier que l'entier `argc` vaut 2.

Le paramètre `argv` est un tableau de chaînes de caractère. La chaîne `argv[0]` contient le nom de l'exécutable, et la chaîne `argv[1]` contient le nom de fichier *PGM* passé dans la ligne de commande par l'utilisateur. Il ne reste plus qu'à passer ce nom de fichier à la fonction `GenereContour`.