IUT de Montpellier, Dép. Informatique
M2101 - Archi 2 V. Poupet

TD n° 4 - Manipulation d'images

Dans ce TD nous allons utiliser les appels systèmes d'entrée/sortie pour manipuler des fichiers représentant des images.

Tous les programmes écrits doivent gérer les éventuelles erreurs qui peuvent se produire lors de l'utilisation des appels systèmes, en particulier `open` (en cas d'erreur, affichez un message adapté à l'aide de `perror` et interrompez l'exécution du programme).

1 Images en niveaux de gris

Dans cette section, nous allons manipuler des images en niveaux de gris stockées au format *Portable Gray Map* (`pgm`). Un fichier `pgm` contient un en-tête suivi d'une partie de données binaires dans laquelle chaque point de l'image (pixel) est représenté par un octet du fichier, correspondant à son intensité en niveau de gris (0 : noir, 255 : blanc).

L'en-tête comporte au moins trois lignes :

— un code (ou *magic number* indiquant le format (« P5 » pour le `pgm`) ;
— la largeur de l'image, suivie de la hauteur. Les valeurs sont représentées par leurs représentations binaires (caractères ASCII) et sont séparées par au moins un espace ;
— la valeur maximale des niveaux de gris utilisés (en général 255), également en décimal ASCII. Cette valeur correspond à du blanc.

L'en-tête peut également contenir d'autres lignes de commentaires commençant par le caractère `#`. Pour simplifier les choses, on supposera dans ce TP que les en-têtes ne contiennent pas de commentaires et que le niveau de gris maximum est 255.

Par exemple, un fichier `pgm` pourrait commencer de la manière suivante :

```
P5
4320 3240
255
"+-6HB59;31.+*+35...
```

1 En utilisant la commande `convert` (vue au premier semestre), convertissez l'image de votre choix au format `pgm`. Renommez l'image produite en `image.pgm`.

Solution :

```
$ convert lapin.jpg image.pgm
```

Remarque : Pour visualiser les images `pgm`, vous pouvez utiliser le programme *gimp*.

2 Écrivez un programme `negative.c` qui prend en argument deux noms de fichiers, ouvre le premier (que l'on suppose être une image au format `pgm`) et écrit dans le second une image correspondant à une inversion d'intensité de chacun des pixels de l'image de départ.

Solution : On utilise la fonction `lireligne` du TD précédent.

```c
int main(int argc, char **argv) {
  int fdin, fdout, i, nbread;
  char buffer[4096];

  // ouverture du premier fichier en lecture
  fdin = open(argv[1], O_RDONLY);
  if (fdin < 0) {
    perror(argv[1]);
    exit(1);
  }

  // ouverture du second fichier en écriture
  fdout = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, 0644);
  if (fdout < 0) {
    perror(argv[2]);
    exit(1);
  }

  // copie les trois premières lignes (en-tête)
  for (i = 0; i < 3; i++) {
    nbread = lireligne(fdin, buffer, 4096);
    write(fdout, buffer, nbread);
  }

  // lecture des pixels de l'image
  do {
    // lecture de pixels dans le buffer
    nbread = read(fdin, buffer, 4096);
    // inversion des pixels lus
    for (i = 0; i < nbread; i++) {
      buffer[i] = 255 - buffer[i];
    }
    // écriture des pixels inversés
    write(fdout, buffer, nbread);
  } while (nbread > 0);
}
```

2 Images en couleur

On s'intéresse maintenant aux images au format *Portable Pixel Map* (`ppm`). Tout comme pour le format `pgm`, les fichiers `ppm` commencent par un en-tête suivi de données binaires où chaque pixel de l'image est représenté par trois octets correspondant aux niveaux de rouge, vert et bleu de la couleur du pixel. Le format de l'en-tête est semblable à celui du format `pgm` :

— un *magic number* (« P6 » pour le format `ppm`) ;
— la largeur et la hauteur de l'image données en décimal ASCII ;
— le nombre de niveaux de chaque composante de couleur (en général 255).

3 Convertissez une image au format `ppm`, en la nommant `image.ppm`.

4 Écrivez un programme `permute.c` qui permute les composantes de chaque pixel d'une image `ppm` (pour chaque pixel, `rgb` → `gbr`).

Solution :

```
int main(int argc, char **argv) {
  int fdin, fdout, i, nbread;
  // on va lire les octets 3 par 3 donc on prend un buffer dont la taille
  // est divisible par 3
  char buffer[3000];
  char c;

  // ouverture du premier fichier en lecture
  fdin = open(argv[1], O_RDONLY);
  if (fdin < 0) {
    perror(argv[1]);
    exit(1);
  }

  // ouverture du second fichier en écriture
  fdout = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, 0644);
  if (fdout < 0) {
    perror(argv[2]);
    exit(1);
  }

  // copie les trois premières lignes (en-tête)
  for (i = 0; i < 3; i++) {
    nbread = lireligne(fdin, buffer, 3000);
    write(fdout, buffer, nbread);
  }

  // lecture des pixels de l'image
  do {
    // lecture de pixels dans le buffer
    nbread = read(fdin, buffer, 3000);
    // permutation des couleurs (on suppose que le nombre d'octets lus
    // est multiple de 3
    for (i = 0; i < nbread/3; i++) {
      c = buffer[3*i];
      buffer[3*i] = buffer[3*i + 1];
      buffer[3*i + 1] = buffer[3*i + 2];
      buffer[3*i + 2] = c;
    }
    // écriture des pixels inversés
    write(fdout, buffer, nbread);
  } while (nbread > 0);
}
```

5 Écrivez un programme `intensite.c` qui modifie l'intensité de chaque composante des

pixels d'une image en lui ajoutant une valeur constante passée en troisième argument (si la constante est de 15 par exemple, alors chaque composante doit être augmentée de 15).
Attention : La valeur de chaque composante doit rester entre 0 et 255.
Indication : Les arguments reçus par le programme sont tous des chaînes de caractères. Ceux qui représentent des entiers doivent donc être convertis en une variable de type `int` [1].

Solution :

```
int main(int argc, char **argv) {
  int fdin, fdout, i, nbread;
  char buffer[4096];
  char *ptr;

  // valeur d'intensité à ajouter (cf. doc de strtol)
  int delta = strtol(argv[3], &ptr, 10);

  // ouverture du premier fichier en lecture
  fdin = open(argv[1], O_RDONLY);
  if (fdin < 0) {
    perror(argv[1]);
    exit(1);
  }

  // ouverture du second fichier en écriture
  fdout = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, 0644);
  if (fdout < 0) {
    perror(argv[2]);
    exit(1);
  }

  // copie les trois premières lignes (en-tête)
  for (i = 0; i < 3; i++) {
    nbread = lireligne(fdin, buffer, 4096);
    write(fdout, buffer, nbread);
  }

  // lecture des pixels de l'image
  do {
    // lecture de pixels dans le buffer
    nbread = read(fdin, buffer, 4096);
    // changement d'intensité (on vérifie qu'on ne sort pas de
    // l'intervalle [0, 255]
    for (i = 0; i < nbread; i++) {
      // Pour gcc les char sont signés (entre -128 et 127) donc il faut
      // les convertir en unsigned char (entre 0 et 255) pour faire les
      // comparaisons
      unsigned char valeur = buffer[i];
      if (valeur > 255 - delta) {
        buffer[i] = 255;
```

1. Vous pouvez utiliser les fonctions `strtol` (un peu compliquée mais recommandée) ou `atoi` (beaucoup plus simple mais ne détecte pas les erreurs).

```
      } else if (valeur < -delta) {
        buffer[i] = 0;
      } else {
        buffer[i] = valeur + delta;
      }
    }
    // écriture des pixels inversés
    write(fdout, buffer, nbread);
  } while (nbread > 0);
}
```

6 Écrivez un programme `miroir.c` qui permet de réaliser un effet de miroir vertical ou horizontal sur une image `pgm` ou `ppm`.

Le programme doit prendre en paramètre une option indiquant le type de symétrie à appliquer ainsi que les noms des fichiers en entrée et en sortie. Si la première option est `-v`, il faut appliquer une symétrie selon un axe vertical, si c'est `-h` l'axe est horizontal.

Indication : Il faut lire les données de l'image et les mémoriser dans un tableau (c'est plus simple pour le miroir vertical parce qu'on peut travailler ligne par ligne). Si l'image est grande, les tableaux ne peuvent pas être alloués de manière automatique et il faut alors utiliser l'allocation dynamique (`malloc`).

Solution : Ce programme est écrit pour les images PGM. Pour les PPM il faut tenir compte du fait que chaque pixel prend 3 octets. On peut faire un programme qui gère les deux cas en regardant le contenu de la première ligne du fichier. On utilise la fonction `strcmp` de la bibliothèque `string.h` pour tester la valeur du premier paramètre.

```
int main(int argc, char **argv) {
  // Verification du nbre d'arguments
  // On attend 3 arguments :
  //   argv[1] : "-v" ou "-h" pour indiquer la direction de l'axe
  //   argv[2] : nom du fichier en entrée (qui doit exister)
  //   argv[3] : nom du fichier en sortie (créé par le programme)
  if (argc != 4){
    printf("Erreur: nbre d'arguments.\n");
    exit(1);
  }

  int fdin, fdout, i, nbread;
  char buffer[4096];

  // ouverture du premier fichier en lecture
  fdin = open(argv[2], O_RDONLY);
  if (fdin < 0) {
    perror(argv[2]);
    exit(1);
  }

  // ouverture du second fichier en écriture
  fdout = open(argv[3], O_WRONLY | O_CREAT | O_TRUNC, 0644);
  if (fdout < 0) {
```

```
  perror(argv[3]);
  exit(1);
}

// copie de la première ligne de l'en-tête
nbread = lireligne(fdin, buffer, 4096);
write(fdout, buffer, nbread);
// lecture et copie de la seconde ligne (pour les dimensions de l'image)
int largeur, hauteur;
nbread = lireligne(fdin, buffer, 4096);
i = 0;
// lecture de la largeur de l'image
largeur = 0;
while ('0' <= buffer[i] && buffer[i] <= '9') {
  largeur = 10 * largeur + (buffer[i] - '0');
  i++;
}
while (buffer[i] == ' ') {
  i++;
}
// lecture de la hauteur de l'image
hauteur = 0;
while ('0' <= buffer[i] && buffer[i] <= '9') {
  hauteur = 10 * hauteur + (buffer[i] - '0');
  i++;
}
write(fdout, buffer, nbread);
// copie de la troisième ligne de l'en-tête
nbread = lireligne(fdin, buffer, 4096);
write(fdout, buffer, nbread);

// copie des données de l'image dans un tableau data
char *data;
data = malloc(sizeof(char) * largeur * hauteur);
i = 0;
do {
  nbread = read(fdin, data + i, largeur * hauteur - i);
  i += nbread;
} while (i < largeur * hauteur);

int x, y;
char c;
if (strcmp(argv[1], "-v") == 0) {
  // miroir selon un axe vertical (ligne par ligne)
  for (y = 0; y < hauteur; y++) {
    // retournement de la ligne y dans le tableau
    for (x = 0; x < largeur/2; x++) {
      c = data[largeur * y + x];
      data[largeur * y + x] = data[largeur * y + largeur - x];
      data[largeur * y + largeur - x] = c;
```

```
      }
    }
    // écriture des données dans le fichier en sortie
    i = 0;
    while (i < largeur * hauteur) {
      nbread = write(fdout, data + i, largeur * hauteur - i);
      i += nbread;
    }
  } else if (strcmp(argv[1], "-h") == 0) {
    // miroir selon un axe horizontal (on écrit les lignes dans l'ordre
    // inverse
    for (y = hauteur-1; y >= 0; y--) {
      write(fdout, data + (largeur * y), largeur);
    }
  } else {
    printf("Le premier argument doit être \"-v\" ou \"-h\".\n");
    exit(1);
  }
}
```