Projet C : Traitement et manipulation d'image

1 Sujet

Ce projet vous permettra de mettre en pratique les connaissances acquises dans le module de programmation en C. Le but est d'implanter quelques outils fondamentaux du traitement d'images : conversion d'une image en couleurs en niveaux de gris puis à partir de celle-ci le calcul et représentation de l'histogramme de l'image, négatif, égalisation, filtre passe-bas et utilisation de l'opérateur de Sobel pour extraire les contours de l'image.

Une image numérique est une matrice de pixels (picture elements), chacun de ces pixels codant une couleur par l'intermédiaire d'un triplet (R, V, B) dont les éléments représentent respectivement la composante rouge, verte et bleue d'une couleur (en synthèse additive). Chaque composante couleur est généralement codée sur 8 bits (ou 1 octet) soit 256 valeurs différentes et donc chaque pixel est d'une couleur choisie parmi 16 millions (approximativement). La position de chaque pixel n'est pas codée, l'image (dont on connait les dimensions) étant simplement représentée par une succession de pixels.

2 Quelques traitements d'image de base

2.1 Conversion en niveaux de gris

La conversion d'une couleur en information de luminance L se fait en récupérant les composantes RVB de chaque pixel de l'image et en utilisant l'une des formules proposées par la Compagnie Internationale de l'Eclairage. Ici $L = 0.299 * R + 0.587 * V + 0.114 * B$.

2.2 Inversion

Le négatif d'une image en niveau de gris se fait simplement en calculant le complment N de la luminance de chaque pixel codée sur 8 bits soit $N = 255 - L$.

2.3 Histogramme d'une image

En imagerie numérique, l'histogramme représente la distribution des luminances dans une image en noir et blanc. C'est un outil fondamental en traitement d'images. Comme nous travaillons sur une image en niveaux de gris, cet histogramme comportera 256 valeurs différentes sur l'axe des abscisses (de 0 à 255) et à chaque valeur correspondra le nombre de pixels ayant cette valeur. Pour sa représentation graphique, il faudra appliquer une mise à l'échelle (penser au cas particulier d'une image monochrome).

2.4 Egalisation

Dans une image naturelle qui a été quantifiée de manière linéaire, une majorité de pixels ont une valeur inférieure à la luminance moyenne. C'est pourquoi les détails dans les régions sombres sont difficilement perceptibles. Une des techniques utilisées pour pallier à cet inconvénient est appelée égalisation d'histogramme, qui est une intégration de l'histogramme initial. L'égalisation d'histogramme consiste à faire en sorte que le nombre de pixels pour chaque niveau N de l'histogramme égalisé idéal (dénommé ici histogramme final) soit constant et égal à $l \times h/N$ avec l la largeur et h la hauteur de l'image. Le nombre de niveaux final N doit être inférieur au nombre de niveaux initial M car l'égalisation n'est autre qu'une intégration de l'histogramme initial, en général on utilise $N = M/2$ qui est l'intégration minimale. L'histogramme final est donc séparé en bandes de taille M/N.

Méthode

- on part du niveau 0 de l'histogramme initial et on somme les niveaux jusqu'à la valeur moyenne idéale $l \times h/N$,
- tous ces niveaux sont recadrés sur un niveau final unique qui se trouve au centre d'une bande de l'histogramme final,
- on recommence l'opération pour les bandes suivantes.

```
Algorithme Egalisation(Image ima, Histogramme histo)
Donnes
  Entier deb, fin  // index de debut et de fin
  Entier lab  // largeur d'une bande soit M/N
  Entier centre  // centre de la bande courante
Dbut
  moy <- l*h/N
  lab <- M/N
  centre <- lab/2
  deb <- fin <- 0
  tantque (fin < M) faire
    cumul <- 0
    tantque (cumul < moy ET fin < M) faire
      cumul <- cumul + histo[fin]
      fin++
    fintq
    pour i <- deb  fin-1 faire
      transfo[i] <- centre
    finpour
    deb <- fin
    centre <- centre + lab
  fintq
  pour tous les pixels de l'image ima faire
    ima[i][j] <- transfo[ima[i][j]]
  finpour
Fin
```

2.5 Filtrage linéaire

Entre une image réelle et l'image numérisée obtenue, un grand nombre de processus sont venus l'altérer. Parmi ceux-ci, le bruit résultat du bruit électronique du capteur et de la qualité de la numérisation joue un role fondamental. On peut définir le bruit dans une image comme un phénomène de brusque variation d'un pixel isolé par rapport à ses voisins. On peut en déduire que pour lutter contre les effets du bruit, il est nécessaire d'opérer des transformations qui tiennent compte du voisinage de chaque pixel. L'une des méthodes visant à atténuer les effets du bruit est un filtre passe-bas : le filtrage linéaire qui est la transformation d'un pixel par une combinaison linéaire des pixels voisins. Une méthode simple consiste à considérer chaque point de l'image et d'en faire la moyenne avec les 8 pixels qui lui sont voisins, ce qui aura pour effet d'adoucir l'image en réduisant les fluctuations de luminosité. Pour chaque pixel, on aura donc :

$$imb[i][j] = (ima[i-1][j-1] + ima[i-1][j] + ima[i-1][j+1] + ima[i][j-$$

$1]+ima[i][j]+ima[i][j+1]+ima[i+1][j-1]+ima[i+1][j]+ima[i+1][j+1])/9$

Attention : il faut travailler avec deux images (pour ne pas modifier l'image initiale pendant le filtrage) et tenir compte des effets de bord (i.e des pixels aux bords qui n'ont pas 8 voisins). Pour ces derniers on peut les ignorer durant le processus ou créer des pixels miroirs (i.e répliquer les pixels du bord de faon à ce qu'ils aient 8 voisins).

2.6 Extraction de contours (filtre de Sobel)

La notion de contour dans une image est fortement liée à ce que l'on désire voir dans celle-ci associée avec une information sur le contexte (taille des objets, contraste moyen de ceux-ci avec le fond, etc.). Mais indépendamment de tout critère subjectif, il faut une définition mathématique d'un contour : c'est la brusque variation de niveau de gris dans une image et on peut caractériser cette variation par son amplitude (et par sa pente). La mise en évidence des contours se fait par une différenciation de l'image : si l'on dérive l'image une fois, on obtient un gradient (deux fois, on obtient un laplacien). On utilise le filtre de Sobel qui est une approximation discrète de la dérivée réelle et tel que (attention aux signes) :

$Ax = -ima[i-1][j-1] - ima[i-1][j] - ima[i-1][j+1] + ima[i+1][j-1] + ima[i+1][j] + ima[i+1][j+1]$
$Ay = -ima[i-1][j-1] - ima[i][j-1] - ima[i+1][j-1] + ima[i-1][j+1] + ima[i][j+1] + ima[i+1][j+1]$

L'amplitude en un point (i,j) de l'image est alors : $A = sqrt(Ax^2 + Ay^2)$. Pour visualiser l'image des contours, il est nécessaire ici encore d'appliquer une mise à l'échelle de faon à ce que l'ensemble des valeurs d'amplitude soit compris entre 0 et 255.

3 Implantation

L'implantation pourrait être facilitée par l'utilisation le module : utv_gd.h et utv_gd.c, basé sur la bibliothèque GD `https://boutell.com/gd/manual2.0.33.html`. On ne traitera que les images PNG en 256 couleurs avec table des couleurs. La commande convert -colors 256 permet la conversion d'une image dans ce format (cf. convert –help).

Votre application doit proposer un menu texte comportant : le chargement d'une image, l'affichage d'informations associées à cette image, le calcul et l'affichage de l'histogramme, le négatif, l'égalisation, le filtrage, l'extraction des contours de l'image et bien entendu de quitter l'application. Elle doit pouvoir prendre une image en paramètre au moment du lancement dans un terminal. L'affichage d'une image dans le programme peut

se faire par l'instruction system() et la commande display, exemple : system("display toto.png");.

Si vous décidez de vous baser sur le module proposé, l'utilisation du type utv_image est obligatoire :

```
typedef struct {
  char* nom; /* nom de l'image */
  unsigned int la, ha, nbc; /* largeur, hauteur, nombre de couleurs */
  int histo[256]; /* histogramme de l'image en niveau de gris */
} utv_image;
```

Aucune variable globale ne doit être utilisée. Les fonctions et les blocs d'instructions doivent être commentés.