PAROIS Angélique
ARIMON Vincent TP3 SMP

SMP-TP3

---

1 Introduction

Le but de ce TP est de réaliser des traitements d'image simples sur des images binaires. La première partie sera consacrée à charger des images à partir de fichiers au format PGM et bien sûr à les sauver. Pour simplifier, nous ne travaillerons qu'avec des images en niveau de gris de taille maximum fixée. La structure de données pour représenter l'image est imposée :

type
t_Image : enregistrement
  entier : w
  entier : h
  tableau d'entiers : im
fin_enregistrement

Elle se traduit en C(++) par le bloc de code suivant :

```
// taille maximum de l'image
const int TMAX = 640;
typedef struct
{
  // largeur de l'image
  int w;
  // hauteur de l'image
  int h;
  // tableau des niveaux de gris de l'image
  unsigned char im[TMAX][TMAX];
 } t_Image;
```

Le tableau im contient les niveaux de gris de l'image. La convention est de représenter le noir par 0 et le blanc par 255. Les autres valeurs correspondent à des valeurs de gris plus ou moins 1 claires. im[i][j] représente le pixel de la ligne i et de la colonne j. Les colonnes et les lignes sont numérotées à partir de (0, 0), ce point d'origine se trouvant en haut à gauche de l'image. Les images peuvent être visualisées avec l'immense majorité des éditeurs d'image du marché (dont gimp qui est disponible sur Windows, Linux et Mac).

2 Chargement et sauvegarde des images

Le format PGM est simple, ce qui présente l'inconvénient d'avoir rapidement affaire à des gros fichiers. Un fichier PGM en format ascii commence par une ligne contenant P2. Sur la ligne suivante on trouvera les dimensions de l'image (largeur puis hauteur). La ligne suivante donne la plus grande valeur de niveaux de gris présente dans l'image (on écrira systématiquement 255). Suivent ensuite les valeurs des niveaux de gris de chaque pixel ligne par ligne. En théorie, les lignes ne doivent pas contenir plus de 70 caractères, mais on ne prendra pas en compte cette contrainte pour ce TP. Par exemple, un fichier PGM peut commencer de la manière suivante :

```
P2
640 480
255 112 113 107 112 113 114 121 113 114 116 111 108 107 117 113 114 111 114 112
114 111 112 116 111 119 120 118 114 119 123 125 119 117 118 115 109
```

Deux fonctions de chargement et de sauvegarde des images vous sont fournies.

Le code associé est contenu dans les fichiers chargesauve.h et charsauve.cpp. L'utilisation des images dans les fonctions se fait systématiquement à l'aide de pointeurs. Il y a trois raisons à cela : la première est bien sûr de pouvoir modifier les images passées en paramètre comme vu en cours. La deuxième est que comme vous le savez, les paramètres 2 des fonctions sont recopiés dans la pile : en passant un pointeur sur une image, on fait une grosse économie d'espace mémoire par rapport à passer une image complète à chaque fois. Enfin, une image allouée dynamiquement utilise de la mémoire dans les tas et non pas dans la pile du programme, là où il y a sensiblement plus d'espaces.

3 Outils

Pour préparer les opérations de dilatation et d'érosion, on va créer des images noir et blanc. Pour ensuite voir les modifications sur les images, on va utiliser une fonction faisant la différence de deux images.

3.1 Seuillage

Le seuillage à un niveau s d'une image consiste à remplacer les pixels d'un niveau de gris inférieur à s par des pixels noirs (valeur 0) et les autres par des pixels blancs (255).

Description de l'algorithme

Fonction Seuillage
Porblème : transformer une image nuancée de gris en noir et blanc
Spécification
Paramètres
s : entier
Image : typedef struct
Resultat : image transformée en noir et blanc
Pré-condition le seuil doit être compris entre 0 et 255

PAROIS Angélique
ARIMON Vincent TP3 SMP

Principe
On vient comparer un à un chaque pixel de l'image et les mettre en noir si leurs valeur est inférieur au seuil ou en blanc si elle est supérieur.

Algorithme

```
Variables
s : entier
Image : typedef struct
Début
0 < s < 255
Pour i allant de 1 à hauteur de l'image
        Pour j allant de 1 à largeur de l'image
                Si  im[i][j] est inférieur au seuil alors
                        im[i][j] ← 0
                Fin Si
                Sinon [i][j] ← 255
        Fin Pour
Fin Pour
Fin
```

Jeux de tests

On paramètre le seuil à 127 (soit une valeur à mi chemin entre 0 et 255) afin de transformer l'image initiale en noir et blanc de la manière la plus "fidèle". A ce stade si l'image transformée est toute noir, toute blanche ou nuancée de gris c'est que le programme ne fonctionne pas.

Voici le résultat obtenu :

D'après le résultat ci dessus, on peut dans un premier valider le fonctionnement de la fonction car elle transforme bien l'image initiale en une image en noir et blanche. De plus, on observe que les teintes claires on bien été transformée en blanc et les teintes sombres en noir.

3.2 Différence entre deux images

On effectue la différence entre deux images en prenant pour chaque pixel de coordonnées (i, j) la valeur absolue de la différence des valeurs de ce pixel sur les deux images.

Description de l'algorithme

Fonction Différence
Problème : effectuer la différence entre deux images en prenant pour chaque pixel de coordonnées (i, j) la valeur absolue de la différence des valeurs de ce pixel sur les deux images.
Spécification
Paramètres
Image1, Image2, ImFin : typedef struct (2 int et 1 tableau)
Resultat : image à la couleur de image1 - image2
Pré-condition la taille des 3 images doit être identique
Principe
On vient soustraire un à un chaque pixel de l'image2 à l'image1 afin d'obtenir la couleur des pixels de l'image finale.

Algorithme

```
Variables
Image1, Image2: typedef struct
ImFin : typedef struc
Début
ImFin->h ← Image1->h
ImFin->w ← Image1->w
Pour i allant de 0 à hauteur de l'image 1
        Pour j allant de 0 à largeur de l'image 1
                ImFin[i][j] ← valeur absolu(Image1->im[i][j] - Image2->im[i][j])
        Fin Pour
Fin Pour
Fin
```

Jeux de tests

Une fois compilé, notre programme créé à partir d'une image de base (logo) une seconde image seuillée puis viens soustraire les couleurs pour chacun des pixels de la nouvelle image à la première. On obtient alors une troisième image avec des valeurs de pixels différentes.

On observe bien les différences sur la troisième image qui nous donne un résultat mixant les deux images.

4.1 Dilatation

Fonction : Dilatation

Problème :
Effectuer la dilatation d'une image binaire en utilisant un élément structurant de taille impaire.
Un pixel de coordonnées (i, j) devient blanc (255) dans l'image finale s'il existe au moins un pixel blanc dans la zone recouverte par l'élément structurant autour de ce pixel.

Spécification

Paramètres	Type
Image, ImFin	typedef struct (w, h, tableau im)
Structurant	typedef struct (w, h, tableau im)

Résultat :
Une image dilatée : la forme (pixels blancs) est agrandie.

Pré-conditions :

- L'image doit être binaire (valeurs 0 ou 255)
- Le structurant doit être de taille impaire
- Dimensions de ImFin identiques à celles de Image

Principe

On parcourt tous les pixels de l'image.
Pour chaque pixel blanc, on recopie l'élément structurant autour de lui dans l'image finale.
Si au moins une cellule du structurant chevauche un pixel blanc dans l'image originale, alors le pixel du centre devient blanc.

Algorithme

Début

```
ImFin->h ← Image->h
ImFin->w ← Image->w

// Mise à zéro de l'image finale
Pour i de 0 à ImFin->h - 1
  Pour j de 0 à ImFin->w - 1
    ImFin->im[i][j] ← 0
  FinPour
FinPour

cx ← Structurant->w / 2
cy ← Structurant->h / 2

Pour i de 0 à Image->h - 1
  Pour j de 0 à Image->w - 1
    Si Image->im[i][j] == 255 alors
      Pour si de 0 à Structurant->h - 1
        Pour sj de 0 à Structurant->w - 1
          Si Structurant->im[si][sj] == 1 alors
            ni ← i + si - cy
            nj ← j + sj - cx
            Si ni dans [0, Image->h - 1] et nj dans [0, Image->w - 1] alors
              ImFin->im[ni][nj] ← 255
            FinSi
          FinSi
        FinPour
      FinPour
    FinSi
  FinPour
FinPour
Fin
```

Jeux de tests

Après compilation, notre programme applique une opération de **dilatation** sur l'image de départ (logo) en utilisant un élément structurant. Cette transformation consiste à étendre les zones blanches (pixels de valeur 1) de l'image : si au moins un pixel voisin défini par l'élément structurant est blanc, alors le pixel central devient blanc dans l'image résultante.

La nouvelle image obtenue présente donc une extension des formes originales : les contours s'épaississent et les zones blanches occupent davantage d'espace. Cette opération permet notamment de combler certains trous ou discontinuités présents dans l'image initiale.

Visuellement, on remarque que l'image dilatée possède des formes plus larges, ce qui met en évidence l'effet d'expansion produit par la transformation morphologique.

4.2 Erosion

Fonction : Érosion

Problème :

PAROIS Angélique
ARIMON Vincent TP3 SMP

Effectuer l'érosion d'une image binaire en utilisant un élément structurant.
Un pixel reste blanc (255) uniquement si **toutes** les positions du structurant recouvrent des pixels blancs.

Spécification

Paramètres	**Type**
Image, ImFin	typedef struct
Structurant	typedef struct

Résultat :
Une image érodée : la forme (pixels blancs) est réduite.

Pré-conditions :

- L'image doit être binaire (0 ou 255)
- Le structurant doit être de taille impaire
- Dimensions de ImFin identiques à celles de Image

Principe

On parcourt tous les pixels.
Pour chaque pixel, on vérifie si **l'ensemble** de l'élément structurant reste sur des pixels blancs de l'image.
Si une seule cellule active du structurant chevauche un pixel noir ou sort de l'image, le pixel est mis à 0.

Algorithme
Variables :
  Image : typedef struct
  ImFin : typedef struct
  Structurant : typedef struct
  garder : bool
  cx, cy : int

Début

ImFin->h ← Image->h
ImFin->w ← Image->w

cx ← Structurant->w / 2
cy ← Structurant->h / 2

Pour i de 0 à Image->h - 1
  Pour j de 0 à Image->w - 1

    garder ← vrai

```
      Pour si de 0 à Structurant->h - 1
        Pour sj de 0 à Structurant->w - 1

          Si Structurant->im[si][sj] == 1 alors
            ni ← i + si - cy
            nj ← j + sj - cx

            Si ni hors image OU nj hors image OU Image->im[ni][nj] == 0 alors
              garder ← faux
            FinSi

          FinSi

        FinPour
      FinPour

      Si garder alors
        ImFin->im[i][j] ← 255
      Sinon
        ImFin->im[i][j] ← 0
      FinSi

    FinPour
FinPour
Fin
```

Jeux de tests

Après compilation, notre programme applique une opération d'**érosion** sur l'image de départ (logo) en utilisant un élément structurant. Cette transformation consiste à réduire les zones blanches (pixels de valeur 1) de l'image : un pixel ne reste blanc que si **tous** les pixels de son voisinage, défini par l'élément structurant, sont également blancs dans l'image d'origine.

L'image obtenue présente donc une diminution de l'épaisseur des formes : les contours deviennent plus fins et certaines petites zones blanches isolées peuvent disparaître. Cette opération est particulièrement utile pour éliminer du bruit, séparer des objets connectés ou affiner les motifs.

Visuellement, on observe que l'image érodée contient des formes plus étroites et parfois fragmentées, mettant clairement en évidence l'effet de réduction produit par la transformation morphologique.

Conclusion

Ce TP nous a permis de revoir des notions essentielles d'algorithmique ainsi que la gestion de la mémoire en langage C, notamment grâce à l'utilisation des pointeurs et de l'allocation dynamique. Nous avons pu mettre en œuvre et vérifier plusieurs fonctions de traitement d'images au format PGM, telles que le **seuillage**, la **différence**, ainsi que les opérations de **morphologie mathématique** : **dilatation** et **érosion**.

La dilatation nous a permis d'observer l'expansion des formes présentes dans l'image, tandis que l'érosion a eu l'effet inverse en réduisant leur épaisseur.

Ce TP nous a donc offert l'occasion de manipuler concrètement des images en niveaux de gris et d'appliquer des concepts théoriques importants en traitement d'image. Il ouvre également des perspectives d'amélioration du programme, notamment en ajoutant d'autres opérateurs morphologiques, des visualisations comparatives ou l'optimisation de l'usage mémoire.