

IMAGE ET SIGNAL

Partie Traitement et Analyse d'Images

Travaux Pratiques

1 Objectif

Il s'agit de programmer en langage C des opérateurs de traitement d'images vus en cours-TD. Chaque opérateur correspond à une fonction dont vous devez écrire le corps en utilisant les outils fournis.

2 Outils et données

2.1 Archive à récupérer

Les programmes, les images et les matrices fournis sont rassemblés dans l'archive TAI-2022-2023.zip. Cette archive contient :

- la bibliothèque `IMaCE` qui fournit des outils pour manipuler des images au format PNM (PBM, PGM et PPM) et des matrices d'entiers de type `int` ou de flottants de type `double` :
 - `limace.pdf` : documentation ;
 - `limace.h` : fichier d'en-tête ;
 - `limace.c` : fichier d'implémentation ;
- le module `Erreurs` qui fournit des fonctions pour l'affichage de messages d'erreurs et de l'usage des opérateurs :
 - `erreurs.h` : fichier d'en-tête ;
 - `erreurs.c` : fichier d'implémentation ;
- le module `TAI` dont vous devez écrire les corps des fonctions :
 - `tai.h` : fichier d'en-tête ;
 - `tai.c` : fichier d'implémentation à compléter ;
- les fichiers contenant les fonctions principales (`main`) des opérateurs (`comparison.c`, etc.) ;
- le fichier de description des dépendances (`Makefile`) :
 - `make` : commande permettant de lancer la compilation de l'ensemble du projet ;
 - `make clean` : commande permettant d'effacer tous les fichiers objets et tous les fichiers exécutables ;
- des images pour tester les opérateurs :
 - images à utiliser en entrée des opérateurs ;
 - images de référence permettant de vérifier le bon fonctionnement des opérateurs (cf. le descriptif de chaque opérateur) ;
- des matrices pour tester les opérateurs.

2.2 Principales fonctions utiles de `IMaCE`

- `ImAlloc` et `ImCAlloc` : créer une nouvelle image ;
- `ImCopy` : copier une image ;
- `ImType` : connaître le type d'une image (`BitMap`, `GrayLevel` ou `Color`) ;
- `ImGetR` : accéder à la composante rouge des pixels d'une image couleur (type `Color`) ;
- `ImGetG` : accéder à la composante verte des pixels d'une image couleur (type `Color`) ;
- `ImGetB` : accéder à la composante bleue des pixels d'une image couleur (type `Color`) ;

- `ImGetI` : accéder aux niveaux de gris d'une image en niveaux de gris ou noir et blanc (types `GrayLevel` et `BitMap`);
- `ImNbRow` : nombre de lignes d'une image;
- `ImNbCol` : nombre de colonnes d'une image;
- `MatAlloc` et `MatCAlloc` : créer une nouvelle matrice;
- `MatType` : type d'une matrice (`Int` ou `Double`);
- `MatGetInt` : accéder aux éléments d'une matrice d'éléments de type `int`;
- `MatGetDouble` : accéder aux éléments d'une matrice d'éléments de type `double`;
- `MatNbRow` : nombre de lignes d'une matrice;
- `MatNbCol` : nombre de colonnes d'une matrice;
- `MatFree` : libérer l'espace mémoire occupé par une matrice.

Les descriptions de ces fonctions, ainsi que d'autres, se trouvent dans le fichier `limace.pdf`. Le fichier d'en-tête `limace.h` contient également des commentaires utiles. En revanche, la lecture du fichier `limace.c` n'est pas nécessaire.

2.3 Attention



En C, l'expression `a/b` vaut le quotient de la division entière de `a` par `b` si `a` et `b` sont deux entiers (familles des `char` ou des `int`). Dans ce cas, passer cette expression à la fonction `round` n'a aucun effet sur la valeur. Pour que `a/b` soit le résultat de la division réelle de `a` par `b`, il faut qu'au moins l'un des deux opérandes soit un représentant des réels (`float`, `double` ou `long double`). Par exemple, `round(11/4)` vaut 2.0 alors que `round(11.0/4)` vaut 3.0.

3 Travail à réaliser

Après avoir décompressé l'archive, lancez la production des exécutables correspondant aux opérateurs décrits dans la section 4 grâce à la commande `make` (des *Warnings* concernant les paramètres inutilisés s'afficheront tant que vous n'aurez pas programmé tous les opérateurs).

Pour connaître la syntaxe d'appel de chaque opérateur, lancez l'exécutable correspondant avec l'option `-h`.

Pour chaque opérateur, dans le fichier `tai.c`, écrivez le corps de la fonction correspondante à la place de l'appel à la macro `AFAIRE()`.

Testez chaque opérateur sur les données indiquées dans le descriptif. Afin de vérifier le résultat, en plus de l'affichage du résultat obtenu, comparez-le avec le résultat de référence fourni en utilisant l'opérateur `comparison` (cet opérateur étant déjà programmé, vous pouvez l'utiliser directement).

4 Opérateurs à programmer

4.1 **RGB2Gray : conversion d'une image couleur en une image de niveaux de gris**



Écrivez la fonction `RGB2Gray` en calculant une moyenne pondérée des trois composantes couleur appelée luminance (les poids correspondent à notre sensibilité aux trois composantes) :

$$I = [0.299 \times R + 0.587 \times G + 0.114 \times B],$$

où :

- R , G et B désignent respectivement les composantes rouge, verte et bleue d'un pixel;

- $[x]$ est l'entier le plus proche de x ;
- I désigne le niveau de gris du pixel.



Testez l'exécutable `rgb2gray` sur l'image `chenille.ppm`.



Vérifiez votre résultat avec l'opérateur `comparison` sur l'image de référence `chenille.pgm`.

4.2 Binarization : binarisation d'une image de niveaux de gris par seuillage global



Écrivez la fonction `Binarization`. Pour chaque pixel, le résultat est 0 si son niveau de gris est strictement inférieur au seuil, 1 sinon.



Testez l'exécutable `binarization` sur l'image `rice.pgm` avec le seuil 118.



Vérifiez votre résultat avec l'opérateur `comparison` sur l'image de référence `rice-bin118.pbm`.

4.3 Inversion : inversion (obtention du négatif) d'une image noir et blanc (binaire), en niveaux de gris ou en couleur



Écrivez la fonction `Inversion` en appliquant un traitement différent selon le type de l'image.



Testez l'exécutable `inversion` sur les images `chenille.pbm`, `chenille.pgm` et `chenille.ppm`.



Vérifiez vos résultats avec l'opérateur `comparison` sur les images de référence correspondantes `chenille-inv.pbm`, `chenille-inv.pgm` et `chenille-inv.ppm`.

4.4 Histogram : calcul de l'histogramme d'une image de niveaux de gris



Écrivez la fonction `Histogram` qui retourne l'histogramme sous la forme d'une `Matrix` d'éléments de type `int` de taille 1×256 (1 ligne et 256 colonnes).



Testez l'exécutable `histogram` sur l'image `chenille.pgm`.



Vérifiez votre résultat avec l'opérateur `comparison` sur l'histogramme de référence `chenille-hist.mx`.



Visualiser sous la forme d'une image de niveaux de gris cet histogramme en utilisant l'opérateur `hist2im` qui est déjà programmé et donc prêt à l'emploi.

4.5 Otsu : calcul du seuil d'Otsu



Écrivez la fonction `Otsu` qui retourne le niveau de gris qui est le seuil optimal selon le critère d'Otsu.



Testez l'exécutable `otsu` sur l'histogramme de référence `rice-hist.mx`.



Vous devez obtenir le niveau de gris 132.



Utilisez l'opérateur `binarization` pour binariser l'image `rice.pgm` avec ce seuil.

4.6 📖 Hist2CumHist : calcul de l'histogramme cumulé à partir de l'histogramme

- ✍ Écrivez la fonction `Hist2CumHist` qui retourne l'histogramme cumulé sous la forme d'une `Matrix` de `int` de taille 1×256 (1 ligne et 256 colonnes).
- ☢ Testez l'exécutable `hist2cumhist` sur l'histogramme `moon-hist.mx`.
- 🔍 Vérifiez votre résultat avec l'opérateur `comparison` sur l'histogramme cumulé de référence `moon-hist-cum.mx`.

4.7 📖 AppLUT : application d'une transformation ponctuelle à une image de niveaux de gris

- ✍ Écrivez la fonction `AppLUT` qui crée et retourne l'image résultat de l'application de la transformation ponctuelle.
- ☢ Testez l'exécutable `applut` sur l'image `chenille.pgm` avec la transformation ponctuelle `lut-inv.mx`.
- 🔍 Vérifiez votre résultat avec l'opérateur `comparison` sur l'image inversée de référence `chenille-inv.pgm`.

4.8 📖 HistSpecif : spécification d'histogramme

- ✍ Écrivez la fonction `HistSpecif` qui retourne la transformation ponctuelle obtenue par l'algorithme de spécification d'histogramme.
- ☢ Testez l'exécutable `histspecif` avec l'histogramme cumulé `moon-hist-cum.mx` de l'image `moon.pgm` et l'histogramme cumulé désiré `hist-cum-unif-262144.mx` correspondant à l'histogramme uniforme pour le nombre de pixels de l'image `moon.pgm`.
- 🔍 Vérifiez votre résultat avec l'opérateur `comparison` sur la transformation ponctuelle de référence `moon-lut-algo-egal.mx`.


4.9 📖 Erosion : érosion d'une image binaire

- ✍ Écrivez la fonction `Erosion`. L'élément structurant est représenté par une `Matrix` de type `Int` contenant uniquement des 0 et des 1 et dont les nombres de lignes et de colonnes sont impairs. Son origine est son centre. La validité de la matrice en argument de l'opérateur est vérifiée dans la fonction `main` par l'appel à la fonction `NotValidBinSE` qui est déjà programmée dans le fichier `tai.c`. Attention, les pixels du bord de l'image doivent être traités en ne considérant que la partie de l'élément structurant qui se trouve à l'intérieur de l'image. Cette manière de traiter les pixels du bord nécessite une gestion fine des indices des boucles.
- ☢ Testez l'exécutable `erosion` sur l'image `im-bin.pbm` avec l'élément structurant `carre3x3.mx` (les fichiers au format `Matrix` sont du texte lisible avec un éditeur ou avec la commande `cat`).
- 🔍 Vérifiez votre résultat avec l'opérateur `comparison` sur l'image de référence `im-bin-ero-carre3.pbm`.

⚠ **Attention**
Il ne faut pas confondre l'érosion et la transformation « tout ou rien ».

- 🔍 Pour vérifier que vous n'avez pas fait la confusion, appliquez l'érosion à l'image `im-bin.pbm` avec l'élément

structurant `croix3x3.mx`.

 Vérifiez votre résultat avec l'opérateur `comparison` sur l'image de référence `im-bin-ero-croix3.pbm`.



4.10 Pour aller plus loin

Si vous avez terminé la mise au point des opérateurs demandés, vous pouvez ajouter un opérateur d'étiquetage des composantes connexes. Vous pouvez appliquer cet étiquetage à l'image `rice-ouv-disque7.pbm`.

Remarque

L'algorithme classique pour gérer les étiquettes équivalentes est *union-find* : <https://fr.wikipedia.org/wiki/Union-find>.