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;
struct  t_Image
{
  // largeur de l'image
  int w;
  // hauteur de l'image
  int h;
  // tableau des niveaux de gris de l'image
  unsigned int im[TMAX][TMAX];
 };
```

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

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 de 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. Voici leur spécification :

Spécification
Fonction loadPgm(t_Image im, chaîne de caractère)
Paramètres d'entrée : chaîne de caractères nom
Paramètres d'entrée/sortie : pointeur sur une t_Image im
Résultat : booléen
Problème : charge en mémoire une `t_Image` pour la traiter et renvoie un booléen lorsque le chargement est effectué

Spécification
Fonction savePgm(t_Image im, chaîne de caractère)
Paramètres d'entrée : chaîne de caractères nom
Paramètres d'entrée/sortie : pointeur sur une t_Image im
Résultat : booléen
Problème : enregistre la `t_Image` renvoie un booléen lorsque la sauvegarde est effectuée

Le code associé est contenu dans les fichiers `chargesauve.h` et `chargesauve.cpp`. Vous noterez que 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 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 le tas et non pas dans la pile du programme, là où il y a sensiblement plus d'espaces. Pour ce TP, vous devrez utiliser uniquement des pointeurs sur les images et les opérateurs d'allocation dynamique `new` et `delete`. Pensez aussi à utiliser les préconditions partout où c'est possible (sous forme d'assertions).

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).

3.2 Différence entre deux images

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

3.3 Travail à faire

Question 1 Écrire les **algorithmes** puis les **fonctions** de seuillage et de différence dans des fichiers `outils.h` et `outils.cpp`. Profitez-en pour écrire une fonction `main()` de test de vos outils et n'oubliez pas de créer un `Makefile` pour la compilation. Fournissez des jeux d'essais dans vos comptes rendus.

4 Dilatation/Érosion d'images noir et blanc

Les opérations de **dilatation** et d'**érosion** sont des opérations de filtrage morphologique. Ce sont des opérations fondamentales pour analyser et transformer les formes présentes dans une image binaire. Dans une image binaire, la distinction entre *fond* (zones noires, souvent associées à la valeur 0) et *forme* (zones blanches, associées à la valeur 1) est essentielle. Les opérations morphologiques comme l'érosion et la dilatation modifient uniquement la forme (la zone blanche). .
Ces opérations s'appuient sur un élément, appelé *élément structurant*, un motif de pixels prédéfini, qui agit comme une sorte de "filtre" pour modifier la forme des objets dans l'image. Les caractéristiques importantes de cet élément structurant sont ses dimensions, la position de son centre et ses valeurs (que l'on prendra égales à 0 ou à 1). La forme de l'élément structurant est définie par ses éléments à 1. L'élément structurant peut avoir différentes formes et

tailles (par exemple un carré, un disque, une croix). Il joue un rôle clé dans la transformation des objets, car il définit les critères selon lesquels les pixels seront ajoutés ou supprimés. Un élément structurant plus grand aura un effet plus radical sur la taille et la forme des objets dans l'image. On ne considérera ici que des éléments structurants de taille impaire et carrés. La figure 4.1 donne un exemple d'élément structurant de taille 3x3.

FIGURE 4.1 – Exemple d'éléments structurants de taille 3x3 avec le centre placé en $(1,1)$.

4.1 DILATATION

Pour effectuer une dilatation, il faut déplacer l'élément structurant sur l'image à dilater. La dilatation permet d'"agrandir" les objets en ajoutant des pixels autour des contours. Ici, l'élément structurant est é déplacé sur toute l'image, mais un pixel est ajouté à la forme dès qu'une partie de l'élément structurant chevauche un pixel de la forme dans l'image originale. Cela a pour effet de combler de petits trous dans les objets, d'élargir les formes, ou de connecter des éléments proches. Quand la forme de l'élément structurant a une intersection non nulle avec la forme de l'image, la position correspondant au centre de l'élément structurant est allumée (ie. sa valeur est mise à 255).

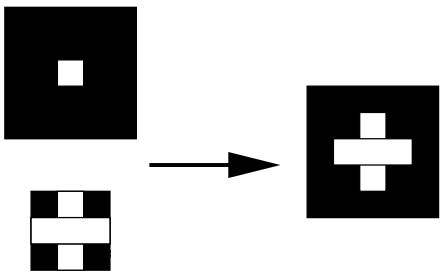

FIGURE 4.2 – Exemple de dilatation avec un élément structurant 3×3. Ici le fond est noir (0) et la forme blanche (1).

4.2 ÉROSION

À l'inverse, l'érosion consiste à réduire les objets dans l'image en supprimant des pixels sur leurs contours. L'élément structurant est "balayé" de même sur toute l'image, et pour chaque pixel de la forme (zones blanches, de valeur 1), l'érosion ne conserve le pixel correspondant au centre de l'élément structurant dans l'image de sortie que si l'élément structurant tient

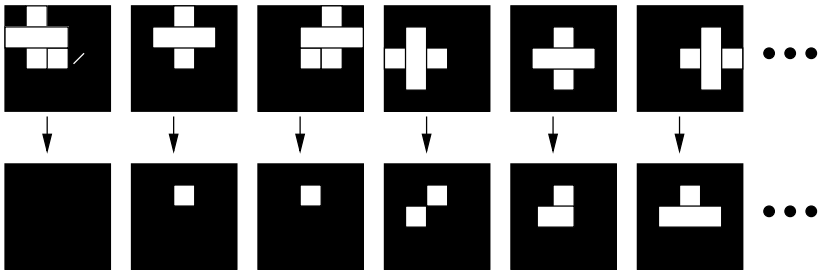

FIGURE 4.3 – Début de dilatation d'une image par un élément structurant. On ne tient pas compte des bords de l'image.

entièrement dans la forme à cet endroit. Si une partie de l'élément structurant dépasse dans le fond (zones noires, de valeur 0), le pixel est supprimé dans l'image érodée. Cela a pour effet de "rétrécir" les objets, d'éliminer de petits détails ou de séparer des objets connectés.

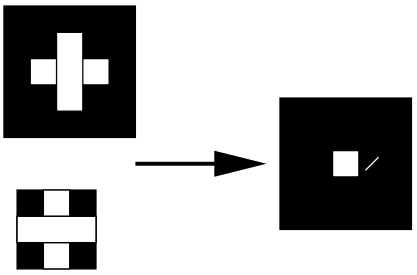

FIGURE 4.4 – Exemple d'érosion avec un élément structurant 3×3. Ici le fond est noir (0) et la forme blanche (1).

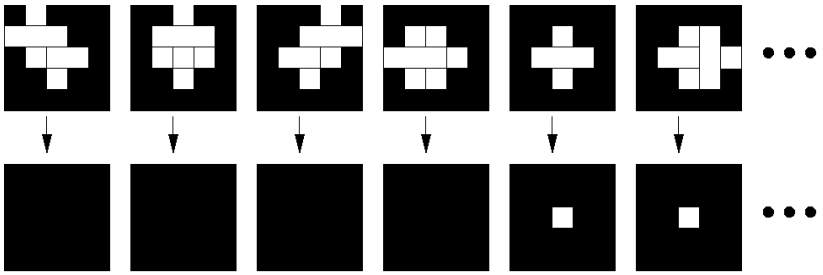

FIGURE 4.5 – Début d'érosion d'une image par un élément structurant. On ne tient pas compte des bords de l'image.

Ces opérations sont souvent combinées pour réaliser des traitements plus complexes, comme l'*ouverture* (érosion suivie de dilatation) pour éliminer le bruit tout en conservant la forme générale, ou la *fermeture* (dilatation suivie d'érosion) pour combler des trous tout en lissant

les contours.

4.3 TRAVAIL À FAIRE

QUESTION 2 Vous devez définir une **structure** décrivant un élément structurant.

QUESTION 3 Écrire les **algorithmes** et le **code** de **deux fonctions** pour la **dilatation** et l'é**rosion** d'une image par un élément structurant qui sera passé en paramètre. Illustrez vos résultats par des jeux d'essais.

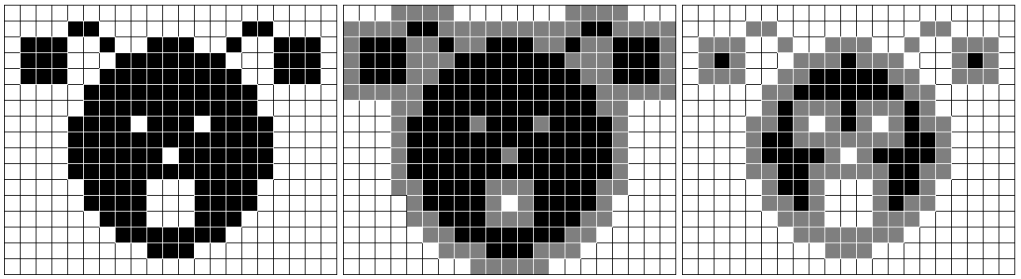

FIGURE 4.6 – Exemple de dilatation et d'érosion d'une image par un carré 3x3. Dans l'image originale à gauche le fond est blanc et la forme est noire. L'élément structurant utilisé est un carré $3x3$. La forme après dilatation (au milieu) est formée des pixels noir et gris. La forme érodée (à droite) n'est formée que des pixels noirs.

QUESTION 4 Écrire les **algorithmes** et le **code** pour permettre les opérations d'**ouverture** et de **fermeture**. Tester ensuite des ouvertures et fermetures sur les images de test. Comparez à l'aide de la fonction de différence les images seuillées et les images qui ont subi des opérations morphologiques. Illustrez vos résultats par des jeux d'essais.
Merci de déposer vos solutions sur hippocampus (code, rapport en pdf et un éventuellement un fichier texte contenant le lien de votre travail GitHub).