Compte Rendu TP2
Opérations morphologies sur des images

Ivan Lejeune

10 février 2024

Table des matières

1 Seuillage d'une image et erosion de l'image binaire

1.1 Choix de l'image

On commence par choisir une image au format *pgm*, dans notre cas, l'image `08.pgm`. On réduit ensuite la taille de l'image pour faciliter le traitement, on choisit une taille de 256x256 pixels. Cela donne alors :

FIGURE 1 – Image originale

FIGURE 2 – Image redimensionnée

1.2 Seuillage de l'image

On commence par modifier le programme `test_grey.cpp` pour que les objets soient noirs et le fond blanc. On peut ensuite tester le seuillage de l'image avec différents seuils. Le plus pertinent semble être un seuil de 80, cela donne alors :

FIGURE 3 – Image redimensionnée

FIGURE 4 – Image modifiée avec un seuil de 80

1.3 Erosion de l'image binaire

On commence par créer un programme `erosion.cpp` pour effectuer l'érosion de l'image. Cela consiste à parcourir l'image et à remplacer chaque pixel par le maximum des pixels voisins.

L'essentiel du code est le suivant :

```
// initialize ImgOut to ImgIn
for (int i = 0; i < nTaille; i++) {
    ImgOut[i] = ImgIn[i];
}

// excluding edge cases
for (int i = 1; i < nH-1; i++)
    for (int j = 1; j < nW-1; j++) {
        // verify if the pixel has a white pixel in its 8-neighborhood
        if (
                ImgIn[(i-1) * nW + j] == 255 || // top
                ImgIn[(i-1) * nW + (j-1)] == 255 || // top-left
                ImgIn[(i-1) * nW + (j+1)] == 255 || // top-right
                ImgIn[i * nW + (j-1)] == 255 ||  // left
                ImgIn[i * nW + j] == 255 ||  // center
                ImgIn[i * nW + (j+1)] == 255 || // right
                ImgIn[(i+1) * nW + j] == 255 || // bottom
                ImgIn[(i+1) * nW + (j-1)] == 255 || // bottom-left
                ImgIn[(i+1) * nW + (j+1)] == 255) { // bottom-right
            ImgOut[i * nW + j] = 255;
        } else {
            ImgOut[i * nW + j] = 0;
        }
    }
```

FIGURE 5 – Code de l'érosion

On peut ensuite tester l'érosion de l'image avec l'image seuillée précédemment. Cela donne alors :

FIGURE 6 – Image modifiée avec un seuil de 80

FIGURE 7 – Image modifiée avec l'érosion

2 Seuillage d'une image et dilatation de l'image binaire

2.1 Dilatation de l'image binaire

On commence par créer un programme `dilatation.cpp` pour effectuer la dilatation de l'image. Cela consiste à parcourir l'image et à remplacer chaque pixel par le minimum des pixels voisins.

L'essentiel du code est le suivant :

```cpp
// initialize ImgOut to ImgIn
for (int i = 0; i < nTaille; i++) {
    ImgOut[i] = ImgIn[i];
}

// excluding edge cases
for (int i = 1; i < nH-1; i++)
    for (int j = 1; j < nW-1; j++) {
        // verify if the pixel has a black pixel in its 8-neighborhood
        if (
            ImgIn[(i-1) * nW + j] == 0 || // top
            ImgIn[(i-1) * nW + (j-1)] == 0 || // top-left
            ImgIn[(i-1) * nW + (j+1)] == 0 || // top-right
            ImgIn[i * nW + (j-1)] == 0 ||  // left
            ImgIn[i * nW + j] == 0 ||  // center
            ImgIn[i * nW + (j+1)] == 0 || // right
            ImgIn[(i+1) * nW + j] == 0 || // bottom
            ImgIn[(i+1) * nW + (j-1)] == 0 || // bottom-left
            ImgIn[(i+1) * nW + (j+1)] == 0) { // bottom-right
            ImgOut[i * nW + j] = 0;
        } else {
            ImgOut[i * nW + j] = 255;
        }
    }
```

FIGURE 8 – Code de la dilatation

On peut ensuite tester la dilatation de l'image avec l'image seuillée précédemment. Cela donne alors :

FIGURE 9 – Image modifiée avec un seuil de 80

FIGURE 10 – Image modifiée avec la dilatation

3 Fermeture et ouverture d'une image et de l'image binaire

3.1 Fermeture de l'image binaire

On commence par créer un programme `fermeture.cpp` pour effectuer la fermeture de l'image. Cela consiste à effectuer une dilatation de l'image puis une érosion de l'image. Cela permet de fermer les trous dans les objets de l'image.

L'essentiel du code est le suivant :

```
// initialize ImgOut to ImgIn
for (int i = 0; i < nTaille; i++) {
    ImgOut[i] = ImgIn[i];
}
// create temporary image
for (int i = 0; i < nTaille; i++) {
    ImgTmp[i] = ImgIn[i];
}
// create temporary image name
char cNomImgTmp[250];
sprintf( Buffer: cNomImgTmp,  Format: "%s-tmp.pgm", cNomImgEcrite);

// dilatation
dilatation(ImgIn,  ImgOut: ImgTmp, nH, nW,  cNomImgEcrite: cNomImgTmp);
// erosion
erosion( ImgIn: ImgTmp, ImgOut, nH, nW, cNomImgEcrite);
```

FIGURE 11 – Code de la fermeture

L'essentiel du travail est de modifier les programmes précédents pour qu'ils puissent être utilisés dans ce programme. Pour cela, on crée des nouveaux fichiers sans *main()* et extrait le contenu d'`image_ppm.h` dans un fichier *cpp*.

On peut ensuite tester la fermeture de l'image avec l'image seuillée précédemment. Cela donne alors :

FIGURE 12 – Image modifiée avec un seuil de 80

FIGURE 13 – Image modifiée avec la fermeture

3.2 Ouverture de l'image binaire

On commence par créer un programme `ouverture.cpp` pour effectuer l'ouverture de l'image. Cela consiste à effectuer une érosion de l'image puis une dilatation de l'image. Cela permet de supprimer les petits objets de l'image.

L'essentiel du code est le suivant :

```
// initialize ImgOut to ImgIn
for (int i = 0; i < nTaille; i++) {
    ImgOut[i] = ImgIn[i];
}
// create temporary image
for (int i = 0; i < nTaille; i++) {
    ImgTmp[i] = ImgIn[i];
}
// create temporary image name
char cNomImgTmp[250];
sprintf( Buffer: cNomImgTmp,  Format: "%s-tmp.pgm", cNomImgEcrite);

// erosion
erosion(ImgIn,  ImgOut: ImgTmp, nH, nW,  cNomImgEcrite: cNomImgTmp);
// dilatation
dilatation( ImgIn: ImgTmp, ImgOut, nH, nW, cNomImgEcrite);
```

FIGURE 14 – Code de l'ouverture

On peut ensuite tester l'ouverture de l'image avec l'image seuillée précédemment. Cela donne alors :

FIGURE 15 – Image modifiée avec un seuil de 80

FIGURE 16 – Image modifiée avec l'ouverture

3.3 Enchainement de fermeture et ouverture

On peut ensuite enchaîner la fermeture et l'ouverture de l'image pour obtenir une image plus propre. Cela donne alors :

FIGURE 17 – Image modifiée avec un seuil de 80

FIGURE 18 – Image modifiée avec fermeture

FIGURE 19 – Image modifiée avec fermeture puis ouverture

3.4 Impact cumulatif de la fermeture et de l'ouverture

On peut ensuite tester l'impact cumulatif de la fermeture et de l'ouverture de l'image. On procède d'abord à 3 dilatations, 6 érosions et enfin 3 dilatations.

Cela donne alors :

FIGURE 20 – Image modifiée avec un seuil de 80

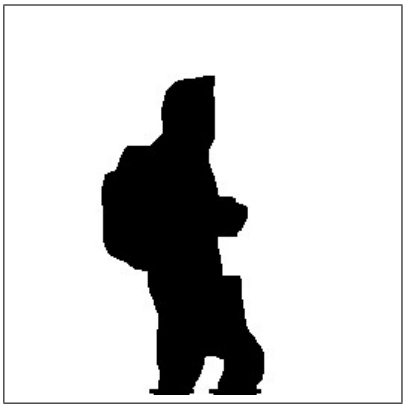

FIGURE 21 – Image modifiée avec cumul

4 Segmentation d'une image

4.1 Detection de contours

On commence par créer un programme `difference.cpp` pour effectuer la détection de contours de l'image. Cela consiste à effectuer une dilatation de l'image puis d'observer la différence entre l'image dilatée et l'image originale.

L'essentiel du code est le suivant :

```
allocation_tableau(ImgIn1, OCTET, nTaille);
lire_image_pgm( nom_image: cNomImgLue1,  pt_image: ImgIn1,  taille_image: nH * nW);
allocation_tableau(ImgIn2, OCTET, nTaille);
lire_image_pgm( nom_image: cNomImgLue2,  pt_image: ImgIn2,  taille_image: nH * nW);
allocation_tableau(ImgOut, OCTET, nTaille);

// initialize ImgOut to ImgIn
for (int i = 0; i < nTaille; i++) {
    ImgOut[i] = ImgIn1[i];
}

// excluding edge cases
for (int i = 1; i < nH-1; i++)
    for (int j = 1; j < nW-1; j++) {
        // verify if both pixels are different
        if (ImgIn1[i * nW + j] != ImgIn2[i * nW + j]) {
            ImgOut[i * nW + j] = 0;
        } else {
            ImgOut[i * nW + j] = 255;
        }
    }

ecrire_image_pgm( nom_image: cNomImgEcrite,  pt_image: ImgOut,  nb_lignes: nH,  nb_colonnes: nW);
```

FIGURE 22 – Code de la détection de contours

On peut ensuite tester la détection de contours de l'image avec l'image seuillée précédemment. Cela donne alors :

FIGURE 23 – Image modifiée avec un seuil de 80

FIGURE 24 – Image modifiée avec la détection de contours

5 Bonus : extension aux images en niveaux de gris

5.1 Erosion de l'image en niveaux de gris

On commence par créer un programme `erosion_grey.cpp` pour effectuer l'érosion de l'image en niveaux de gris. Cela consiste à parcourir l'image et à remplacer chaque pixel par le maximum des pixels voisins.

L'essentiel du code est le suivant :

```
// initialize ImgOut to ImgIn
for (int i = 0; i < nTaille; i++) {
    ImgOut[i] = ImgIn[i];
}

int min;

// excluding edge cases
for (int i = 1; i < nH-1; i++) {
    for (int j = 1; j < nW - 1; j++) {
        // replace pixel with minimum value of its neighbors
        min = 255;
        for (int k = -1; k <= 1; k++)
            for (int l = -1; l <= 1; l++) {
                if (ImgIn[(i + k) * nW + (j + l)] < min) {
                    min = ImgIn[(i + k) * nW + (j + l)];
                }
            }
        ImgOut[i * nW + j] = min;
    }
}
```

FIGURE 25 – Code de l'érosion en niveaux de gris

On peut ensuite tester l'érosion de l'image avec l'image en niveaux de gris `08.pgm`. Cela donne alors :

FIGURE 26 – Image originale

FIGURE 27 – Image modifiée avec l'érosion en niveaux de gris

5.2 Dilatation de l'image en niveaux de gris

On commence par créer un programme `dilatation_grey.cpp` pour effectuer la dilatation de l'image en niveaux de gris. Cela consiste à parcourir l'image et à remplacer chaque pixel par le minimum des pixels voisins.

L'essentiel du code est le suivant :

```
// initialize ImgOut to ImgIn
for (int i = 0; i < nTaille; i++) {
    ImgOut[i] = ImgIn[i];
}

int max;

// excluding edge cases
for (int i = 1; i < nH-1; i++) {
    for (int j = 1; j < nW-1; j++) {
        // replace pixel with maximum value of its neighbors
        max = 0;
        for (int k = -1; k <= 1; k++) {
            for (int l = -1; l <= 1; l++) {
                if (ImgIn[(i+k)*nW + (j+l)] > max) {
                    max = ImgIn[(i+k)*nW + (j+l)];
                }
            }
        }
        ImgOut[i*nW + j] = max;
    }
}
```

FIGURE 28 – Code de la dilatation en niveaux de gris

On peut ensuite tester la dilatation de l'image avec l'image en niveaux de gris `08.pgm`. Cela donne alors :

FIGURE 29 – Image originale

FIGURE 30 – Image modifiée avec la dilatation en niveaux de gris

5.3 Fermeture de l'image en niveaux de gris

On commence par créer un programme `fermeture_grey.cpp` pour effectuer la fermeture de l'image en niveaux de gris. Cela consiste à effectuer une dilatation de l'image puis une érosion de l'image. Cela permet de fermer les trous dans les objets de l'image.

L'essentiel du code est le suivant :

```
// initialize ImgOut to ImgIn
for (int i = 0; i < nTaille; i++) {
    ImgOut[i] = ImgIn[i];
}
// create temporary image
for (int i = 0; i < nTaille; i++) {
    ImgTmp[i] = ImgIn[i];
}
// create temporary image name
char cNomImgTmp[250];
sprintf( Buffer: cNomImgTmp,  Format: "%s-tmp.pgm", cNomImgEcrite);

// dilatation
dilatation_grey(ImgIn,  ImgOut: ImgTmp, nH, nW,  cNomImgEcrite: cNomImgTmp);
// erosion
erosion_grey( ImgIn: ImgTmp, ImgOut, nH, nW, cNomImgEcrite);
```

FIGURE 31 – Code de la fermeture en niveaux de gris

On peut ensuite tester la fermeture de l'image avec l'image en niveaux de gris `08.pgm`. Cela donne alors :

FIGURE 32 – Image originale

FIGURE 33 – Image modifiée avec la fermeture en niveaux de gris

5.4 Ouverture de l'image en niveaux de gris

On commence par créer un programme `ouverture_grey.cpp` pour effectuer l'ouverture de l'image en niveaux de gris. Cela consiste à effectuer une érosion de l'image puis une dilatation de l'image. Cela permet de supprimer les petits objets de l'image.

L'essentiel du code est le suivant :

```
// initialize ImgOut to ImgIn
for (int i = 0; i < nTaille; i++) {
    ImgOut[i] = ImgIn[i];
}
// create temporary image
for (int i = 0; i < nTaille; i++) {
    ImgTmp[i] = ImgIn[i];
}
// create temporary image name
char cNomImgTmp[250];
sprintf( Buffer: cNomImgTmp,  Format: "%s-tmp.pgm", cNomImgEcrite);

// dilatation
erosion_grey(ImgIn,  ImgOut: ImgTmp, nH, nW,  cNomImgEcrite: cNomImgTmp);
// erosion
dilatation_grey( ImgIn: ImgTmp, ImgOut, nH, nW, cNomImgEcrite);
```

FIGURE 34 – Code de l'ouverture en niveaux de gris

On peut ensuite tester l'ouverture de l'image avec l'image en niveaux de gris `08.pgm`. Cela donne alors :

FIGURE 35 – Image originale

FIGURE 36 – Image modifiée avec l'ouverture en niveaux de gris

5.5 Enchainement de fermeture et ouverture en niveaux de gris

On peut ensuite enchaîner la fermeture et l'ouverture de l'image pour obtenir une image plus propre. Cela donne alors :

FIGURE 37 – Image originale

FIGURE 38 – Image modifiée avec fermeture

FIGURE 39 – Image modifiée avec fermeture puis ouverture

5.6 Impact cumulatif de la fermeture et de l'ouverture en niveaux de gris

On peut ensuite tester l'impact cumulatif de la fermeture et de l'ouverture de l'image en niveaux de gris. On procède d'abord à 3 dilatations, 6 érosions et enfin 3 dilatations.

Cela donne alors :

FIGURE 40 – Image originale

FIGURE 41 – Image modifiée avec cumul

5.7 Segmentation d'une image en niveaux de gris

On peut directement réutiliser le programme `difference.cpp` pour effectuer la segmentation de l'image en niveaux de gris.

On peut ensuite tester la segmentation de l'image avec l'image en niveaux de gris `08.pgm`. Cela donne alors :

FIGURE 42 – Image originale

FIGURE 43 – Image modifiée avec la difference en niveaux de gris

On constate que cela donne un résultat très surprenant, on peut alors essayer d'adapter le programme pour qu'il soit plus pertinent. On décide donc de regarder si la différence entre deux pixels est supérieure à un certain seuil. Le programme `difference_grey_seuil.cpp` est ainsi créé pour un meilleur résultat.

Le code est le suivant :

```
// initialize ImgOut to ImgIn
for (int i = 0; i < nTaille; i++) {
    ImgOut[i] = ImgIn1[i];
}

// excluding edge cases
for (int i = 1; i < nH-1; i++)
    for (int j = 1; j < nW-1; j++) {
        // verify if difference is greater than threshold
        if (abs( X ImgIn1[i * nW + j] - ImgIn2[i * nW + j]) > seuil) {
            ImgOut[i * nW + j] = 0;
        } else {
            ImgOut[i * nW + j] = 255;
        }
    }
```

FIGURE 44 – Code de la différence en niveaux de gris avec seuil

On peut ensuite tester la segmentation de l'image avec l'image en niveaux de gris `08.pgm`. Pour des différents seuils, on obtient les transformations suivantes :

FIGURE 45 – Image originale

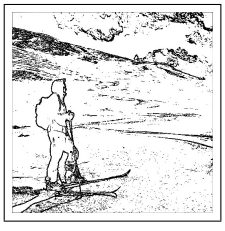

FIGURE 46 – Image modifiée avec seuil de 10

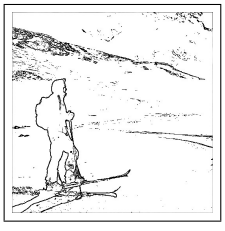

FIGURE 47 – Image modifiée avec seuil de 20

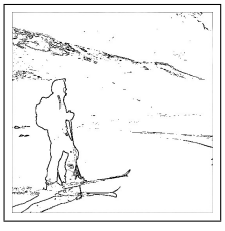

FIGURE 48 – Image modifiée avec seuil de 30

6 Conclusion

En conclusion, on a pu voir que les opérations morphologiques permettent de modifier les images de manière intéressante. On a pu voir que l'érosion et la dilatation permettent de modifier les images binaires, la fermeture et l'ouverture permettent de nettoyer les images binaires et la détection de contours permet de segmenter les images en niveaux de gris. On a pu voir que l'extension aux images en niveaux de gris est possible et permet de segmenter les images de manière intéressante.