

Compte Rendu TP1

Prise en main d'une librairie de traitement d'images

Ivan Lejeune

7 février 2024

1 Seuillage d'une image au format pgm

1.1 Ouverture des fichiers

On commence par télécharger les fichiers se trouvant à https://www.lirmm.fr/~wpuech/enseignement/donnees_multimedia/librairie/. Ensuite, on les ouvre dans l'éditeur de texte voulu, dans notre cas, on se servira de *CLion* (et parfois de *VSCodium*).

1.2 Téléchargement des fichiers nécessaires

On télécharge les fichiers depuis https://www.lirmm.fr/~wpuech/enseignement/donnees_multimedia/images/.

C'est parmi ces images qu'on effectuera la majorité du travail, notamment avec *08.pgm* et *peppers.pgm*.

1.3 Compilation, exécution et test

On commence par assurer d'avoir placé les fichiers `test_grey.cpp` et `image_pgm.cpp` dans le même répertoire.

On compile le programme `test_grey.cpp` avec un seuil de 80 pour passer de l'image de gauche à celle de droite :



FIGURE 1 – Image originale



FIGURE 2 – Image modifiée avec un seuil de 80

2 Seuillage d'une image pgm avec plusieurs niveaux

2.1 Seuillage en 3 parties

On reprend le programme `test_grey.cpp` pour le modifier afin de pouvoir effectuer un seuillage en 3 parties. Il suffit de rajouter deux seuils supplémentaires, S2 et S3, et de les appliquer à l'image. Ici, on a pris des seuils à 80, 140 et 200 pour obtenir la transformation suivante :



FIGURE 3 – Image originale



FIGURE 4 – Image modifiée avec 3 seuils

2.2 Seuillage en 4 parties

On reprend le programme `test_grey.cpp` pour le modifier afin de pouvoir effectuer un seuillage en 4 parties. Il suffit de rajouter un seuil supplémentaire, S4, et de l'appliquer à l'image. Ici, on a pris des seuils à 40, 100, 160 et 220 pour obtenir la transformation suivante :



FIGURE 5 – Image originale



FIGURE 6 – Image modifiée avec 4 seuils

3 Histogramme d'une image pgm

3.1 Développement du programme

Pour cette partie, on a développé un programme `histo.cpp` qui permet de générer l'histogramme d'une image pgm. Les données sont d'abord affichées dans la console, puis on les enregistre dans un fichier `histo.dat` pour pouvoir les visualiser avec *gnuplot*. Le nom du fichier de sortie peut être rajouté en argument lors de l'exécution du programme.

L'essentiel du code est le suivant :

```
// initialize tab to 0
int tab[256];
for (int i = 0; i < 256; i++) {
    tab[i] = 0;
}

// Open file for writing
FILE *fp = fopen("cNomFicSort", "w+");

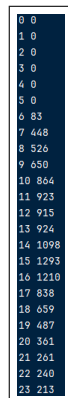
// Update array
for (int i = 0; i < nTaille; i++) {
    tab[ImgIn[i]] += 1;
}

// Output data
for (int i = 0; i < 256; i++) {
    // printf("%d %d\n", i, tab[i]);
    fprintf(fp, "%d %d\n", i, tab[i]);
}
```

FIGURE 7 – Code de `histo.cpp`

3.2 Résultats

On a utilisé l'image *08.pgm* pour tester le programme. Cela donne les résultats suivants :



0	0
1	0
2	0
3	0
4	0
5	0
6	83
7	448
8	326
9	450
10	864
11	923
12	915
13	924
14	1098
15	1293
16	1210
17	838
18	659
19	487
20	361
21	261
22	240
23	215

FIGURE 8 – Données de l'histogramme

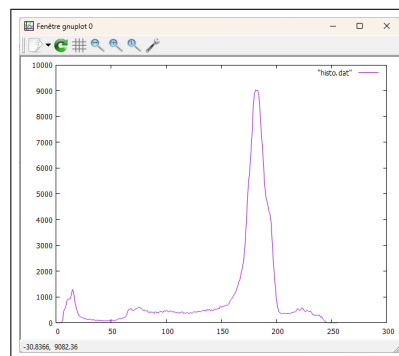


FIGURE 9 – Histogramme de l'image

4 Profil d'une ligne ou d'une colonne d'une image pgm

On a développé un programme `profil.cpp` qui permet de générer le profil d'une ligne ou d'une colonne d'une image pgm. Les données sont d'abord affichées dans la console, puis on les enregistre dans un fichier `profil.dat` pour pouvoir les visualiser avec *gnuplot*. Le nom du fichier de sortie peut être rajouté en argument lors de l'exécution du programme.

L'essentiel du code est le suivant :

```
// Open file for writing
FILE *fp = fopen(FileName: cNomFicSort, Mode: "w+");

// Output data
if (l_or_c[0] == 'l') { // if l_or_c is 'l' then we get the values of the specific line
    for (int i = 0; i < nW; i++) {
        // printf("%d %d\n", i, ImgIn[num * nW + i]);
        fprintf(Stream: fp, Format: "%d %d\n", i, ImgIn[num * nW + i]);
    }
} else { // if l_or_c is 'c' then we get the values of the specific column
    for (int i = 0; i < nH; i++) {
        // printf("%d %d\n", i, ImgIn[num * nW + i]);
        fprintf(Stream: fp, Format: "%d %d\n", i, ImgIn[i * nW + num]);
    }
}
```

FIGURE 10 – Code de `profil.cpp`

4.1 Résultats

On a utilisé l'image *08.pgm* pour tester le programme sur la 200-ième ligne. Cela donne les résultats suivants :

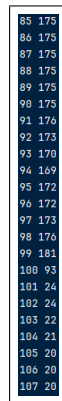


FIGURE 11 – Données de l'histogramme

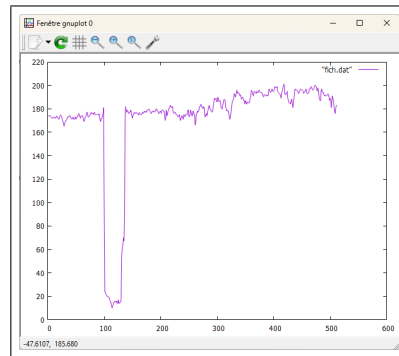


FIGURE 12 – Histogramme de l'image

5 Seuillage d'une image couleur (ppm)

On reprend le programme `test_couleur.cpp` pour le modifier afin de pouvoir effectuer un seuillage en 3 parties. Il suffit de rajouter deux seuils supplémentaires, `S_G` et `S_B`, et de les appliquer à l'image.

L'essentiel du code est le suivant :

```
for (int i = 0; i < nTaille3; i += 3) {  
    nR = ImgIn[i];  
    nG = ImgIn[i + 1];  
    nB = ImgIn[i + 2];  
    if (nR < S_R) ImgOut[i] = 0; else ImgOut[i] = 255;  
    if (nG < S_G) ImgOut[i + 1] = 0; else ImgOut[i + 1] = 255;  
    if (nB < S_B) ImgOut[i + 2] = 0; else ImgOut[i + 2] = 255;  
}
```

FIGURE 13 – Code de `test_couleur.cpp`

On a utilisé l'image *peppers.ppm* pour tester le programme avec des seuils à 80, 140 et 200 pour obtenir la transformation suivante :



FIGURE 14 – Image originale



FIGURE 15 – Image modifiée

6 Histogrammes de 3 composantes d'une image couleur (ppm)

Pour cette partie, on a développé un programme `histo_couleur.cpp` qui permet de générer l'histogramme d'une image ppm. Les données sont d'abord affichées dans la console, puis on les enregistre dans un fichier `histo-col.dat` pour pouvoir les visualiser avec *gnuplot*. Le nom du fichier de sortie peut être rajouté en argument lors de l'exécution du programme.

L'essentiel du code est le suivant :

```
// initialize tabs to 0
int tabR[256], tabG[256], tabB[256];
for (int i = 0; i < 256; i++) {
    tabR[i] = 0;
    tabG[i] = 0;
    tabB[i] = 0;
}

// Open file for writing
FILE *fp = fopen( FileName: cNomFicSort, Mode: "w+");

// Update array
for (int i = 0; i < nTaille3; i += 3) {
    tabR[ImgIn[i]]++;
    tabG[ImgIn[i+1]]++;
    tabB[ImgIn[i+2]]++;
}

// Output data
for (int i = 0; i < 256; i++) {
    // printf("%d %d %d %d\n", i, tabR[i], tabG[i], tabB[i]);
    fprintf( Stream: fp, Format: "%d %d %d %d\n", i, tabR[i], tabG[i], tabB[i]);
}
```

FIGURE 16 – Code de `histo_couleur.cpp`

On a utilisé l'image *peppers.ppm* pour tester le programme. Cela donne les résultats suivants :

```
0 0 191 143
1 0 155 84
2 0 251 147
3 0 1625 256
4 0 755 2182
5 0 533 1115
6 0 610 848
7 1 345 438
8 1 207 279
9 1 246 292
10 1 267 264
11 2 340 292
12 3 487 274
13 4 328 357
14 9 251 458
15 15 222 382
16 18 226 287
17 36 189 297
18 63 195 296
19 40 210 322
20 34 217 343
21 42 204 389
22 37 244 405
```

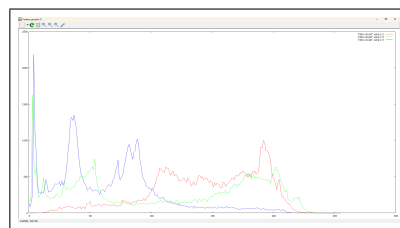


FIGURE 18 – Histogramme de l'image

FIGURE 17 – Données de l'histogramme