Compte Rendu TP3
Filtre inverse vidéeo et floutages d'images

Ivan Lejeune

11 février 2024

Table des matières

1 Choix des images, histogramme et profil de ligne **2**
1.1 Choix de l'image 2
1.2 Histogramme 2

2 Inverse vidéo **3**
2.1 Comparaison des profils de ligne 3

3 Filtre flou 1 **4**
3.1 Programme 4
3.2 Comparaison des profils de ligne 5

4 Filtre flou 2 **6**
4.1 Programme 6
4.2 Floutage répété 7
4.3 Comparaison des profils de ligne 8
4.4 Comparaison des histogrammes 9

5 Floutage de l'image couleur **10**
5.1 Programme 10
5.2 Comparaison des histogrammes 10

1 Choix des images, histogramme et profil de ligne

1.1 Choix de l'image

On commence par choisir une image au format *ppm*, dans notre cas, l'image `peppers.pgm`. On la transforme ensuite en *pgm*. Cela donne alors :

FIGURE 1 – Image originale

FIGURE 2 – Image en niveaux de gris

1.2 Histogramme

On commence par calculer l'histogramme de l'image. Et ensuite le profil de ligne de l'image. On choisit ici la ligne 30. Cela donne :

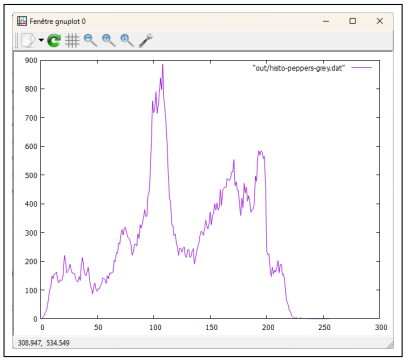

FIGURE 3 – Histogramme de l'image en niveaux de gris

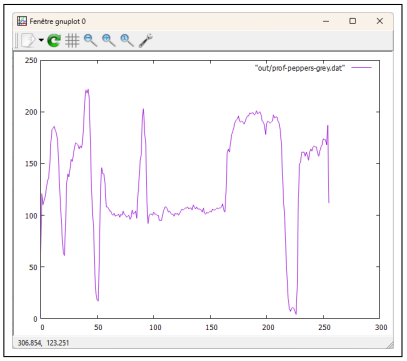

FIGURE 4 – Profil de ligne de l'image en niveaux de gris

2 Inverse vidéo

On commence par créer le programme `inverse.cpp`. L'essentiel du code est le suivant :

```
for (int i = 0; i < nH; i++){
    for (int j = 0; j < nW; j++) {
        ImgOut[i * nW + j] = 255 - ImgIn[i * nW + j];
    }
}
```

FIGURE 5 – Code du programme `inverse.cpp`

On peut alors appliquer le programme à l'image `peppers.pgm`. Cela donne :

FIGURE 6 – Image originale

FIGURE 7 – Image inversée

2.1 Comparaison des profils de ligne

On compare ensuite le profil de ligne de l'image originale et de l'image inversée. Cela donne :

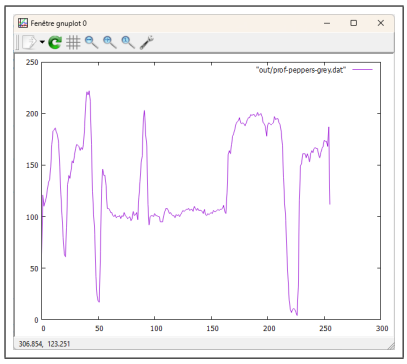

FIGURE 8 – Profil de ligne de l'image originale

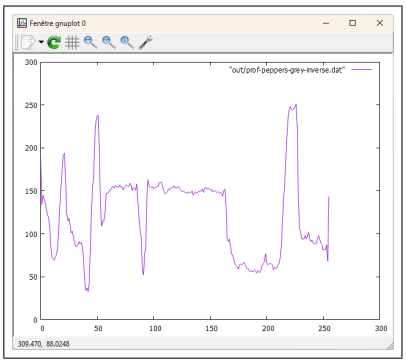

FIGURE 9 – Profil de ligne de l'image inversée

On remarque que le profil de ligne de l'image inversée est l'opposé de celui de l'image originale.

3 Filtre flou 1

3.1 Programme

On commence par créer le programme `flou1.cpp`. L'essentiel du code est le suivant :

```
// initialise output image to be the same as input
for (int i = 0; i < nTaille; i++) {
    ImgOut[i] = ImgIn[i];
}

// apply the filter
for (int i = 1; i < nH-1; i++){
    for (int j = 1; j < nW-1; j++) {
        // p_out(i,j)= ( p(i,j)+p(i-1,j)+p(i+1,j)+p(i,j-1)+p(i,j+1) )/5.
        ImgOut[i * nW + j] = (ImgIn[i * nW + j]
                + ImgIn[(i-1) * nW + j]
                + ImgIn[(i+1) * nW + j]
                + ImgIn[i * nW + j-1]
                + ImgIn[i * nW + j+1]) / 5;
    }
}
```

FIGURE 10 – Code du programme `filtre_flou1.cpp`

On peut alors appliquer le programme à l'image `peppers.pgm`. Cela donne :

FIGURE 11 – Image originale

FIGURE 12 – Image floutée

3.2 Comparaison des profils de ligne

On compare ensuite le profil de ligne de l'image originale et de l'image floutée. Cela donne :

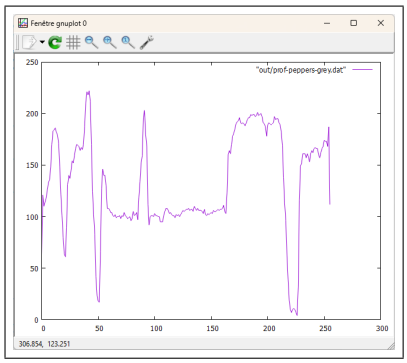

FIGURE 13 – Profil de ligne de l'image originale

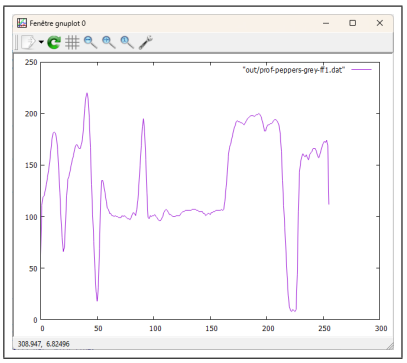

FIGURE 14 – Profil de ligne de l'image floutée

On remarque que le profil de ligne de l'image floutée est plus lisse que celui de l'image originale.

4 Filtre flou 2

4.1 Programme

On commence par créer le programme `flou2.cpp`. L'essentiel du code est le suivant :

```
// initialise output image to be the same as input
for (int i = 0; i < nTaille; i++) {
    ImgOut[i] = ImgIn[i];
}


for (int i = 1; i < nH-1; i++){
    for (int j = 1; j < nW-1; j++) {
        // p_out(i,j)= ( p(i,j)+p(i-1,j)+p(i+1,j)+p(i,j-1)+p(i,j+1) )/5.
        ImgOut[i * nW + j] =
                (ImgIn[i * nW + j] + ImgIn[(i-1) * nW + j] + ImgIn[(i+1) * nW + j]
                +ImgIn[i * nW + j+1] + ImgIn[(i-1) * nW + j+1] + ImgIn[(i+1) * nW + j+1]
                +ImgIn[i * nW + j-1] + ImgIn[(i-1) * nW + j-1] + ImgIn[(i+1) * nW + j-1])/9;
    }
}
```

FIGURE 15 – Code du programme `filtre_flou2.cpp`

On peut alors appliquer le programme à l'image `peppers.pgm`. Cela donne :

FIGURE 16 – Image originale

FIGURE 17 – Image floutée

4.2 Floutage répété

On applique ensuite le programme `flou2.cpp` à l'image floutée plusieurs fois. Cela donne :

FIGURE 18 – Image floutée deux fois

FIGURE 19 – Image floutée cinq fois

4.3 Comparaison des profils de ligne

On compare ensuite le profil de ligne de l'image originale et de toutes les images floutées. Cela donne :

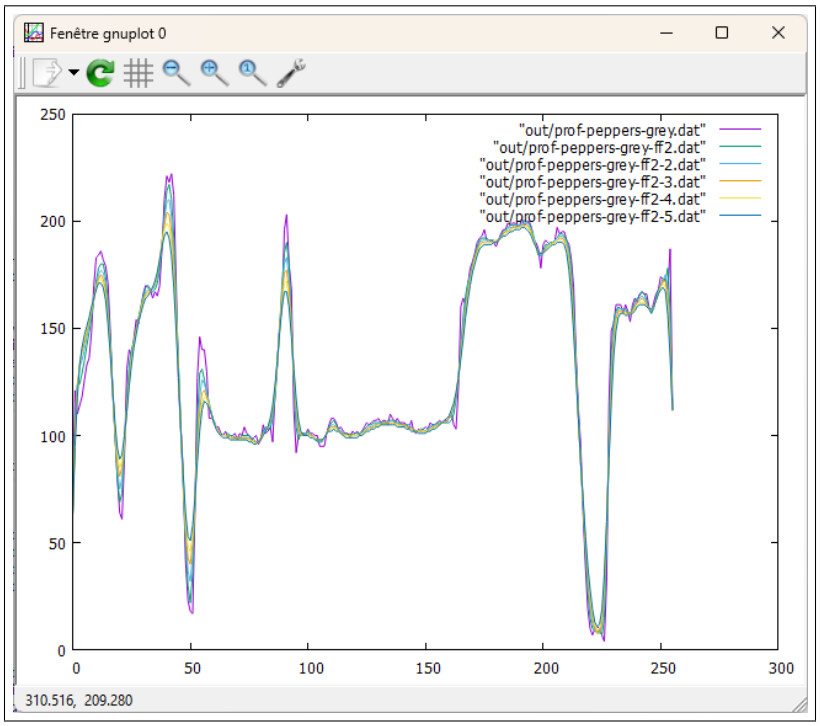

FIGURE 20 – Profil de ligne de l'image floutée

On peut clairement voir ici que plus on floute l'image, plus le profil de ligne devient lisse.

4.4 Comparaison des histogrammes

On compare ensuite les histogrammes de l'image originale et de toutes les images floutées. Cela donne :

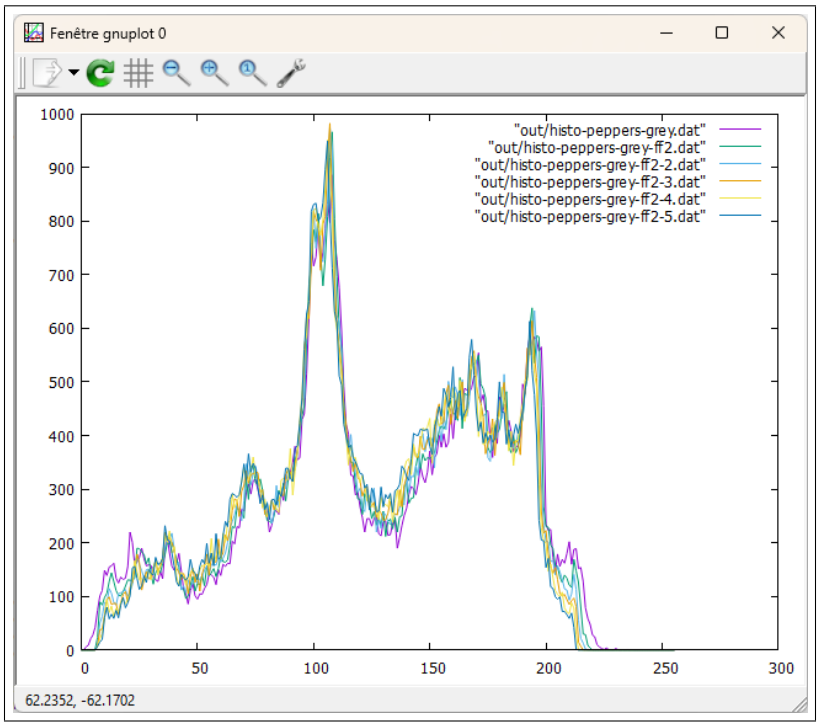

FIGURE 21 – Histogramme de l'image floutée

Comme pour le profil de ligne, on peut clairement voir ici que plus on floute l'image, plus l'histogramme devient lisse.

5 Floutage de l'image couleur

5.1 Programme

On commence par créer le programme `filtre_flou_couleur.cpp`. L'essentiel du code est le suivant :

```
// initialise output image to be the same as input
for (int i = 0; i < nTaille3; i++) {
    ImgOut[i] = ImgIn[i];
}

// apply the filter
for (int i = 1; i < nH-1; i++){
    for (int j = 3; j < (nW-1)*3; j+=3) {
        // p_out(i,j)= ( p(i,j)+p(i-1,j)+p(i+1,j)+p(i,j-3)+p(i,j+3)+p(i-1,j-3)+p(i-1,j+3)+p(i+1,j-3)+p(i+1,j+3) )/9.
        nR = (ImgIn[i * nW*3 + j] + ImgIn[(i-1) * nW*3 + j] + ImgIn[(i+1) * nW*3 + j]
                + ImgIn[i * nW*3 + j-3] + ImgIn[i * nW*3 + j+3] + ImgIn[(i-1) * nW*3 + j-3]
                + ImgIn[(i-1) * nW*3 + j+3] + ImgIn[(i+1) * nW*3 + j-3] + ImgIn[(i+1) * nW*3 + j+3]) / 9;
        nG = (ImgIn[i * nW*3 + j + 1] + ImgIn[(i-1) * nW*3 + j + 1] + ImgIn[(i+1) * nW*3 + j + 1]
                + ImgIn[i * nW*3 + j - 2] + ImgIn[i * nW*3 + j + 4] + ImgIn[(i-1) * nW*3 + j - 2]
                + ImgIn[(i-1) * nW*3 + j + 4] + ImgIn[(i+1) * nW*3 + j - 2] + ImgIn[(i+1) * nW*3 + j + 4]) / 9;
        nB = (ImgIn[i * nW*3 + j + 2] + ImgIn[(i-1) * nW*3 + j + 2] + ImgIn[(i+1) * nW*3 + j + 2]
                + ImgIn[i * nW*3 + j - 1] + ImgIn[i * nW*3 + j + 5] + ImgIn[(i-1) * nW*3 + j - 1]
                + ImgIn[(i-1) * nW*3 + j + 5] + ImgIn[(i+1) * nW*3 + j - 1] + ImgIn[(i+1) * nW*3 + j + 5]) / 9;
        ImgOut[i * nW*3 + j] = nR;
        ImgOut[i * nW*3 + j + 1] = nG;
        ImgOut[i * nW*3 + j + 2] = nB;
    }
}
```

FIGURE 22 – Code du programme `filtre_flou_couleur.cpp`

On peut alors appliquer le programme à l'image `peppers.ppm`. Cela donne :

FIGURE 23 – Image originale

FIGURE 24 – Image floutée

5.2 Comparaison des histogrammes

On compare ensuite les histogrammes de l'image originale et de l'image floutée. Cela donne :

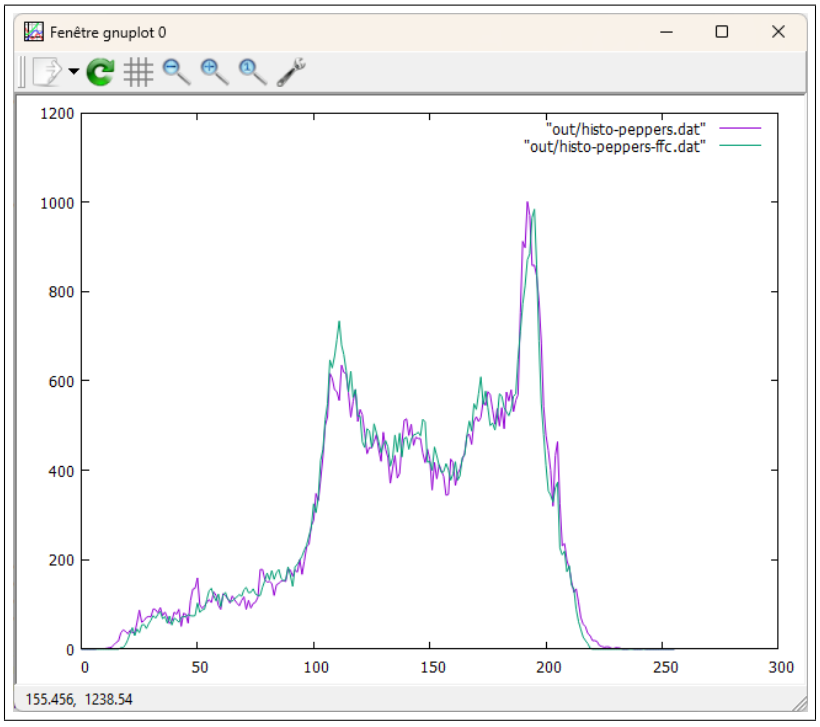

FIGURE 25 – Histogramme de l'image floutée