

Projet de mathématiques : Rééquilibrage d'histogramme et ses options

GRONDIN Corinne - DEMAY Nicolas

28 novembre 2005

Institut Charles Cros
- Val d'Europe -



1 Introduction

Le rapport qui suit est un explicatif des fonctions et des idées que nous avons eu pour réaliser le projet de mathématiques pour l'informatique s'intitulant "Histogramme rééquilibré".

Le but de l'exercice était de faire un programme permettant de rééquilibrer un histogramme c'est-à-dire rendre une image plus visible.

Voilà comment nous y sommes pris pour le réaliser.

Table des matières

1	Introduction	2
2	Le programme	4
2.1	Fonctionnement	4
2.2	Pseudo-Code	4
2.2.1	Pseudo-code i : fonction de l'image de l'histogramme	4
2.2.2	Pseudo-code ii : fonction de l'histogramme	5
2.2.3	Pseudo-code iii : fonction de la lookup table	5
2.2.4	Pseudo-code iv : fonction de l'histogramme cumulé	6
2.2.5	Pseudo-code v : fonction de transformation	6
3	Les options et optimisation	7
3.1	Complexités et robustesse	7
3.1.1	Complexités	7
3.1.2	Robustesse	7
3.2	Les ajouts	7
3.2.1	option -n : négatif	7
3.2.2	option -t : seuillage	8
3.2.3	option -g : transformation en niveau de gris	9
3.2.4	option -l : luminosité	9
4	Difficultés et souhaits	11
4.1	Problèmes	11
4.2	Fonctions qui ne marchent pas et idées	11
5	Conclusion	12

2 Le programme

2.1 Fonctionnement

Le but du programme étant de rééquilibrer l'histogramme, il faut déjà un histogramme (*cf. pseudo-code ii*). Ensuite, il faut créer la *lookup table* qui, pour chaque niveau des gris associe un nouveau niveau de gris. On utilise la fonction mathématique suivante afin de trouver les nouveaux niveaux de gris. (*cf. pseudo-code iii*)

$$\text{niveau_gris} = \text{moyenne} * \text{histo_cumul}[\text{niveau_gris}]$$

avec

$$\text{moyenne} = \frac{255}{(\text{hauteur} * \text{largeur} * 3)}$$

255 est le nombre de niveau de gris et $(\text{hauteur} * \text{largeur} * 3)$ le nombre de valeurs.

Il faut donc calculer l'histogramme cumulé (*cf. pseudo-code iv*). Il suffit ensuite de remplacer ces nouvelles valeurs dans l'image. (*cf. pseudo-code v*).

Cette transformation fonctionne aussi avec les images couleurs. En effet, la fonction rééquilibrage s'applique sur chaque valeur et par conséquent aussi bien pour les images en niveaux de gris ou couleurs. Une image en niveaux de gris n'est autre qu'une image ayant les mêmes valeurs RVB.

2.2 Pseudo-Code

2.2.1 Pseudo-code i : fonction de l'image de l'histogramme

```
ENTIER creeHistoPPM (ENTIER histo [], ENTIER fileName[i], ENTIER type)
{
  FAIRE UN FICHIER file [];
  ENTIER i=0;
  ENTIER c, l;
  ENTIER largeur=256;
  ENTIER hauteur=300;
  CHAINE heading[100];
  CHAINE name [100];
  REEL max=0;
  POUR i DE 0 A 256 POUR UN PAS DE 1
  {
    SI max EST PLUS PETIT QUE histo[i]
    {
      max=histo[i];
    }
  }
}
```

```

REEL unit = hauteur/max;
POUR i DE 0 A 256 POUR UN PAS DE 1
{
  histo[i]*=unit;
}
ENTIER buffer [largeur*hauteur*3];
POUR i DE 0 A (largeur*hauteur*3) POUR UN PAS DE 1
{
  buffer[i]=0;
}
POUR c DE 0 A (largeur*3) POUR UN PAS DE 1
{
  POUR l DE 0 A (hauteur-histo[c/3]) POUR UN PAS DE 1
  buffer[(3*largeur*(l-1))+c]=255;
}
}
RENOYER histoCumul;
}

```

2.2.2 Pseudo-code ii : fonction de l'histogramme

```

ENTIER creeHisto (ENTIER image [], ENTIER largeur, ENTIER hauteur)
{
  ENTIER i=0;
  ENTIER histo[256];
  POUR i DE 0 A 256 POUR UN PAS DE 1
  {
    histo[i]=0;
  }
  POUR i DE 0 A (largeur*hauteur*3) POUR UN PAS DE 1
  {
    histo[image[i]]++;
  }
  RENVOYER histo;
}

```

2.2.3 Pseudo-code iii : fonction de la lookup table

```

ENTIER equilibre (ENTIER histoCumul [], ENTIER largeur, ENTIER hauteur)
{
  REEL moyenne=255.0/(largeur*hauteur*3.0);
  ENTIER i=0;
  ENTIER lookuo[256];

```

```
POUR i DE 0 A 255 POUR UN PAS DE 1
{
lookup[i]=moyenne*histoCumul[i];
}
RENVoyer lookup;
}
```

2.2.4 Pseudo-code iv : fonction de l'histogramme cumulé

```
ENTIER creeHistoCumul (ENTIER histo [])
{
ENTIER i=0;
ENTIER histoCumul[256];
histoCumul[0]=histo[0];
POUR i DE 1 A 256 POUR UN PAS DE 1
{
histoCumul[i]=histoCumul[i-1]+histo[i];
}
RENVoyer histoCumul;
}
```

2.2.5 Pseudo-code v : fonction de transformation

```
ENTIER transfo (ENTIER buffer [], ENTIER lookup [], ENTIER largeur, ENTIER hauteur)
{
ENTIER i=0;
POUR i DE 0 A (hauteur*largeur*3) POUR UN PAS DE 1
{
buffer[i]=lookup[buffer[i]];
}
RENVoyer buffer;
}
```

3 Les options et optimisation

3.1 Complexités et robustesse

3.1.1 Complexités

La fonction `histo` : tout d'abord, elle crée et initialise l'histogramme à 0. Puis, elle le remplit en parcourant chaque pixel une fois. La complexité est donc de $O(n+256)$.

La fonction `histo_cumul` : cette fonction crée et initialise l'histogramme cumulé. Ensuite, elle le remplit en parcourant les valeurs d'histo. La complexité est donc constante soit de $O(256)$.

La fonction `equilibre` : elle crée la *lookup table*. La fonction va donc la remplir en parcourant les valeurs d'histo_cumul. Sa complexité est donc constante à savoir de $O(256)$.

La fonction `transfo` : elle charge les valeurs de buffers. Sa complexité est de $O(n)$.

La complexité pour modifier (équilibrer) une image est donc de $O(2n+3*256)$. En effet, on parcourt deux fois tous les pixels (une fois pour récupérer les valeurs et une deuxième fois pour les modifier).

Le traitement, lui, est constant et ne dépend pas du nombre de pixels. L'optimisation au niveau de la complexité est donc acceptable. On pourrait pousser le vice à chercher à modifier la valeur des pixels que si cette dernière a réellement été modifiée.

Mais l'avantage ne serait pas très significatif voire pas du tout car, il faudrait tout de même tester toutes les valeurs des pixels de l'image.

3.1.2 Robustesse

En ce qui concerne le programme de base, des tests sont effectués pour vérifier que l'image envoyée en paramètre correspond bien à une image existante et au format ppm. Le code est donc robuste dans ce domaine.

Le code est moins robuste en ce qui concerne les options. En effet, le programme traite d'abord l'image et ensuite, il applique les options.

3.2 Les ajouts

3.2.1 option -n : négatif

Cette fonction permet de renvoyer le négatif de l'image passée en paramètre. Mais comment celle-ci fonctionne ?

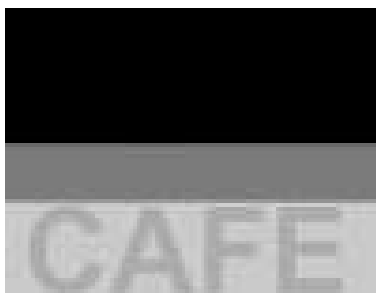


FIG. 1 – Voici une image en négatif

La fonction `-n` prend les valeurs des pixels et va les remplacer par ces valeurs otées de 255 c'est-à-dire elle enlève à chaque pixels 255 qui est la valeur d'une couleurs.

```
ENTIER negatif (ENTIER image [], ENTIER largeur, ENTIER hauteur)
{
  ENTIER i=0;
  TANT QUE i EST PLUS PETIT QUE (largeur*hauteur*3)
  {
    image[i]=255-image[i];
    i++;
  }
  RENVOYER image;
}
```

3.2.2 option `-t` : seuillage

L'option seuillage (`-t` pour `threshold` à savoir seuillage en anglais). Elle consiste à mettre à zero tous les pixels ayant un niveau de gris inférieur à une certaine valeur. Nous avons initialisé cette valeur appelée `valeurSeuillage` à 128. Ainsi le résultat du seuillage est une image binaire contenant des pixels noirs et blancs. Le seuillage permet de mettre en évidence des formes ou des objets dans une image.

```
ENTIER threshold (ENTIER image [], ENTIER largeur, ENTIER hauteur)
{
  ENTIER i;
  ENTIER valeurSeuillage=128;
  POUR i DE 0 A (largeur*hauteur*3) POUR UN PAS DE 1 FAIRE
  {
    SI buffer[i] EST PLUS PETIT QUE valeurSeuillage
    {
```



```

buffer[i]=0;
\}
SINON
{
buffer[i]=255;
}
}
RENVoyer buffer;
}

```

3.2.3 option -g : transformation en niveau de gris

Cette fonction fait la moyenne des trois valeurs (RVB) et change les trois valeurs du pixel. Voici son pseudo-code.

```

ENTIER grayscale(ENTIER buffer[], ENTIER hauteur, ENTIER largeur) {
ENTIER i=0;
POUR i DE 0 A (3*hauteur*largeur) POUR UNE PAS DE 3) {
buffer[i] =(buffer[i] + buffer[i+1] + buffer[i+2])/3;
buffer[i+1] = buffer[i];
buffer[i+2] = buffer[i];
}
RENVoyer buffer;
}

```

3.2.4 option -l : luminosité

Elle augmente ou diminue d'une valeur alpha toutes les valeurs de l'image.

```

ENTIER luminosite(ENTIER histo[], ENTIER largeur, ENTIER hauteur)
{
int i=0;
int lumino = -50;
ENTIER lookup[256];

SI lumino EST PLUS GRAND QUE 0) {
POUR i DE 0 A (256-lumino) POUR UN PAS DE 1 {
lookup[i] = i + lumino;
}
POUR i DE(256-lumino) A 256 POUR UN PAS DE 1) {
lookup[i] = 255;
}
}
SINON {

```

```
SI lumino EST PLUS PETIT QUE 0) {
POUR i DE 0 A (-lumino) POUR UNE PAS DE 1) {
lookup[i] = 0;
}
POUR i DE (-lumino) A 256 POUR UNE PAS DE 1 {
lookup[i] = i + lumino;
}
}
SINON {
SORTIR;
}
}
RENNVOYER lookup;
}
```

4 Difficultés et souhaits

4.1 Problèmes

Notre premier et plus important problème est que notre programme n'est pas assez convivial. En effet, on a essayé de faire en sorte que les options soient facile à utiliser. Nous n'avons pas réellement réussi. Nous avons trouvé un compromis : le programme fait le traitement de l'image avant de faire les options.

De plus, l'utilisateur est obligé d'écrire à chaque fois les options en entier : -h -v etc au lieu de -h v. Ce problème est quand même important.

Les images doivent être placés dans le même dossier que le programme. Mais comme nous avons vu ce problème assez tard, nous avons préféré nous attarder sur autre chose.

Les fonctions telles que seuillage ou luminosité devraient prendre en paramètre des valeurs saisies par l'utilisateur mais ce n'est pas disponible pour le moment ce qui est dû au problème précédent.

4.2 Fonctions qui ne marchent pas et idées

Plusieurs fonctions sont venues à nous. Nous en avons testé quelques unes.

- Ce qu'on a tenté de faire

L'étirement d'histogramme. Il consiste répartir les fréquences d'apparition des pixels sur la largeur de l'histogramme. Ceci revient étendre l'histogramme afin que la valeur d'intensité la plus faible soit à zéro et que la plus haute soit à la valeur maximale. Cela se voit sur l'image car les pixels clairs sont plus clairs et les pixels foncés sont plus foncés.

Le flou. Il consiste à faire la moyenne des valeurs des pixels autour d'un pixel et ce pour tout les pixels. Nous avons essayer de faire un flou mais comme il n'avait pas réellement d'attrait avec la *lookup table*, nous nous sommes pas réellement attardé sur le problème.

- Les idées

La sépia. Sur une image en noir et blanc, l'effet sépia se fait facilement. Il faut ajouter 25% de rouge sur la première valeur (rouge) et enlever 20% sur la troisième valeur (bleu). En effet, une couleur correspond à un mélange RVB (Rouge Vert Bleu) et donc faire ce traitement à chaque pixel va modifier l'image de sorte qu'elle soit dans une nuance de brun.

Traitement d'image 16 bits. Il correspond à 2^5 pour le Rouge, 2^6 pour le Vert et 2^5 pour le Bleu. Il faudrait pour cela refaire toutes les fonctions pour traiter différemment les pixels Rouge et Bleu des pixels Vert.

5 Conclusion

Comme vous l'avez vu, nous avons réussi à traiter le sujet en soit mais, par manque de temps, nous n'avons pas réussi à gérer toutes les options auxquelles nous avons pensé et à la robustesse du programme.

Le traitement de l'image est très vaste. Ainsi, beaucoup d'options sont possibles mais certaines sont hors sujet. Nous aurions aimé faire plus d'options afin d'explorer encore plus le monde du traitement de l'image.

De plus, ceci est limité car ne nous connaissons pas totalement le langage de programmation.