TP n°1 Prise en main d'une librairie de traitement d'images :

(git : https://github.com/GarciaPena-Loris/Donnees-Multimedia)

GARCIA-PENA Loris L3

Le but de ce TP est d'expérimenter la modification d'image en manipulant des images en c++. Pour ce faire, nous utiliserons l'image suivante :

1) Seuillage d'une image au format pgm :

Dans cette première partie, le but est de tester le code fourni pour les images en n&b.

Après compilation avec la commande : `***g++ -o grey1 test_grey.cpp***`
Il faut exécuter l'exécutable généré avec la commande : `***grey1.exe images/lena.pgm newImages/newLena.pgm 100***`

Dans cet exemple cela va modifier l'image lena.pgm stocké dans le dossier images et générer une nouvelle image newLena.pgm dans le dossier newImages avec comme seuil 100.

Cela signifie que tous les pixels (codé entre 0 et 255 pour les niveaux de gris) situés en dessous de 100 seront blanc, ceux au-dessus seront noirs.

Cela donne le résultat suivant :

Pour 200 cela donne :

Pour générer ce résultat le programme procède de cette manière :

Si pixel d'entrée (i,j) < Seuil *--le pixel aux coordonnées (i,j) est inférieur au seuil*
alors pixel de sortie (i,j) = 0, (noir) *--le pixel aux coordonnées (i,j) devient noir*
Sinon pixel de sortie (i,j) = 255 (blanc) *--le pixel aux coordonnées (i,j) devient blanc*

2) Seuillage d'une image pgm avec plusieurs niveaux S1, S2, S3

Dans cette deuxième partie, le but est de créer deux nouveaux fichiers à partir du code précédent dans le but d'augmenter le nombre de seuils des images.

Pour cela, il faut rajouter au code précédent de nouvelles vérifications pour les valeurs de seuil souhaité :

Si pixel d'entrée (i,j) < Seuil1 *--le pixel aux coordonnées (i,j) est inférieur au seuil1*
alors pixel de sortie (i,j) = 0,
sinon si pixel d'entrée (i,j) < Seuil2 *--le pixel aux coordonnées (i,j) est inférieur au seuil2*
alors pixel de sortie (i,j) = 128, *--le pixel aux coordonnées (i,j) devient gris*
Sinon pixel de sortie (i,j) = 255

En appelant le nouveau code avec des nouveaux seuils, on obtient les images suivantes :

Deux seuils : 100, 180 | Deux seuils : 50, 150 | Trois seuils : 50, 128, 200



Finalement, le code est passé de (code 1) à (code 2) pour la modification avec 1 seuil et celle à 3 seuils. Il est bien entendu possible d'augmenter le nombre de seuils autant de fois que souhaités.

code 1

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

```
for (int i = 0; i < nH; i++)
  for (int j = 0; j < nW; j++)
  {
    if (ImgIn[i * nW + j] < S1) {
      ImgOut[i * nW + j] = 0;
    }
    else if (ImgIn[i * nW + j] < S2) {
      ImgOut[i * nW + j] = 90;
    }
    else if (ImgIn[i * nW + j] < S3) {
      ImgOut[i * nW + j] = 180;
    }
    else {
      ImgOut[i * nW + j] = 255;
    }
  }
```

code 2

3) Histogramme d'une image pgm

Dans cette troisième partie, le but est d'afficher puis d'enregistrer les données de l'histogramme d'une image. Un histogramme est une présentation graphique des fréquences relatives à un caractère. Dans notre cas, il faut afficher le nombre d'occurrences d'un même niveau de gris.

Pour cela, il faut dans un premier temps créer un nouveau fichier basé sur le même principe que les précédents, mais sans la partie de création d'une nouvelle image. De plus, nous devons mettre en place le système permettant de récupérer le nombre d'occurrences d'une même nuance de pixel.

Pour faire cela, il suffit de créer un tableau qui va stocker nos résultats et de faire des boucles qui parcourent tous les pixels d'une image afin de compter les occurrences.

```
for (int i = 0; i < nH; i++)
    for (int j = 0; j < nW; j++)
    {
        tabHisto[ImgIn[i * nW + j]]++;
    }
for (int m = 0; m < 256; m++)
{
    if (tabHisto[m] != 0)
    {
        printf("%d pixel de valeur: %d\n", tabHisto[m], m);
    }
}
```

Première boucle qui compte le nombre d'occurrences d'une nuance.

Deuxième boucle qui affiche le résultat.

Sur notre image témoin 'lena.pgm' on obtient le résultat suivant :
(c'est un résultat partiel, car il y a 255 nuances différentes).

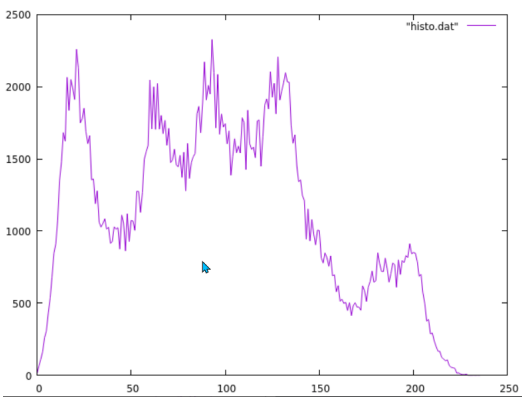
```
11 pixel de valeur: 3
65 pixel de valeur: 4
111 pixel de valeur: 5
164 pixel de valeur: 6
261 pixel de valeur: 7
308 pixel de valeur: 8
431 pixel de valeur: 9
537 pixel de valeur: 10
682 pixel de valeur: 11
846 pixel de valeur: 12
912 pixel de valeur: 13
```

Pour fournir un résultat plus visuel, on utilise le logiciel GNUPLOT qui peut visualiser des histogrammes :

Pour ce faire, on enregistre notre résultat dans un fichier .dat à l'aide de cette commande : ***`histo images/lena.pgm > histo.dat`.***

Ensuite, il faut lancer le terminal de gnuplot en faisant : `***gluplot***` dans notre terminal.
Pour finir, il faut exécuter : `***plot "histo.dat" with line***` pour afficher le résultat sous forme de diagramme.

On obtient le diagramme suivant :



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

Cette partie reprend le principe de la précédente, mais cette fois-ci la recherche d'occurrences se fait sur une seule ligne / colonne.

Pour ce faire, on peut réutiliser le même procédé que précédemment, mais en changeant le système de comptage en fonction du comptage d'une ligne ou d'une colonne.

Dans les grosses ligne, il faut :

1. Modifier les arguments pour prendre en compte s'il s'agit d'une ligne ou d'une colonne et le numéro de la ligne / colonne :

```
sscanf(argv[1], "%s", cNomImgLue);
sscanf(argv[2], "%s", emplacement);
sscanf(argv[3], "%d", &indice);
```

2. Ajouter une vérification pour savoir si on veut compter une ligne ou une colonne ::

```
if (strcmp(emplacement, "ligne") == 0)
```

3. Compte le nombre d'occurrences en fonction de la ligne ou de la colonne :
   a. Ligne :

```
// recherche par ligne
for (int j = 0; j < nW; j++)
{
    tabHisto[ImgIn[indice * nW + j]]++;
}
```

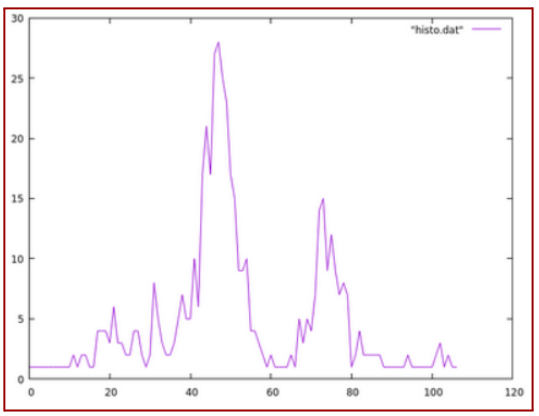
   b. Colonne :

```
// recherche par colonne
for (int i = 0; i < nH; i++)
{
    tabHisto[ImgIn[i * nW + indice]]++;
}
```

On remarque que le **i** et le **j** change en fonction de la ligne ou de la conne.

Comme précédemment, on peut afficher le résultat sous forme textuelle ou graphique :

```
1 pixel de valeur: 18
2 pixel de valeur: 19
1 pixel de valeur: 21
2 pixel de valeur: 22
2 pixel de valeur: 24
```



5) Seuillage d'une image couleur (ppm)

Dans cette partie, le but est de seuiller une image ppm en fonction des trois composantes de couleurs (R,G,B) :

Avant cela nous allons tester le seuillage d'une image ppm selon un seul niveau général au R,G,B grâce à la fonction fournie test_couleur.
De manière analogue à la première partie, il faut compiler le programme et l'exécuter avec des valeurs différentes, nous avons pris cette image comme image de départ :

seuil : 80

seuil : 150

seuil : 220

Maintenant nous allons seuiller cette image selon les niveau de Red, Green et Blue :

Pour ce faire il faut rajouter 3 seuils au lieux de 1 dans les arguments, et seuiller en fonction de ces seuils :

```
if (nR < SR) ImgOut[i]=0; else ImgOut[i]=255;
if (nG < SG) ImgOut[i+1]=0; else ImgOut[i+1]=255;
if (nB < SB) ImgOut[i+2]=0; else ImgOut[i+2]=255;
```

On obtient ainsi les résultat suivant :

seuils : 50 100 150

seuil : 150 50 100

seuil : 100 150 50

On remarque que l'on peut changer la couleur d'une image de cette façon.

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

Dans cette dernière partie, le but est d'afficher un histogramme comme dans la partie précédente mais cette fois ci pour chaque composantes (R, G, B) :

Pour cela il faut reprendre le principe du programme d'histogramme précédent, mais en modifiant le code pour compter individuellement chaque composante :

```
for (int i = 0; i < nTaille3; i += 3)
{
    tabHistoR[ImgIn[i]]++;
    tabHistoG[ImgIn[i + 1]]++;
    tabHistoB[ImgIn[i + 2]]++;
}
```

De manière analogue pour l'affichage il faut séparer pour chaque composante :

```
for (int m = 0; m < 256; m++)
{
    printf("%i %i %i %i\n", m, tabHistoR[m], tabHistoG[m], tabHistoB[m]);
}
```

Sur notre image témoin 'lena.ppm on obtient le résultat suivant : (c'est un résultat partiel, car il y a 255 nuances différentes).

```
193 260 11 6
194 249 13 4
195 266 10 6
196 219 6 7
197 213 13 3
198 199 12 4
```

Pour conclure, ce premier TP nous a permis de découvrir les bases de la modification d'image ainsi que des thématiques importantes telles que les histogrammes qui permettent rapidement de visualiser les valeurs d'une image.