# Traitement d'images [dx06] - Examen

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Université Haute-Alsace

Version 20 mai 2018

## Table des matières



## C++ - Traitement d'images (Solution)

**Mots-Clés** Exercices généraux ■
**Requis** Axiomatique impérative ■
**Difficulté** ●●○ (45 min) ■

**Objectif**
Cet exercice réalise quelques traitements d'images.

# 1 Traitement d'images (9 points)

## 1.1 Type Image

Une image est représentée par ses pixels $(x, y)$ où chaque pixel a une couleur uniforme. Pour simplifier, on manipulera des images en niveau de gris, c.-à-d. que la valeur des pixels sera un entier compris entre 0 et 255 (bornes incluses) représentant l'intensité lumineuse : 0 (pas d'intensité = noir) et 255 (intensité maximale = blanc).

**(0.5 point)** Définissez les constantes entière `LGMAX=1280` et `HTMAX=1024` représentant la largeur et la hauteur maximales d'une image, le type `TabPixels` comme étant un tableau de `HTMAX` lignes et `LGMAX` colonnes d'entiers, puis le type `Image` structure contenant :

- Un `TabPixels p` des pixels.
- Un entier `largeur` de la largeur effective de l'image.
- Un entier `hauteur` de la hauteur effective de l'image.

## 1.2 Image au format PGM

On veut afficher une image au format PGM (on pourrait ensuite écrire cet affichage dans un fichier) : il faut afficher `P2`, puis la largeur et la hauteur de l'image, puis `255`, puis enfin les valeurs des pixels ligne par ligne, comme dans l'exemple suivant :

```
P2
12 7
255
0   0   0   0   0   0   0   0   0   0   0   0
0  92  92  92  92   0   0 205 205 205 205   0
0  92   0   0   0   0   0  15   0   0 205   0
0  92  92  92   0   0   0 205 205 205 205   0
0  92   0   0   0   0   0 205   0   0   0   0
0  92  92  92  92   0   0 205   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0
```

**(1.5 point)** Écrivez une procédure `afficherImagePGM(im)` qui affiche une `Image im` au format PGM. Utilisez `setw(4)` pour qu'une valeur soit affichée sur 4 caractères.

**(2 points)** Écrivez une procédure `saisirImagePGM(im)` qui saisit une `Image` au format PGM dans `im`. Supposez que toutes les données nécessaires sont données sans erreur.

## 1.3 Amélioration du contraste d'une image

On veut renforcer le contraste d'une image, c.-à-d. rendre les couleurs foncées encore plus foncées et les couleurs claires encore plus claires.

On veut même contraster à fond une image par rapport au valeurs $p_{min}$ et $p_{max}$, c.-à-d. augmenter le contraste de façon à ce que les couleurs inférieures à $p_{min}$ deviennent noires et les couleurs supérieures à $p_{max}$ deviennent blanches.

Pour chaque valeur de pixel $p$, on peut calculer sa nouvelle valeur à partir de la formule suivante :

$$\begin{cases} p \leq p_{min} & \to 0 \\ p_{min} < p < p_{max} & \to \dfrac{255}{2}\left(1 - \cos\dfrac{\pi(p - p_{min})}{p_{max} - p_{min}}\right) \\ p \geq p_{max} & \to 255 \end{cases}$$

**(2 points)** Écrivez une procédure `contrasterImage(im,pmin,pmax)` qui contraste une `Image im` par rapport aux valeurs `pmin` et `pmax`. Appliquez les calculs précédents en évitant de recalculer plusieurs fois certaines valeurs.

**(2 points)** Pour contraster au maximum sans perdre de couleur, il faut contraster par rapport aux valeurs minimales et maximales des pixels de l'image. Écrivez une procédure `calculerExtremaImage(im,pmin,pmax)` qui calcule (en ne faisant qu'une seule boucle), la plus petite valeur dans `pmin` et la plus grande valeur dans `pmax` des pixels d'une `Image im`.

**(1 point)** Déduisez une procédure `constraterMaxImage(im)` qui contraste une `Image im` par rapport à ses valeurs extrémales.

## 1.4 Programme de test

Écrivez un programme qui saisit une `Image`, l'affiche, la contraste au maximum puis la réaffiche.

Validez votre programme avec la solution.

**Solution C++** @[pgtimage.cpp]

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

// Definitions
const int LGMAX = 100/*1280*/, HTMAX = 100/*1024*/;
typedef int TabPixels[HTMAX][LGMAX];
struct Image{
   TabPixels p;
   int largeur;
   int hauteur;
};
int evalPixel(const Image& im,int j,int k){
   return im.p[j][k];
```

```
}
int fixerPixel(Image& im,int j,int k,int valeur){
   im.p[j][k] = valeur;
}

// Affichage d'une Image
void afficherImagePGM(const Image& im){
   cout<<"P2"<<endl;
   cout<<im.largeur<<" "<<im.hauteur<<endl;
   cout<<255<<endl;
   for (int j = 0 ; j < im.hauteur ; ++j){
      for (int k = 0 ; k < im.largeur ; ++k){
         cout<<setw(4)<<evalPixel(im,j,k);
      }
      cout<<endl;
   }
}

// Saisie d'une Image
void saisirImagePGM(Image& im){
   string s;
   cin>>s;
   cin>>im.largeur>>im.hauteur;
   int x;
   cin>>x;
   for (int j = 0 ; j < im.hauteur ; ++j){
      for (int k = 0 ; k < im.largeur ; ++k){
         cin>>x;
         fixerPixel(im,j,k,x);
      }
   }
}

// Constrate d'une Image
const double M_PI = 3.141592653589793116;
void contrasterImage(Image& im,int pmin,int pmax){
   int delta = pmax - pmin;
   double coeff = 255 / 2.0;
   int val, nval;
   for (int j = 0 ; j < im.hauteur ; ++j){
      for (int k = 0 ; k < im.largeur ; ++k){
         int val = evalPixel(im,j,k);
         if (val <= pmin){
            nval = 0;
         }
         else if (val >= pmax){
            nval = 255;
         }
         else{
            nval = int(coeff * (1 - cos( (M_PI*(val-pmin)) / delta)));
         }
         fixerPixel(im,j,k,nval);
      }
   }
}

// Extrema d'une Image
void calculerExtremaImage(const Image& im,int& pmin,int& pmax){
```

```
   pmin = pmax = evalPixel(im,0,0);
   for (int j = 0 ; j < im.hauteur ; ++j){
      for (int k = 0 ; k < im.largeur ; ++k){
         int val = evalPixel(im,j,k);
         if (val < pmin){
            pmin = val;
         }
         else if (val > pmax){
            pmax = val;
         }
      }
   }
}

// Constraste maximum d'une Image
void constraterMaxImage(Image& im){
   int pmin, pmax;
   calculerExtremaImage(im,pmin,pmax);
   contrasterImage(im,pmin,pmax);
}

// Histogramme d'une Image
const int NCOULEURS = 256;
typedef int Histo[NCOULEURS];
void calcHistoImage(const Image& im,Histo& h){
   for (int j=0 ; j<NCOULEURS ; ++j){
      h[j] = 0;
   }
   for (int j = 0 ; j < im.hauteur ; ++j){
      for (int k = 0 ; k < im.largeur ; ++k){
         h[evalPixel(im,j,k)] += 1;
      }
   }
}

void afficherHisto(const Histo& h){
   for (int j=0 ; j<NCOULEURS ; ++j){
      cout<<j<<h[j]<<" ";
   }
   cout<<endl;
}

// Binarise une Image
void binariserImage(Image& im,int seuil){
   for (int j = 0 ; j < im.hauteur ; ++j){
      for (int k = 0 ; k < im.largeur ; ++k){
         int val = evalPixel(im,j,k);
         val = (val<seuil ? 0 : 1);
         fixerPixel(im,j,k,val);
      }
   }
}

int main()
{
   Image im;
   saisirImagePGM(im);
   afficherImagePGM(im);
```

```
    constraterMaxImage(im);
    afficherImagePGM(im);
}
```

## 1.5 Complément : Histogramme d'une image

L'histogramme d'une image étudie la répartition statistique de chaque valeur de couleur dans une image en 256 couleurs. Son principe consiste, dans un tableau histogramme de 256 cases de type entier, à compter combien l'image contient de pixels de la couleur 0 et à stocker cette valeur dans la case 0 du tableau histogramme, puis combien l'image contient de pixels de couleur 1 à ranger le nombre obtenu dans la case 1 du tableau histogramme... ainsi de suite pour toutes les couleurs jusqu'à la couleur 255 comprise.

Calculez l'histogramme de l'image puis affichez le résultat de manière lisible à l'écran.

## 1.6 Complément : Binarisation de l'image

La binarisation d'une image consiste, dans une image secondaire afin de ne pas modifier l'image originale, à mettre à 0 tous les pixels de l'image originale inférieurs à une valeur seuil entrée par l'utilisateur, et à mettre à 255 tous les pixels de l'image originale supérieurs ou égaux à cette valeur seuil.

Calculez la binarisation de l'image puis affichez le résultat.