Bibliothèque de gestion d'images

1 Introduction

Cette bibliothèque de gestion d'images permet, assez simplement, de créer des images (en niveaux de gris ou en couleurs), de les manipuler et de les afficher.
De plus, 3 formats d'images sont pris en charge pour la lecture ou l'écriture de fichiers image : `.ras` (Sun Rasterfile), `.ppm` et `.pgm` .

2 Les classes `Image` et `ImageRVB`

Cette section décrit brièvement les classes permettant de manipuler des images :

- `class Image` ; pour la manipulation d'images en niveaux de gris sur 1 plan où chaque pixel est codé sur 1 octet (`typedef unsigned char octet;`).

- `class ImageRVB` ; pour la manipulation d'images couleurs sur 3 plans (3 objets de la classe `Image`).

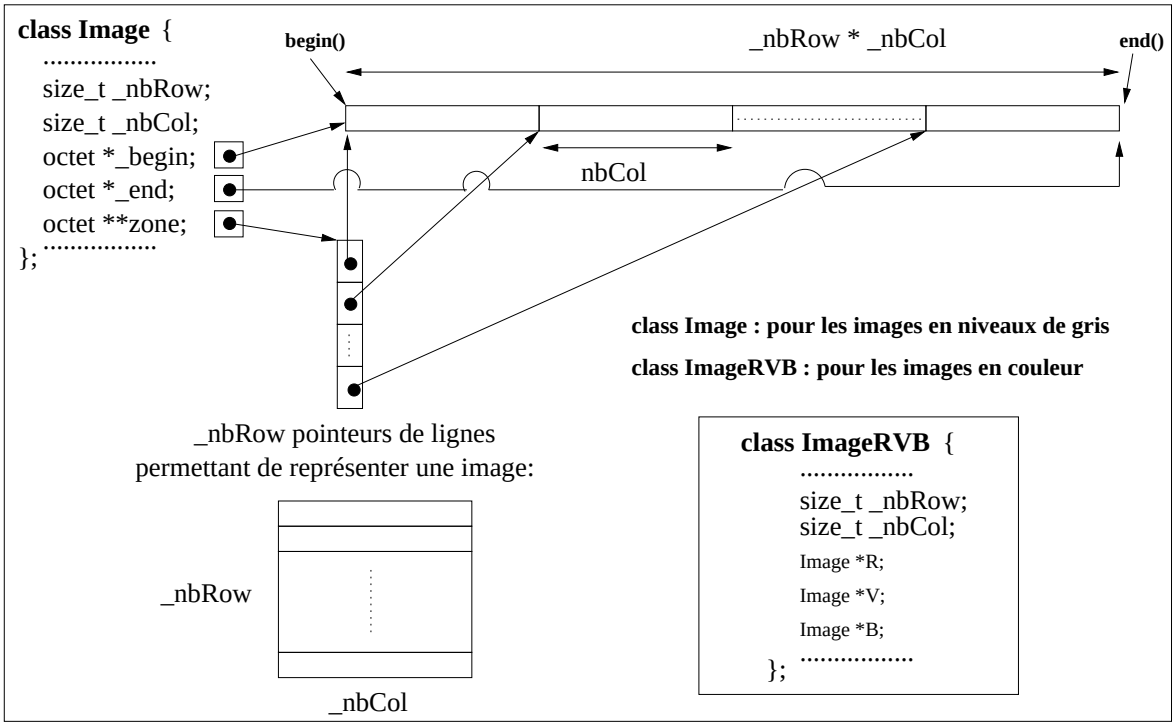

Figure 1: Description des structures internes des deux classes `Image` et `ImageRVB`.

3 Interface publique de la classe Image

```
class Image {
              friend ostream& operator<<(ostream& os, const Image& anImage);
  public:
    // Allocateurs/Desallocateurs
           Image(size_t nbRow=512, size_t nbCol=512);
           Image(const char* fileName);
           Image(const Image& anImage);
           Image(const ImageRVB& anImageRVB);            // Cousin... I_NB <- I_RVB
           Image& operator=(const Image& anImage);       // Affectation
           Image& operator=(const ImageRVB& anImageRVB); // Affectation : I_NB <- I_RVB
  virtual ~Image(void);

           void  setImageSize(size_t nbRow, size_t nbCol);

           void  loadImage(const char* fileName);
           void  saveImage(const char* fileName);

    // Operateurs
           Image& operator+=(const Image& anImage);
    friend Image  operator+ (const Image& anImage1,const Image& anImage2);
           Image& operator-=(const Image& anImage);
    friend Image  operator- (const Image& anImage1,const Image& anImage2);

    // Comparaisons
  friend   bool operator==(const Image& anImage1, const Image& anImage2);
  friend   bool operator!=(const Image& anImage1, const Image& anImage2);

    // Inspecteurs
           size_t  getNbRow(void) const;
           size_t  getNbCol(void) const;

     const octet*  begin(void) const; // Un pointeur sur le DEBUT
           octet*  begin(void);       // de la zone image
     const octet*  end(void) const;   // Un pointeur sur la FIN
           octet*  end(void);         // de la zone image

    // Modificateurs/Inspecteurs
           void  writePix(size_t row, size_t col, octet val);
           void  readPix(size_t row, size_t col, octet& val) const;
           octet readPix(size_t row, size_t col) const;

           const octet* operator[] (size_t row) const;    // Pour autoriser
                 octet* operator[] (size_t row);          // anImage[row][col]

           octet  operator() (size_t row, size_t col) const; // Pour autoriser
           octet& operator() (size_t row, size_t col);       // anImage(row,col)

           void  setImage(octet val); // Tous les pixels initialises a val
           void  lineImage(int X1,int Y1, int X2,int Y2,octet val); // Ligne (x1,y1)-(x2,y2)

  protected:
  //  display a appeler dans une classe derivee (methode appelee dans operator<<)
   virtual void display(ostream& os) const;
  //  isEqualTo a appeler dans une classe derivee (methode appelee dans operator==)
   virtual bool isEqualTo(const Image& anImage) const;
};
```

4 Interface publique de la classe ImageRVB

```
class ImageRVB {
                friend ostream& operator<<(ostream& os, const ImageRVB& anImageRVB);
  public:
    // Allocateurs/Desallocateurs
          ImageRVB(size_t nbRow=512, size_t nbCol=512);
          ImageRVB(const char* fileName);
          ImageRVB(const ImageRVB& anImageRVB);
          ImageRVB(const Image& anImage);                  // Cousin... I_RVB <- I_NB
          ImageRVB& operator=(const ImageRVB& anImageRVB); // Affectation
          ImageRVB& operator=(const Image& anImage);       // Affectation: I_RVB <- I_NB
  virtual ~ImageRVB(void);

          void  setImageSize(size_t nbRow, size_t nbCol);

          void  loadImage(const char* fileName);
          void  saveImage(const char* fileName);
    // Operateurs
          ImageRVB& operator+=(const ImageRVB& anImageRVB);
    friend ImageRVB  operator+ (const ImageRVB& anImageRVB1, const ImageRVB& anImageRVB2);
          ImageRVB& operator-=(const ImageRVB& anImageRVB);
    friend ImageRVB  operator- (const ImageRVB& anImageRVB1, const ImageRVB& anImageRVB2);
    // Comparaisons
  friend   bool operator==(const ImageRVB& anImageRVB1, const ImageRVB& anImageRVB2);
  friend   bool operator!=(const ImageRVB& anImageRVB1, const ImageRVB& anImageRVB2);

    // Inspecteurs
          size_t  getNbRow(void) const;
          size_t  getNbCol(void) const;

     const Image&  getR(void) const;          // Pour obtenir
          Image&  getR(void);                 // le plan R

     const Image&  getV(void) const;          // Pour obtenir
          Image&  getV(void);                 // le plan V

     const Image&  getB(void) const;          // Pour obtenir
          Image&  getB(void);                 // le plan B
    // Modificateurs/Inspecteurs
          void  writePix(size_t row, size_t col, octet valR, octet valV, octet valB);
          void  readPix(size_t row, size_t col, octet& valR, octet& valV, octet& valB) const;

          void  setImage(octet valR, octet valV, octet valB); // Tous les pixels initialises
                                                              // a valR, valV et valB

          void  lineImage(int X1,int Y1, int X2,int Y2, octet valR, // Ligne (x1,y1)-(x2,y2)
                                                        octet valV,
                                                        octet valB);
  protected:
  //  display a appeler dans une classe derivee (methode appelee dans operator<<)
   virtual void display(ostream& os) const;

  //  isEqualTo a appeler dans une classe derivee (methode appelee dans operator==)
   virtual bool isEqualTo(const ImageRVB& anImageRVB) const;
};
```

5 Méthodes d'accès aux images de la classe `Image`

Afin d'illustrer les différentes possibilités d'accès à une image en niveaux de gris, nous prenons l'exemple d'une fonction `raz` permettant de mettre à `0` tous les pixels d'une image.

5.1 Avec des [ ][ ]

```
void raz(Image& out)
{
 int nbRow = out.getNbRow(), nbCol = out.getNbCol();
 for(int l=0;l<nbRow;l++)
 {
  for(int c=0;c<nbCol;c++)
  {
   out[l][c] = 0;       // Pas de test de validite de l et c ... plus rapide
  }                     //                                   ... moins fiable
 }
}
```

5.2 Avec des (,)

```
void raz(Image& out)
{
 int nbRow = out.getNbRow(), nbCol = out.getNbCol();
 for(int l=0;l<nbRow;l++)
 {
  for(int c=0;c<nbCol;c++)
  {
   out(l,c) = 0;              // Test de validite de l et c ... moins rapide
  }                           //                            ... plus fiable
 }
}
```

5.3 Avec un accès via un pointeur (parcours séquentiel)

```
void raz(Image& out)
{
 for(octet* ptrOut = out.begin() ; ptrOut < out.end() ; ptrOut++)
 {
  *ptrOut = 0;
 }
}
```

5.4 Avec un appel de mèthode `readPix` ou `writePix`

```
void raz(Image& out)
{
 int nbRow = out.getNbRow(), nbCol = out.getNbCol();
 for(int l=0;l<nbRow;l++)
 {
  for(int c=0;c<nbCol;c++)
  {
   out.writePix(l,c,0);       //  Test de validite de l et c ... moins rapide
  }                           //                             ... plus fiable
 }
}
```

6 Méthodes d'accès aux images de la classe `ImageRVB`

Afin d'illustrer les différentes possibilités d'accès à une image en couleur, nous prenons l'exemple d'une fonction `raz` permettant de mettre à `0` tous les pixels d'une image.

6.1 Avec un appel de méthode `readPix` ou `writePix`

```
void raz(ImageRVB& out)
{
 int nbRow = out.getNbRow(), nbCol = out.getNbCol();
 for(int l=0;l<nbRow;l++)
 {
  for(int c=0;c<nbCol;c++)
  {
   out.writePix(l,c,0,0,0);   //  Test de validite de l et c
  }
 }
}
```

6.2 En obtenant des références sur les différents plans `Image`

```
void raz(ImageRVB& out)
{
 Image& R = out.getR();
 Image& V = out.getV();
 Image& B = out.getB();

 raz(R);  // Appels a la methode raz pour chaque plan, methode
 raz(V);  // decrite lors de la section precedente... sur les
 raz(B);  // objet de la classe Image
}
```

7 Redimensionnement d'une image

Il peut parfois être nécessaire de redimensionner une image avant un traitement avec la méthode `setImageSize` de la classe `Image` ou de la classe `ImageRVB`. Ainsi, si l'on considère l'exemple suivant :

```
void seuillage(const Image& in, Image& out, octet seuil)
{
 int nbRow = in.getNbRow(), nbCol = in.getNbCol();

 out.setImageSize(nbRow,nbCol);  // Redimensionnement, ... au cas ou

 for(int l=0;l<nbRow;l++)
 {
  for(int c=0;c<nbCol;c++)
  {
   out[l][c] = (in[l][c] > seuil) ? 0 : 255;   // Pas de test de validite de l et c
  }
 }
}
```

un redimensionnement de l'image de sortie sera ainsi effectué. Remarque : il n'y a réallocation de la zone dédiée à l'image que si les dimensions n'étaient pas celles désirées.

8 Passage d'une Image à une ImageRVB et inversement

Afin de facilier la transformation d'une ImageRVB en une Image, il existe dans la classe Image un constructeur par cousinage et un opérateur d'affectation permettant de faire la conversion d'une image codée sur 3 plans en une image codée sur 1 plan :

```
Image(const ImageRVB& anImageRVB);            // Cousin... I_NB <- I_RVB
Image& operator=(const ImageRVB& anImageRVB); // Affectation : I_NB <- I_RVB
```

Dans la classe ImageRVB, il existe également la possibilité de convertir une Image en une ImageRVB :

```
ImageRVB(const Image& anImage);            // Cousin... I_RVB <- I_NB
ImageRVB& operator=(const Image& anImage); // Affectation : I_RVB <- I_NB
```

9 Affichage d'une image (Image ou ImageRVB)

Afin d'afficher une image, if faut déclarer un objet de type XAffichage en indiquant les dimensions de la fenêtre d'affichage.
Ensuite, sur cet objet, il faut appeler :

- la méthode Afficher en passant l'image (Image ou ImageRVB) à afficher.
- Puis, la méthode XEvenement (toujours en passant l'image). Cette méthode retourne un éventuel caractère ayant été entré par l'utilisateur.

Ainsi, le programme type d'affichage d'une image est le suivant.

```
      // g++ ex.cpp -I../LibImages/include -L/usr/X11R6/lib -lX11 -L../LibImages/lib -lImages -o ex

#include <iostream>              // Fichier ex.cpp

using namespace std;

#include "LibImages.h"

int main(void)
{
 ImageRVB im("../Images/Lena/lena24FullColor.ras");

 XAffichage *Fim = new XAffichage(im.getNbRow(),im.getNbCol());
 Fim->setLabel("lena24FullColor.ras");

 while (1)
 {
  char cim;

  Fim->Afficher(im);
  cim=Fim->XEvenement(im);
  cim = tolower(cim);

  if (cim=='q')  break;
 }

 delete(Fim);

 return 0;
}
```

10 Affichage d'une image (Image ou ImageRVB) avec gestion de la souris

```
// g++ exCurseur.cpp -I../LibImages/include -L/usr/X11R6/lib -lX11 -L../LibImages/lib -lImages -o exCurseur

#include <iostream>             // Fichier exCurseur.cpp

using namespace std;

#include "LibImages.h"

int main(void)
{
 ImageRVB im("../Images/Lena/lena24FullColor.ras");

 XAffichage *Fim = new XAffichage(im.getNbRow(),im.getNbCol());
 Fim->setLabel("lena24FullColor.ras");

 while (1)
 {                                /*************************************/
  XAffichageEvent event;          /* Dans un XAffichageEvent, il y a:  */
                                  /*                                   */
  /* typedef                   */ /* 1) Recuperation caractere appuye: */
  /*    struct {               */ /*                                   */
  /*      char key;            */ /* char key (le caractere appuye ou  */
  /*      int  button;         */ /*           -1 si aucun caractere)  */
  /*      int  row;            */ /* key...aussi le retour de XEvenement */
  /*      int  col;            */ /*                                   */
  /*      bool onButtonPress;  */ /* 2) Gestion souris:                */
  /*      bool onButtonRelease; */ /*                                  */
  /*      bool onButtonMotion; */ /* int button (1,2 ou 3 si evt souris */
  /*    } XAffichageEvent;     */ /*             -1 si aucun evt souris) */
                                  /* Si button!= -1 :                  */
  char cim;                       /* . int row,int col(coord y,x souris) */
                                  /*     (row,col)=(-1,-1) si en dehors */
  Fim->Afficher(im);              /* . bool onButtonPress  (true,false), */
  cim=Fim->XEvenement(im,&event); /* . bool onButtonRelease(true,false), */
  cim = tolower(cim);             /* . bool onButtonMotion (true,false). */
  if (event.button>0)             /*************************************/
  {
   if (event.onButtonPress)   { cout << "  Press";   }
   else
   if (event.onButtonRelease) { cout << "Release"; }
   else
   if (event.onButtonMotion)  { cout << " Motion"; }
   else { cout << "???????"; }

   cout << "(" << event.button << "):";
   cout<<"("<<event.button<<"):"<<" row: "<<event.row<<" col: "<< event.col;
   cout << endl;
  }

  if (cim=='q')  break;
 }

 delete(Fim);

 return 0;
}
```

11 Affichage avec un visalisateur externe

La fonction

```
void displayImage(const char *nomFichier, char *visualiseur = "xv");
```

fonction décrite dans le fichier `LibImages/src/ESImages.cpp`, permet d'afficher le contenu d'un fichier image à l'aide d'un visualisateur externe.
Par défaut, le visualisateur externe utilisé est `xv`.

Exemples d'utilisation :

```
displayImage("../Images/Lena/lena24FullColor.ras");        // Affichage avec xv

displayImage("../Images/Lena/lena24FullColor.ras","gimp"); // Affichage avec gimp
```

12 Formats image pris en charge lors du chargement ou de la sauvegarde d'une image

12.1 Lors du chargement

Lors du chargement d'une image (`Image` ou `ImageRVB`) avec le constructeur ou la méthode `loadImage`, les formats pris en charge sont `.ras`, `.ppm` et `.pgm` .
La détection du format se fait via l'extension du fichier :

- `.ras` ; format de Sun : rasterfile
- `.ppm` ; format PPM : Portable PixMap (binaire ou ascii)
- `.pgm` ; format PGM : Portable GreyMap (binaire ou ascii)

12.2 Lors de la sauvegarde

Lors de la sauvegarde d'une image (`Image` ou `ImageRVB`) avec la méthode `saveImage`, les formats pris en charge sont `.ras`, `.ppm` et `.pgm` .
La détection du format se fait via l'extension du fichier :

- `.ras` ; format de Sun : rasterfile
- `.ppm` ; format PPM (Portable PixMap) binaire
- `.ascii.ppm` ; format PPM (Portable PixMap) acsii
- `.pgm` ; format PGM (Portable GreyMap) binaire
- `.ascii.pgm` ; format PGM (Portable GreyMap) acsii

13 A savoir

Il est possible:

- de tester l'égalité ou l'inégalité entre deux images
  $\Longrightarrow$ `operator==` et `operator!=`
- d'affecter une image dans une autre
  $\Longrightarrow$ `operator=`
- de calculer la somme de deux images
  $\Longrightarrow$ `operator+=` et `operator+`
- de calculer la différence de deux images
  $\Longrightarrow$ `operator-=` et `operator-`

Remarque: ces opérateurs existent pour la classe `Image` et pour la classe `ImageRVB`

14 Installation

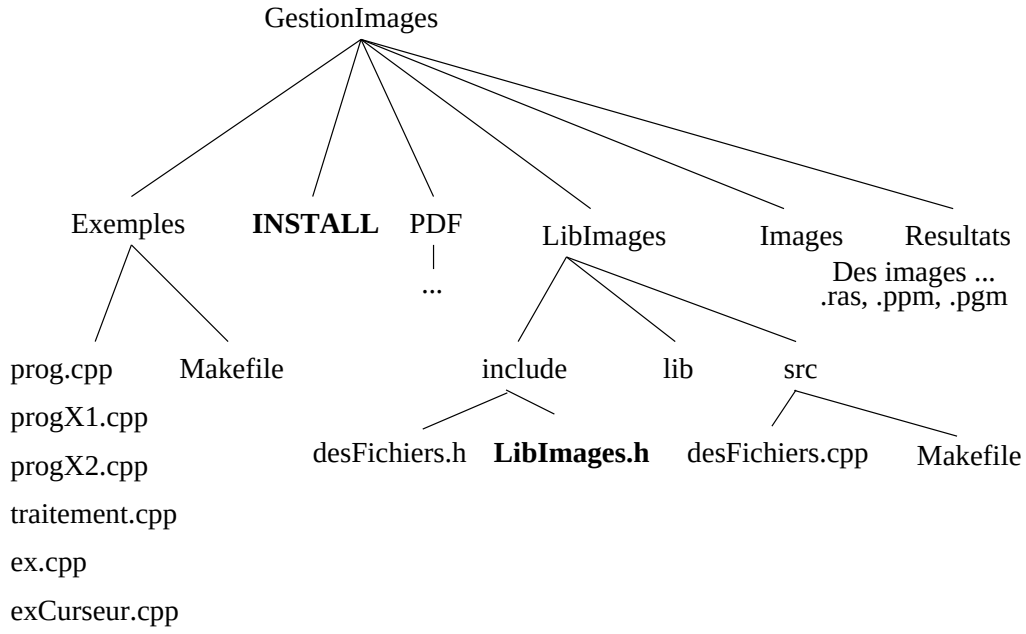

Figure 2: Description de l'arborescence de la bibliothèque

Aller dans le répertoire `LibImages/src` et faire `$ make`
Une bibliothèque `libImages.a` est alors placée dans le répertoire `LibImages/lib`
Des exemples de programmes sont disponibles dans le répertoire `Exemples` (un fichier `Makefile` est donné).

... mais on peut faire plus simple !

En étant dans le répertoire `GestionImages`, faire tout simplement `$ ./INSTALL`
Il faut ensuite aller dans le répertoire `Exemples` et faire `$ make`