UNIVERSITE DE NANTES
FACULTE DE SCIENCES ET DE TECHNIQUES

JUIN 2004

TER BMP – RAPPORT

Abdeslam MOKRANI
ENCADRANT : Gilles RIMBERT

SOMMAIRE

I. Introduction

Le présent TER (Travail d'études et de recherche) se consacre à la réalisation d'un logiciel de traitement d'images numériques. Il permet d'ouvrir des images codées dans un format standard (le codage BMP par exemple) Il fournit des outils pour visualiser et manipuler les images ouvertes. Enfin, il permet de sauvegarder les images modifiées.

Le logiciel utilise la bibliothèque graphique GTK+. Ce choix étant effectué car celle-ci est libre et portable vers d'autres systèmes d'exploitation. Le langage de programmation choisis est le langage C. Ce langage est plus adapté car, d'une part, la bibliothèque graphique utilisée est entièrement écrite avec ce langage (pour une cohérence du code). D'autre part, ce langage est très puissant pour l'écriture d'algorithmes de traitement d'images. En effet, il permet d'écrire des instructions bas-niveau pour traiter les données d'une image notamment au niveau bit.

Le programme de l'application est divisé en deux grandes parties. Dans la première partie sont implémentées structures relatives aux images et leur traitement.. La seconde partie correspond à l'interface graphique du logiciel.

Une liste de définitions de quelques mots clés et de conventions utilisées se trouve dans en page 17. Celle-ci à pour a pour but de faciliter la compréhension de ce document. Il est conseillé d'y aller jeter un coup d'œil avant de continuer.

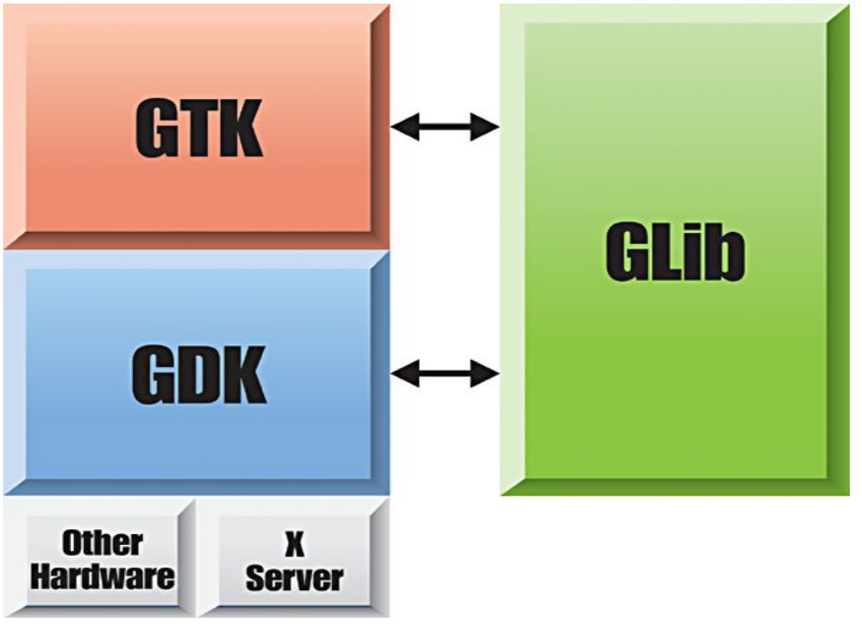
Avant d''entamer l'étude du programme, nous allons faire une brève présentation de la bibliothèque graphique GTK+.

II. GTK+

1. Présentation

C'est une bibliothèque graphique libre (sous la licence GNU LGPL.) Elle est initialement conçue pour le logiciel de retouche d'image The GIMP (GNU Image Manipulation Program, GTK signifie Gimp Tool Kit)

GTK+ utilise une autre bibliothèque graphique de bas niveau : la bibliothèque GDK. Cette dernière est écrite pour les différentes plates-formes en vue de supporter GTK+. Sous un system de type UNIX par exemple, le GDK effectue une encapsulation des structures du serveur graphique X Window et utilise les fonctions q'il fournit. Une autre bibliothèque dont GTK+ dépend est GLib. Celle-ci n'est pas une bibliothèque graphique, mais elle fournit un ensemble d'utilités comme la redéfinition des types standard du langage c, des fonctions et macros de manipulation de structures de données, de fichiers, de chaînes de caractères etc. tout pour augmenter la portabilité du code et sa sûreté.



GTK+

Voici un schéma qui illustre la conception de GTK+ sous un system de type Unix :

2. Conventions

Tous les noms de fonctions et de types de données des trois bibliothèques décrites précédemment commencent par leur initiale. On retrouve ainsi des types `gint` ou `gfloat` qui correspond à la redéfinition des types `int` et `float` du langage c par GLib. Ou `gint32`, introduit par Glib (représente un entier dont on garantit que la taille reste de 4 octets quelle que

soit la plate-forme, à la différence du type `long` du langage C qui ne grandit pas toujours une taille de 4 octets)

Il est très important d'utiliser les types de base redéfinis par Glib dans les structures d'images pour s'assurer de la taille en octets qu'occupe un tel type et ce pour toutes les plates-forme.

Tous les types de GTK commencent par "Gtk" et toutes ses fonctions par "gtk_". De même pour GDK, les types commencent par "Gdk" et les fonctions par "gdk_".

Pour repercer cette convention, les type implémentés dans notre programme commenceront par "Tb" (pour Ter Bmp), et les fonctions par "tb_".

3. fonctionnement

GTK+ permet de construire facilement des interfaces graphiques en utilisant des composants affichables (Widgets) comme les boutons, les menu, surface de dessins etc. Ceux-ci sont mis dans des conteneurs (fenêtres, boites verticales, tableaux etc.) qui permettent de gérer leur position et leur taille.

GTK+ est basée sur la programmation événementielle. Elle associe à chaque composant graphique un ensemble d'événements (click pour un bouton out déplacement du curseur de la souris dans une surface de dessin par exemple) Il suffit d'attacher des fonctions aux évènements voulus et composants qui détiennent ces événements. Ceci s'effectue en utilisant des fonctions fournis par cette bibliothèque. Celles-ci prennent le type d'événement, le composant graphique et la référence vers la fonction qui doit traiter l'événement et les ajoute dans une table de gestion d'événement, interne au GTK+, pour que la fonction soit puisse être appeler.

Pour utiliser GTK+, il faut tout d'abord initialiser son environnement en appelant la fonction `gtk_init (gint argc, gchar *argv)` qui prend les argument de la fonction `main` en vue de récupérer les options qui doivent être passer à GTK+ par le programme. Ensuite, construire l'interface graphique et les gestionnaires d'événements. En fin, il faut lancer la fonction qui permet de traiter les évènements `gtk_main ()` Cette dernière est arrêtée si elle reçoit l'événement produit par la fonction `gtk_main_quit ()`, ce qui induit un arrêt du programme.

Pour obtenir une documentation complète (en anglais) ou le code source de GTK+ et les bibliothèques dont elle dépend, allez sur le site Internet www.gtk.org.

III. Structures et traitement des images

Cette partie traite tout ce qui a un rapport avec les données en mémoire d'une l'image. On peut spécifier deux types de données : l'image entière qui correspond à l'image lue à partir d'un fichier ou une région d'image déterminée par un ensemble de rectangles et les données pixels de chaque rectangle.

1. Les images BMP

Les images BMP peuvent avoir trois formats : Le format RGB (les composantes rouge, vert et bleu dans chaque pixel), le format niveaux de gris (l'intensité lumineuse du pixel est comme valeur de celui-ci) et enfin le format indexé (une table de couleur est utilisée, chaque pixel contient un indexe de sa couleur dans cette table)

Les algorithmes de lecture et d'écriture des images BMP ont étés pris à partir du code source du logiciel The Gimp. Ils ont été en suite adaptés à la structure de l'image utilisé par notre application. Ces algorithmes traitent notamment les images BMP compressées.

2. La structure et le traitement d'une image

Notre programme peut traiter trois types d'images : les images RGB, les images en niveaux de gris et les images indexées. On définit donc une union qui permet de différencier ces trois types :

```
typedef enum
{
  RGB,
  GRAY,
  INDEXED
} TbImageBaseType;
```

Une image possède un certain nombre d'attributs comme le montre la structure suivante (chaque attribut est expliqué en commentaire) :

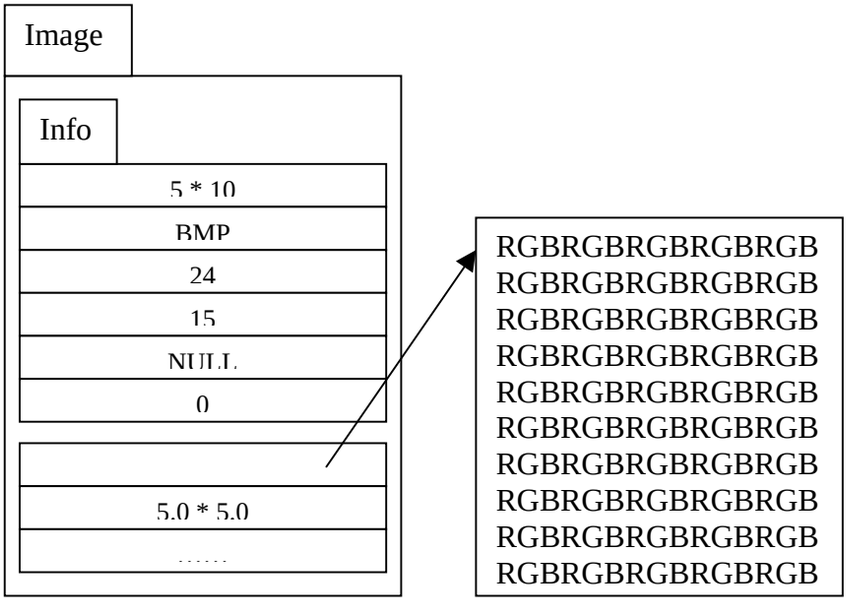
```
struct _TbImageInfo
{
  gint               width, height; /* Largeur et hauteur de l'image        */
  TbImageBaseType    base_type;     /* Le type de l'image RGB, GRIS..       */
  gint               bpp;           /* Nombre de bits par pixel             */
  gint               rowstride;     /* Nombre d'octets par ligne de d'image */
  guchar            *cmap;          /* La colormap pour les indéxées        */
  gint               nb_cols;       /* Nombre de couleurs pour les indéxées */
};
```

La "colormap" est la table de couleurs si l'image est indexée, le nombre de couleurs dans cette table est "nb_cols".

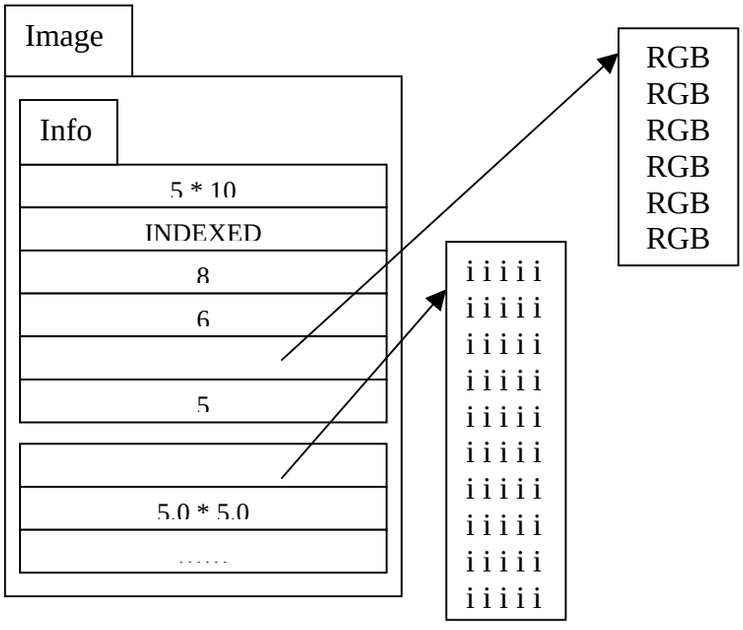
L'image est représentée en mémoire par ses attributs et ses données pixels (un tableau de caractères de dimension `bpp / 8 * rowstride * height`) et sa résolution en x et en y. Voici la structure commentée d'une image en mémoire :

```
struct _TbImage
{
  /* TbImageInfo (pour permettre le cast) */
  gint               width, height; /* Largeur et hauteur de l'image        */
  TbImageBaseType    base_type;     /* Le type de l'image RGB, GRIS..       */
  gint               bpp;           /* Nombre de bits par pixel             */
```

```
  gint                rowstride;     /* Nombre d'octets par ligne de pixels  */
  guchar             *cmap;          /* La colormap pour les indéxées        */
  gint                nb_cols;       /* Nombre de couleurs pour les indéxées */

  guchar             *data;          /* Les données pixels                   */

  gdouble             xresolution;   /* Resolution en x en dpi               */
  gdouble             yresolution;   /* Resolution en y en dpi               */

  gboolean            modified;      /* Indique si l'image a été modifié     */
  guchar             *filename;      /* Le nom du fichier de l'image         */
};
```



Exemple d'une image RGB



Exemple d'une image indexée

Un ensemble de fonctions de modification des données d'une image est implémenté. Celles-ci prennent en paramètre les coordonnées du rectangle de pixels à modifier dans l'image. Ceci étant permet plus de flexibilité, à savoir que toute l'image peut être tout de même modifiée en entier en spécifiant le rectangle de coordonnées (0, 0, largeur de l'image; hauteur de l'image). Ces fonctions prennent aussi en paramètres les attributs de l'image afin de savoir comment effectuer les modifications.

Voici les modifications de base qu'on peut appliquer à une image :

Renvoyer un rectangle de pixels :

```
guchar *tb_image_get_area_data  (TbImageInfo *src_info, guchar *src_data,
                                 gint x, gint y, gint w, gint h);
```

Cette fonction renvoie une copie des données d'une région rectangulaire (x, y, w, h) à l'intérieure d'une image dont les attributs (src_info) et les données pixels (src_data) sont données. Cette fonction ne prend pas directement une image en paramètre. Ceci pour pouvoir l'utiliser avec des images n'ayant pas une structure complète.

Copie d'une région rectangulaire d'une image vers une autre :

```
void    tb_image_move_area_data (TbImageInfo *src_info, guchar *src_data,
                                 TbImageInfo *dst_info, guchar *dst_data,
                                 gint src_x, gint src_y,
                                 gint dst_x, gint dst_y,
                                 gint w, gint h);
```

Cette fonction permet de copier une région rectangulaire de taille w*h d'une image vers une autre image ayant les mêmes attributs type et bpp (la vérification d'égalité des attributs type et bpp est effectuée). Pour cela on donne les attributs de l'image source et de l'image destination

et les coordonnées du rectangle dans les deux images. Le débordement du rectangle dans les deux images est traité grâce aux attributs de celles-ci.

Effacement d'une région rectangulaire dans l'image

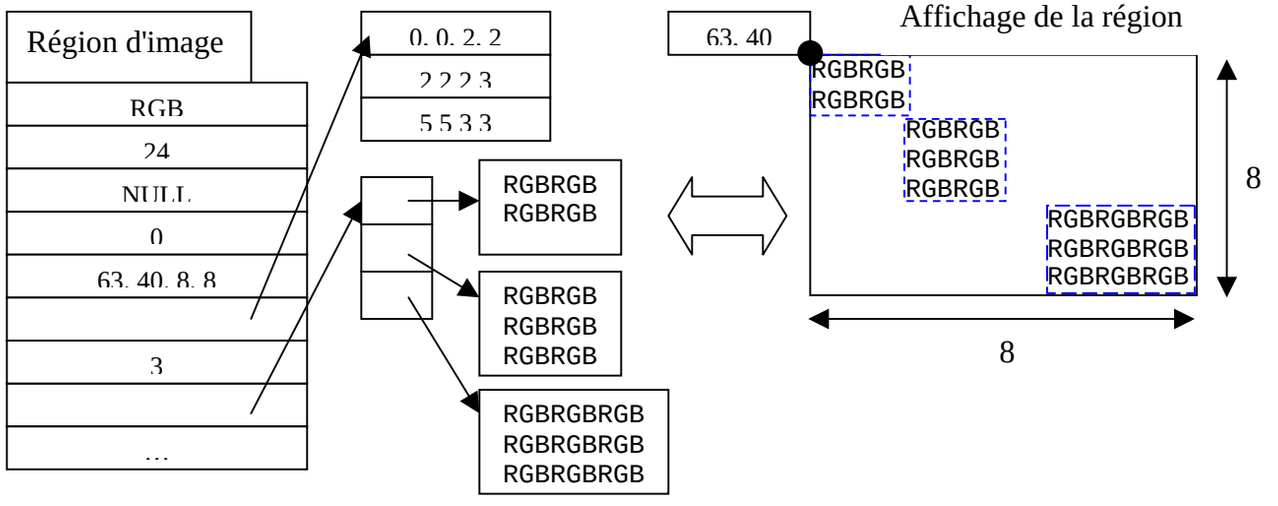
```
void    tb_image_blank_area_data (TbImageInfo *info, guchar *data, gint x, gint y,
                                  gint w, gint h);
```

3. La structure et le traitement d'une région d'image

Une région d'image est constituée d'attributs de l'image, un ensemble de rectangles de l'image qui constitue la région et les données pixels de chaque rectangle.

Voici la structure d'une région d'image :

```
struct _TbImageRegion
{
  TbImageBaseType   base_type;     /* Le type d'image de la région (rgb, niveau de
                                      gris...) */
  gint              bpp;           /* Nombre de bits par pixel */
  guchar           *cmap;          /* La colormap si la région est indéxée */
  gint              nb_cols;       /* Nombre de couleurs dans la colormap */

  GdkRectangle      clipbox;       /* Le plus petit rectangle qui inclut toute la
                                      région en coordonnées image */
  GdkRectangle     *rectangles;    /* Les rectangles correspondant à la région de
                                      l'image en coordonnées / clipbox */
```



Exemple d'une région d'image

```
  gint              nb_rectangles; /* Le nombre de rectangles */
  guchar          **data;          /* Les données pixels de tous les rectangles */
  guchar           *name;          /* Un nom donné à la région d'image */
};
```

Une région d'image peut être initialisée en donnant l'ensemble des rectangles qu'il faut copier et les données où il faut effectuer la copie ainsi que les attributs de ces données. En suite, une région peut être convertit en lui donnant des nouveaux attributs. Ses données pixel sont alors converties. Une région d'image ne fait qu'utiliser les fonctions définies précédemment pour un rectangle d'image. En général, elle parcourt toutes ses surfaces rectangulaires (à l'aide d'une boucle pour), et appelle la fonction en question avec comme paramètres la surface rectangulaire et les attributs de la région et du rectangle (largeur et hauteur)

4. Les filtres

Toutes les fonctions qui appliquent un filtre à une image prennent en paramètre l'image à traiter et la région (ensemble de rectangles) sur laquelle appliquer le filtre.

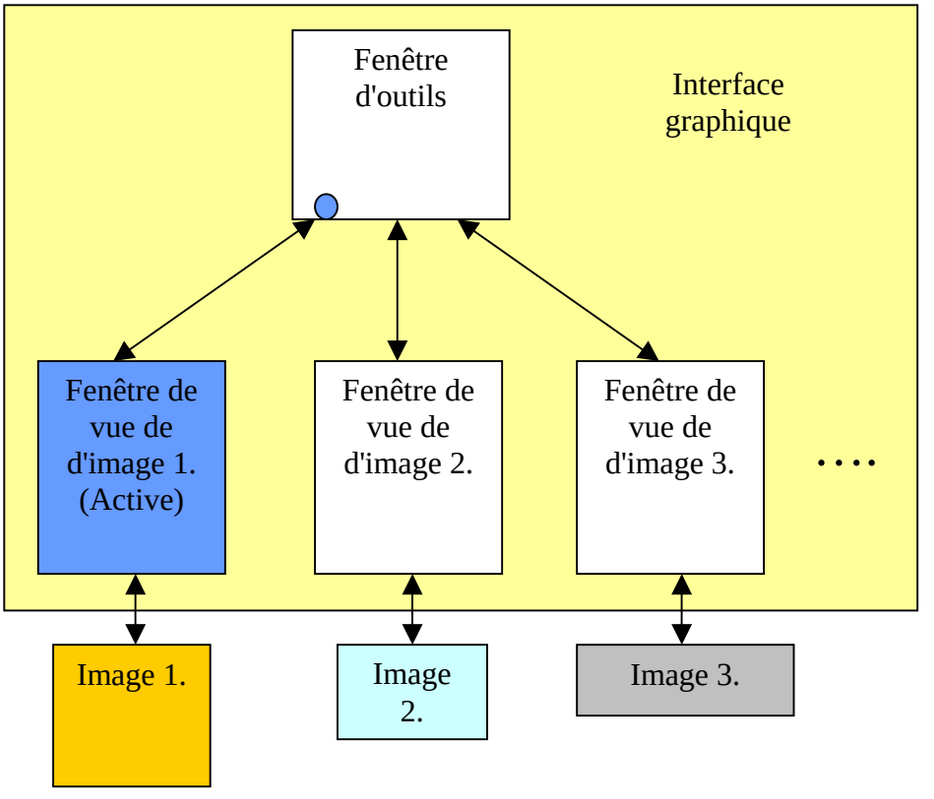
Seule la fonction qui génère un négatif est implémentée.

```
void tb_filters_negatif (TbImage *image, GdkRectangle *rectangles, gint
                         nb_rectangles);
```

IV. L'interface graphique

Notre application fournit un environnement d'édition constitué d'une fenêtre principale appelée la fenêtre d'outils et d'une fenêtre affectée à chaque image ouverte pour l'afficher. Cette dernière sera appelée la fenêtre de vue d'une image.

La première fenêtre permet de sélectionner des outils à utiliser et dispose d'un ensemble de commandes accessibles à partir d'un menu.



Organisation des fenêtres de l'application

Les deux types de fenêtres sont implémentés à l'aide de deux structures, avec une référence de chaque fenêtre de vue vers la fenêtre d'outils, ce qui lui permet d'accéder à ses fonctionnalités (accès au menu par exemple) la fenêtre d'outils dispose d'une liste chaînée (`Glist` utilisée) de références sur toutes les fenêtres de vues. Cela permet de naviguer entre ces fenêtres ou de les libérer à la fin du programme par exemple.

A tout moment une seule fenêtre de vue est activée. La fenêtre d'outils dispose de la référence vers cette fenêtre. C'est l'image affectée à celle-ci qui reçoit les opérations courantes.

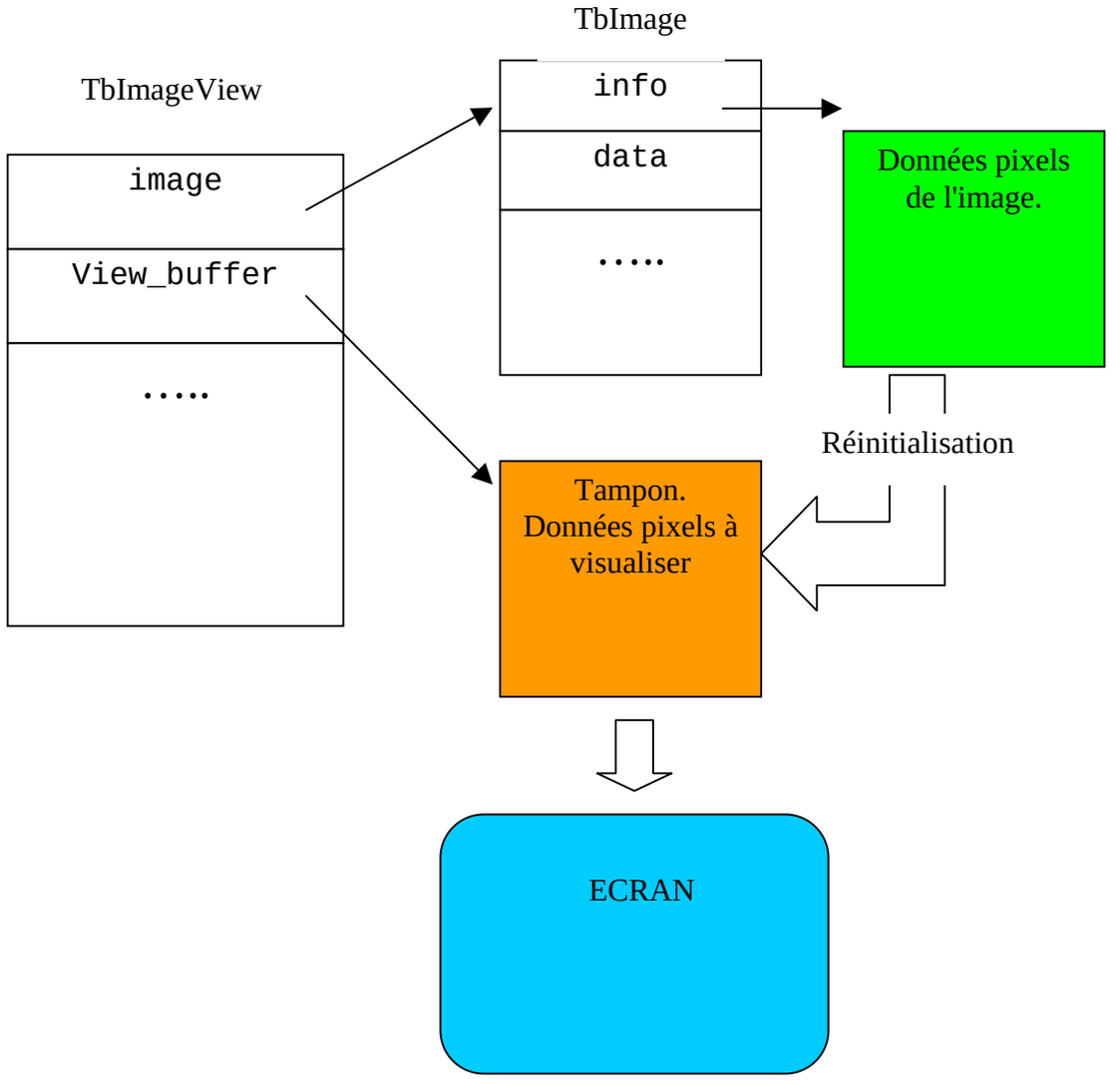
Chacune des deux structures dispose d'une référence vers une fenêtre GTK de haut niveau (avec décorations : barre de titre, bordures etc.) C'est cette fenêtre qui sera affichée à l'écran.

Au niveau de la fenêtre d'outils sont traités, le menu principal qui est unique pour toutes les fenêtres de vue et une barre d'outils.

Au niveau de la fenêtre de vue, un ensemble de composant graphique permettant d'afficher l'état actuel de l'image (sont type, le zoom…), et bien entendu une surface de dessin où est projetée l'image.

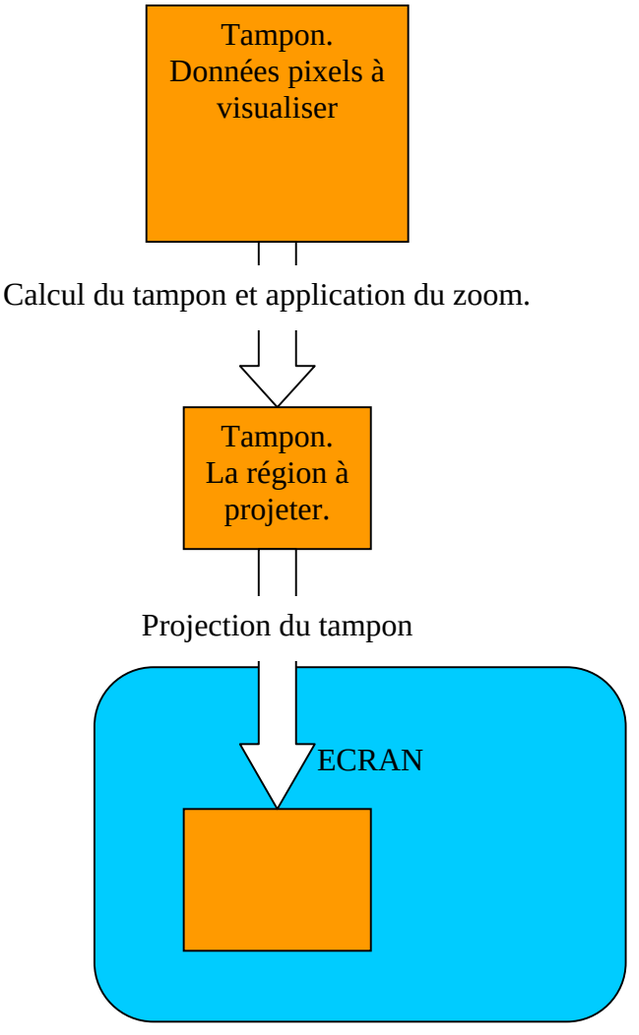
1. Projection d'une image

La structure d'une image de vue (`TbImageView`) possède un tampon de pixels (`view_buffer`) de même type et de même taille que celui de l'image. Celui-ci sert à modifier le type de visualisation de l'image sans modifier ses données pixels. Par exemple, si l'on veut visualiser la composante rouge des pixels de l'image (désactivation des canaux vert et bleu), alors seul le tampon de visualisation modifié. En effet ces opérations ne correspondent pas à des opérations de modification de l'image.



Utilisation d'un tampon à visualiser

Projeter le tampon de pixels directement sur l'écran ne suffit pas pour avoir toutes les visualisations possibles. En effet, en cas d'application d'un zoom sur l'image, celui-ci doit être agrandit ou diminué selon le zoom. Or un agrandissement important va conduire à une l'allocation d'une zone mémoire aussi importante pour toute la surface de l'image. L'idée est d'utiliser un troisième tampon (`render_buffer`) qui va avoir seulement la taille de la zone d'image visible à l'écran. Le zoom va maintenant s'appliquer sur ce dernier tampon sans l'agrandir car ça ne sert à rien, puisque tout ce qui sera alloué de plus ne sera pas visible à l'écran. L'inconvénient du tampon (`render_buffer`) est qu'il doit être réallouer et recalculer à chaque fois que la zone de l'image à afficher change (en utilisant les barres de défilement par exemple) Une amélioration serait de garder la taille de ce tampon à la taille de l'écran entier pour éviter la ré allocation de la mémoire et diminuer les calculs de copie de mémoire.



Projection de l'image

Une vue d'image possède toutes les informations sur la zone de l'image à projeter, notamment les coordonnées de cette zone dans l'image et dans la surface de dessin (donc à l'écran puisqu'une surface de dessin représente une surface dessinable dans l'écran)

La projection du tampon final sur l'écran est effectuée par appel à une primitive de dessin de GDK (`gdk_draw_rgb_image`), cette fonction prend le tampon de type RGB (Red Green Blue) et l'affiche dans un objet dessinable telle que la surface de dessin (`GtkDrawingArea`). Cela implique que le tampon de projection doit être toujours de type RGB, une conversion doit donc être effectuée si l'image d'origine n'est pas de type RGB (opération très rapide).

2. Calcul du zoom

Un zoom est un nombre réel égale au rapport entre la taille de l'image sans zoom et taille de l'image après le zoom.

Algorithme : il faut un indice réel a sur le pixel courant dans le tampon de visualisation sans zoom. Un indice entier i suit le pixel courant dans le tampon de projection à calculer. Pour i parcourant tous les pixel du tampon de projection faire à chaque fois une copie du pixel à l'indice a du tampon de visualisation (avec un arrondi) vers le pixel à l'indice i du tampon de projection et l'augmentation de a de zoom (`a <- a + zoom`)

Cet algorithme est relativement lent à cause de l'utilisation de réels. Pour éviter cela, il faut utiliser deux entiers a et b pour spécifier le zoom plutôt qu'un réel. Le zoom sera alors égale à a / b (a longueur dans l'image avant le zoom devient égale à b après le zoom). Les calculs ne s'effectuent alors qu'avec des entiers en utilisant des boucles pour.

3. Calcul des sélections

Conversion d'une sélection polygonale à un ensemble de rectangles

L'idée est de sauvegarder la sélection en mémoire sous forme d'un ensemble re rectangles. Cela simplifie largement les calcules d'union, d'intersection, … entre sélections. Il faut donc dès le départ convertir les sélections polygonales (ensemble de points) effectuées par l'utilisateur vers des rectangles.

Il existe en GDK la structure GdkRegion qui permet de manipuler un ensemble de rectangles et qui permet de générer cet ensemble à partir d'un polygone. Cette structure a été utilisée pour simplifier le travail. Mais le problème qui se pose, c'est comment reconvertir l'ensemble des rectangles vers un polygone qui les entoure pour pouvoir l'afficher. Sachant que cette fonction n'existe dans le GDK, elle a été donc implémentée localement.

Chaque sélection effectuée est aussitôt affichée dans la surface de dessin en faisant un appel à une primitive de dessin de GDK qui prend en paramètre un polygone à afficher.

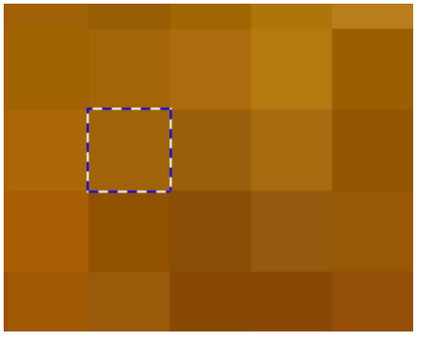
Pour l'animation de la sélection, une fonction de Glib, qui permet de créer un processus à exécution régulière dans un temps à spécifier, a été utilisée. A chaque pas d'animation une icône que le GDK utilise pour remplir le polygone à afficher est décaler pour donner au polygone affiché l'apparence d'une fourmilière.

4. Copie et collage de région d'image

Un presse-papiers est intégré dans la fenêtre d'outils pour pouvoir faire le lien entre les différentes images ouvertes. Celui-ci sauvegarde une copie de la région d'image copiée. Au collage de la région, celle-ci est dessinée dans le tampon de visualisation. Ce-ci permet de faire déplacer cette région, car les zones qu'elle écrase dans le tampon de visualisation sont rechargées à partir de l'image (jusqu'ici pas encore modifiée) Pour coller définitivement la région, celle-ci est dessinée dans l'image même.

V. Glossaire et conventions

Pixel d'image : La plus petite partie homogène d'une image (qui ne peut avoir qu'une seule couleur). Le pixel d'une image n'a pas toujours la taille d'un pixel écran (notamment en cas d'un zoom).



Un pixel sélectionné par le logiciel après un zoom

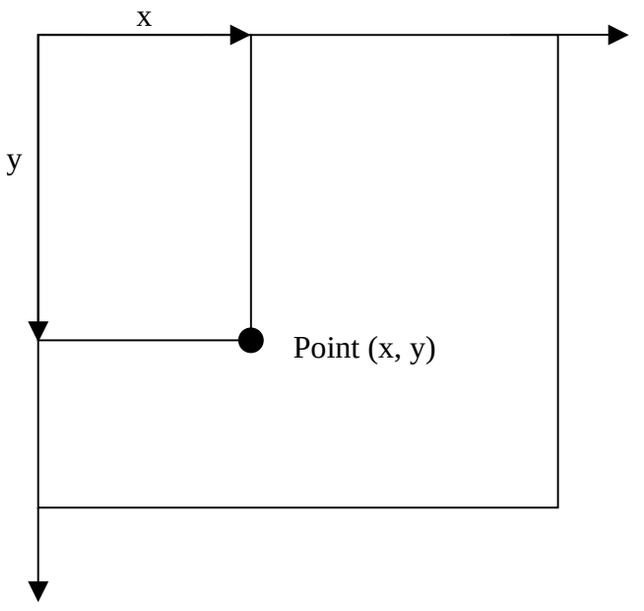
bpp : (bits per pixel) le nombre bits dans lesquels est codé un pixel (3 * 8 bits pour les images RGB par exemple)

rowstride : (le grand pas d'une ligne) le nombre d'octets sur lesquels est codée une ligne d'image. Ceci set à effectuer des changements de ligne lorsqu'on traite un rectangle dans l'image. Ce nombre ne peut pas être toujours déterminé par la largeur de l'image (par exemple pour les images BMP, cette valeur est toujours paire pour pouvoir lire l'image mot par mot)

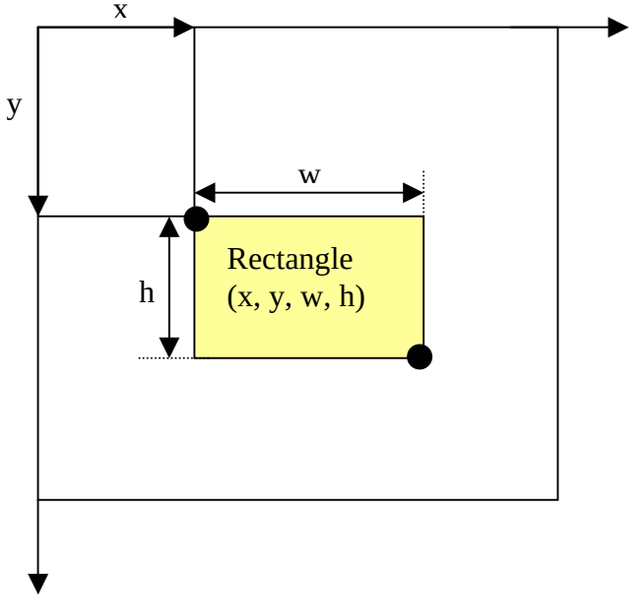
Résolution en x d'une image : C'est le nombre de pixels horizontales qu'il faut pour constituer une longueur égale à un pouce (inch en anglais, 25.4 mm) dans l'image affichée à l'écran ou imprimée sans agrandissement ou réduction.

Résolution en y d'une image : La même chose mais verticalement.

Coordonnées (x, y) dans une image : C'est les coordonnées par rapport au coin supérieur gauche d'une image en nombre de pixels en partant de gauche à droite et de haut en bas.



Rectangle (x, y, w, h) dans une image : C'est le rectangle dont les coordonnées du coin supérieur gauche par rapport à l'image sont (x, y) et ceux du coin opposé sont (x + w, y + h). w et h sont alors respectivement la largeur et la hauteur du rectangle.



VI. Améliorations

- Pour pouvoir annuler les modifications récentes appliquées à une image, on peut intégrer un service de "annuler/refaire" qui permet de sauvegarder les opérations réversibles effectuées ou les régions modifiées pour les opérations irréversibles, ce qui permet d'annuler ces opérations.
- On peut ajouter d'autres filtres d'images.
- La transparence peut être traitée en introduisant un canal alpha aux images (RGBA), celui-ci représente la valeur de transparence de la couleur RGB.
- On peut intégrer l'utilisation de calques d'images. Les calques sont des surfaces pouvant avoir une transparence et qui lorsqu'on les superpose, on obtient la surface de l'image finale. L'utilité des calques est de pouvoir modifier la couleur d'un pixel différemment selon les calques actifs.
- On peut ajouter une fonctionnalité pour pouvoir utiliser des modules chargeable à l'exécution (des fichier d'extension .so sous UNIX ou .dll sous windows). Ces modules peuvent être édités et compilés en ligne.
- On peut intégrer un module pour automatiser les actions à réaliser sur une image en spécifiant un langage de script.
- Beaucoup d'algorithmes utilisés peuvent être optimisés pour une vitesse d'exécution maximale. On en site par exemple l'algorithme d'application d'un zoom et celui de calcule de segments d'une sélection.
- D'autres services : informations sur une image, boites de dialogue diverses …
- Le traitement d'images indexées n'a pas été terminé.

VII. Moyens utilisés

- La bibliothèque GTK+
- Le compilateur gcc et l'outil make
- Les éditeurs de fichiers source xemacs et dev-cpp
- The Gimp pour les captures d'écran
- Le code source de The Gimp pour la lecture et écriture d'images BMP
- Microsoft Word pour l'édition du rapport
- Environnements : Linux et Windows.