



EISTI

---

## Projet Quadtree

---

ALGORITHMIQUE ET PROGRAMMATION  
FONCTIONNELLE

CPI1 C2 - GROUPE 2

Amine TAGHROUT  
David RIGAUX  
Mehdi DALAA

7 janvier 2017

## Table des matières

	Page
<b>Introduction</b>	<b>2</b>
<b>1 Étude du projet</b>	<b>2</b>
<b>2 LablGTK</b>	<b>3</b>
2.1 GTK+ . . . . .	3
<b>3 Arbre Quadtree</b>	<b>3</b>
<b>4 Fonction I/O</b>	<b>4</b>
4.1 Chargement . . . . .	4
4.2 Enregistrement . . . . .	4
<b>5 Fonctions de manipulation</b>	<b>5</b>
5.1 Opérations Simples . . . . .	5
5.1.1 Rotation de 90° . . . . .	5
5.1.2 Miroir . . . . .	5
5.1.3 Inversion . . . . .	5
5.2 Opérations Avancées . . . . .	6
5.2.1 Compression . . . . .	6
5.2.2 Segmentation . . . . .	6
<b>6 Graphiques</b>	<b>6</b>
6.1 View 1 . . . . .	9
<b>7 Bilans Personnel</b>	<b>12</b>
7.1 David RIGAUX . . . . .	12
7.2 Amine TAGRHOUT . . . . .	13
7.3 Mehdi DALAA . . . . .	13
<b>Conclusion</b>	<b>13</b>

## Introduction

L'objectif de notre projet en informatique est de réaliser un programme pouvant manipuler des images encodées sous forme d'arbre. Pour réaliser ce projet, nous avions un temps imparti d'un mois, et le langage de programmation devait être le OCaml. Les images que nous devons traiter sont de type *.ppm*.

Le *portable pixmap file format* (PPM) est un format de fichier utilisé pour les échanges. Il a été défini et est utilisé par le projet NetPBM. Le format de fichier est utilisé pour des images couleur. Chaque pixel est codé par trois valeurs (rouge, vert et bleu). Le codage d'une image en PPM est le suivant :

1. P3 (Numéro magique)
2. Largeur Hauteur (Dimensions)
3. MaxVal (Valeur maximum des couleurs)
4. Les données de l'image : succession des valeurs associées à chaque pixel

Comme on l'avait fait remarquer, on ne s'intéresse qu'aux images carrées de taille  $2^n * 2^n$  une image unicolore se représente par sa couleur, tandis qu'une image composite se divise naturellement en quatre sous-images carrées. Les feuilles d'un *quadtree* correspondent donc à une zone de l'image ayant une couleur unique. Les noeuds internes stockent la moyenne des couleurs de leurs fils

## 1 Étude du projet

Tout d'abord et puisque une image ppm est représentée par des matrices (listes de listes) il fallait faire des manipulations sur des fichiers constitués de matrices. La première manipulation qu'on avait faite était de lire un fichier en utilisant la fonction *readfile*. Cette dernière prend en paramètre un fichier et permet de le lire ligne par ligne. Cependant, le fichier lu peut éventuellement contenir des commentaires, des caractères spéciaux ou des espaces. C'est pour cela qu'on avait fait la fonction *filtre* qui prend en paramètre une liste de string et qui enlève tous les éléments indésirables de façon à avoir à la fin un fichier qui ne contient que le nom, la taille de l'image et les suites de pixels. L'étape suivante était de créer une fonction qui permet de regrouper les pixels sous forme de triplets et pour cela on n'avait besoin que des pixels. C'est pour cela qu'on avait besoin de la fonction *clean* qui enlève le titre et les dimensions de l'image de façon à ne garder que les pixels. A partir de ce moment, on pouvait utiliser la fonction *regroupe* qui prend en paramètre une liste de string et qui renvoie une liste de triplets d'entiers. On avait créé aussi des fonctions complémentaires comme *superlecture* qui permet à l'utilisateur de lire et en même

temps filtrer le fichier voulu. De plus, et comme on peut clairement le voir dans le quadtree, les noeuds internes stockent la moyenne des couleurs de leurs fils. C'est pour cela qu'une fonction moyenne était nécessaire mais avant cela il fallait créer un fonction qui calcule la somme des composantes des triplets de la liste des pixels et c'est ce que fait la fonction somme qui prend en paramètre une liste de triplet et qui renvoie le triplet qui a pour composantes la somme de toutes les composantes. Comme vous pouvez le remarquez, la fonction moyenne prend en paramètre le triplet renvoyé par la fonction somme et la liste initiale et elle renvoie un triplet qui a comme composantes la moyenne de toutes les autres. Après cela, nous avons également fait des fonctions supplémentaires comme *superlecture* qui permet de lire et de filtrer en même temps le fichier voulu, ou encore la fonction *moyenne* qui prend en paramètre le fichier initiale et qui renvoie directement la moyenne de pixels. La deuxième partie du projet consistait à créer le type *quadtree* et biensûr créer l'arbre *quadtree* en utilisant la fonction *creerArbre*. Tous cela, pour pouvoir faire différentes manipulations et opérations sur l'image. Parmi ces opérations on trouve les simples et les complexes.

De plus, pour faciliter l'utilisation du programme codé, nous avons créé une interface graphique faite à l'aide de la librairie graphique *LablGTK2*.

## 2 LablGTK

LablGTK est une interface OCaml du projet GTK+ 1.2 et 2.x. Cette librairie nous permet de créer des interfaces graphiques orienté-objets.

### 2.1 GTK+

**GTK+** (*The GIMP Toolkit*) est un ensemble de bibliothèques logicielles, c'est-à-dire un ensemble de fonctions permettant de réaliser des interfaces graphiques. Cette bibliothèque a été développée originellement pour les besoins du logiciels de traitement d'image GIMP. GTK+ est maintenant utilisé dans de nombreux projets, dont les environnements de bureau *GNOME*, *Xfce*, *Lxde* et *ROX*.

GTK+ est un projet libre (licence GNU LGPL 2.1) et multiplateforme.

## 3 Arbre Quadtree

Qu'est ce qu'un arbre Quadtree et comment on l'a créé

## 4 Fonction I/O

Les fonctions I/O (*In/Out*) dans notre projet sont indispensable pour pouvoir faire des manipulations d'images. Nous avons en tout quatres fonctions I/O que nous allons vous présenter dans les sous-parties suivantes.

### 4.1 Chargement

Pour ce qui est du chargement, nous avons créé deux fonctions qui font deux opérations distinctes l'un de l'autre.

Sachant que nous ne pouvons pas stocker un arbre en OCaml nous avons donc décider de créer un fichier temporaire au moment où l'utilisateur fait le choix de l'image à modifier. En effet, cela nous permet donc de travailler avec l'image mais sans modifier l'image de départ et nous permet également ainsi de stocker les modifications. Donc nous travaillons principalement avec le fichier temporaire créer suite au chargement de l'image.

La deuxième fonction de chargement est la fonction qui nous permet de lire le fichier et de ressortir un arbre de type *quadtree* pour faire ensuite les modifications en questions.

### 4.2 Enregistrement

Pour l'enregistrement, nous avons, comme pour le chargement, deux fonctions distincts.

Une fois le fichier chargé sous forme d'arbre et modifié nous avons fais une fonction qui ressort l'arbre dans le fichier temporaire pour pouvoir stocker l'opération faite à l'image.

La deuxième fonction, n'entre en jeu dès que l'on veut sauvegarder l'image et ressortir la version finale. L'utilisateur a la possibilité d'enregistrer-sous ou tout simplement d'écraser l'ancienne image en enregistrant. Commençons par expliquer la fonction permettant d'enregistrer. Sachant que nous travaillons avec un fichier temporaire, qui a toute les modifications déjà faite, il ne nous suffit qu'à supprimer l'image original et de renommer ensuite le fichier temporaire au nom de base. Cependant, nous recréons un fichier temporaire aussitôt pour laisser la possibilité à l'utilisateur de continuer à travailler sur son image. Maintenant pour ce qui est de l'enregistrement-sous, il y a la même logique que l'enregistrement simple, mais cette fois nous ne supprimons pas l'image de original. En effet, nous ne faisons que déplacer le fichier temporaire, en le renommant avec le nom que l'utilisateur souhaite. Si l'utilisateur

ne mets pas l'extension `.ppm` à la fin du nom de fichier souhaité, nous avons créé une petite fonction qui permet de faire la vérification et en cas d'absence de cette extension nous la rajoutons à la fin du nom au moment de renommer le fichier. De plus, comme pour l'enregistrement, nous créons aussitôt un autre fichier temporaire pour laisser la possibilité à l'utilisateur de continuer à travailler avec cette image.

## 5 Fonctions de manipulation

### 5.1 Opérations Simples

#### 5.1.1 Rotation de 90°

1. Rotation vers la droite : cette opération consiste à effectuer une rotation de l'image vers la droite. Pour cela, on a utilisé la fonction `rotated` qui décale les fils d'une place vers la droite
2. Rotation vers la gauche : cette opération consiste à effectuer une rotation de l'image vers la gauche. Pour cela, on a utilisé la fonction `rotateg` qui décale les fils d'une place vers la gauche

#### 5.1.2 Miroir

1. Miroir haut-bas : cette opération consiste à créer un arbre de façon à ce que les feuilles soient interverties de haut en bas. Par exemple :
 

1 2	devient	4 3
4 3		1 2
2. Miroir gauche-droite : cette opération consiste à créer un arbre de façon à ce que les feuilles soient interverties de haut en bas. Par exemple :
 

1 2	devient	2 1
4 3		3 4

#### 5.1.3 Inversion

Cette opération consiste à inverser les couleurs de l'image. Ce que fait la fonction `inverse` qui prend en paramètre le triplet représentant la moyenne des couleurs et qui renvoie le triplet dont les composantes sont la différence avec 255 qui est le maximum des pixels.

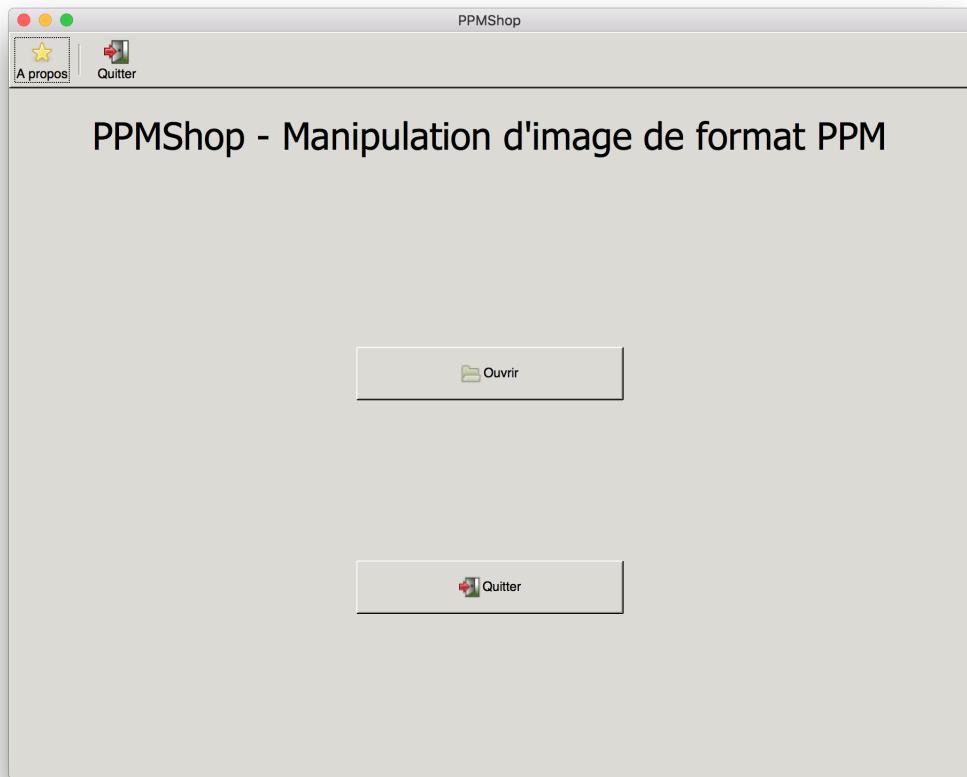
## 5.2 Opérations Avancées

### 5.2.1 Compression

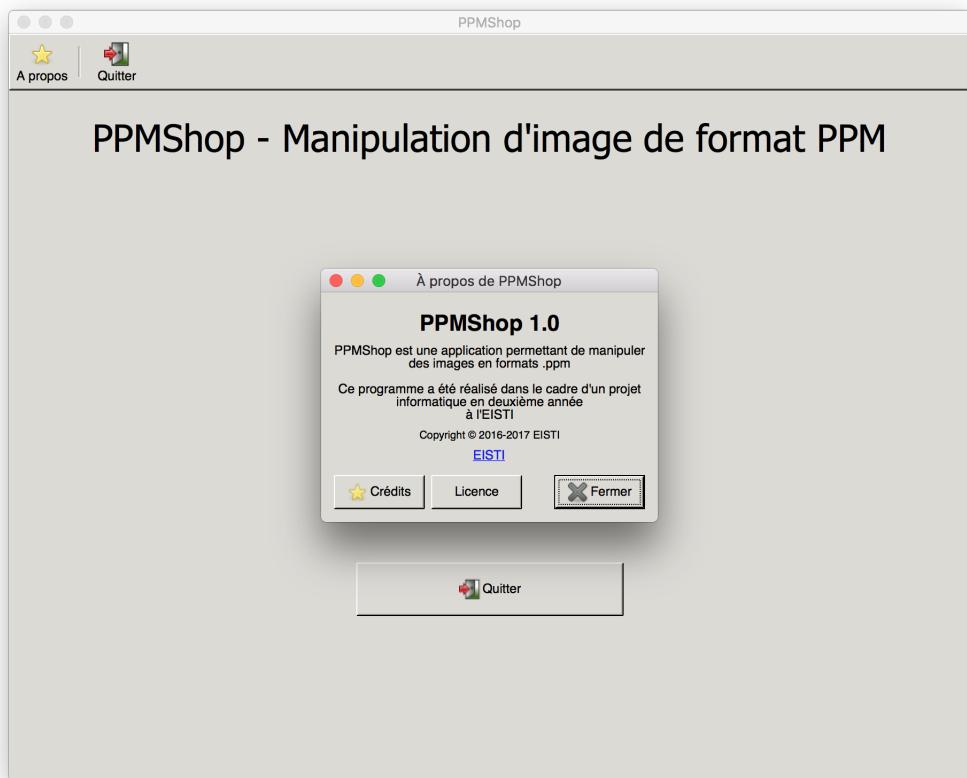
### 5.2.2 Segmentation

## 6 Graphiques

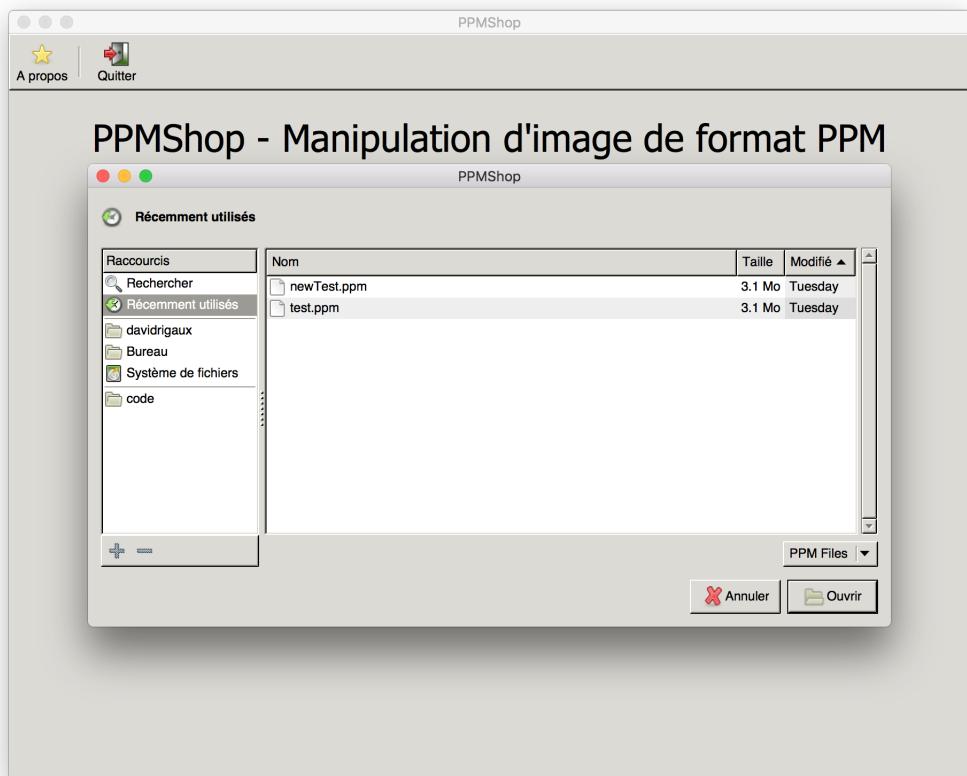
Pour pouvoir créer une bonne interface graphique, il nous a fallu penser à tout ce qu'un utilisateur aurait besoin d'avoir et à quel moment. Voici ce que l'utilisateur aperçoit dès l'ouverture de l'application :



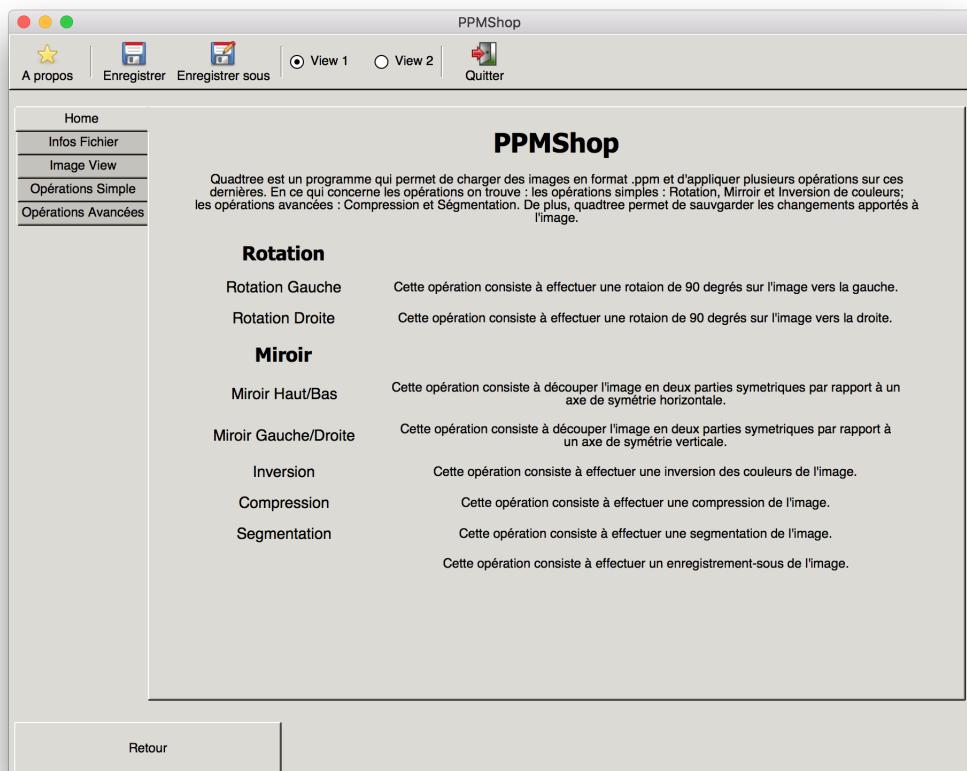
Il peut ici choisir entre avoir plus d'ample informations sur l'application *PPMShop*, quitter l'application et charger une image. Si l'utilisateur appuie sur le bouton *À propos*, cet écran sera affiché :



Maintenant s'il préfère commencer à vraiment utiliser *PPMShop* et ainsi charger une image en appuyant le bouton *Ouvrir* la fenêtre suivante s'ouvrira :



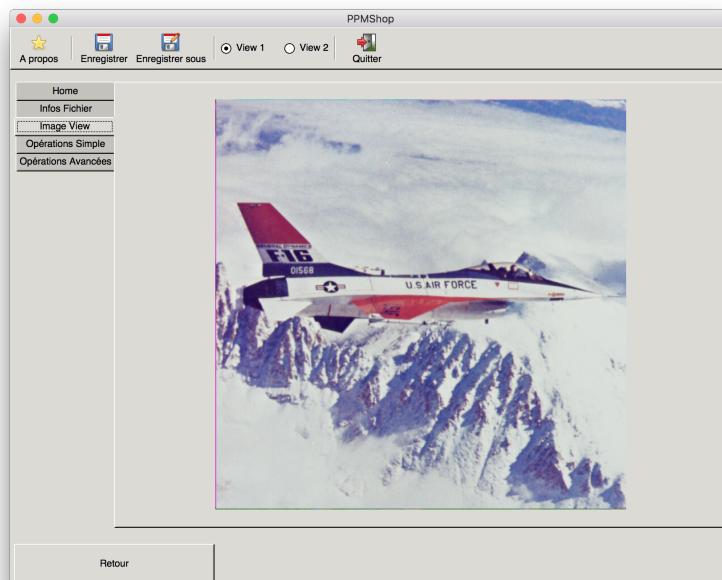
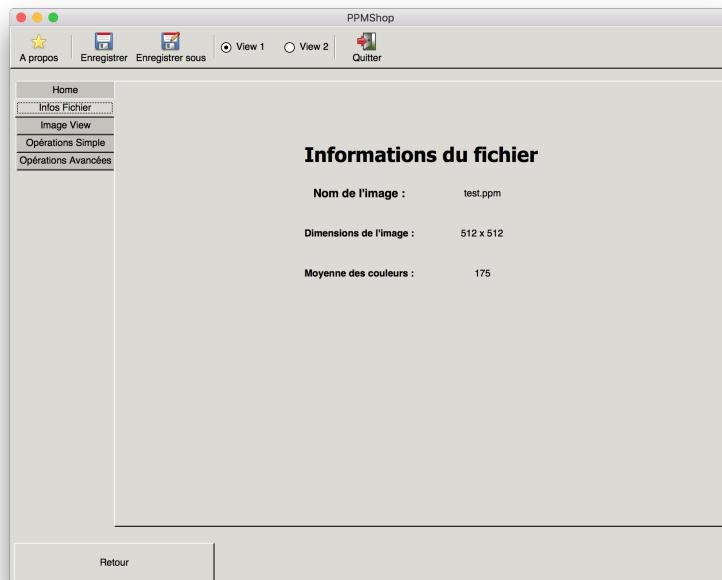
Ici il pourra alors naviguer parmi tout les fichiers présent sur son ordinateur et ainsi choisir son image en format *.ppm*. On peut également remarquer que dans la partie inférieure droite de cette fenêtre nous avons rajouté un filtre pour les fichiers qui permet d'afficher que les fichiers ayant une extension *.ppm*. Après avoir choisis son fichier, nous faisons certaines vérifications sur l'image choisie (dimensions carrées du fichier, d'extension *.ppm* et de format **P3**). Une fois les vérifications passées, notre programme ayant plusieurs interface possible, il commence systématiquement par afficher la *View 1*, qui correspond à l'interface par défaut.

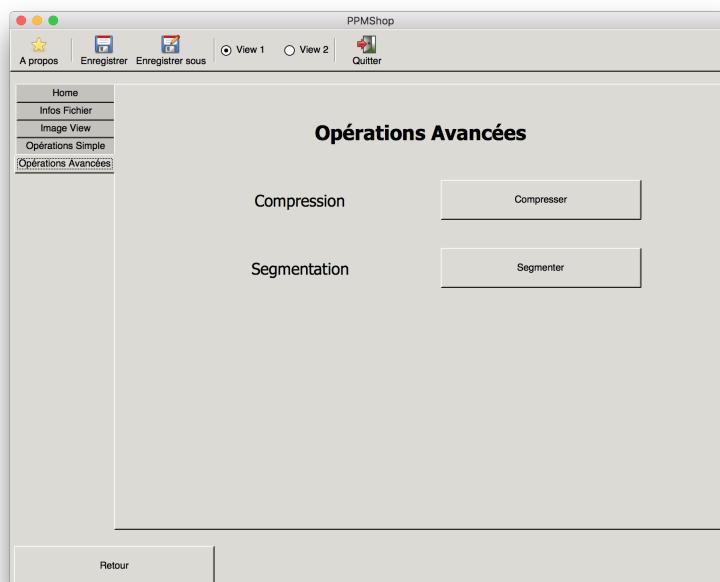
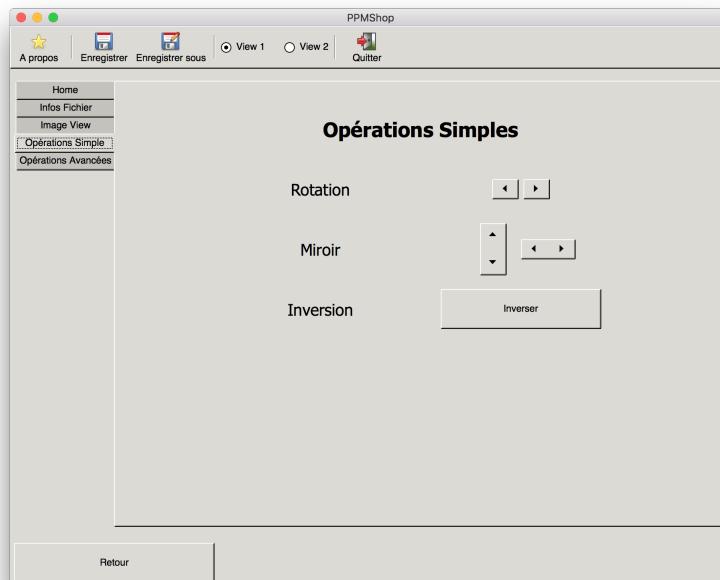


Nous avons ici, accès à toute les informations par rapport à l'image, que ce soit ces dimensions, son nom, un affichage de la photo dans son état actuel, et la possibilité d'appliquer les opérations simples et avancées.

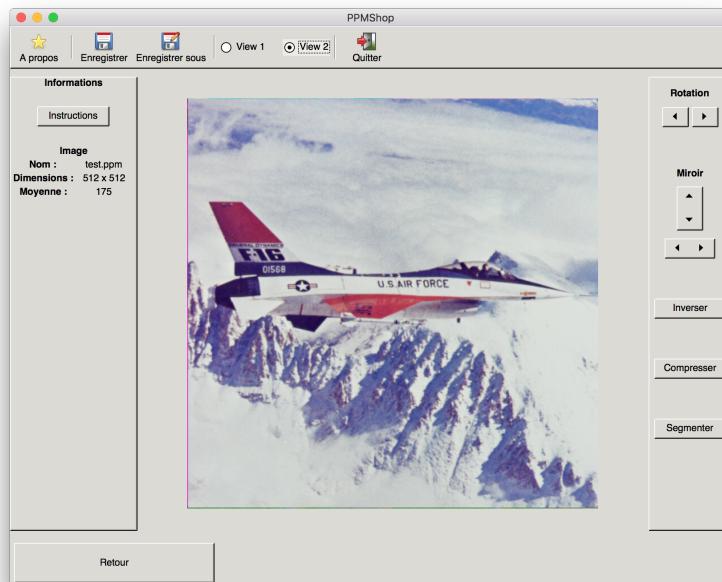
## 6.1 View 1

Dans la *View 1*, nous avons décidé d'utiliser des onglets qui affiche les informations et les boutons que l'on souhaite avoir ou utiliser.





Si l'utilisateur préfère utiliser la *View 2*, il lui suffit d'appuyer sur le bouton radio qui se situe sur la barre à outils sur la partie supérieur de l'application. Celle-ci ressemble à ceci :



Qui est une vue plus compacte, qui donne accès à toute les fonctionnalités sur un écran et la possibilité de voir le changement de l'image en temps réel.

## 7 Bilans Personnel

### 7.1 David RIGAUX

Pour ma part je me suis essentiellement occupé de l'interface graphique et j'ai trouvé ça super intéressant à faire. En effet, j'ai déjà essayé de faire une interface graphique il y a quelques années pour un autre projet d'informatique mais je n'y comprenais pas vraiment avec tout ce qui était objets, widgets et conteneurs, mais cette fois ci j'ai réussi à comprendre le principe et à apprendre comment ces notions marchaient. Même si j'ai eu quand même beaucoup de mal à arriver au résultat final (avec des problèmes incompréhensible, le fait que l'ordinateur calcul toute les positions des objets). Cependant, je suis content de ce que j'ai réussis à créer.

## 7.2 Amine TAGRHOUT

## 7.3 Mehdi DALAA

## Conclusion

Le projet de manipulation d’images semblait facile à faire mais quand on y intègre un arbre cela rajoute un certain nombre de difficultés. De plus intégrer toute les fonctions demandés dans l’interface graphique était une certaine épreuve mais nous avons réussis à faire ce qui était demandé. Nous avons tous appris un peu plus que ce soit sur les structures d’arbres, les fonctionnalités d’OCaml et surtout l’utilisation de ce langage.