

# Rapport de Mini Projet

Elaboration d'une nouvelle technique de compression .IRM

Réalisé par :

BELMAHI Abderafii

HAMDANI Marwane

OUAJBIR Othman

Encadre par

ADIB Abdellah

## **Remerciement**

Nous tenons tout d'abord à remercier Dieu le tout puissant et miséricordieux, qui nous a donné la force et la patience d'accomplir ce Modeste travail.

En second lieu, nous tenons à remercier notre encadrant Monsieur Abdellah ADIB, pour son précieux conseil et son aide durant toute la période du travail, pour l'orientation, la confiance et la patience qui ont constitué un apport considérable sans lequel ce travail n'aurait pas pu être mené au bon port. Qu'il trouve dans ce travail un hommage vivant à sa haute personnalité.

Nous tenons à exprimer nos sincères remerciements à tous les professeurs qui nous ont enseigné et qui par leurs compétences nous ont soutenu dans la poursuite de nos études.

Enfin, nous tenons également à remercier toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail.

## Table of Contents

1	Etude théorique	5
1.1	Algorithmes de Compression	5
1.1.1	RLE (Run length encoding)	5
1.1.2	Algorithme de Huffman	5
1.2	Etude de formats de compression existantes	6
1.2.1	BMP (Windows bitmap)	6
1.2.2	PNG (Portable Network Graphics)	9
2	La. IRM	11
2.1	La technique utilisée	11
2.2	La structure de fichier. IRM	14
2.3	Résultats et discussions	16
3	Conclusion	18
	References	19
	Lien complémentaire	19

## Liste des figures

Figure 1: Les champs de l'entête du fichier .bmp.....	7
Figure 2: Processus d'acquisition de l'image BMP.....	9
Figure 3 : Transformation de la matrice en vecteur.....	11
Figure 4 : Les trois parcours de la vectorisation (ligne, colonne, et zigzag).....	12
Figure 5 : Application de RLE sur le vecteur de l'image.....	12
Figure 6 : Application de Rle après juxtaposition des vecteurs .....	13
Figure 7 : Transformation de l'image RBG avant Rle.....	14
Figure 8 : Rle ajusté pour l'image RGB.....	14
Figure 9 : Objet contenant les informations de l'entête .....	14
Figure 10 : Représentation de la palette .....	15
Figure 11 : BELMAHI ABDERAFII .....	16
Figure 12 : Arbres - image binaire .....	17
Figure 13 : Les frères - Image indexée.....	17

## Liste des tableaux

Tableau 1: Structure d'un fichier BMP.....	6
Tableau 2 : Les champs de l'entête de l'image.....	8
Tableau 3 : Disposition des segments (Chunks).....	10
Tableau 4 : Champs de l'Entête du fichier .IRM .....	15
Tableau 5 : Résultats de test .....	18

## Introduction

Avec l'évolution technologique, nous sommes amenés à manipuler des données de plus en plus importantes, à les échanger et les stocker. Bien que les capacités de stockage et le débit internet augmentent eux aussi, trouver comment diminuer efficacement la taille prise par ces données reste d'un intérêt majeur.

Ce rapport présente la mise en œuvre d'un nouveau format d'image qui implémente la compression sans perte par l'algorithme RLE, cet algorithme qui convient parfaitement aux images RVB.

Nous allons donc aussi nous intéresser à deux autres formats d'images assez utilisées.

# 1 Etude théorique

La compression d'image sans perte, comme leur nom indique, différentes techniques de compression de données appliquées sur les images numériques qui n'impliquent aucune perte d'informations, si l'image a été compressé sans perte, l'image d'origine peut-être récupérer exactement à partir de fichier compressé.

Les méthodes de compression d'images sans perte sont utiles lorsqu'on veut garder une grande précision, tel que pour des balayages médicaux, ou des numérisations d'images destinées à l'archivage.

Pour cet objet, La compression d'image sans perte peut toujours être modélisée comme une procédure en deux étapes : décorrélation et codage entropique<sup>1</sup> :

- La première étape supprime la redondance spatiale ou la redondance entre pixels principalement à l'aide de l'algorithme RLE
- La deuxième étape, qui comprend le codage Huffman, le codage arithmétique et LZW, supprime la redondance de codage.

## 1.1 Algorithmes de Compression

### 1.1.1 RLE (Run length encoding)

L'algorithme RLE repose sur une idée assez simple : au lieu de répéter plusieurs fois un même symbole, on indique le nombre de fois qu'on le répète, puis ce symbole (une seule fois). Pour des données très redondantes, on peut ainsi gagner facilement de la place.

S'il y a peu de doublons à la suite, on a un problème : la taille peut être jusqu'à deux fois plus grande ! Il faut donc améliorer l'idée : un entier inférieur à 128 indique le nombre de répétition de la lettre suivante ; un entier supérieur à 128 indique le nombre de lettres (+ 128) qu'il y a à lire après, sans répétition

La compression RLE est utilisée pour les images .bmp (qu'on détaillera après). Mais elle présente, dans l'immédiat, peu d'intérêt : il n'y a jamais deux pixels consécutifs de même couleur dans une photo.

### 1.1.2 Algorithme de Huffman

L'algorithme de Huffman est plus complexe : il s'agit de travailler au niveau binaire, et d'utiliser moins de bits pour coder les symboles qui reviennent le plus souvent. On va construire un arbre binaire où les feuilles sont les symboles de la table, et où les branches les plus profondes portent les symboles les plus rares.

On construit d'abord une file de priorité où les priorités sont les fréquences d'apparition des symboles, en tant que feuilles d'un futur arbre. (Les premiers sortis de la file seront les symboles les moins fréquents).

---

<sup>1</sup> Source: <https://ieeexplore.ieee.org/abstract/document/1594297>

Tant qu'il y a plus de deux éléments dans la file : on sort les deux premiers éléments de la file g et d (c'est-à-dire les moins fréquents), on construit un Nœud (g, d), qu'on rajoute dans la file avec comme priorité la somme des priorités de g et d.

On a alors un arbre, dont on indice les branches de la sorte : 0 pour une branche gauche, 1 pour une branche droite.

Enfin, on remplace les symboles du message à coder par la suite de bits nécessaires pour les atteindre dans l'arbre.

Pour écrire ce code dans un fichier, il suffit d'écrire la lecture préfixe de l'arbre (avec un « identifiant » pour distinguer Feuille et Nœud), puis la suite de bits (complétés du nombre de zéros nécessaires pour avoir un nombre entiers d'octets, en indiquant ce nombre avant).

La décompression s'effectue en reconstituant l'arbre, puis la suite de bits associée à la suite d'entiers, puis en lisant l'arbre.

## 1.2 Etude de formats de compression existantes

Dans le cadre de cette étude, notre group a été sélectionner pour étudier deux formats d'image qui utilisent une compression sans perte, le format BMP et le PNG, afin de comparer les résultats du nouveau format qu'on va proposer avec ce qui existe dans le marché.

### 1.2.1 BMP (Windows bitmap)

Le premier format qu'on va découvrir est un des plus simples, BMP est l'abréviation de bitmap image aussi nome « Device Independent Bitmap (**DIB**) » été conçue par Microsoft Windows pour faciliter l'échange des images matricielles indépendamment du matériel d'affichage.

Le format BMP support les images avec 1, 2, 4, 8, 16, 24, et 32. En plus Il support la compression par l'algorithme RLE pour les images avec 4 et 8 bit par pixel.

Un fichier BMP contient des différentes sections, chaque section contient des informations sur le fichier, sur l'image, la palette, et la matrice des pixels.

Section	Description	Taille	Nb de champs
<b>En-tête de fichier</b>	informations sur le fichier image	14	5
<b>En-tête d'image</b>	Informations sur l'image	40	11
<b>Palette de couleurs</b>	Info sur les couleurs utilisés	~	~
<b>données de pixel</b>	Les valeurs de chaque pixel	~	~

Tableau 1: Structure d'un fichier BMP

Comprenons le rôle de chacun de ces blocs.

### En-tête de fichier :

Chaque fichier BMP commence avec ce bloc comme c'est montré dans le Tableau 1 qui joue le rôle de la signature de ce fichier, ce bloc fournit des informations sur le type de fichier, sa taille, aussi précise ou commence les données de l'image enveloppé dans ce fichier.

Le champs « *File Type* » code sur 2 octets, sont deux caractère qui indiquent qu'il s'agit d'un fichier BMP, ils doivent être 'BM' ou « 0\*42 0\*4D » en hexadécimal. Suivi par « *File Size* », un entier naturel code sur 64 bit donne la taille du fichier en octets. Deux champs réservées de 2 octets pour chacun. En fin « *Pixel Data Offset* », un entier de 32 bit représentant le décalage des données de pixels en octets.

Field Name	Size in Bytes	Description
bfType	2	Contains the characters "BM" that identify the file type
bfSize	4	File size
bfReserved1	2	Unused
bfReserved2	2	Unused
bfOffBits	4	Offset to start of pixel data

Figure 1: Les champs de l'entête du fichier .bmp

### En-tête d'image :

Ce bloc fournit des informations sur l'image et les couleurs, un Total de 11 champs et 40 octets de large, sont mentionnés dans le Tableau 2

Noms de champs	Taille	Description
HeaderSize	4	Taille de l'en-tête: doit être d'au moins 40
ImageWidth	4	Largeur de l'image
ImageHeight	4	Longueur de l'image
Planes	2	Doit être 1
BitsPerPixel	2	Bits par pixel: 1, 4, 8, 16, 24 ou 32



<b>Compression</b>	4	Type de compression, RVB = 0, RLE8 = 1, RLE4 = 2 ou BITFIELDS = 3
<b>ImageSize</b>	4	La taille totale de l'image
<b>XpixelsPerMeter</b>	4	La résolution horizontale
<b>YpixelsPerMeter</b>	4	La résolution verticale
<b>TotalColors</b>	4	Le nombre de couleurs de la palette
<b>ImportantColors</b>	4	Le nombre de couleurs importantes de la palette

Tableau 2 : Les champs de l'entête de l'image

### Palette de couleurs :

Cette section contient la liste des couleurs utilisées dans le pixel de l'image, les images BMP à profondeur de couleur de 1, 4 et 8 bits sont des images indexes, et la valeur du pixel contient un index qui fait référence à une couleur de la palette. Lorsque  $BPP^2 > 8$ , nous devons ignorer l'ajout de palette de couleurs à l'image BMP.

### Données de pixel (le Corps de l'image) :

Les valeurs de chaque pixel sont présentées à partir le « *Pixel Data Offset* » nième octet, les pixels sont classés de bas vers le haut (comme le montre le Figure 2) :

- Les images en 2 couleurs utilisent 1 bit par pixel, ce qui signifie qu'un octet permet de coder 8 pixels
- Les images en 16 couleurs utilisent 4 bits par pixel, ce qui signifie qu'un octet permet de coder 2 pixels
- Les images en 256 couleurs utilisent 8 bits par pixel, ce qui signifie qu'un octet code chaque pixel
- Les images en couleurs réelles utilisent 24 bits par pixel, ce qui signifie qu'il faut 3 octets pour coder chaque pixel, en prenant soin de respecter l'ordre de l'alternance bleu, vert et rouge.

---

<sup>2</sup> BPP: Bits Per Pixel

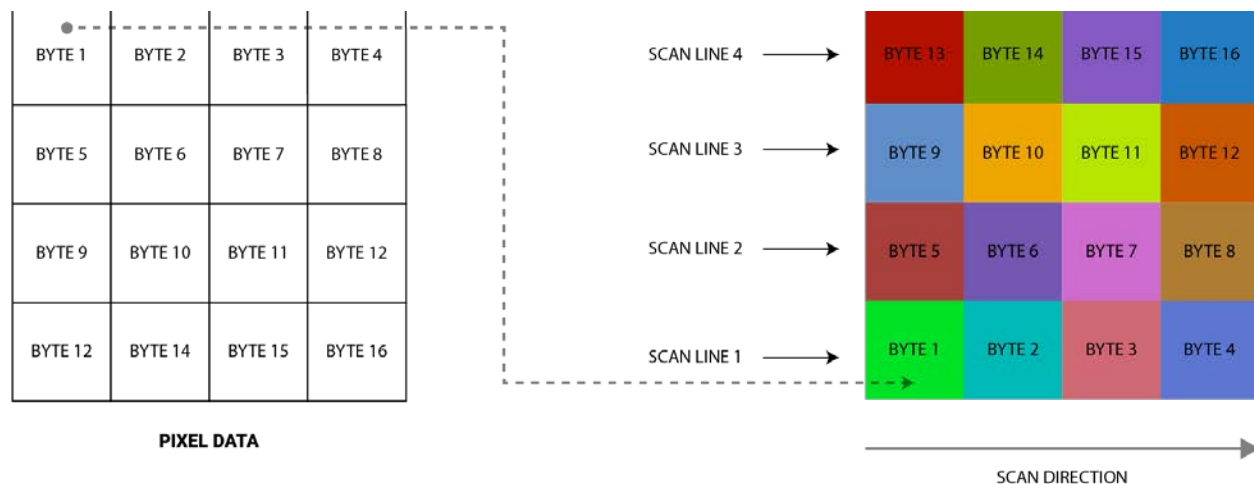


Figure 2: Processus d'acquisition de l'image BMP

### 1.2.2 PNG (Portable Network Graphics)

Le format PNG (Portable Network Graphics) fournit une norme portable, légalement libre, bien compressée et bien spécifiée pour les fichiers d'image bitmap sans perte de qualité. On dit que c'est un format d'image non destructeur.

Il a été mis au point en 1995 afin de fournir une alternative libre au format GIF, format propriétaire dont les droits sont détenus par la société Unisys (propriétaire de l'algorithme de compression LZW), ce qui oblige chaque éditeur de logiciel manipulant ce type de format à leur verser des royalties. Ainsi est également un acronyme récursif pour PNG's not GIF

#### Caractéristiques

Le format PNG permet de stocker des images en noir et blanc (jusqu'à 16 bits par pixels de profondeur de codage), en couleurs réelles (jusqu'à 48 bits par pixels de profondeur de codage) ainsi que des images indexées faisant usage d'une palette de 156 couleurs.

Le format PNG 8 bits offre la possibilité d'enregistrer une image avec une palette de couleurs de 2 à 256 couleurs (comme le GIF).

Le format PNG 24 bits permet quant à lui l'enregistrement de l'image sous une palette de plus de 16 millions de couleurs, ce que l'on a appelé dans le langage les couleurs vraies. On peut aujourd'hui enregistrer ces images sous des palettes encore plus élevées, mais pour Internet elles ces palettes de couleurs ne sont pas utilisées.

La compression proposée par ce format est une compression sans perte (lossless compression) 5 à 25% meilleure que la compression GIF.

Enfin PNG embarque des informations sur le gamma de l'image, ce qui rend possible une correction gamma et permet une indépendance vis-à-vis des périphériques d'affichage. Des mécanismes de correction d'erreurs sont également embarquées dans le fichier afin de garantir son intégrité.

## L'Entête de fichier PNG :

Un fichier PNG est constitué d'une signature, permettant de signaler qu'il s'agit d'un fichier PNG, puis d'une série d'éléments appelés segments (chunks) est la suivante :

137 80 78 71 13 10 26 10

Chaque segment (chunk) est composé de 4 parties :

Partie	Taille	Description
<b>La taille</b>	4	Décrivant la taille du segment
<b>Le type de segment (chunk type)</b>	4	Composés de caractères ASCII alphanumériques (A-Z, a-z, 65 à 90 et 97 à 122) permettant de qualifier la nature du segment
<b>Les données du segment (chunk data)</b>	~	~
<b>Le CRC (cyclic redundancy check)</b>	4	Un code correcteur permettant de vérifier l'intégrité du segment

Tableau 3 : Disposition des segments (Chunks)

Les segments peuvent être présents dans n'importe quel ordre si ce n'est qu'ils doivent commencer par le segment d'en-tête (IHDR chunk) et finir par le segment de fin (IEND chunk). Les principaux segments (appelés critical chunks) sont :

- IHDR Image header
- PLTE Palette
- IDAT Image data
- IEND Image trailer

## 2 La. IRM

Afin de contribuer dans le monde du « Data Compression » et l'enrichir, l'idée est de proposer un nouveau format d'image qui utilisent le fameux algorithme de compression RLE (voir section 1.1.1), pour compresser tous les types d'images existants <sup>3</sup> sans aucune perte d'information.

Donc sans plus tarder, découvrons notre nouveau-né le format «. IRM »

### 2.1 La technique utilisée

Comme déjà dit, notre technique consiste à compresser la matrice de l'image avec RLE, d'abord il faut transformer la matrice des pixels en vecteur, pour appliquer l'algorithme de compression, puisque plus d'occurrence successive, le taux de compression sera élevé, on doit choisir le parcours qui va nous donner le mieux taux de compression.

$$\text{Taux de compression} = \frac{\text{la taille initiale} - \text{la taille finale}}{\text{la taille initiale}}$$

$$\begin{bmatrix} 25 & 23 & 250 & 250 & 255 & \dots & 150 \\ 25 & 23 & 111 & 200 & 180 & \dots & 63 \\ & & \vdots & & & \ddots & \vdots \\ 40 & 50 & 50 & 70 & 156 & \dots & 30 \end{bmatrix} \rightarrow [25 \quad 23 \quad 250 \quad \dots \quad 30]$$

Figure 3 : Transformation de la matrice en vecteur

Après avoir su le type de l'image entre avec ce simple test :

```
1    [l,c,p] = size(Im);
2    [lm,~] = size(map);
3    if (p>1)
4        type = 'Vrais couleurs';
5    else
6        if(lm>2)
7            type = 'Couleurs indexes';
8        else
9            if(max(A)<2)
10               type = 'Binaire';
11            else
12               type = 'Niveaux de Gris';
13            end
14        end
15    end
```

---

<sup>3</sup> - Vrais couleurs – Couleurs indexées – Niveaux de Gris - Binaire

On construit trois vecteurs selon trois types de parcours (figure 4), l'image en vrais couleurs est traitée différemment car il contient trois plans (RGB).

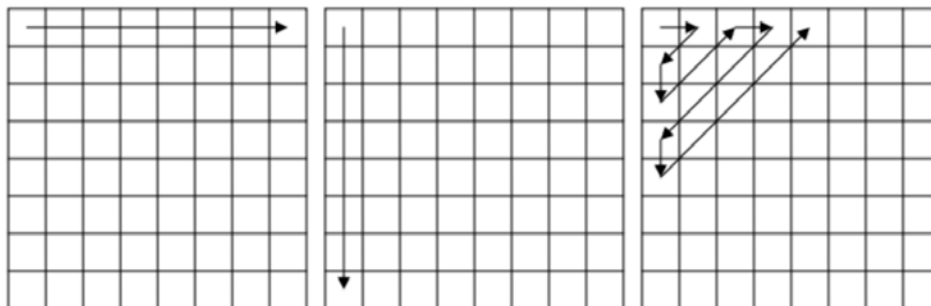


Figure 4 : Les trois parcours de la vectorisation (ligne, colonne, et zigzag)

```

1  imVectorH = reshape(image.',1,[]); // parcours ligne
2  imVectorV = reshape(image,1,[]);   // parcours colonne
3  imVectorZ = zigzag(image);          // parcours zigzag

```

La fonction « zigzag » prend en paramètres une matrice 2D et retourne un vecteur ligne contenant la valeur de chaque pixel. Après la phase de vectorisation on procède à la phase de compression avec RLE. Avec la fonction « rle » on élimine les répétitions et créons un nouveau vecteur qui contient des couples { *valeur* | *occurrence* }.

```

1  function [vectRle] = rle(im)
2      vectRle = struct('valeur',{},'occurrence',{});
3      ...
17 end

```

$$[5 \ 5 \ 2 \ 150 \ 150 \ \dots \ 68] \rightarrow \begin{bmatrix} \{ 5 | 2 \} \\ \{ 2 | 1 \} \\ \{ 150 | 12 \} \\ \vdots \\ \{ 68 | 20 \} \end{bmatrix}$$

Figure 5 : Application de RLE sur le vecteur de l'image

Et là on arrive à l'étape où le programme choisit quel parcours qui a donné le meilleur taux de compression, ce dernier est calculé par la formule donnée précédemment, donc on doit calculer les tailles initiales et finales pour chaque parcours, en fonction de nombre de pixels en largeur « L », nombre de pixels en longueur « l », nombre de bits nécessaires pour coder la valeur du pixel « BPP », nombre de bits nécessaires pour coder l'occurrence « BPO », et la taille du vecteur d'occurrence « rleLength » :

- Pour l'image binaire :

$$taille\ initiale = L * l$$

$$taille\ finale = rleLength * BPO$$

- Pour l'image niveau de gris :

$$taille\ initiale = L * l * 8$$

$$taille\ finale = rleLength * 8 + rleLength * BPO$$

- Pour l'image indexée :

$$taille\ initiale = L * l * BPP$$

$$taille\ finale = rleLength * (BPP + BPO)$$

On calcule la valeur de BitsParOccurrence « BPO » par la formule :

$$BPO = E(\log_2(\max(vectRle.occurrence))) + 1$$

Et enfin le taux de compression :

$$Taux\ de\ compression = \frac{la\ taille\ initiale - la\ taille\ finale}{la\ taille\ initiale}$$

### Les images vrais couleurs RGB

On a dit que ce type nécessite un traitement un peu spécial, ici chaque pixel est représenté par trois valeurs, chaque une stocke dans un plan, quelle est l'approche qui va nous donner les meilleurs résultats ?

Initialement on a testé de juxtaposer les vecteurs de chaque plan dans un seul vecteur et le compresser par Rle, mais cette méthode nous menés à une extension de la taille du fichier, et après tester ça sur plusieurs images (toujours une extension), on est sorti avec une explication qui paraît logique :

$$\begin{array}{cc} 25 & 25 \\ 144 & 144 \\ 88 & 88 \end{array} \rightarrow \left[ \begin{array}{l} \{25|2\} \\ \{144|2\} \\ \{88|2\} \end{array} \right]$$

Figure 6 : Application de Rle après juxtaposition des vecteurs

Imagine un pixel avec ses trois valeurs répétées 2 fois, pour stocker ça on a besoin de  $3*2*8 = 48$  bits, après la compression on se trouve avec  $3*(8+8) = 48$  bits, c'est comme si on a rien fait. Donc l'idée était de factoriser par l'occurrence qui va nous donner  $3*8+8 = 32$  bits.

Pour cela, on doit ajuster Rle pour qu'il compare trois valeurs à la fois. Au lieu de juxtaposer les vecteurs, la fonction « RleRGB » attend en paramètre une matrice de  $(3, L*1)$ , chaque ligne correspond à un plan.

$$\begin{bmatrix} 23 & 23 \\ 14 & 56 \\ 30 & 30 \\ 40 & 40 \\ 16 & 16 \\ 20 & 5 \end{bmatrix} \rightarrow \begin{bmatrix} 23 & 23 & 14 & 56 \\ 30 & 30 & 40 & 40 \\ 16 & 16 & 20 & 5 \end{bmatrix}$$

Figure 7 : Transformation de l'image RBG avant Rle

Après le « RleRGB » (Figure 8), l'étape de choix de parcours où on doit calculer les taux de compression. La taille finale après la compression est  $RleLength * (24 + BPO)$ .

$$\begin{bmatrix} 23 & 23 & 14 & 56 \\ 30 & 30 & 40 & 40 \\ 16 & 16 & 20 & 5 \end{bmatrix} \rightarrow \begin{bmatrix} \{ [23 & 30 & 16] \mid 2 \} \\ \{ [14 & 40 & 20] \mid 1 \} \\ \{ [56 & 40 & 5] \mid 1 \} \end{bmatrix}$$

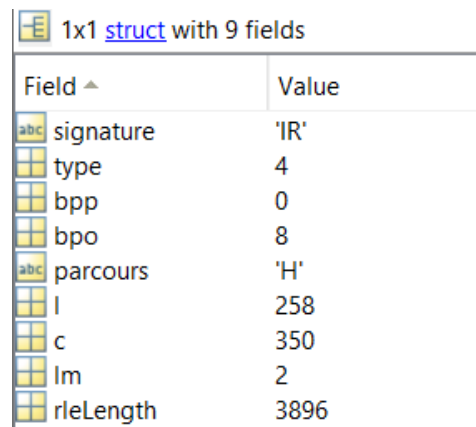
Figure 8 : Rle ajusté pour l'image RGB

## 2.2 La structure de fichier. IRM

Le format de fichier. Irm qu'on a proposé est composé de trois sections, un entête qui contient les informations relatives à l'image compressée, et qui aide à la décompression. Une section réserve au données de la palette. Et la dernière pour stocker le vecteur Rle.

### Entête. IRM

Notre entête contient 9 champs, une structure contenant ces informations doit être en mémoire avant d'écrire dans de l'image.



Field	Value
signature	'IR'
type	4
bpp	0
bpo	8
parcours	'H'
l	258
c	350
lm	2
rleLength	3896

Figure 9 : Objet contenant les informations de l'entête

Champs	Taille	Description
<b>Signature</b>	2	Deux caractères en Ascii « IR » indiquant qu'il s'agit de .IRM.
<b>Le type d'image</b>	1	Un entier de 8 bit. $\left\{ \begin{array}{l} = 1, \forall \text{ image vrais couleurs} \\ = 2, \forall \text{ image indexe} \\ = 3, \forall \text{ image greyscale} \\ = 4, \forall \text{ image binaire} \end{array} \right.$
<b>BitPerPixel</b>	1	Nombre de bits pour chaque pixel.
<b>BitPerOccur</b>	1	Nombre de bits pour l'occurrence de chaque pixel.
<b>Parcours</b>	1	Un caractère qui indique le parcours utilise pour vectorisaer la matrice d'image.
<b>Longueur</b>	2	~
<b>Largeur</b>	2	~
<b>Longueur de la palette</b>	2	Nombre de couleurs dans la palette.
<b>RleLength</b>	4	~

Tableau 4 : Champs de l'Entête du fichier .IRM

### Palette de couleurs

Cette section est utilisée par les images indexées pour stocker les couleurs, chaque couleur est présentée par 3 octets consécutifs, le rouge 1 octet, le vert 1 octet, et le bleu 1 octet. Suivi par la couleur suivante (Figure 10).

$$\begin{bmatrix} 23 & 20 & 75 \\ 100 & 40 & 30 \\ \vdots & & \\ 200 & 150 & 255 \end{bmatrix}$$

Figure 10 : Représentation de la palette



## Les Données de l'image

Ici on trouve les pixels et ses occurrences.

- Pour l'image RGB

Rouge	Vert	Bleu	occurrence
-------	------	------	------------

- Pour l'image indexée

Index	Occurrence
-------	------------

- Pour l'image niveau de gris

Niveau de gris	Occurrence
----------------	------------

- Pour l'image binaire

Occurrence du 0	Occurrence du 1	Occurrence du 0	...
-----------------	-----------------	-----------------	-----

## 2.3 Résultats et discussions

Cette technique de compression s'avère très efficace pour les images avec une grande disposition de pixels similaire. On a testé ça sur plusieurs images pour chaque type, les résultats étaient satisfaisants avec un moyen taux de compression de 30%.



Figure 11 : BELMAHI ABDERAFII

Taux de compression : 33.9%



Figure 12 : Arbres - image binaire

Taux de compression : 61.1%

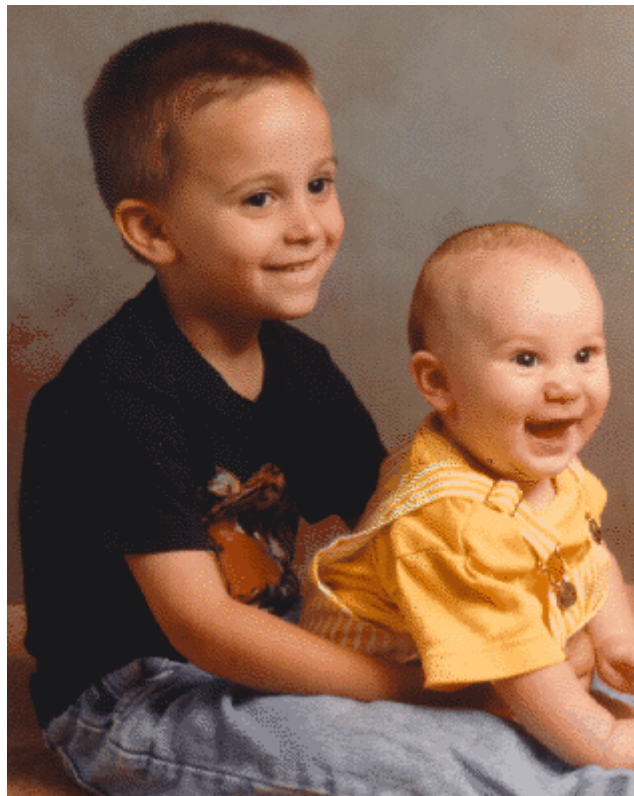


Figure 13 : Les frères - Image indexée

Taux de compression : 9.4%

Voici un tableau qui englobe et détaille les résultats des tests effectués :

Image	Type	Dimension	Taille d'image	Taille d'image compressée	Taux de compression	Parcours	Taille de fichier .IRM
Belmahi	Rgb	300*300	270000	178332	33.9%	Vertical	174 ko
Arbres	Bin	258*350	11288	4383	61.1%	Horizontal	4.30 ko
Les frères	Indx	400*318	127200	115200	9.4%	Horizontal	113 ko
Linux	Gr	160*160	25600	9760	61.8%	Vertical	4.35 ko

Tableau 5 : Résultats de test

### 3 Conclusion

Selon les résultats ci-dessus du test, la technique de compression qu'on a proposé fonctionne efficacement là où les grands des zones de valeur de pixel similaire ont lieu dans les données d'image, on ne peut pas cacher qu'il y a des images qu'on a testées, ont donné des mauvais résultats.

En gros, il paraît que nous avons atteint le but de ce mini-projet. On espère que notre travail et ce rapport reflètent bien nos ambitions, et nos rêves de faire partie de la communauté des chercheurs, et de contribuer un jour avec des grands projets dans ce domaine, partageant notre passion, et montrant au monde que les Marocains sont des vrais génies.

Mais sans oublier que « le seul endroit où le succès vient avant le travail, c'est le dictionnaire ! ».

## References

- [1] Varsha Bansal, “THE IMPLEMENTATION OF RUN LENGTH ENCODING FOR RGB IMAGE COMPRESSION”, International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 3 Issue 12, December 2014. <http://ijarcet.org/wp-content/uploads/IJARCET-VOL-3-ISSUE-12-4397-4401.pdf>
- [2] John Miano, “COMPRESSED IMAGE FILE FORMATS JPEG, PNG, GIF, XBM, BMP”, Copyright © 1999 by the ACM Press, a division of the Association for Computing Machinery, Inc. (ACM). <https://bit.ly/34vWrIO>
- [3] Robin Champenois, “TIPE - COMPRESSION D’IMAGES”, Juin 2012. <https://www.robin-champenois.fr/prepa/tipe.pdf>
- [4] Uday Hiwarale, “Bits to Bitmaps: A simple walkthrough of BMP Image Format”, ItNext 19 Oct 2019. <https://itnext.io/bits-to-bitmaps-a-simple-walkthrough-of-bmp-image-format-765dc6857393>
- [5] Glenn Randers-Pehrson, “PNG (Portable Network Graphics) Specification, Version 1.2”, Copyright © 1996 by: Massachusetts Institute of Technology (MIT). <http://www.libpng.org/pub/png/spec/1.2/>

## Lien complémentaire

Répertoire git, contenant les fonctions et les scripts développé avec Matlab, et l’interface utilisateur qu’on a utilisé pour faire les tests :

- <https://github.com/TheHonorRise/Image-compression-using-Rle> .

Contient aussi les images de test et ses fichiers «. IRM » compressés.

- <https://github.com/TheHonorRise/Image-compression-using-Rle/tree/master/Test%20images> .