EXTENSION IRM

pour la compression d'images sans perte

1. TRAITEMENT ET COMPRESSION D'IMAGES
2. ETUDE THÉORIQUE: PNG & BMP
3. ÉTUDE PRATIQUE: PROJET RÉALISÉ

RÉALISÉ PAR:

JALLOULI CHAIMAE - BARRADE FATIHA
MIFTAH IDRISSI MOUAD - EL BZIOUI OUSAMA

FACULTÉ DES SCIENCES ET DES TECHNIQUES MOHAMMEDIA
LICENCE SCIENTIFIQUE ET TECHNIQUE EN INFORMATIQUE, RÉSEAUX & MULTIMÉDIA
TRANSMISSION DES DONNÉES MULTIMÉDIA ENCADRÉ PAR MR: ABDELAH ADIB

Table des matières

TABLE DES MATIERES 2

TABLE DES FIGURES 4

TABLE DES TABLEAUX 4

1. NOTIONS GENERALES SUR LA COMPRESSION DE DONNEES 7

1.1 Introduction 8

1.2 Traitement de l'image numérique 8
1.2.1 Notions sur l'image numérique 8
1.2.2 Différents formats de l'image 9

1.3 Compression des données 9
1.3.1 Introduction 9
1.3.2 Types de compression des données 10
1.3.2.1 Compression sans perte 11
1.3.2.2 Compression avec perte 11

1.4 Techniques de compression d'images 12
1.4.1 Redondances et compression des images 12
1.4.2 Modèles de compression des images 13
1.4.3 Avantages et inconvénients de la compression des images 14

2. ALGORITHMES DE COMPRESSIONS & CRITERES DE PERFORMANCE 16

2.1 Introduction 17

2.2 Algorithmes de compression sans perte 17
2.2.1 Codage arithmétique 17
2.2.2 Code de Huffman 21
2.2.3 Run-Length Encoding 23
2.2.4 Codage de Lempel_Ziv_78 24
2.2.5 Codage de Lempel_Ziv_Welch 26

2.3 Algorithmes de compression avec perte 27
2.3.1 Transformations d'espaces de couleurs 28
2.3.1.1 Les espaces de couleurs 28
2.3.1.2 L'espace de couleurs HSL 28
2.3.1.3 Sous-échantillonnage 30
2.3.2 Transformations discrètes 32
2.3.2.1 Transformée de Fourier discrète 32
2.3.2.2 Transformée en ondelettes discrète (DWT) 32
2.3.2.3 Transformée en cosinus discrète (DCT) 33
2.3.3 Quantification 34

2.4 Critères de performance 34
2.4.1 Taux de compression 34

2.4.2 Erreur quadratique moyenne (MSE) ... 35
2.4.3 Rapport signal/bruit (PSNR) ... 35

3. PARTIE RECHERCHE : LES FORMATS BMP & PNG ... 37

3.1 Les formats des fichiers image ... 38
3.1.1 L'entête du fichier image ... 38
3.1.2 Le corps du fichier image ... 39
3.1.3 Le pied du fichier image ... 39

3.2 Le format BMP ... 39
3.2.1 L'entête du fichier BMP ... 40
3.2.1.1 BMP header ... 40
3.2.1.2 BMP InfoHeader ... 41
3.2.2 La table des couleurs BMP ... 41
3.2.3 Pixel Data de l'image BMP ... 42
3.2.3.1 BMP Scanning ... 42
3.2.3.2 BMP Scan Lines Padding ... 43

3.3 Le format PNG ... 43
3.3.1 Signature du fichier png ... 44
3.3.2 Chunks du fichier PNG ... 44
3.3.2.1 IHDR chunk : Image header ... 45
3.3.2.2 PLTE Chunk : Image palette ... 45
3.3.2.3 IDAT chunk : Image Data ... 46
3.3.2.4 IEND Chunk : Image trailer ... 47
3.3.2.5 Ancillary chunks ... 47
3.3.3 Deflate / Inflate compression ... 48
3.3.4 Algorithmes de filtrage ... 48

4. PARTIE REALISATION : EXTENSION « IRM » POUR LA COMPRESSION D'IMAGES ... 50

4.1 Extension IRM pour la compression sans perte ... 51

4.2 Entête IRM ... 51
4.2.1 Encodeur sans perte ... 52
4.2.2 Décodeur sans perte ... 54

4.3 Extension IRM pour la compression avec perte ... 54
4.3.1 Transformation de l'espace de couleur: ... 54
4.3.2 Quantification: ... 55
4.3.3 Codage par Huffman ... 55

4.4 Écriture dans un fichier binaire ... 56

4.5 Interface graphique ... 56

4.6 Mesures de performance ... 58

4.7 Résultats et conclusions générales ... 60

Liste des figures

Figure 1: Compression et reconstruction 10
Figure 2: modèle général de système de compression 14
Figure 3: Différentes variantes de RLE 24
Figure 4: Etapes de compression avec perte 28
Figure 5: Hue, saturation, lightness 29
Figure 6: Les différentes structures du sous-échantillonnage 32
Figure 7: transformée en ondelettes discrète 33
Figure 8: Processus du scanning BMP 43
Figure 9: Interface graphique (1) 57
Figure 10: Interface graphique - sans perte 57
Figure 11: Interface graphique - avec perte 58
Figure 12: interface graphique - décompression 58

Liste des tableaux

Tableau 1: Différents formats de l'image 9
Tableau 2: les composantes d'un fichier BMP 40
Tableau 3: composantes du BMP header 40
Tableau 4: Elements du BMP InfoHeader 41
Tableau 5: BMP Color table 42
Tableau 6: PNG Chunk Layout 44
Tableau 7: PNG IHDR Chunk 45
Tableau 8: Filtrage PNG 48
Tableau 9: IRM FILE HEADER 51
Tableau 10 : IRM IMAGE HEADER 52
Tableau 11: IRM IMAGE DATA 52
Tableau 12: mesures de performances 59

Liste des abréviations

BPP : Bit Per Pixel

DCT : Discrete Cosine Transform

GIF : Graphic Interchange Format

JPEG : Joint Photographic Experts Group

PNG : Portable Network Graphic

LZ : Lempel Ziv

LZW : Lempel Ziv Welch

MSE : Mean Square Error

PSNR : Peak Signal to Noise Ratio

RLE : Run-Length Encoding

RGB : Red Green Blue

HSL : Hue Saturation Lightness

TIFF : Tagged Image File Format

Résumé

Ce rapport est rédigé dans le cadre du mini-projet effectué pour le module "Transmission des données multimédia", filière d'informatique, réseau et multimédia au sein de la Faculté des Sciences et Techniques de Mohammedia. Le but de ce mini-projet est de réaliser une extension « irm » de compression sans perte des images, en effectuant une étude théorique et s'inspirant des extensions déjà présentes sur ce domaine, BMP et PNG dans le cas de notre projet.

Abstract

The goal of this mini-project is to create an "irm" extension for lossless image compression, by carrying out a theoretical study and drawing inspiration from the extensions already present in this field, BMP and PNG in the case of our project.

1. Notions générales sur la compression des données

1.1 Introduction

Cette première partie représente une introduction générale du rapport. Dans cette partie, on va parler de l'image numérique, ses caractéristiques, ses types, et les opérations qu'elle subit au cours de son traitement. Dans un deuxième temps, on va aborder la compression des images, son rôle, en expliquant les différents types d'algorithmes utilisés.

1.2 Traitement de l'image numérique

1.2.1 Notions sur l'image numérique

Une image numérique est toute image acquise, créée, traitée et stockée sous forme binaire. Une image numérique est composée d'un ensemble de points appelés Pixels (Picture Elements), chacun avec une représentation numérique de son intensité ou son niveau de gris. C'est une représentation numérique d'un objet physique qu'on souhaite traiter par un ordinateur.

Souvent, la numérisation donne à l'image une représentation matricielle, qui est caractérisée par deux paramètres :

La résolution : La finesse de représentation de l'image exprimée en point par pouce (dpi=dot per inch).

La dynamique : La plage de couleurs disponible pour coder l'image.

Pour une image binaire, chaque pixel est codé sur un seul bit. On aura donc $2^1 = 2$ couleurs possibles (0 pour le noir, 1 pour le blanc).

Pour une image en niveaux de gris ou une image à palette (fausses couleurs), chaque pixel est codé sur 8 bits. On aura donc $2^8 = 256$ couleurs possibles.

Enfin, pour une image en « vraies couleurs » ou images RGB (Red-Green-Bleu), on utilise un octet pour chacun des plans de

l'espace de représentation des couleurs, on aura donc 16 millions couleurs possibles.

1.2.2 Différents formats de l'image

Tableau 1: Différents formats de l'image

Nom du format	***Description***	***Type d'image***	***Compression des données***	***Nombre de couleurs supportées***
TIFF	Images volumineuses de grandes tailles et bonne qualité.	Images matricielles	Images non compressés ou compressés avec ou sans perte	De monochrome à 16 millions couleurs.
BMP	Images volumineuses, seulement supporté par Windows.	Images matricielles	Sans compression ou compression sans perte	16 millions couleurs.
JPEG	Format d'image le plus utilisé sur internet, souvent utilisé pour les photographies.	Images matricielles	Compression avec perte	JPEG (16 millions) et JPEG 2000 (32 millions)
GIF	Très utilisé malgré ses faiblesses, supporte les animations.	Images matricielles	Compression sans perte	256 maxi (palette)
PNG	Supporte les images détourées, et donne des images de bonne qualité.	Images matricielles	Compression sans perte	Palettisé (256 couleurs ou moins) ou 16 millions

1.3 Compression des données

1.3.1 Introduction

Le terme « compression des données » fait référence au processus de la diminution de la donnée nécessaire pour représenter une quantité d'information. Une distinction claire doit se faire entre la donnée et l'information. En fait, la donnée signifie l'outil utilisé pour transmettre l'information, et cette dernière peut être représentée par différents types des données.

Prenons l'exemple de deux personnes qui veulent raconter une histoire. Ici, l'information est l'histoire transmise, alors que la donnée est la parole. Si ces deux personnes utilisent un nombre

différent de mots pour s'expliquer, deux versions de l'histoire sont créées, et une (sinon les deux), peut contenir des données supplémentaires non nécessaires, qui ne procure aucune information importante, ou déclare ce qui a déjà été dit. On dit que cette version contient une redondance des données.

La compression des données consiste à réaliser une représentation compacte de l'information, en exploitant l'aspect de redondance y présent.

On définit la redondance comme étant la partie de la donnée ou du message qu'on peut éliminer sans toucher ou perdre l'information essentielle.

Après avoir enlevé les redondances, vient la phase de numérisation. Dans cette phase, il faut encoder l'information avec une représentation binaire. A ce point, on profite du fait que l'information est représentée en utilisant un alphabet particulier, dont les lettres apparaissent aves des probabilités différentes. Il nous sera utile donc de coder les lettres les plus fréquents avec un code plus court, ce qui diminue le nombre moyen des bits utilisé pour représenter chaque lettre.

La réalisation d'un algorithme de compression consiste à comprendre les types de redondances présents dans la donnée, et l'exploiter d'une manière qui permet une représentation plus compacte de l'information. Cette information sera reconstruite au moment de la réception.

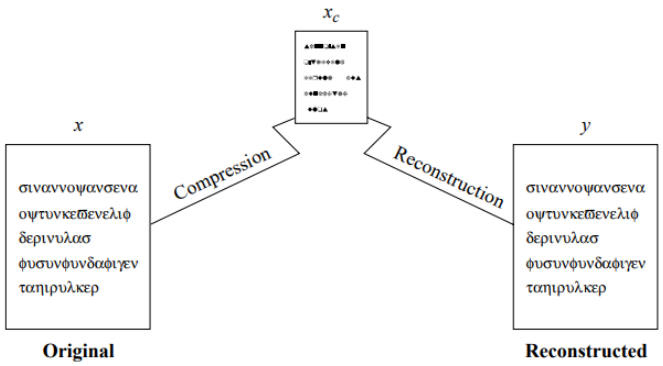


Figure 1: Compression et reconstruction

1.3.2 Types de compression des données

Plus populairement, la compression des données existe sous deux formes :

1.3.2.1 Compression sans perte

Comme son nom l'indique, préserve toute l'information, ce qui mène à une donnée identique au moment de sa reconstitution. Ce type de compression est généralement utilisé pour les applications qui ne tolèrent aucune différence entre la donnée originale et la donnée reconstituée.

La compression du texte est un exemple très pertinent de la compression sans perte, puisque les plus petites différences peuvent changer le sens du texte. Le même argument s'applique pour les fichiers exécutables.

Si la donnée en question va être « traitée » par la suite, il est déconseillé d'utiliser ce genre de compression à cause de la difficulté de remonter à l'information originale. Les images radiologiques et les images satellitaires sont deux exemples appropriés où l'utilisation de la donnée sans perte peut donner des résultats non précis.

Il existe plusieurs situations où l'on préfère utiliser une compression qui permet la reconstitution identique de la donnée, mais il existe également des situations où on peut allouer une certaine perte, dans le but d'achever une compression meilleure.

1.3.2.2 Compression avec perte

Comprend une certaine perte de donnée, qui ne peut pas être récupérée ou reconstruite. En échange d'accepter cette perte de donnée, on arrive à obtenir un ratio de compression plus augmenté que celui obtenu par la compression sans perte.

Dans plusieurs situations, cette perte de donnée n'est pas un inconvénient. Par exemple, le stockage ou la transmission du fichier son n'ont pas besoin des valeurs exactes. On peut donc tolérer des quantités différentes de perte de l'information selon

la qualité désirée du son reconstruit. Si on voudrait que la qualité du son reconstruit soit similaire à celle du téléphone, une perte considérable de la donnée peut être tolérée. Néanmoins, moins de perte sera tolérée si une qualité plus élevée est voulue.

Similairement, la différence entre la séquence vidéo reconstruite et la séquence vidéo originale est généralement sans importance aussi longtemps que cette différence n'influence pas l'information transmise.

1.4 Techniques de compression d'images

Les données multimédias sont de grande taille par rapport aux données en texte brut. Et donc pour les transmettre sur un canal de communication à faible bande passante, il doit être compressé. Dans ce qui vient, on va aborder les différentes techniques de la compression des images, ainsi que les métriques utilisées pour évaluer leur qualité.

Les objectifs de la compression des images se résument en trois axes principaux :

- Minimiser l'espace de stockage
- Réduire la largeur de bande
- Réduire le temps de transfert.

1.4.1 Redondances et compression des images

Afin de transmettre l'image d'une manière efficace, il faut d'abord la compresser en utilisant des algorithmes de compression, et la décompresser à l'arrivée pour remonter à l'image originale. Comme tous les autres types des données, la compression des images se base sur l'élimination des redondances. Dans la compression d'images numériques, trois redondances de données de base peuvent être identifiées et exploitées :

Redondance inter-pixel : Due à la corrélation entre les pixels voisins dans une image. Cela signifie que les pixels voisins ne

sont pas statistiquement indépendants. La valeur d'un pixel donné peut être déterminée à partir de la valeur de ses voisins, c'est-à-dire qu'ils sont fortement corrélés. Pour réduire la redondance inter-pixel, la différence entre pixels adjacents peut être utilisée pour représenter une image.

Redondance de codage : Associée à la représentation de l'information. Si les valeurs de pixels d'une image sont codées d'une manière qui utilise plus de symboles de code qu'absolument nécessaire, on dit que l'image résultante contient une redondance de codage.

Redondance psycho-visuelle : Existent parce que la perception humaine n'implique pas d'analyse quantitative de chaque pixel ou valeur de luminance de l'image. Son élimination n'est possible que parce que l'information elle-même n'est pas essentielle pour le traitement visuel normal.

1.4.2 Modèles de compression des images

La figure 2 montre un système de compression qui consiste de deux structures distinctes : un encodeur et un décodeur. Une image d'entrée f(x,y) est fournie au décodeur, qui la transforme en un ensemble de symboles. Après la transmission dans le canal, la représentation encodée de l'information est donnée à un décodeur, qui se charge de reconstruire l'image de sortie f^(x,y).

L'encodeur est composé d'un encodeur source, qui supprime les redondances d'entrée, et d'un encodeur de canal, qui augmente l'immunité au bruit de la sortie de l'encodeur source. Comme on pouvait s'y attendre, le décodeur comprend un décodeur de canal suivi d'un décodeur de source. Si le canal entre l'encodeur et le décodeur est sans bruit (non sujet aux erreurs), l'encodeur et le décodeur de canal sont omis, et l'encodeur et le décodeur généraux deviennent respectivement l'encodeur et le décodeur source.

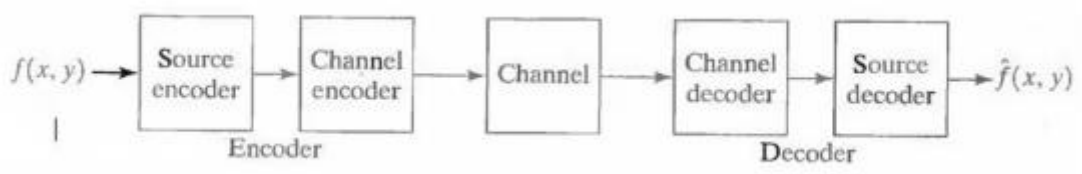


Figure 2: modèle général de système de compression

1.4.3 Avantages et inconvénients de la compression des images

La compression d'une image peut avoir plusieurs avantages :

- Réduction de la taille du fichier : Grâce à la compression, la taille du fichier image peut être réduite jusqu'à atteindre la taille désirée. Par conséquent, le fichier occupe moins d'espace de stockage sur le disque dur tout en gardant les mêmes dimensions de l'image. Cet avantage, permet aux webmasters, à titre d'exemple, de créer des sites riches en images sans perdre trop d'espace d'hébergement ni d'allocation de bande passante.
- Vitesse de chargement plus rapide : Bénéfique surtout dans le cas des appareils électroniques qui n'arrivent pas à lire rapidement les images de grande taille et, qui par la suit, fonctionnent mieux avec les images compressées. Un autre exemple, est l'exemple des services web, ou la compression des images évite la lenteur du transfert des données.
- Le coût du transfert des images larges dans des canaux à faible bande passante est réduit, alors que son efficacité augmente.
- Les images compressées ne peuvent pas être accédées par des moniteurs illégaux, et donc procurent un certain niveau de sécurité.

La compression des images peut également mener à des complexités :

- Basse qualité : La dégradation de la qualité de l'image après compression est un résultat attendu, ce qui est inconvénient si on a besoin d'afficher les images avec une résolution élevée, ou faire subir l'image aux transformations, pour l'utiliser dans le domaine scientifique, médical…
- Perte de données : Certains types de fichiers entraînent à une perte permanente des données après compression, ce qui change le but initial de la compression, qui est de réduire la taille sans toucher à l'information.

2. Algorithmes de compressions & critères de performance

2.1 Introduction

Dans la partie qui précède, on a parlé des deux types de compression : Compression sans perte qui consiste à réécrire la donnée autrement de telle manière qu'elle prend moins de taille, en se basant sur les propriétés de la source, et la compression avec perte qui fait subir l'image à différentes transformations en se basant sur les limites de la perception visuelle humaine.

Dans cette partie, on va donner des exemples précis et détaillés de chaque type de compression.

2.2 Algorithmes de compression sans perte

Dans les techniques de compression d'images sans perte, la qualité de l'image reconstruite après la décompression est idéale, presque la même qualité que l'image d'entrée. Les techniques de compression sans perte encodent l'image originale pour la compresser, sans pouvoir atteindre un taux de compression élevé puisqu'il y a aucune perte de donnée.

Les techniques de compression sans perte sans expliquées ci-dessous :

2.2.1 Codage arithmétique

Une forme de de codage entropique à longueur fixe utilisé dans la compression des données sans perte. Cette technique permet de réduire le nombre de bits sur lesquels une chaîne de caractères et codée en stockant les caractères les plus fréquents d'une chaîne avec moins de bits, et les caractères moins fréquents avec plus de bits, au lieu d'utiliser un nombre de fixe de bits comme le cas du code ASCII.

Contrairement aux autres formes de codage entropique, telles que la technique de Huffman qu'on va aborder par la suite, le

codage arithmétique code le message entier en un seul nombre flottant, une fraction de précision arbitraire q entre 0 et 1, au lieu de séparer l'entrée en symboles composants et de remplacer chacun par un code.

Dans le cas du codage arithmétique, l'information est représentée sous forme de plage définie par deux nombres.

Etapes de compression par le codage arithmétique

Pour effectuer la compression en utilisant le codage arithmétique, on crée un tableau contenant :

- Les symboles *s* que contient le message à compresser
- La probabilité *p* de chaque symbole dans le message
- L'intervalle [0 ;1[découpé en sous intervalles, proportionnel à la probabilité p de chaque symbole s. Par exemple, si le symbole a une probabilité de 50%, son intervalle sera de longueur 0.5.

Par la suite, on modifie les bornes d'un intervalle donnée, (couramment on utilise l'intervalle [0 ;1[), à chaque ajout d'un symbole en lui appliquant une suite d'opération, afin de limiter le nombre de possibilités du nombre de sortie.

Lors de l'ajout du symbole s, on applique les étapes suivantes :

- La borne supérieure de l'intervalle prend la valeur :

$$BI + (BS - BI) * (\text{BI du symbole s})$$

- La borne inférieure de l'intervalle prend la valeur :

$$BI + (BS - BI) * (\text{BS du symbole s})$$

Après avoir rempli le tableau des bornes inférieures et supérieurs en utilisant les calculs précédents, tout nombre qui appartient au dernier intervalle obtenu peut être le format compressé du mot d'entrée.

Etapes de décompression par le codage arithmétique

On peut remonter à la chaîne du départ en se basant sur le nombre trouvé lors de l'opération de la compression, en répétant les étapes suivantes jusqu'à l'obtention du mot original :

- La prochaine lettre du mot est celle dont l'intervalle contient le nombre du mot actuel.
- On modifie le nombre présent représentant le mot en utilisant la formule :

(Nombre du mot – BI du symbole) / probabilité du symbole

Ainsi, on arrive à récupérer notre mot de départ.

Faiblesses du codage arithmétique

Cette technique de compression pose deux problèmes principaux :

- L'utilisation d'un tableau de statistiques fixe, qui peut mener à une augmentation de taille pour les fichiers utilisant des statistiques inhabituels. Pour éviter ce défaut, il est conseillé d'utiliser un tableau adaptif à fréquences égales, et le mettre à jour pour adapter les intervalles à fur à mesure de la rencontre de nouveaux symboles.
- Utilisation des nombres à virgules, qui sont difficiles à manipuler, et qui peuvent entraîner des erreurs au niveau de la décompression.

Exemple de compression en utilisant le codage arithmétique :

Codant le mot « ESIPE » en utilisant le codage arithmétique.

Étape 1 : Tableau des symboles et leurs probabilités

Symbole s	***Probabilité p***	***Intervalle***
E	4/10	[0,0.4[
S	2/10	[0.4,0.6[
I	2/10	[0.6,0.8[
P	2/10	[0.8,1.0[

Etape 2 : calcul des bornes des intervalles

Symbole s	***Borne Inférieure***	***Borne supérieure***
	0.0	0.1
E	0.0	0.4
S	0.16	0.24
I	0.208	0.224
P	0.2208	0.224
E	0.2208	0.22208

Résultat : le mot « ESIPE » est compressé par tout nombre compris entre 0.2208 et 0.2208.

Exemple de décompression en utilisant le codage arithmétique :

Prenons le format compressé du mot « ESIPE », il s'agit du nombre 0.2208. Pour le décompresser, on applique les étapes déjà mentionnées, pour enfin arriver au tableau suivant :

Mot	***Symbole***	***Nouveau code***
	E	0.552
E	S	0.76
ES	I	0.8
ESI	P	0.0
ESIP	E	

Comme ça, on a pu reconstruire le mot original à partir d'un seul chiffre.

2.2.2 Code de Huffman

Une forme de codage entropique, proposée par David Huffman 1952. Similairement à la méthode du codage arithmétique, le codage de Huffman est un codage à longueur variable attribut un mot de code binaire plus grand aux symboles (caractères, pixels…) moins fréquents, et inversement pour les symboles plus fréquents, on les attributs des mots de codes plus courts.

Etapes de compression par le code de Huffman

Pendant la phase de compression en utilisant le code de Huffman, on crée un arbre d'une manière récursive, en se basant sur les fréquences des symboles. La création de l'arbre de Huffman passe par les étapes suivantes :

- Ordonner dans un tableau les symboles selon leurs probabilités
- Retirer les deux symboles de plus faible fréquence, et les rattacher à un nœud, qui prend comme valeur la somme des fréquences des deux symboles. Successivement, en considérant chaque nœud comme un nouveau symbole, jusqu'à arriver à la racine de l'arbre.
- Les branches à gauches prennent la valeur 0, alors que les branches à droite prennent la valeur 1. On trouve le code de chaque symbole par la suite des codes du chemin allant de la racine et menant vers le symbole.

Par conséquent, les caractères à fréquence minimale, aurons un code plus long puisqu'ils sont profonds dans l'arbre.

Faiblesses du code de Huffman :

Un inconvénient du codage de Huffman est la possibilité de seulement assigner des mots de code entiers. Par exemple, on code un symbole sur 3 bits alors que le contenu de son information est seulement 2.32 bits. Le seul cas où le code de Huffman peut être optimal, est le cas où toutes les probabilités

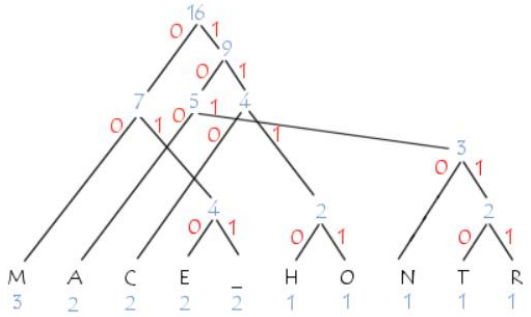
sont des puissances de 2. Cette limitation est dépassée par le codage arithmétique.

Exemple de compression en utilisant le codage de Huffman :

Pour mieux comprendre, on va appliquer les étapes précédentes sur la phrase « comment ca marche ». On commence tout d'abord par créer le tableau symboles-fréquences :

Symbole	***Fréquence***
M	3
A	2
C	2
E	2
_	2
H	1
O	1
N	1
T	1
R	1

A partir de ce tableau, on obtient l'arbre suivant :



A partir de l'arbre en trouve les codes de chaque caractère :

Symbole	***Code***
M	00
A	100
C	110
E	010

_	011
H	1110
O	1111
N	1010
T	10110
R	10111

2.2.3 Run-Length Encoding

RLE est un algorithme de compression de données sans perte, breveté par Hitachi en 1983, initialement utilisé pour transmettre les signaux de télévision analogiques. Actuellement, cette technique est utilisée par la plupart des formats bitmap (TIFF, BMP, PCX…). Même si RLE est capable de compresser n'importe quel type de données indépendamment du contenu de son information, ce contenu influence son taux de compression. RLE est une technique facile et simple à implémenter, rapide à exécuter, constituant une bonne alternative aux autres algorithmes de compression plus complexes.

RLE fonctionne en faisant réduire la taille des plages de valeurs répétitives identiques appelées « run ». Cette chaîne et typiquement codée sur 2 octets, le premier octet présente le nombre des caractères dans la plage de valeurs, nommé « run count ». Le deuxième octet contient la valeur du caractère répété, compris entre 0 et 355, et appelé « run value ».

Ce type de compression fonctionne mieux avec les images simples et les animations qui contiennent un grand nombre de pixels redondants, comme les images en noir et blanc, où on utilise un seul bit pour chaque d'image, commençant toujours par le pixel blanc. Pour les images et animations plus complexes, cette technique peut potentiellement doubler la taille du fichier au lieu de le compresser. Par conséquent, comprendre le contenu de l'information est nécessaire avant d'appliquer l'algorithme.

Variantes de Run-Length Encoding

Avec RLE, l'image est typiquement encodée sous forme d'un vecteur unidimensionnel au lieu d'une matrice à deux dimensions, du gauche au droit, en commençant par le coin gauche supérieur. RLE peut utiliser d'autre schémas en encodant la donnée par colonnes, ou par zig-zag diagonal.

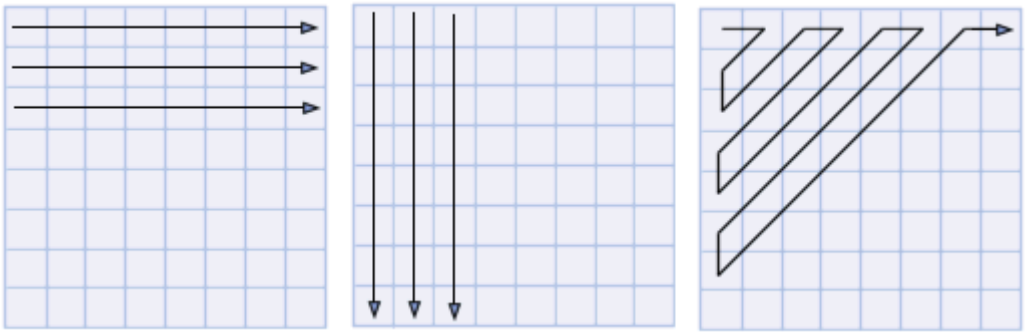


Figure 3: Différentes variantes de RLE

Faiblesses de Run-Length Encoding

En plus de la nécessité d'utiliser RLE avec une donnée simple et redondante, cet algorithme a d'autres inconvénients :

La donnée originale n'est pas instantanément accessible. Il faut décoder toute la donnée avant d'y accéder.

La taille de la donnée initiale est inconnue, on a donc toujours besoin de stocker la taille de la donnée avec la séquence compressée.

2.2.4 Codage de Lempel_Ziv_78

Un algorithme de compression sans perte, proposé par Abraham Lempel et Jacob Ziv en 1987. C'est un algorithme fameux, utilisé à la base de plusieurs logiciels, notamment le format des fichiers ZIP. Cet algorithme est un algorithme à dictionnaire dynamique, qui est généré à fur et à mesure pendant le codage et le décodage.

LZ78 fonctionne sur des sources encodées par blocs de taille constante. L'alphabet d'entrée est donc tous les symboles que

contient le bloc, alors que l'alphabet de sortie est composé des préfixes (entiers), et des symboles.

Le principe de LZ87 consiste à parcourir la source, et stocker les préfixes dans le dictionnaire. Plus une chaîne de caractère se répète, elle donnera lieu aux longs préfixes dans le dictionnaire.

Etapes de compression par LZ78

- L'algorithme commence avec un dictionnaire vide. On parcourt la source en cherchant et ajoutant au dictionnaire les nouveaux préfixes qui ne s'y trouvent pas, en leur donnant le premier indice disponible.
- Après l'ajout d'un nouveau préfixe au dictionnaire, on commence à rédiger le message de sortie en ajoutant le dernier symbole rencontré et le dernier préfixe du symbole dans le dictionnaire.

Etapes de décompression par LZ78

Pendant la décompression, l'algorithme L789 met à jour le dictionnaire à chaque fois qu'il lit un nouveau pair (préfixe, symbole).

Pour décompresser, on ajoute à la sortie le préfixe, suivi du symbole, en mettant à jour le dictionnaire avec le nouveau code trouvé : la concaténation du préfixe et symbole.

Exemple de compression en utilisant le codage LZ78

Prenons l'exemple d'une source contenant la suite des symboles « bbbabbaabbbb ».

- Dans l'exemple, on commence avec le préfixe vide de code 0. Le premier symbole « b » ne se trouve pas encore dans le dictionnaire, il est donc rajouté avec le premier indice 1. Le couple (0, b) est ajouté à la sortie.
- Pour la deuxième itération, on reprend avec un préfixe vide. On rencontre le symbole « b » qu'on a déjà ajouté au dictionnaire. On garde le symbole, et on lui rajoute le symbole suivant qui est aussi

un « b ». La suite « bb » ne se trouve pas dans le dictionnaire, elle est donc rajoutée avec le deuxième indice disponible 2. Le couple qu'on ajoute à la sortie est (1, b). Le nombre 1 constitue l'indice du préfixe de la chaîne, qui est dans ce cas la lettre « b ».

2.2.5 Codage de Lempel_Ziv_Welch

Algorithme universel de compression sans perte, crée par Abraham Lempel, Jacob Ziv, Terry Welch, et publié par ce dernier en 1984. Cet algorithme est une version améliorée de l'algorithme LZ78 publié par Lempel et Ziv en 1978.

LZW est une technique de compression qui utilise sur un dictionnaire. Si le dictionnaire est fixé, on parle du codage de dictionnaire statique. Inversement, si le dictionnaire est créé à fur et à mesure des opérations du codage et décodage, on parle du codage de dictionnaire dynamique. Cette technique est vastement utilisée grâce à sa simplicité à implémenter, sa rapidité, sa capacité à s'adapter aux différents types de données.

Il s'agit de l'algorithme utilisé dans le format d'image GIF, fichiers UNIX et autre.

Etapes de compression par LZW

Le dictionnaire de LZW est d'abord initialisé par les 256 caractères de la table ASCII et leurs valeurs correspondantes. Le principe de l'algorithme consiste à compléter le tableau afin d'avoir tous les préfixes de chaque mot, présents dans le dictionnaire.

Les étapes de l'algorithme sont comme suit :

- A l'origine, on commence avec un mot vide, auquel on ajoute une lettre du mot à coder.

- Si le mot actuel existe dans le dictionnaire, on garde le mot en lui ajoutant une autre lettre, et ainsi de suite jusqu'à obtenir un mot qui n'appartient pas au dictionnaire
- Si le mot ne figure pas dans le dictionnaire, on l'ajoute au dictionnaire en supprimant ça dernière lettre et recommencer la lecture depuis cette lettre.

Etapes de décompression par LZW

La décompression par LZW consiste à créer le même dictionnaire utiliser par la compression, à partir du fichier compressé.

- On lit chaque code du fichier compressé, et on l'ajoute au dictionnaire.
- On ajoute le mot correspondant au mot précédent auquel a été ajouté la première lettre du mot actuel.

Ainsi, on arrive à reconstruire le dictionnaire, et récupérer le mot initial.

2.3 Algorithmes de compression avec perte

Aussi appelée la compression destructive, la compression avec perte dégradent la donnée compressée en exploitant les aspects inaperçus par l'utilisateur, à cause des limites de perception chez l'être humain.

Ces algorithmes, puisqu'ils touchent au contenu de la donnée, ne peuvent pas être appliqués sur des données textuelles, ou sur les fichiers exécutables qui risquent de perdre leur lisibilité à cause de la perte permanente et irréversible de la donnée.

La compression avec perte influence légèrement la qualité du fichier, mais elle est très bénéfique en termes de vitesse de transmission et espace de stockage.

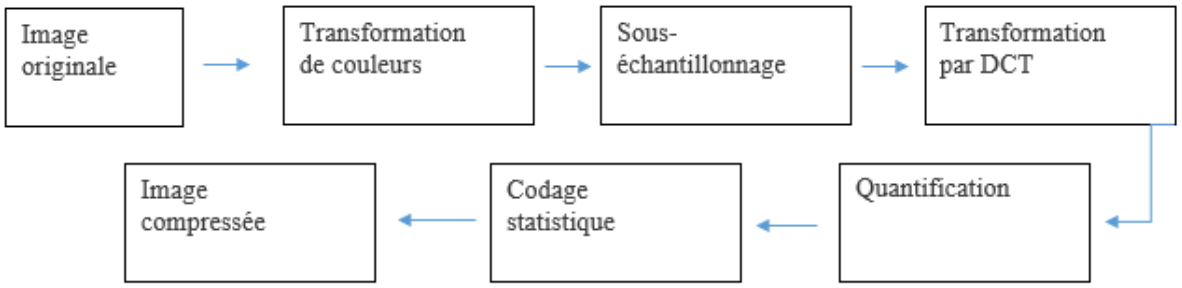


Figure 4: Etapes de compression avec perte

La compression avec perte passe des étapes de transformation, quantification et codage.

Parmi les techniques utilisées dans la compression avec perte :

2.3.1 Transformations d'espaces de couleurs

2.3.1.1 Les espaces de couleurs

Un espace de couleurs est une façon de spécifier, créer et visualiser les couleurs. Il existe plusieurs espaces de couleur qui attire notre attention en termes de la perception humain et la compression.

RGB (Red, Green, Blue) est l'espace de couleur le plus utilisé. Il dépend des moniteurs et des écrans des périphériques sur lesquels il est utilisé. Pour la compression ? YCbCr est l'espace de couleur le plus répandu. Y représente la luminance, Cb et Cr représentent la chrominance. Puisque le système humain est plus sensible aux changements dans la luminance qu'aux changements dans la chrominance, les images sont plus efficacement compressées à partir d'un sous échantillonnage avec cet espace de couleur.

2.3.1.2 L'espace de couleurs HSL

Le système visuel humain est moins sensible aux changements de teint « hue », ou saturation. Par conséquent, on peut

manipuler les valeurs de ces derniers un peu plus que la luminosité avant de remarquer une distorsion au niveau de l'image.

L'espace de couleurs HSL (Hue, Saturation, Lightness) est optimal pour cet objectif. Cet espace de couleur représente les couleurs sous forme d'un cône, avec l'axe principal représentant Ligthness, alors que les deux autres dimensions sont le teint, un angle sur la roue des couleurs, et la saturation, la distance de l'axe central. L'image semble plus colorée le plus loin qu'on se trouve de l'axe principal.

Les trois composantes du système de couleur HSL sont :

- **H – Hue (teinte) :** La forme pure de la couleur

Sa valeur varie entre 0 et 360, mais est parfois normalisée en 0-100%.

- **S – Saturation :** L'intensité de la couleur

Parfois appelée « pureté », sa valeur varie entre 0 et 100%. Plus la saturation d'une couleur est faible, plus l'image est « grisée » et fade.

- **L – Lightness (luminosité) :** La brillance de la couleur

Située entre le minimum et le maximum que peur produire un système. Sa valeur peut varier entre 0 et 1 (ou 0 et 100%).

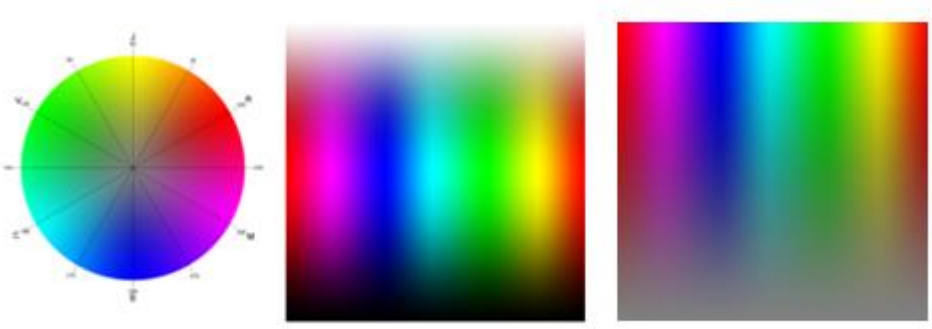


Figure 5: Hue, saturation, lightness

- Passage de RGB vers HSL :

Soit r, g, b ∈ [0,1] les cordonnées rouges, vertes et bleus d'un couleur dans l'espace RGB.

Pour trouver l'angle hue h ∈ [0,360], on calcule :

$$h = \begin{cases} 0 & if\ max = min \\ (60° * \dfrac{g-b}{max-min} + 360°)\ mod\ 360° & if\ max = r \\ 60° * \dfrac{b-r}{max-min} + 120° & if\ max = g \\ 60° * \dfrac{r-g}{max-min} + 240° & if\ max = b \end{cases}$$

On trouve la luminosité par la formule suivante :

$$l = \frac{1}{2}(max + min)$$

On trouve la saturation par la formule suivante :

$$s = \begin{cases} 0, & if\ max = min \\ \dfrac{max-min}{max+min} = \dfrac{max+min}{2l} & if\ l \leq \dfrac{1}{2} \\ \dfrac{max-min}{2-(max+min)} = \dfrac{max-min}{2-2l} & if\ l > \dfrac{1}{2} \end{cases}$$

2.3.1.3 Sous-échantillonnage

Le sous-échantillonnage est une des techniques de compression de l'image qui opère en diminuant le nombre d'échantillons à traiter afin de réduire la bande passante requise sans toucher de façon visible la qualité de l'image. Les techniques de sous-échantillonnage sont donc largement utilisées dans le traitement et transmission des données puisqu'ils conservent le temps de transmission, l'espace de stockage et la bande passante.

Exemple de structure de sous-échantillonnage de chrominance

Lorsqu'on décrit les signaux en termes de sous-échantillonnage, on les représente sous forme de 3 chiffres : 4 :4 :4, 4 :2 :2, ou 4 :4 :0.

Le premier chiffre qui vaut 4 représente le nombre d'échantillons de luminance, alors que les deux deuxièmes chiffres correspondent aux couleurs et définissent l'échantillonnage horizontal et vertical.

- ***Un signal 4 :4 :4*** présente une donnée non sous-échantillonnée et donc non compressé.
- ***Un signal 4 :2 :2*** présente la moitié de l'information de chrominance, puisqu'il a la moitié de l'échantillonnage horizontal sans toucher à l'échantillonnage vertical. Il y'a donc deux fois moins d'échantillons.
- ***Un signal 4 :2 :0*** ne porte plus qu'un quart de l'information. Ce signal échantillonne la moitié des pixels de la première ligne et ignore tous les pixels de la deuxième.

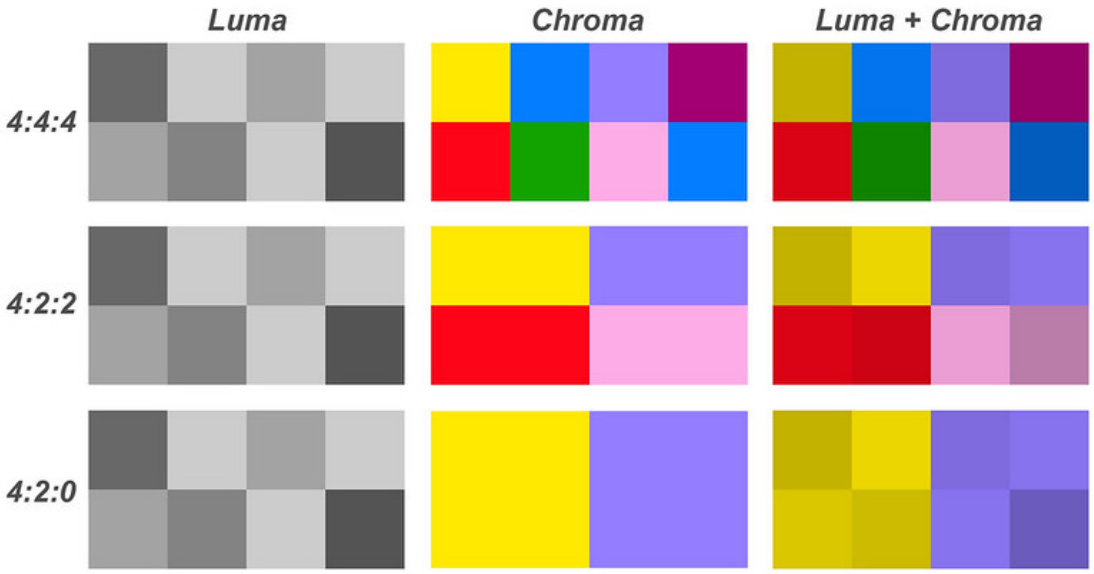


Figure 6: Les différentes structures du sous-échantillonnage

2.3.2 Transformations discrètes

2.3.2.1 Transformée de Fourier discrète

Un outil mathématique utilisé en traitement du signal, qui représente un équivalent discret de la transformée de Fourier continue utilisée dans le traitement du signal analogique.

Cette transformée permet l'évaluation du spectre échantillonnée d'un signal discret échantillonné sur une fenêtre de temps finie.

2.3.2.2 Transformée en ondelettes discrète (DWT)

Une ondelette est un signal élémentaire, à partir duquel on peut créer un signal complexe. La compression par ondelette fonctionne en décomposant l'image en un ensemble d'images de plus petite résolution.

Cette méthode de compression, qui est utilisée à la base de la compression JPEG 2000 offre une compression très importante,

une meilleure définition des détails et permet le téléchargement progressif de l'image.

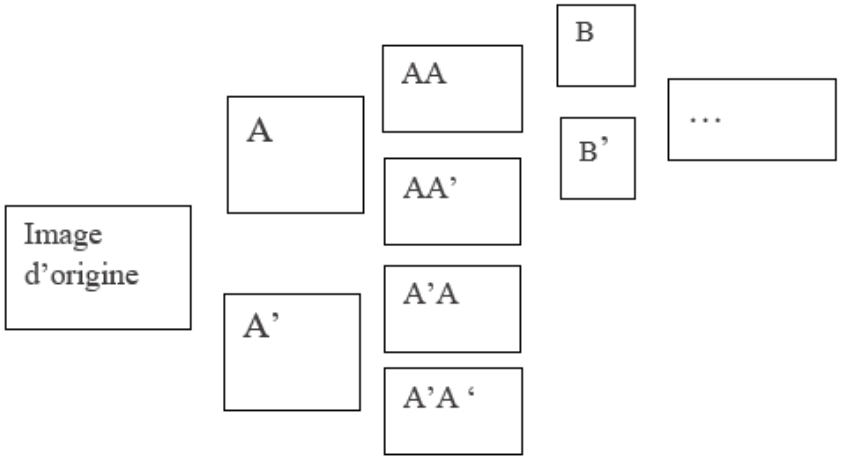


Figure 7: transformée en ondelettes discrète

2.3.2.3 Transformée en cosinus discrète (DCT)

La DCT est l'une des transformées très utilisées en traitement de l'image, spécialement en compression. Cette transformée fonctionne essentiellement en portant l'information par les coefficients basses fréquences.

Cette transformée permet un changement de domaine d'étude, tout en gardant la même donnée étudiée. Dans le cas des images, la DCT permet le passage du domaine spatial au domaine fréquentiel.

On applique la DCT sur une matrice carrée, pour obtenir une matrice carrée de la même dimension. Dans cette matrice, les valeurs en haut à gauche représentent les basses fréquences, alors que les hautes fréquences se trouvent en bas à droit.

La DCT est accompagnée par la DCTI, une méthode d'inversion, permettant le retour vers le domaine spatial.

2.3.3 Quantification

Le processus de quantification est l'étape principale qui mène à la perte de la donnée, et au niveau de laquelle le résultat de la compression et concrètement remarqué. Le taux de compression obtenu par la quantification dépend essentiellement de son facteur.

A partir de ce facteur, on crée une table de quantification de taille 8x8 à partir de la formule suivante :

$$Quantification[i][j] = 1 + (i + j + 1) * qualite;$$

Ensuite, chaque coefficient crée par une des transformée (DCT à titre d'exemple), sera quantifié comme suivant :

$$Coefficient\ [ligne][colonne = ROUND(\frac{Coefficient[ligne][colonne]}{Quantification[ligne][colonne]})$$

Après traitement, on ne garde que la composante continue (la valeur moyenne), et les basses fréquences.

2.4 Critères de performance

Après avoir effectué les différentes opérations de compressions, sans ou avec perte, il est nécessaire de mesurer la performance de l'algorithme utilisé, pour pouvoir l'améliorer. Pour cet objectif, plusieurs critères de performance de compression peuvent être utilisée.

2.4.1 Taux de compression

Le taux de compression représente le gain en termes de taille par rapport à la taille initiale. Plus le taux de compression est élevé, plus la taille du fichier compressé est faible.

Le taux de la compression et donné par la formule suivante :

$$\tau = 1 - \frac{taille\ du\ fichier\ initial}{taille\ du\ fichier\ compressé}$$

2.4.2 Erreur quadratique moyenne (MSE)

MSE (Mean Square Error) est un outil de mesure de qualité, dérivé de la distance euclidienne. Il représente, comme son nom l'indique, l'erreur quadratique cumulée entre l'image originale et l'image compressée. Plus sa valeur est faible, plus l'erreur est faible.

L'erreur quadratique moyenne est calculée à partir de la formule :

$$MSE = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M}(I(i,j) - I'(i,j))^2}{NxM}$$

Avec :

- I et I' : respectivement l'image originale et l'image compressée
- N et M : respectivement le nombre des lignes et des colonnes

2.4.3 Rapport signal/bruit (PSNR)

Le PSNR (Peak Signal Noise Ratio) est une métrique objective souvent utilisée pour mesurer la distorsion, donnée en décibel (dB). Plus la valeur du PSNR est élevée, meilleur est la qualité de l'image reconstruite.

Le PSNR est calculé par la formule suivante :

$$PSNR = 10\log_{10}(\frac{R^2}{MSE})$$

Avec :

- R : La valeur maximale que peut prendre le pixel de l'image
- MSE : L'erreur quadratique moyenne

3. Partie recherche : les formats BMP & PNG

Dans ce chapitre, on va traiter la partie recherche de notre projet. Dans cette partie, on va traiter les deux formats d'image : BMP & PNG, en expliquant le contenu de leurs entêtes, ainsi que les différents algorithmes de compression d'image y utilisée. Ces deux formats seront notre inspiration pour réaliser notre propre extension « irm » qu'on va présenter dans le dernier chapitre.

3.1 Les formats des fichiers image

Sur Internet, ainsi que nos périphériques, l'image existe sous différents formats (voir Tableau 1). Chaque format a ses propres caractéristiques, et utilisé pour des objectifs différents.

Les fichiers images sont composées de 3 parties principales, l'entête (Header), le corps (Content data), et le pieds (Footer).

3.1.1 L'entête du fichier image

L'entête de l'image est une section des données écrites en binaire ou en ASCII, typiquement trouvée au début du fichier, contenant des informations, nécessaires pour extraire et traiter les données. Tous les fichiers bitmap possèdent un entête, mais l'information qu'il contient varie d'un format à un autre. Néanmoins, voici les composantes principales qu'on trouve dans la plupart, sinon dans tous les fichiers :

- Identificateur ou extension du fichier
- Taille de l'entête
- Description de l'image : nombre de lignes et colonnes, nombre de bits par pixel, nombre de plans de l'image…
- Type de compression si le format prend en charge une sorte d'encodage
- Etc…

3.1.2 Le corps du fichier image

Les données bitmap constituent la majorité du fichier. Elles sont aussi appelées « charge utile » du fichier puisqu'elles contiennent les valeurs des pixels de l'image. Dans la plupart des fichiers, les données se trouvent immédiatement après l'entête bitmap, sauf s'il existe une donnée supplémentaire (palette…). Pour éviter toute confusion, on indique au niveau de l'entête le début des données de l'image.

3.1.3 Le pied du fichier image

Le pied du fichier image est une composante similaire à son en-tête, sauf qu'il est ajouté à la fin du fichier image pour indiquer aux applications les différentes versions de son format, présenter des fonctionnalités spéciales… Le pied du fichier image se trouve à un décalage spécifié par rapport à la fin du corps du fichier image. La valeur de ce décalage peut être ajoutée au niveau de l'entête.

3.2 Le format BMP

Le format fichier BMP est un format fichier ouvert développé par Microsoft et IBM, utilisé pour stocker les images matricielles, monochromes ou en couleur, sous forme des matrices à deux dimensions. Optionnellement, le fichier BMP peut aussi contenir une compression des données, les canaux alpha… BMP a été introduit avec Windows 3.0 en 1990, et est lisible par quasiment tous les visualiseurs et éditeurs d'images.

Les fichiers BMP utilisent le système de little-endian pour stocker un nombre supérieur à un octet. C'est-à-dire que l'octet le moins significatif des données est placé à l'octet avec l'adresse la plus basse. Le reste des données est placé dans l'ordre dans les trois octets suivants.

Un fichier BMP est composé des éléments suivants :

Tableau 2: les composantes d'un fichier BMP

Structure	***Octets correspondants***	***Description***
Header	0x00 – 0x0D (14octets)	Information à propos du fichier : Type, taille, Data Offset…
InfoHeader	0x0E – 0x35 (40 octets)	Informations à propos de l'image : dimensions, type de compression, nombre de plans…
ColorTable	0x36 – variable (4*NumColors)	Indique les couleurs utilisées pour une image indexée
RasterData	Variable (InfoHeader.ImageSize)	Les pixels de l'image

Expliquons en détails le contenu de chacune de ces structures.

3.2.1 L'entête du fichier BMP

Le fichier BMP est composé de deux entêtes : entête du fichier (Header), et entête de l'image (InfoHeader).

3.2.1.1 BMP header

BMP header prend 14octets de la totalité du fichier, et contient 5 champs qui donnent des informations générales à propos du fichier images, expliqués dans le tableau ci-dessous :

Tableau 3: composantes du BMP header

Nom	***Taille***	***Description***
Signature	2 octets	'BM' : indique l'extension du fichier
FileSize	4 octets	Taille du fichier bmp en octets
Reserved	4 octets	Non utilisé (=0)
DataOffset	4 octets	L'octet de départ des pixels de l'image

3.2.1.2 BMP InfoHeader

BMP InfoHeader prend 40octets de la taille du fichier, et se charge de donner l'application différentes informations utilisées pour afficher l'image sur l'écran. BMP InfoHeader contient 11 champs à tailles différentes, expliquées dans le tableau ci-dessous :

Tableau 4: Elements du BMP InfoHeader

Nom	***Taille***	***Description***
Size	4 octets	Taille du InfoHeader (=40 octets)
Width	4 octets	Largeur de l'image en pixels
Height	4 octets	Longueur de l'image en pixels
Planes	2 octets	(=1)
Bits Per Pixel	2 octets	Indique les valeurs de couleurs possibles. 1 : NumColors=1 4 : NumColors=16 8 : NumColors=256 16 : NumColors=65536 24 : NumColors=16M
Compression	4 octets	Type de compression 0 : Sans compression 1 : Encodage RLE 8bits 2 : Encodage RLE 4bits
ImageSize	4 octets	Taille de l'image compressée, (=0) si l'image est non compressée.
XPixelsPerM	4 octets	Résolution horizontale
YPixelsPerM	4 octets	Résolution verticale
Colors Used	4 octets	Nombre de couleurs utilisées (vaut 256 si Bits/Pixel=8)
Important Colors	4 octets	Nombre des couleurs importantes, vaut 0 si toutes les couleurs sont importantes.

3.2.2 La table des couleurs BMP

Dans un fichier BMP, la table des couleurs et obligatoire lorsque le nombre de bits par pixel de l'image est inférieur ou égale à 8. Si le nombre des bits par pixel est 16,24 ou 32 la valeur du pixel est calculée à partir de la combinaison des valeurs individuelles du rouge, vert et bleu.

Elle prend la taille du numéro des couleurs possibles pour un pixel multiplié par 4. Il s'agit d'une table indexée commençant par 0. La valeur entière du pixel pointe sur l'indice du couleur dans la table, pour que la couleur soit affichée sur l'écran. Dans la table des couleurs BMP, les couleurs doivent être citées par ordre d'importance.

Tableau 5: BMP Color table

Nom	***Taille***	***Description***
Red	1 octet	L'intensité du rouge
Green	1 octet	L'intensité du vert
Blue	1 octet	L'intensité du bleu
Reserved	1 octet	(=0)

Ces valeurs sont répétées selon le nombre de couleurs possibles.

3.2.3 Pixel Data de l'image BMP

Cette partie contient des nombres binaires dédiés à représenter les valeurs uniques de la couleur de chaque pixel selon le nombre de bits par pixel.

La taille de ce block ne sera calculée qu'après avoir écrit tous les octets.

3.2.3.1 BMP Scanning

Les cordonnées d'une image BMP commence du coin bas à gauche. C'est-à-dire que le premier pixel de la dernière ligne est représenté par le premier octet dans Pixel Data. Ainsi, la première ligne scannée constitue le dernière ligne de l'image BMP.

Ce processus est illustré par la figure ci-dessous :

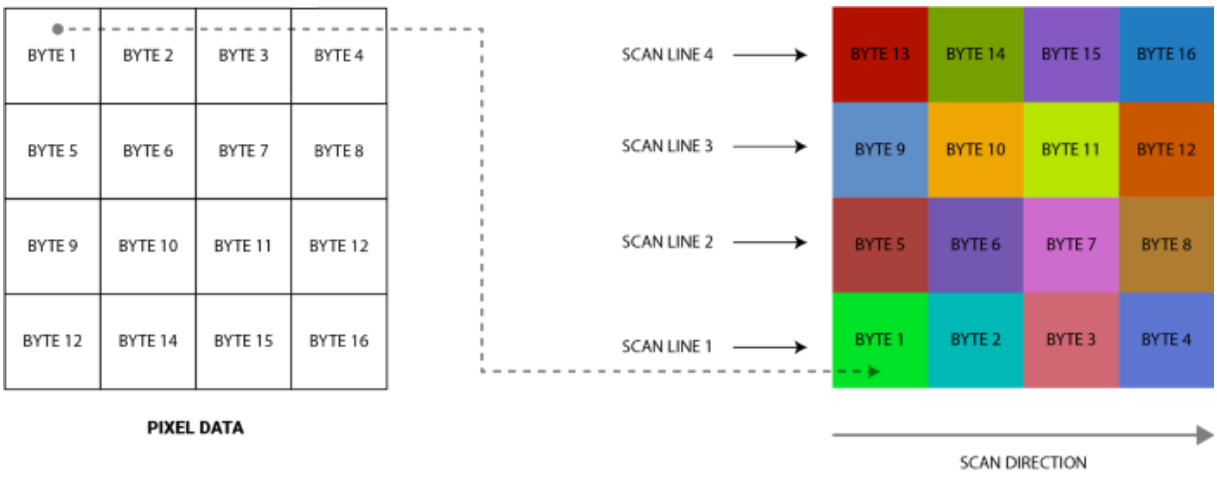


Figure 8: Processus du scanning BMP

3.2.3.2 BMP Scan Lines Padding

Dans le monde réel, un pixel peut prendre quelques bits à quelques octets pour définir sa couleur. Cela dépend uniquement de la valeur du bits par pixel. Mais dans tous les cas, chaque pixel d'un BMP est défini par une taille égale.

Cependant, pour des raisons de cohérence et de simplicité, chaque ligne de balayage est remplie de 0 à la limite de 4 octets la plus proche. Cela signifie que lorsque BMP scanne une ligne de l'image, il considère un bloc de pixels divisible par 4 octets.

3.3 Le format PNG

Le format fichier PNG est un format d'images matricielles qui utilisent une compression sans perte. Ce format a été créer comme alternatif pour le format GIF, sauf qu'elle ne supporte pas les animations.

Le format PNG présente les avantages suivants :

- Simplicité et portabilité : les développeurs doivent être capable d'implémenter PNG facilement.

- Bonne compression : Les images indexée et en vraies couleurs sont effectivement compressées, similairement aux formats qui utilisent une compression avec perte.
- Interchangeabilité : Tour décodeur qui conforme à la norme PNG doit lire tous les fichiers PNG.
- Flexibilité et robustesse : supporte l'intégrité du fichier, et détecte simplement et rapidement les erreurs de transmission.

Un fichier PNG est composé d'une signature PNG, suivie d'une série de "chunks".

3.3.1 Signature du fichier png

Les 8 premiers octets d'un fichier PNG contiennent toujours les valeurs décimales suivantes :

137 80 78 71 13 10 26 10

La signature indique que le reste du fichier contient une image PNG, composé d'une série de « chunks » commençant par un chunk IHDR et terminant par un chunk IEND.

3.3.2 Chunks du fichier PNG

Chaque chunk du fichier PNG contient 3 champs standards :

Tableau 6: PNG Chunk Layout

Nom	***Taille***	***Description***
Length	4 octets	Un entier représentant le nombre d'octets du champ Chunk Data.
Chunk Type	4 octets	Montre le type du chunk (IHDR, IDAT, IEND…)
Chunk Data	-	Contient les octets des données relatifs au Chunk Type.
CRC 4	4 octets	Calculé sur les champs du type et des données, mais non pas sur le champ du Length.

Les chunks d'un fichier PNG sont divisés en deux types : critiques ou auxiliaires, basés sur les 4 octets du champ Chunk Type. Une image PNG valide doit contenir un IHDR Chunk, un IDAT Chunk et un IEND chunk.

3.3.2.1 IHDR chunk : Image header

Le IHDR Chunk doit être le premier à figurer dans le fichier. Il contient 7 champs expliqués dans le tableau suivant :

Tableau 7: PNG IHDR Chunk

Nom	***Taille***	***Description***
Width	4 octets	Dimensions de l'image en pixel (largeur & longueur).
Height	4 octets	
Bit Depth	1 octets	Un entier exprimant le nombre de bits par indice de palette. Peut prendre les valeurs 1,2,4,8, et 16.
Color Type	1 octets	Un entier qui décrit l'interprétation de la donnée (palette utilisée, couleur utilisée, canal alpha utilisé…). Peut prendre les valeurs 0,2,3,4,6
Compression Method	1 octets	Un entier indiquant le type de compression (Deflat/Inflate compression) utilisé. Prend la valeur 0.
Filter Method	1 octets	Un entier indiquant la méthode du pré-traitement appliquée sur l'image avant la compression. Prend la valeur 0.
Interlace Method	1 octets	Un entier indiquant l'ordre de transmission de l'image. Peut prendre la valeur 0 (no interlace), ou 1 (Adam7 interlace).

3.3.2.2 PLTE Chunk : Image palette

Le PLTE chunk contient 1 jusqu'à 256 entrées de palette, chacune prend 3 octets de la forme suivante :

$$\text{Red: 1byte } (0 = \text{black}, 255 = \text{red})$$
$$\text{Green: 1byte } (0 = \text{black}, 255 = \text{green})$$
$$\text{Blue: 1byte } (0 = \text{black}, 255 = \text{blue})$$

Le nombre d'entrées et déterminé par le champ Length. Une valeur qui n'est pas un multiple de 3 génère une erreur. Le PLTE Chunk doit figurer pour le Color Type 3, et peut exister pour les Color Type 2 et 6. Il ne peut pas être présent avec les Color Type 0 et 4.

Pour le Color Type 3 (couleurs indexées), le chunk PLTE est obligatoire. La première entrée du PLTE est référencée par la valeur 0, le deuxième par la valeur 1, et ainsi de suite… Le nombre des entrées de la palette doit être égale ou inférieur à l'intervalle qui peut être représenté par le Bit Depth. Tout pixel de l'image qui a une valeur qui dépasse l'intervalle, donnera une erreur.

3.3.2.3 IDAT chunk : Image Data

Conceptuellement, une image PNG est un tableau rectangulaire de pixels, apparaissant du gauche à droit dans chaque ligne de balayage qui apparaissent du haut ver le bas.

Le chunk IDAT contient la donnée de l'image. Pour créer cette donner :

- Commencer pas des lignes da balayages comme expliqué ci-dessus. La taille totale de la donnée image est déterminée par les champs du IHDR.
- Filtrer la donnée de l'image selon la méthode du filtrage spécifié dans le IHDR.
- Compresser la donnée filtrée en utilisant le type de compression spécifié par le IHDR.

Il peut y avoir plusieurs IDAT chunks. Si c'est le cas, ils doivent apparaitre consécutivement sans aucuns autre chunks intervenant. Le flux de données compressé est la concaténation du contenu de tous les chunks IDAT.

3.3.2.4 IEND Chunk : Image trailer

Le IEND chunk doit être le dernier à apparaître. Il marque la fin du flux de données. Le champ du IEND chunk reste vide.

3.3.2.5 Ancillary chunks

Le fichier PNG contient des chunks supplémentaires optionnels.

- bKGD (background color) : spécifie l'image de l'arrière-plan contre laquelle l'image sera présentée.
- cHRM (Primary chromaticities and white point) : utilisé par les applications qui nécessitent une spécification des couleurs du fichier PNG indépendante de l'appareil.
- gAMA (Image Gamma) : Spécifie la gamma de la caméra qui a produit l'image.
- hIST (Image Histogram) : Donne la fréquence approximative de l'utilisation de chaque couleur dans la palette
- pHYs (Physical Pixel Dimension) : spécifie la taille de pixel ou le rapport hauteur/largeur prévu pour l'affichage de l'image.
- sBIT (Significant Bits) : Stocke le nombre initial de bits significatifs.
- tEXT (Textual Data) : Informations textuelles que l'encodeur désire enregistrer avec l'image.
- tIME (Image Last-modification Time) : donne l'heure de la dernière modification de l'image.
- tRNS (Transparency) : spécifie que l'image utilise une transparence simple
- zTXt (Compressed Textual Data) : similaire à tEXt, mais zTXt est recommandé pour stocker de gros blocs de texte.

3.3.3 Deflate / Inflate compression

La méthode de compression 0, qui est la seule méthode actuellement présente dans PNG, spécifie une compression Deflate/Inflate avec une fenêtre glissante. La compression Deflate est une compression dérivative de la technique LZ77 utilisé dans le format zip.

Les flux de données compressés avec la compression Deflate dans png, sont stockés sous le format « zlib », qui a la structure suivante :

Compression method/flags code:	1 byte
Additional flags/check bits:	1 byte
Compressed data blocks:	n bytes
Check value:	4 bytes

3.3.4 Algorithmes de filtrage

La méthode de filtre PNG 0 définit cinq types de filtres de base :

Tableau 8: Filtrage PNG

Type	***Nom***
0	None
1	Sub
2	Up
3	Average
4	Paeth

- Filtre type 0 (None) :
- Filtre type 1 (Sub) : Transmet la différence entre chaque octet et la valeur de l'octet correspondant du pixel précédent.

$$\mathrm{Sub}(x) = \mathrm{Raw}(x) - \mathrm{Raw}(x - \mathrm{bpp})$$

- Filtre type 2 (Up) : Similaire au filtre Sub, sauf que le pixel immédiatement au-dessus du pixel actuel, plutôt que juste à sa gauche, est utilisé comme prédicteur.

$$\mathrm{Up}(x) = \mathrm{Raw}(x) - \mathrm{Prior}(x)$$

- Filtre type 3 (Average) : Utilise la moyenne des deux pixels voisins (à gauche et au-dessus) pour prédire la valeur d'un pixel.

$$\mathrm{Average}(x) = \mathrm{Raw}(x) - \mathrm{floor}((\mathrm{Raw}(x - \mathrm{bpp}) + \mathrm{Prior}(x))/2)$$

- Filtre type 4 (Paeth) : Calcule une fonction linéaire simple des trois pixels voisins (gauche, haut, haut gauche), puis choisit comme prédicteur le pixel voisin le plus proche de la valeur calculée.

$$\mathrm{Paeth}(x) = \mathrm{Raw}(x) - \mathrm{PaethPredictor}(\mathrm{Raw}(x - \mathrm{bpp}), \mathrm{Prior}(x), \mathrm{Prior}(x - \mathrm{bpp}))$$

4. Partie réalisation : extension « irm » pour la compression d'images

Dans cette partie, on va aborder la partie réalisation de notre projet. Cette partie consiste à réaliser notre propre extension appelée « irm », qui permet la compression d'images. On a utilisé deux techniques, une sans perte et l'autre avec perte. Dans ce qui vient, on va expliquer les étapes de notre programme, présenter et critiquer les différentes mesures de performances, et terminer par donner des conclusions et perspectives.

4.1 Extension IRM pour la compression sans perte

Dans la partie de compression « irm » sans perte, on a effectué une application directe du code de Huffman sur l'image.

4.2 Entête IRM

Au niveau de tous les formats d'images existants, et particulièrement au niveau des deux formats traités au niveau de la partie théorique (BMP et PNG), l'entête constitue une structure obligatoire pour tous les formats images compressés et non compressés. Organiquement, notre extension « irm » ne sera pas valide sans un entête qui offre les différentes informations à propos du fichier et de l'image.

Un fichier IRM est divisé en 3 parties : FILE HEADER, IMAGE HEADER et IMAGE DATA.

- FILE HEADER

Tableau 9: IRM FILE HEADER

Nom	***Taille***	***Description***
FILE HEADER	8 octets	Contient des informations à propos du fichier
Signature	1 octet	L'extension « irm »
File Size	4 octets	La taille totale du fichier
Data Offset	2 octets	Point de départ des pixels

- IMAGE HEADER

Tableau 10 : IRM IMAGE HEADER

Nom	***Taille***	***Description***
IMAGE HEADER	16 octets	Contient des informations à propos de l'image
Height	4 octets	Les dimensions de l'image
Width	4 octets	
Planes Nbr	4 bits	Nombres de plans de l'image
Compression Type	4 bits	Type de compression (0 : sans perte, 1 :quantification, 2: rgb2hsl, 3 : quantification + rgb2hsl)

- IMAGE DATA

Tableau 11: IRM IMAGE DATA

Nom	***Taille***	***Description***
IMAGE DATA	-	
DATA HEADER	4 octets	Dépend du type de compression
Maxbit	2 octets	Longueur maximale des codes
Len Huffman_dict	2 octets	Longueur du dictionnaire Huffman
Dictionnaire + code Huffman	4 bits	Le dictionnaire Huffman + les pixels codés

4.2.1 Encodeur sans perte

- Le nombre de plans de l'image :

Si l'image a trois plans, on constitue un seul dictionnaire pour les trois plans afin de créer des répétitions.

- La méthode d'écriture :

L'écriture binaire du dictionnaire de Huffman se fait de la manière suivante :

- La longueur du dictionnaire
- Le max des longueurs du code pour gagner de l'espace.

Pour les symboles du dictionnaire de Huffman, on aura besoin de transmettre les symboles et leurs codes. Puisque les codes ont une longueur variable, on aura besoin de coder la longueur de chaque code.

- Le symbole est codé sur 8 bits, puisque les valeurs des pixels varient entre 0 et 255.
- La longueur de chaque code est codée sur le maximum des longueurs des codes.
- Le code est codé sur la longueur lui assignée par l'arbre de Huffman.

Par la suite, on code les valeurs des pixels de l'image en utilisant le dictionnaire créé dans l'étape précédente.

Exemple de compression

Soit le dictionnaire suivant : {5 :010, 32 :00110000, 8 :10000}.

Le code de la valeur 32 a une longueur maximale qui vaut 8, et qui peut être codée sur 4 bits. Codant les longueurs des autres codes sur 4 bits.

- Len (010) = 3= 0011
- Len (00110000) =8 =1000
- Len (1000) =5= 0101

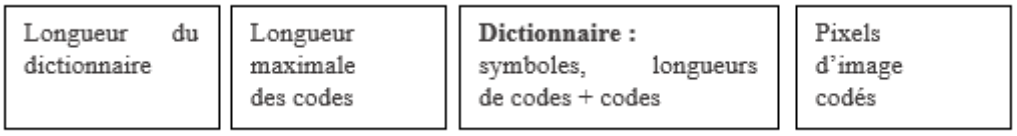
On code la longueur maximale qui est 4 bits sur 8 bits, on aura : 00000100.

Le codage du symbole 5 à titre d'exemple sera comme suivant :

5 : 000001010011010 (5 en binaire sur 8 bits+longueur de son code =3 sur 4 bits et son code sur 3 bits)

4.2.2 Décodeur sans perte

Après avoir reçu la séquence codée, elle contient les informations nécessaires pour la parcourir et la décoder, notamment la longueur du dictionnaire, la longueur des codes, et les codes dans l'ordre suivant :



- On récupère la longueur du dictionnaire ainsi que la longueur maximale des codes.
- On utilise la longueur du dictionnaire pour récupérer le dictionnaire.
- Pour pouvoir récupérer les symboles et leurs codes dans le dictionnaire, on parcourt par un pas de 8 pour avoir le symbole (clé), par un pas de longueur_maximale_des_codes pour avoir la longueur du code, et puis par un pas de cette longueur pour récupérer le code (valeur).
- A fur et à mesure qu'on trouve la clé et la valeur correspondante, on les ajoute dans le dictionnaire qu'on va utiliser par la suite pour décompresser les pixels de l'image.

Exemple de décompression

4.3 Extension IRM pour la compression avec perte

L'algorithme de compression avec perte opère en effectuant une transformation de l'espace de couleurs du RGB vers HSL, suivi d'une étape de quantification.

4.3.1 Transformation de l'espace de couleur:

La première transformation par laquelle passe l'image pendant la compression, est une transformation de l'espace de couleurs RGB (Red, Green, Blue), vers l'espace de couleurs HSL (Hue,

Saturation, Lightness) à partir des équations mathématiques linéaires expliquées dans la section 2.3.1.2.

Le choix de l'espace de couleurs HSL est basé sur la similarité des valeurs de teinte, de saturation et de luminosité même entre les pixels de couleurs différentes. La chose qui va mener à plus de redondances.

Les valeurs obtenues lors de cette transformation sont des valeurs normalisées entre 0 et 1. On passe vers des valeurs entières en multipliant par 255 et en arrondissant.

4.3.2 Quantification:

Pour encore réduire la taille de notre image après l'étape du changement de l'espace de couleurs, on effectue une quantification.

L'idée de cette étape de quantification consiste à rendre tous les pixels des multiples de 8 pour avoir plus de répétitions. On effectue une division entière par 8 de toutes les valeurs des pixels, et on envoi le quotient. Au niveau de la reconstruction, cette valeur va être multipliée par 8, et on ne va perdre que 0 à 7 des valeurs des pixels, puisqu'elles représentent les restes possibles pour une division par 8.

4.3.3 Codage par Huffman

Après avoir quantifié les values des pixels qui sont dans l'espace de couleur HSL, on passe à la dernière étape de notre compression avec perte. Il s'agit du codage par Huffman.

L'algorithme de Huffman expliqué dans la partie de la compression sans perte, les symboles sont codés sur 8 bits puisque la valeur maximale des pixels vaut 255. Après avoir quantifié les pixels de l'image, cette valeur devient 31 (résultat de la division entière de 255 par 8). Cette valeur maximale, n'a besoin que de 5 bits pour être codée.

Par conséquent, on optimise le code de Huffman en réservant seulement 5 bits pour chaque valeur de pixel, au lieu de 8 bits comme dans le cas de Huffman sans perte.

4.4 Écriture dans un fichier binaire

Après toutes ces étapes de compression, que ça soit sans ou avec perte, vient l'étape finale. Il s'agit de l'écriture des pixels de l'image accompagnés de l'entête dans un fichier binaire sous l'extension « irm ».

L'écriture dans un fichier binaire se fait selon les étapes suivantes :

- On parcourt notre séquence par blocs de 8 bits, on convertit chaque bloc en un entier qui sera compacté dans l'objet byte en utilisant la fonction to_bytes.
- Au niveau de l'entête, on envoi le pas de la lecture qui sera utilisé dans la relecture de notre séquence binaire.

Remarque : l'écriture dans le fichier binaire se fait en mode 'wb' avec l'extension 'irm'.

4.5 Interface graphique

Pour bien présenter et visualiser les résultats de notre projet, on a réalisé une interface graphique à l'aide du package « tkinter » pour réalisation des GUI.

L'interface graphique initiale, sans image entrée contient les éléments suivants :

- Un panneau « compression » à partir duquel l'utilisateur peut choisir le type de la compression désiré (avec ou sans perte), choisir l'image qu'il désire compresser et confirmer avec le bouton « compress ».

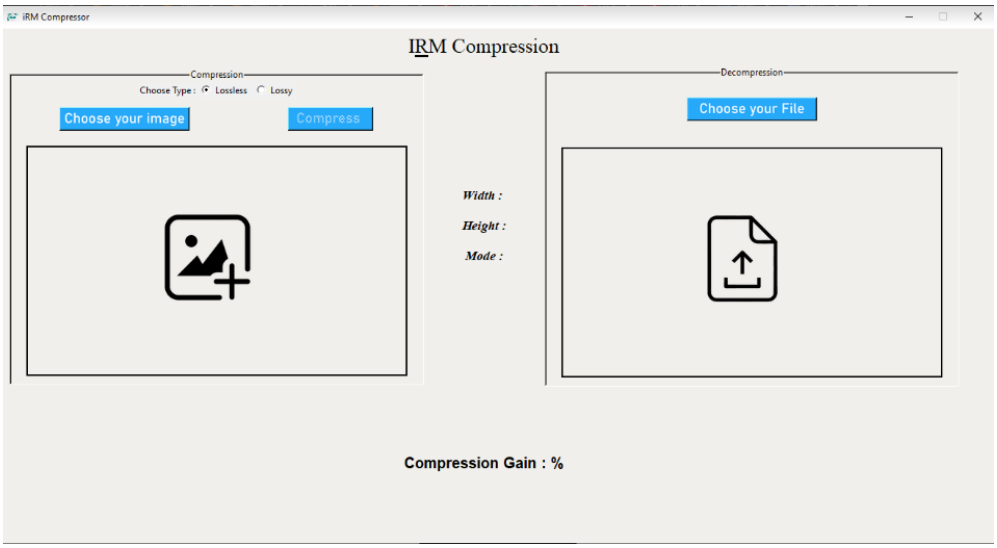


Figure 9: Interface graphique (1)

- Après avoir choisi l'image et cliqué sur le bouton « compress », l'interface diffuse les différentes informations relatives à l'image choisie, voire la largeur, la longueur, le mode. Au-dessous de l'image, une comparaison entre la taille initiale et la taille compressée ainsi que le chemin du fichier compressé et le gain de compression seront affichées sous format « irm ».

- Pour une compression sans perte :

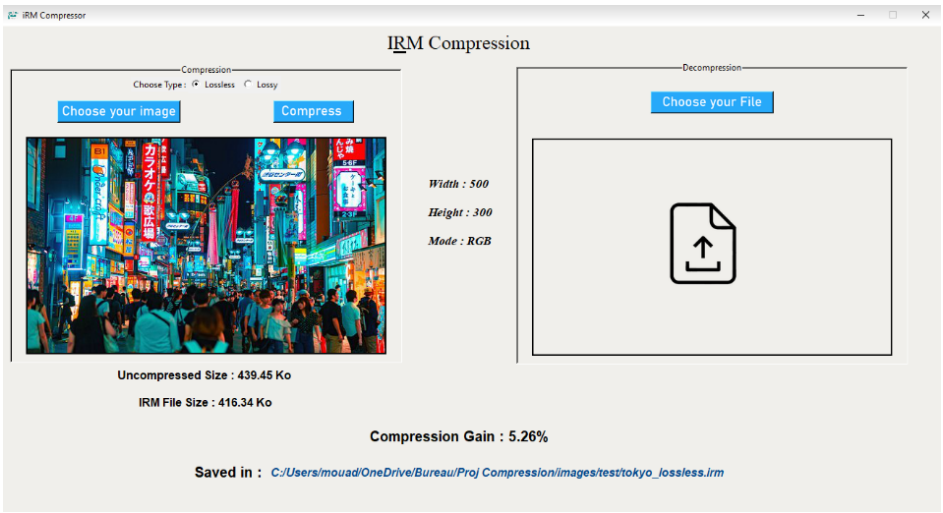


Figure 10: Interface graphique - sans perte

- Pour une compression avec perte :

Figure 11: Interface graphique - avec perte

- Le deuxième panneau « decompression », permet d'afficher une image « .irm » après décompression.

Figure 12: interface graphique - décompression

4.6 Mesures de performance

Pour tester et pouvoir analyser les résultats obtenus à partir de nos algorithmes réalisés, il est nécessaire de mesurer la

performance de notre programme en utilisant les différents critères. Le tableau suivant présente quelques tests effectués.

Tableau 12: mesures de performances

Image	***Type de compression***	***Taux***	***MSE***	***Efficacité***	***Entropie***
	Huffman sans perte	15.73%	0.0	99.6%	6.69
	Quantification	45.49%	6.22	-	
	Rgb2hsl	6.78%	0.55		
	Quantification+rgb2hsl	42.62%	19.28		
	Huffman sans perte	8.19%	0.0	99.5%	7.31
	Quantification	45.2%	5.9	-	
	Rgb2hsl	22.92%	0.41		
	Quantification+rgb2hsl	57.9%	27.63		
	Huffman sans perte	1.07%	0.0	99.7%	7.88
	Quantification	38.51%	5.79	-	
	Rgb2hsl	4.62%	0.33		
	Quantification+rgb2hsl	40.94%	12.49		

4.7 Résultats et conclusions générales

D'après tout ce qui précède et après avoir analysé nos résultats de recherche et de codage on peut conclure que :

- La compression d'images est une discipline très vaste et riche en techniques diverses et algorithmes de compression qui peuvent être classifiés de plus simple au plus compliqué.
- Les transformations que peut subir un pixel d'une image sont nombreuses ce qui montre la force de la représentation de la donnée dans son aspect spatial, mais aussi dans son aspect fréquentiel.
- Les méthodes de compression les plus répandues prennent leur force de la diversité en leurs algorithmes implémentés.
- Une très grande partie de la qualité de la compression, la compression avec perte en particulier, dépend de pouvoir exploiter les limites du système visuel humain et son incapacité à remarques les changements subtiles au niveau de l'image.
- Il est évident que l'algorithme choisi en fonction de la nature de la donnée joue un rôle énorme dans l'efficacité de la compression, mais un aspect aussi important est la représentation binaire de la donnée compressée en mémoire, qui peut améliorer l'algorithme pour donner des meilleurs résultats, mais qui peut également le détériorer en augmentant la taille de l'image.
- Parmi les points forts de l'image numérique, la diversité des espaces couleurs par laquelle elle peut être présentée.
- Grâce aux limites de la perception visuelle humaine, on ne se contente pas seulement de manipuler les valeurs des pixels, mais on exploite également d'autre critères : luminance, chrominance, …

- La qualité de la compression dépend également de l'image, son niveau de saturation, sa texture, les redondances au niveau de ses pixels… Par conséquent, un seul algorithme peut donner des résultats différents d'une image à une autre. Par exemple, RLE augmente la taille de l'image qui ne présente pas beaucoup de redondances (ce qui est le cas pour la majorité des images), et LZW qui fonctionne mieux sur le texte que sur les images à textures désordonnées.
- Pour évaluer un algorithme, le taux de compression n'est pas la seule chose à considérer, mais aussi la complexité et le temps du calcul, ainsi que le temps nécessaire pour la transmission.

Fin