Alexandre Boucher & Pierre-Olivier Parisé
Maîtrise en mathématiques et informatique appliquées
Concepts avancés en mathématiques et informatique appliquées, MAP6014

Compression d'images avec les fractales

Rapport final

Travail présenté à
François Meunier

Université du Québec à Trois-Rivières
11 janvier 2016

Table des matières

Liste des figures

Liste des tableaux

Rapport

Introduction

Les images sont des supports qu'utilisent de nombreuses personnes à travers le monde. Que ce soit pour partager des photos de vacances sur son ordinateur, placer une image sur la Toile, créer des dessins, photocopier des documents, observer la Terre, etc., les images demeurent omniprésentes dans notre quotidien. Cependant, qui sait ce qui se cache derrière le support tant répandu ? Qui sait ce qui se passe lorsqu'un ordinateur affiche une image à l'écran ? L'une des clés afin de répondre à ces questions est la compression d'images.

Il existe plusieurs algorithmes de compression d'images : soit les algorithmes JPEG, JPEG 2000, PNG pour ne nommer que les plus connus. Par contre, les développements informatiques des dernières décennies ont permis d'observer une toute nouvelle catégorie d'objets : les fractales. Les fractales ont été formalisées par le mathématicien Benoît Mandelbrot. Cette formalisation a été complétée par Micheal F. Barnsley du point de vue mathématique. Ainsi, ces objets mathématiques particuliers sont au cœur de notre travail.

Dans notre projet, nous voulons étudier dans ses moindres détails un type d'algorithme qui permet d'enregistrer des images sur un certain support : la méthode de *Jacquin* pour la compression fractale. C'est une méthode très peu connue qui gagne en popularité. Notre problématique repose donc sur la manière dont les fractales peuvent servir le domaine de la compression d'images (voir la section 1.3). Pour y arriver, nous allons discuter brièvement des principes généraux de la compression d'images et de la problématique de notre projet. Ensuite, dans le second chapitre, nous allons introduire quelques concepts de base en mathématiques et quelques notations utiles pour ce travail. Puis, nous allons présenter la méthode de la compression fractale. Enfin, nous énonçons puis analyserons les résultats obtenus en utilisant l'algorithme de compression d'images par les fractales sur différents formats d'images.

Chapitre 1

Généricités sur la compression d'images

1.1 La compression d'images

À la base, la compression d'images a pour but de réduire la taille d'un fichier de type image. Pour ce faire, elle analyse l'image afin de rechercher des redondances et les éliminer. Ainsi, l'information à enregistrer est réduite et, par ricochet, la mémoire utilisée (taille du fichier) l'est aussi.

La compression d'images peut s'effectuer avec ou sans perte d'informations. Plusieurs méthodes sont proposées dans la littérature afin de réduire la taille d'une image. Le tableau suivant contient quelques noms de méthodes fréquemment utilisées dans le domaine de la compression d'images [Wik15].

Sans perte d'information	Avec perte d'information
Codage des répétitions	Codage par transformations (Fourier, Ondelettes)
Codage entropique	Compression fractale
Réduction d'espace des couleurs	
Sous-échantillonnage de la chrominance	

TABLE 1.1 – Différentes méthodes employées dans le domaine de la compression d'images

1.2 Les formats d'image

Ces méthodes doivent enregistrer soit l'image directement, soit l'image compressée en un format particulier. Plusieurs formats sont disponibles. Leur nom varie selon l'algorithme utilisé pour compresser l'image, le système d'exploitation utilisé ou le groupe de travail auquel est associé

ce format. Voici quelques exemples de formats d'image utilisés le plus souvent dans l'univers de l'informatique [SAR15] :

(a) PNG : Portable Network Graphics ;

(b) JPEG : Joint Photograpic Expert Group ;

(c) JPEG 2000 : JPEG amélioré ;

(d) BMP : Windows BitMap ;

(e) TIFF : Tagged Image File Format ;

(f) GIF : Graphic Interchange Format ;

(g) FIF : Fractal Image Format.

Cette liste n'est pas exhaustive, mais elle présente les principaux formats d'image utilisés et ceux qui seront utilisés dans le futur (.FIF) [1].

1.3 Problématique en compression d'images

La sauvegarde d'images nécessite beaucoup de mémoire dès que l'image dépasse une certaine taille et que la résolution de l'image est grande (par exemple une image haute définition de 1440 par 1920 pixels). Dans cette situation, certaines méthodes doivent être développées afin de réduire le cout en mémoire de la sauvegarde de l'image sur un support (disque dur, clé USB, CD-ROM, etc.). Par exemple, le codage entropique (codage de Huffman), la transformation en cosinus discrets (DCT), les ondelettes, etc. sont des méthodes pour réduire le cout en mémoire sur le support informatique.

Cependant, cette réduction peut engendrer des pertes significatives d'informations contenues dans l'image selon le ratio de compression souhaité. Aussi, certaines images construites artificiellement ou captées par des appareils peuvent être très complexes. Par exemple, si une image a la majorité de son spectre de Fourier dans les hautes fréquences, alors l'algorithme JPEG, qui utilise les DCT pour éliminer les hautes fréquences dans l'image, réduit grandement la qualité de l'image lors de la compression. De plus, la méthode JPEG est dépendante de la résolution de l'image.

1. Dans le cadre de notre travail, le fichier a une extension .fcci pour *fractal compressed color image*. Le format .FIF est un format utilisant un algorithme plus complexe que celui utilisé dans le cadre de notre projet.

Ainsi, quelles sont les méthodes qui permettraient de résoudre ces problèmes ? Nous proposons une méthode de compression, à l'aide des fractales, qui ne dépend pas des fréquences dans le domaine de Fourier et peut être utilisée pour générer des images à une résolution plus grande qu'au départ. Comme la compression JPEG est très documentée et utilisée comme standard, nous allons nous attarder à expliquer la compression fractale. Plus particulièrement, nous allons détailler la méthode dite de *Jacquin* [SAN15 ; Fis95].

Avant tout cela, nous introduisons quelques concepts mathématiques de base et quelques notations utiles.

Chapitre 2

Les mathématiques de base et notations

Dans ce chapitre, nous spécifions les outils mathématiques à la base de la compression fractale et quelques notions de base sur les images polychromatiques.

2.1 Les espaces métriques

Soit X un espace vectoriel non vide quelconque. Sur cet espace, on définit une **fonction distance** $d : X \times X \to \mathbb{R}$ qui satisfait les propriétés suivantes :

1. $d(x, x) \geq 0$ pour tout $x \in X$;
2. $d(x, y) = d(y, x)$ pour tout $x, y \in X$;
3. $d(x, z) \leq d(x, y) + d(y, z)$ pour tout $x, y, z \in X$.

Le couple (X, d) s'appelle un espace métrique. Un exemple classique est l'espace $\mathbb{R}^n$ avec la distance euclidienne définie par

$$d(\mathbf{x}, \mathbf{y}) := \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

où

$$\mathbf{x} := (x_1, x_2, \ldots, x_n) \quad \text{et} \quad \mathbf{y} := (y_1, y_2, \ldots, y_n).$$

Un autre exemple intéressant est l'espace des fonctions continues

$$C(K) := \{f : K \to \mathbb{R} : f \text{ est continue sur } K\}$$

muni de la fonction distance

$$d(f, g) := \sup_{x \in K} \{|f(x) - g(x)|\}$$

où $f, g \in C(K)$ et K est un ensemble compact de $\mathbb{R}$ (ou $\mathbb{R}^n$). Dans ce cas $(C(K), d)$ est un espace métrique. L'espace $C(K)$ peut être une fonction qui représente les niveaux de gris dans une image K.

Nous verrons plus loin un autre exemple d'espace métrique : l'espace des fractales.

2.2 Des outils de statistiques et probabilités

Quelques outils puisés dans les domaines de la Statistique et des Probabilités sont utilisés afin de réaliser la compression fractale.

Supposons que nous disposons d'un jeu de données $X := \{x_1, x_2, \ldots, x_n\}$. Un estimateur sans biais pour la moyenne de ces données est

$$\overline{x} = \frac{1}{n} \sum_{j=1}^{n} x_j.$$

Dans le domaine de la compression d'image, la moyenne est utilisée afin de lisser une région d'une image et de réduire la quantité de données qui représente cette région. En effet, il suffira d'un nombre (la moyenne des intensités des pixels de la région) afin de représenter l'intensité de la couleur de cette région de l'image. Dans le cours, nous avons aussi vu que ce procédé correspond à un certain filtre pour lisser une image.

Dans la prochaine section, nous introduisons les transformations affines. Ces transformations sont utilisées dans la compression fractale, mais les paramètres de ces transformations doivent être estimés. L'outil utilisé afin de réaliser cette estimation est la *méthode des moindres carrés*. Supposons que nous disposons de deux jeux de données $A := \{a_1, a_2, \ldots, a_n\}$ et $B := \{b_1, b_2, \ldots, b_n\}$. Nous souhaitons construire une droite $s \cdot a + o$ qui passe le plus près des points du jeu de données

B. Pour déterminer les coefficients s et o, on doit minimiser la quantité χ^2 définie par

$$\chi^2 := \sum_{i=1}^{n} (s \cdot a_i + o - b_i)^2.$$

La solution de ce problème d'optimisation est

$$s = \frac{n \sum_{i=1}^{n} a_i b_i - \left(\sum_{i=1}^{n} a_i\right)\left(\sum_{i=1}^{n} b_i\right)}{n \sum_{i=1}^{n} a_i^2 - \left(\sum_{i=1}^{n} a_i\right)^2}$$

et

$$o = \frac{1}{n}\left[\sum_{i=1}^{n} b_i - s \sum_{i=1}^{n} a_i\right].$$

L'erreur commise est donnée par la racine carrée de la quantité χ^2 après avoir remplacé s et o par les valeurs trouvées. On définira aussi l'erreur entre deux images à partir d'une formule similaire plus loin.

2.3 Types de transformations

Dans ce travail, nous allons considérer certains types de transformation qui agissent sur les éléments d'un espace métrique, plus particulièrement les éléments de l'espace $(\mathbb{R}^n, d)$.

On définit une **fonction contractive** ou simplement **une contraction** comme étant une fonction $w : X \to X$ qui a la propriété suivante :

$$d(w(\mathbf{x}), w(\mathbf{y})) \leq s \cdot d(\mathbf{x}, \mathbf{y})$$

pour tout $\mathbf{x}$, $\mathbf{y} \in X$ et où $0 \leq s < 1$ est un **facteur de contraction**. Une contraction a donc pour effet de réduire la distance entre deux éléments de l'espace.

On définit une **transformation affine** de l'espace $\mathbb{R}^n$ vers un espace $\mathbb{R}^m$ comme suit

$$w(\mathbf{x}) := A\mathbf{x} + \mathbf{t}$$

où $\mathbf{x} \in \mathbb{R}^n$, la matrice A est de dimensions $m \times n$ et le vecteur $\mathbf{t}$ de dimensions $m \times 1$. Par exemple, une transformation affine $w : \mathbb{R}^2 \to \mathbb{R}^2$ ressemble à ceci :

$$w(x, y) = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}.$$

2.4 Les fractales en nD

En 1980, un mathématicien révolutionne les mathématiques en décrivant des objets qu'il a nommés fractales. Il s'agit de Benoît Mandelbrot. Trois propriétés sont données par ce dernier mathématicien afin de qualifier si un objet a un caractère fractal :

(i) Autosimilarité ;

(ii) Détails à toutes les échelles ;

(iii) Objet irrégulier.

D'abord, la propriété d'autosimilarité est simple. Il s'agit de voir si les parties de l'objet ressemblent à l'objet lui-même. Ensuite, un objet dont on peut s'approcher aussi près que l'on veut sans perte de détails possède la propriété des détails à toutes les échelles. Puis, un objet très fragmenté, irrégulier ou brisé satisfait la troisième propriété. La deuxième propriété est très intéressante pour la compression d'images puisque l'outil introduit plus loin est un type particulier de fractales. Ainsi, il serait possible d'agrandir une image sans subir l'effet de pixelisation, souvent rencontrer dans les formats JPEG ou PNG.

La figure 2.1 illustre quelques exemples de fractales provenant de la nature et de formules mathématiques.

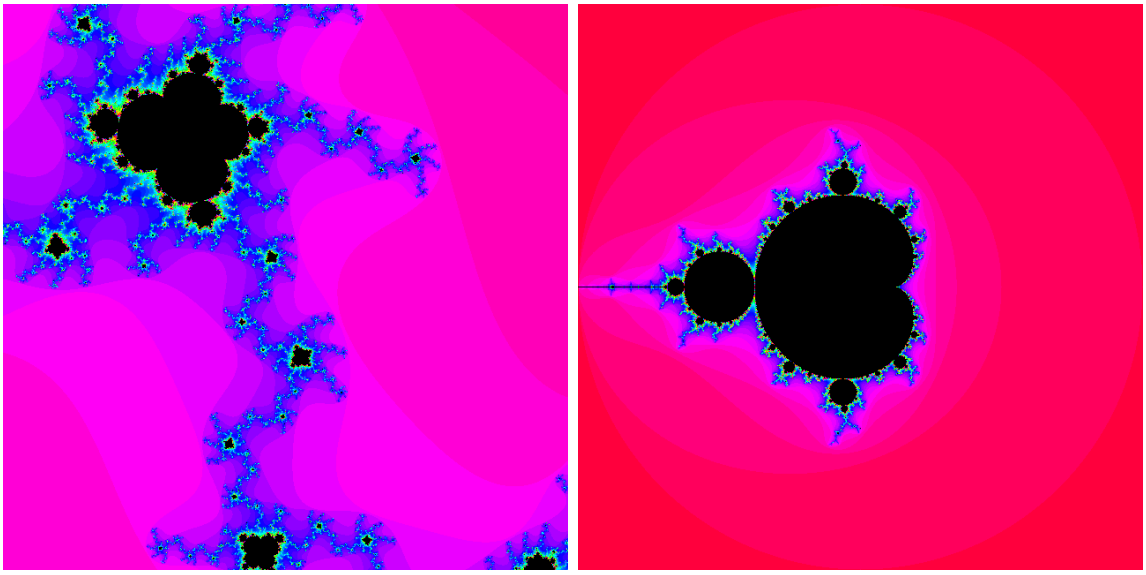
Selon M. F. Barnsley [BR93], les fractales sont présentes partout dans la nature. Il explique cela par le fait que les êtres vivants utilisent des processus répétitifs afin de se former et d'évoluer. D'ailleurs, M. F. Barnsley est le premier à formuler rigoureusement l'espace des fractales $\mathcal{H}(\mathbb{R}^n)$. On définit cet espace comme ceci

$$\mathcal{H}(\mathbb{R}^n) := \{A \subset \mathbb{R}^n \,:\, A \text{ est un ensemble compact de } \mathbb{R}^n\}$$

et sa fonction distance $h : \mathcal{H}(\mathbb{R}^n) \to \mathcal{H}(\mathbb{R}^n)$ est définie par l'expression suivante

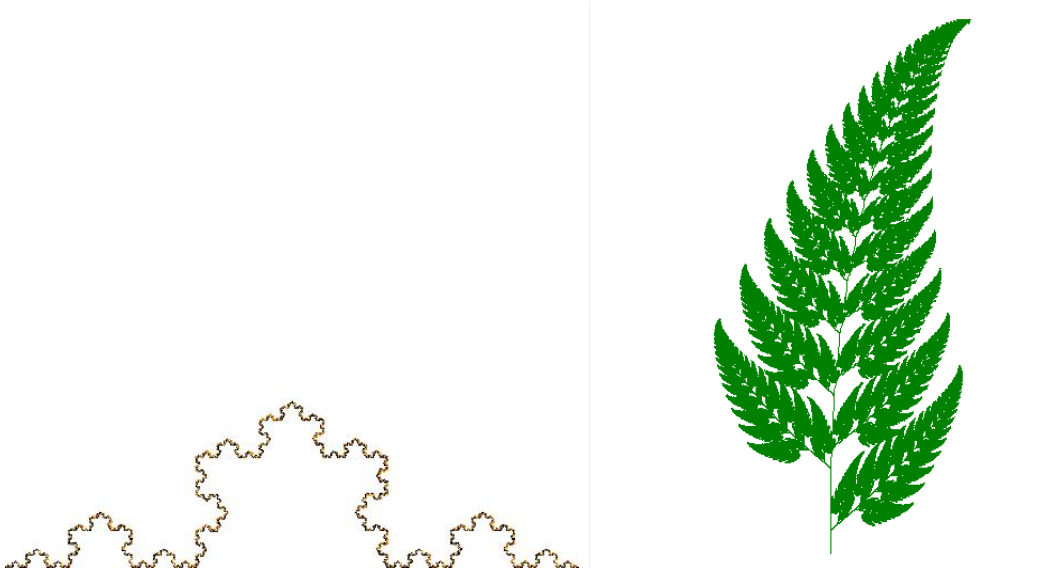
$$h(A, B) := \max\{d(A, B), d(B, A)\}$$

où A et B sont des sous-ensembles compacts de $\mathbb{R}^n$. Le nombre $d(A, B)$ est la plus grande distance qui sépare les points de l'ensemble A à l'ensemble B. Tandis que le nombre $d(B, A)$ est la plus grande distance qui sépare les points de l'ensemble B à l'ensemble A. Dans ce cas, $(\mathcal{H}(\mathbb{R}^n), h)$ est un espace métrique. La distance d est la distance euclidienne habituelle sur $\mathbb{R}^n$ telle que définie à la section 2.1. De plus, l'espace des fractales est un espace métrique complet, c'est-à -dire que



(a) Multibrot ordre 5

(b) Mandelbrot



(c) La courbe de Koch

(d) Fougère de Barnsley [Laj05]

FIGURE 2.1 – Exemples d'objets fractals

toutes les suites de Cauchy [1] dans l'espace converge vers un élément $A \in \mathcal{H}(\mathbb{R}^n)$. Ceci est important puisque le processus de la compression d'images par les fractales nécessite que l'espace soit complet : les images appartiennent à l'espace des fractales puisque ce sont des sous-ensembles compacts de $\mathbb{R}^n$.

2.5 La représentation d'une image

Les images traitées dans ce travail sont composées de pixels. Chaque pixel a trois composantes afin de représenter sa couleur : Rouge (R), Vert (G) et Bleu (B). Donc, une image I de taille h pixels par l pixels est représentée par un ensemble de vecteurs à cinq composantes :

$$I := \{(x, y, R(x, y), G(x, y), B(x, y)) \ : \ 0 \leq x \leq l, \ 0 \leq y \leq h, \quad 0 \leq R, G, B \leq 255\}$$

où (x, y) est la position du pixel dans l'image, h et l sont la hauteur et la largeur de l'image respectivement et $R(x, y)$, $G(x, y)$ et $B(x, y)$ correspondent aux intensités de rouge, vert et bleu à la position (x, y) du pixel dans l'image.

Une image peut n'avoir qu'une seule teinte. Ainsi, un seul paramètre, disons N, est suffisant pour représenter la couleur de l'image. Dans ce cas, une image est appelée une *image en niveau de gris* et est représentée par un ensemble de vecteurs à trois composantes :

$$I := \{(x, y, N(x, y)) \ : \ 0 \leq x \leq l, \ 0 \leq y \leq h, \quad 0 \leq N \leq 255\}$$

Il existe plusieurs représentations de l'espace des couleurs dans une image. Nous avons vu la représentation RGB. Nous nous concentrons sur deux autres représentations des couleurs d'une image : les composantes YUV et YIQ.

La représentation des couleurs d'une images selon les composante YUV est utilisée pour la diffusion des images digitales sur les télévisions européennes [Mic93] (les fameux câble à trois composantes). La composante Y est une somme pondérée des composantes R, G et B et c'est pour cela qu'elle est appelée la luminosité. La composante U est la différence entre la luminosité Y et la composante rouge R de l'image. La composante V est la différence entre la luminosité Y et la composante bleu B de l'image. Afin de passer du système RGB au système YUV, on exécute la

1. Une suite (x_n) est de Cauchy si et seulement si $\forall \varepsilon > 0$, il existe un entier $N(\varepsilon) > 0$ tel que si $n, m \geq N(\varepsilon)$, alors $d(x_n, x_m) < \varepsilon$. Autrement dit, les éléments de la suite sont aussi proches que l'on veut de l'un et de l'autre.

transformation suivante :

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0,299 & 0,587 & 0,114 \\ -0,147 & -0,289 & 0,436 \\ 0,615 & -0,515 & -0,100 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}.$$

L'opération inverse s'effectue en calculant l'inverse de la matrice de passage de RGB à YUV ci-dessus. On trouve le lien inverse suivant :

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1,000 & 0,000 & 1,140 \\ 1,000 & -0,395 & -0,581 \\ 1,000 & 2,032 & 0,000 \end{pmatrix} \begin{pmatrix} Y \\ U \\ V \end{pmatrix}.$$

La deuxième alternative pour représenter une image, soit les composantes YIQ est utilisée pour diffuser les images sur les téléviseurs américains [Mic93]. La composante Y est aussi appelée la luminosité. Par contre, les composantes I et Q sont respectivement appelées la *teinte* et la *saturation*. On nomme aussi les composantes I et Q par *chrominance*. Le passage du système RGB au système YIQ se fait comme ceci :

$$\begin{pmatrix} Y \\ I \\ Q \end{pmatrix} = \begin{pmatrix} 0,299 & 0,587 & 0,114 \\ 0,596 & -0,274 & -0,322 \\ 0,211 & -0,523 & 0,312 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}.$$

L'opération inverse s'effectue en calculant l'inverse de la matrice de passage de RGB à YIQ ci-dessus. On établit le lien inverse suivant :

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1,003 & 0,956 & 0,621 \\ 0,997 & -0,272 & -0,647 \\ 1,008 & -1,106 & 1,703 \end{pmatrix} \begin{pmatrix} Y \\ I \\ Q \end{pmatrix}.$$

La représentation YIQ est utilisée dans l'algorithme de compression fractale (voir section 3.2) puisqu'elle permet de réduire l'espace mémoire pour l'enregistrement de chacune des composantes. En effet, on peut réduire les composantes I et Q à un espace de valeurs plus petit puisque la variance des valeurs associée à la teinte et la saturation sont plus petites que celle de la luminosité [Fis95 ; Mic93]. Aussi, les composantes YIQ sont moins corrélées entre elles que les composantes RGB le sont entre elles. Puis, l'œil humain est moins sensible à la teinte et la saturation, ce qui nous permet de manipuler les composantes I et Q sans trop introduire de distorsions colorimétriques dans l'image qui soient perceptibles par l'homme [AHZ14 ; Ah15].

2.6 Comparaison de deux images

Afin de vérifier si la compression d'une image est bonne ou non, une métrique de bruit est calculée à partir de l'image originale et de l'image compressée. Cette métrique, appelée PSNR pour *Peak Signal-to-Noise Ratio* utilise la métrique MSE (*Mean Squared Error*) pour calculer le bruit d'une image en décibels [Mat15]. Plus sa valeur est élevée, plus l'image compressée est fidèle à l'image originale et donc plus sa qualité est élevée. Voici donc la formule requise pour calculer l'erreur MSE entre deux images :

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$

où I est l'image originale, K est l'image compressée, m est la largeur de l'image et n est la hauteur de celle-ci. Aussi, $I(i,j)$ et $K(i,j)$ représentent la valeur de la composante au pixel (i,j) des images I et K respectivement pour laquelle on calcul la MSE. Ensuite, la métrique PSNR est calculée avec la formule suivante :

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_i^2}{MSE} \right)$$

où MAX_i est la fluctuation maximale possible pour la composante dont le PSNR est calculé. Par exemple, si on calcule le PSNR pour la composante Y d'une image, MAX_i sera égal à 255, étant donné que la composante Y se situe dans l'intervalle $[0, 255]$. Enfin, étant donné que le PSNR est calculé pour une composante de l'image, le PSNR d'une image en couleurs sera la somme du PSNR de chacune des composantes de l'image divisé par le nombre de composantes de l'image, soit 3.

Avec ces outils mathématiques et ces notations, nous sommes prêts à aborder la méthode de la compression fractale.

Chapitre 3

La compression fractale

3.1 Les systèmes de fonctions itérées

La compression fractale se base sur une notion importante : les systèmes de fonctions itérées. Un système de fonctions itérées (IFS) est construit à l'aide d'un certain nombre de contractions et de leur facteur de contraction s. Explicitement, un IFS est :

$$\{\mathbb{R}^n;\, w_i,\, s,\, i = 1, 2, \ldots, m\}$$

où n est la dimension des objets considérés, m le nombre de contractions et s est le facteur de contraction global du système donné par :

$$s := \max\{s_i \,:\, i = 1, 2, \ldots, m\}$$

et chaque s_i est le facteur de contraction de la contraction w_i. Dans notre cas, on considère l'espace $\mathbb{R}^5$. Cet espace couvre l'ensemble des images I définies à la section 2.5. À l'aide des contractions de l'IFS, on définit un opérateur $W : \mathcal{H}(\mathbb{R}^5) \to \mathcal{H}(\mathbb{R}^5)$ comme étant la réunion des ensembles $w_i(I)$:

$$W(I) := \bigcup_{i=1}^{m} w_i(I)$$

où

$$w_i(I) := \{w(x) \,:\, x \in I\}$$

est l'ensemble image de chaque contraction.

Regardons un exemple simple. Il s'agit du triangle de Sierpinski. Cette fractale est codée à l'aide de trois contractions w_1, w_2 et w_3 définies comme ceci :

$$w_1(x,y) := \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{h}{2} \end{pmatrix} \quad , \quad w_2(x,y) := \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{l}{4} \\ 0 \end{pmatrix} \quad \text{et}$$

$$w_3(x,y) := \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{l}{2} \\ \frac{h}{2} \end{pmatrix}$$

où h est la hauteur de l'image et l est la largeur de l'image. La première contraction w_1 a le rôle de déplacer l'image au milieu et de réduire ses dimensions de moitié. Les deux autres transformations w_1 et w_2 amènent respectivement l'image aux coins inférieur gauche et inférieur droit. Elles réduisent aussi la taille de l'image d'un facteur 2.

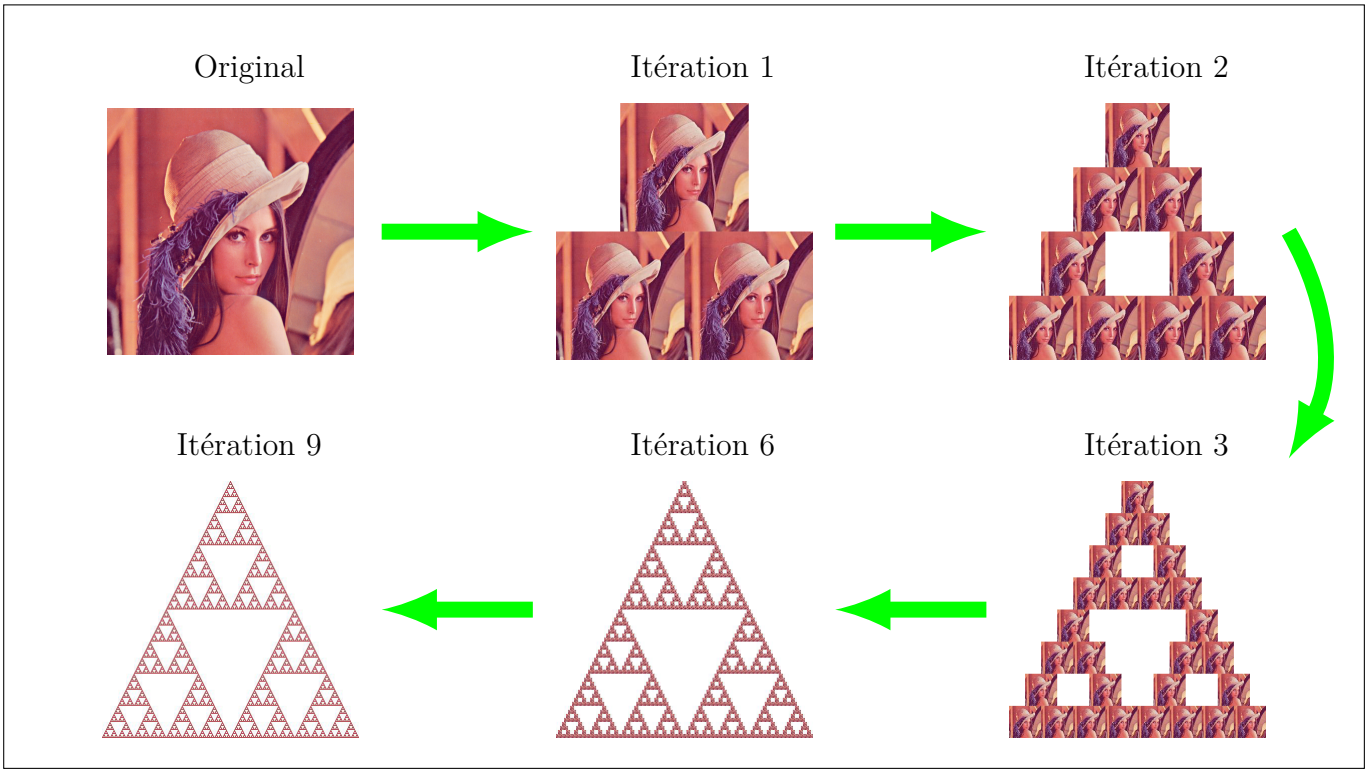


FIGURE 3.1 – Le triangle de Sierpinski

Dans la méthode de compression fractale, nous avons besoin de définir un type de contraction sur les images (ou un sous-ensemble de pixels de l'image). Sur chacune des composantes représentant les couleurs d'une image, on pose la fonction affine suivante :

$$w(x,y,u) := \begin{pmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & s_u \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ u \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \\ o_u \end{pmatrix}$$

où x_0 et y_0 est la position finale de l'image (ou d'un sous-ensemble de pixels de l'image) et $u = Y$, I ou Q. Pour que cette application w soit une contraction, il faut seulement que

$$ad - bc < 1$$

puisque c'est seulement la taille de l'image qui doit être réduite. En fait, le déterminant de la sous-matrice

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

doit être inférieur à 1. De plus, dans notre application informatique, huit matrices particulières qui agissent sur la géométrie de l'image sont considérées : quatre pour chaque symétrie (horizontale, verticale et les diagonales) et quatre pour les rotations (identité, 90, 180 et 270 degrés).

3.2 La méthodologie

La compression fractale est une méthode de compression d'images avec perte d'information qui s'exécute en cinq étapes. Celles-ci sont présentées ici et constituent le cœur de l'algorithme qui a été implémenté dans le programme. Ce programme est disponible dans le dossier de notre projet. Aussi, les étapes présentées ci-dessous sont tirées principalement de [Fis95] et [Ah15].

3.2.1 Première étape : de RGB à YIQ

Avant d'opérer sur l'image à compresser, la représentation des couleurs doit être changée. Le modèle de représentation des couleurs basé sur les composantes RGB est transposée vers le modèle basé sur les composantes YIQ. Les matrices de changements de base sont présentées à la section 2.5. Donc, dans le code du programme, ceci se fait en définissant une nouvelle image de la même taille que l'image initiale et en affectant à chaque pixel de l'image, les valeurs des composantes Y, I et Q.

3.2.2 Deuxième étape : Partitions

On partitionne l'image en blocs disjoints (qui ne se croisent pas) de pixels. Ces blocs sont appelés les « range blocks » et ils sont notés R_i. Dans notre programme, ces blocs sont constitués

de 64 pixels, soit un bloc de 8×8 pixels. À titre d'illustration, la figure suivante présente des « range blocks » de taille 4×4.

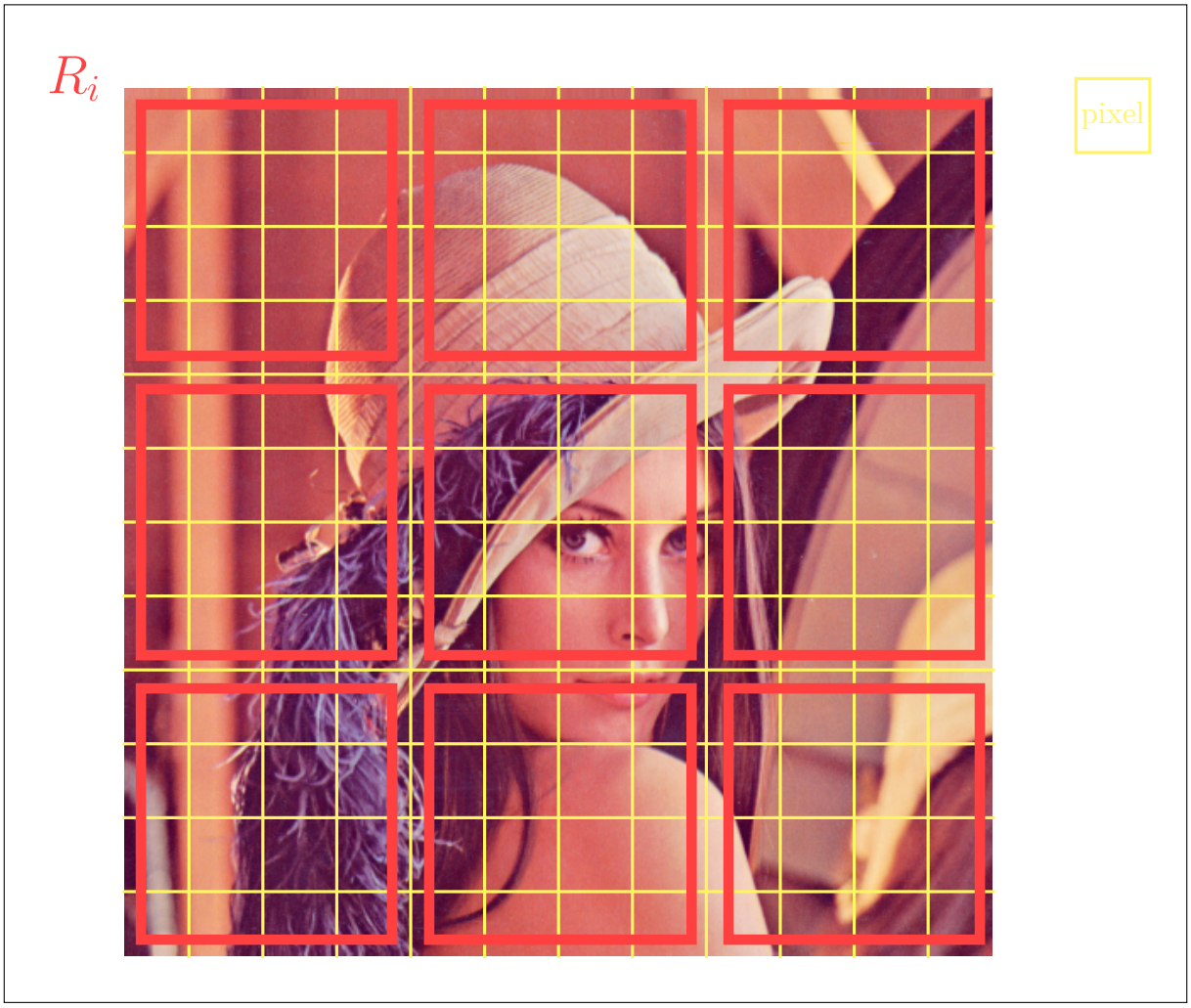


FIGURE 3.2 – Illustration des « range blocks »

Précisément, dans notre programme, des images de taille 64 par 64, 128 par 128, 256 par 256 et 1920 par 960 sont utilisées pour les tests. La quantité de R_i est donnée par le rapport du nombre de pixels total de l'image sur le nombre de pixels dans un bloc R_i. Les variables pour ces données sont rXCount, rYCount et totalRCount.

3.2.3 Troisième étape : Recherche de la meilleure région et de la contraction

Ensuite, on fixe un $R_i =: R$ et on considère une autre famille de blocs. Ces blocs sont appelés les « domain blocks » et ils sont notés D_k où k est l'indice du bloc dans l'ensemble des « domain blocks ». Ces blocs font deux fois la taille du bloc R_i. En général, ces blocs sont de taille 16×16

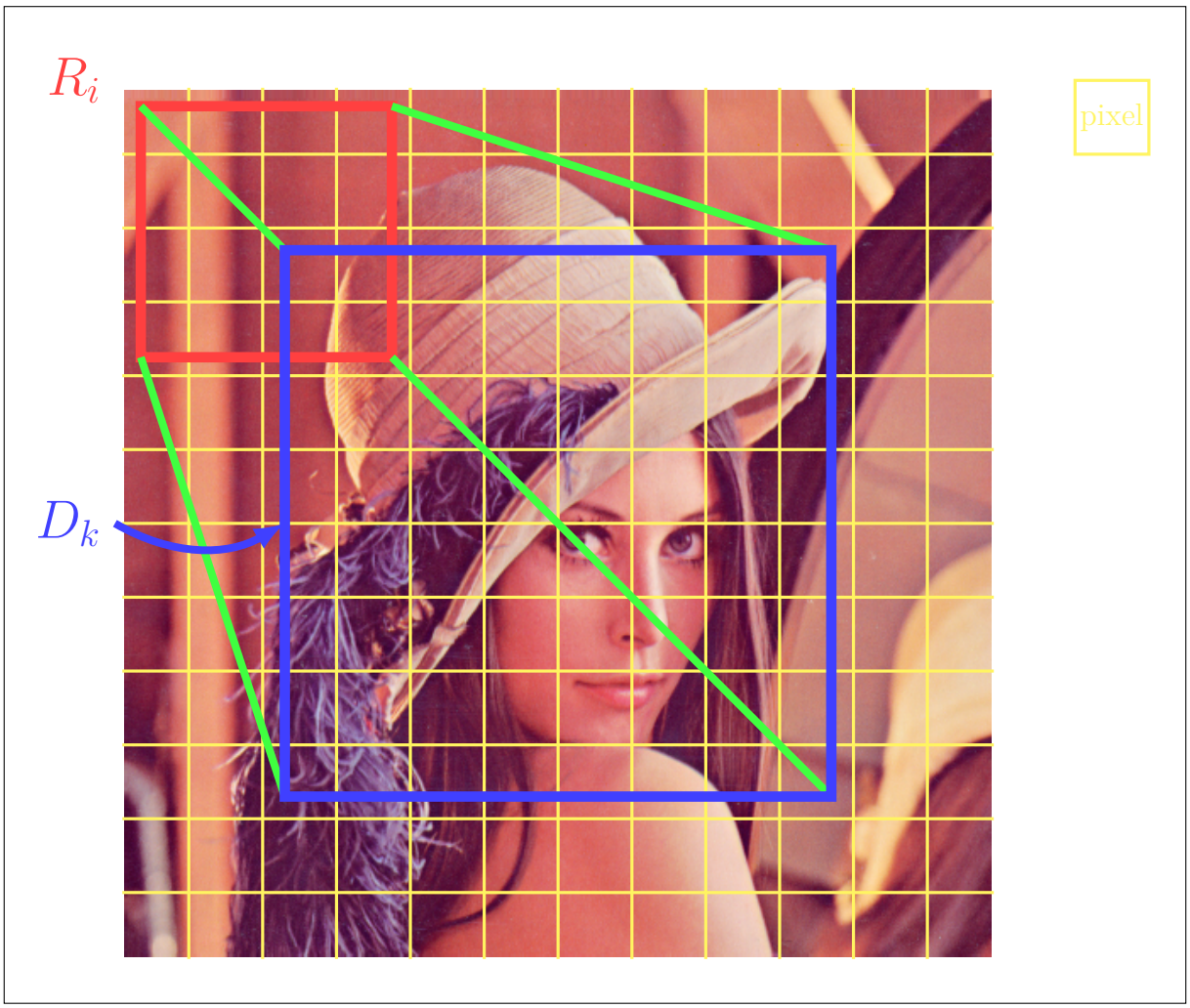


FIGURE 3.3 – Illustration du « Domain block »

puisque l'on choisi une taille de 8×8 pour les « range blocks ». Par contre, contrairement aux « range blocks », les D_k ne sont pas disjoints. Il y en donc beaucoup plus que les R_i. La figure 3.3 illustre le R (de taille 4×4) et un des « domain block » (de taille 8×8) pour l'image de Lenna.

Dans l'application informatique, on utilise trois tableaux pour chaque composantes Y, I et Q. Ces tableaux enregistrent les valeurs des composantes appartenant aux « domain blocks ». Pour la suite, posons l'ensemble des « domain blocks » par D.

Lorsque la famille des « domain blocks » est construite, on recherche dans l'ensemble D le bloc qui ressemble au R_i. Afin de réaliser cette tâche, on utilise la méthode des moindres carrées. Cependant, les blocs D_k font deux fois la taille du R sélectionné. Afin d'appliquer la méthode des moindres carrées, on réduit la taille des blocs de l'ensemble D en effectuant la moyenne des valeurs Y, I et Q pour des blocs de taille 2×2 (voir la figure 3.4). Le nouveau bloc D_k ainsi obtenu est noté $\overline{D_k}$.

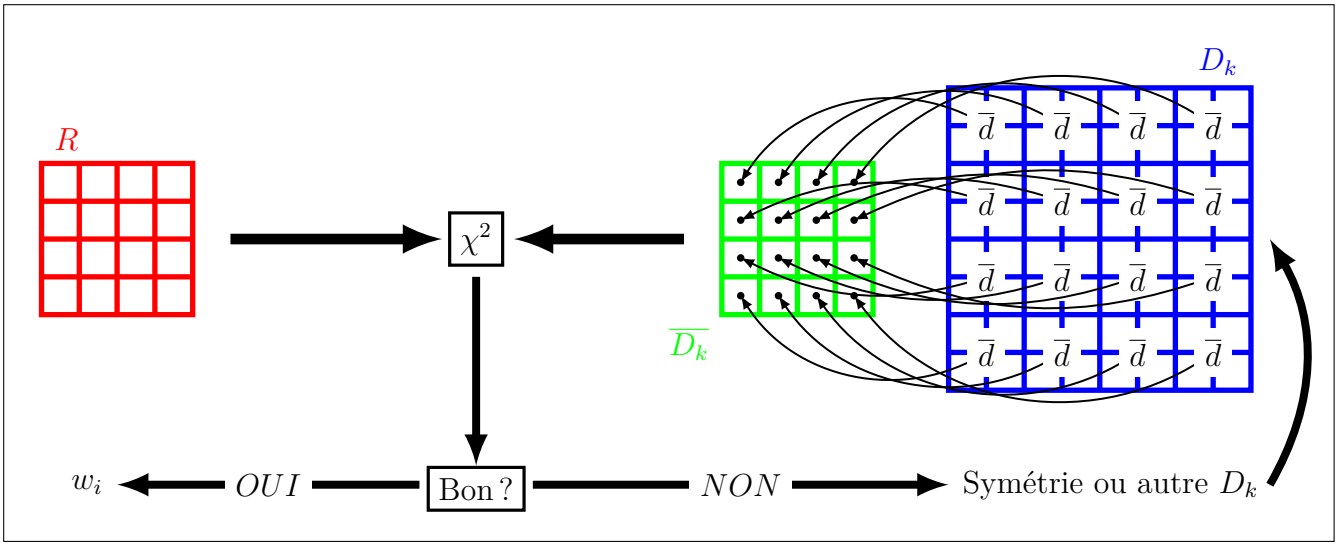


FIGURE 3.4 – Schéma de la procédure à faire pour chaque D_k dans l'ensemble D en appliquant la méthode des moindres carrées

Après cet ajustement des dimensions des blocs D_k, on peut calculer les coefficients s_Y, s_I, s_Q, o_Y, o_I et o_Q à l'aide des formules fournies à la section 2.2.

Les valeurs des autres coefficients de la matrice associées aux transformations affines (voir section 2.5) sont différentes selon la transformation de symétrie et de rotation effectuée. Puis, les coefficient x_0 et y_0 correspondent à l'opération de translation à effectuer pour chacun des blocs $\overline{D_k}$ réduits à la taille des blocs R_i afin qu'ils soient parfaitement alignés après la transformation sur le bloc D_k.

Le calcul des coefficients de la matrice associée à la transformation affine w_i sur une image est exécuté pour chaque D_k. Afin de déterminer quelle est la meilleure transformation, une erreur est calculée à partir de la transformation résultante des D_k et de l'image R_i. Cette erreur commise est donnée par la racine carrée de la quantité suivante :

$$\chi^2 := \sum_{j=1}^{n} (s_u \cdot a_j + o_u - b_j)^2.$$

où $u = Y$, I ou Q, b_j est l'intensité du pixel j dans le bloc R et a_j est l'intensité du pixel j dans le bloc $\overline{D_k}$. L'entier n est le nombre total de pixels dans le bloc R. Il est donc égal à 16 s'il est de dimension 4×4, 64 s'il est de dimension 8×8, etc. De plus, pour chaque D_k, on doit appliquer une symétrie ou une rotation sur celui-ci parmi les huit suivantes :

$$S_0 := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad S_1 := \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}, \quad S_2 := \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad S_3 := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$
$$S_4 := \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}, \quad S_5 := \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}, \quad S_6 := \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad \text{et} \quad S_7 := \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

et refaire la procédure. La figure 3.4 illustre ce dernier aspect de l'algorithme de la compression fractale. Lorsque les meilleures coefficients, au sens de l'estimation des moindres carrées, sont trouvés, la transformation affine w_i est construite et la symétrie est gardée en mémoire.

Cette procédure est répétée pour chaque R_i appartenant à notre ensemble de « range block ».

3.2.4 Quatrième étape : Enregistrer toutes les contractions

Après avoir généré toutes les contractions, on doit les enregistrer. Cette enregistrement se fait avec un minimum de bits. Afin d'atteindre une utilisation minimum de mémoire, on enregistre seulement les nombres de la contraction et non la matrice et le vecteur de translation entièrement. Ainsi, on a seulement 5 nombres pour chacune des composantes Y, I et Q (s, o, D_x, D_y et i, où D_x et D_y sont les positions en x et en y respectivement du D_k retenu et où i est l'index entre 0 et 7 inclusivement de la symétrie retenue) à considérer et à enregistrer dans le fichier pour chaque contraction de chaque composante.

Ensuite, toutes ces données sont compressées dans un fichier qui est, éventuellement, nommé .FCCI.

3.2.5 Cinquième étape : Décompression

La dernière étape consiste à utiliser une image de décodage et à appliquer chaque contraction à l'image. C'est la même méthode qui est utilisée dans notre exemple du triangle de Sierpinski.

Comme l'image a été enregistrée dans le système YIQ, on effectue la transformation inverse présentée à la section 2.5 afin de retrouver les composantes du système RGB et d'afficher l'image correctement sur un plus grand nombre de moniteurs.

Après la décompression, les intersections des carrés R_i dans l'image sont parfois visibles [Fis95]. Pour remédier à ce problème, une étape d'adoucissement peut être ajoutée à la suite de la décompression afin de lisser les intersections des différents carrés R_i. Cet adoucissement fait une moyenne pondérée de chacune des composantes des 2 pixels présents sur les bords des carrés R_i afin que l'adoucissement soit léger mais améliore la qualité de l'image [Fis95].

3.2.6 Justification de la méthode

Un théorème d'Analyse nous permet d'affirmer que la méthode fonctionne. En effet, ce théorème porte sur l'existence et l'unicité du point fixe d'une contraction sur un espace métrique complet. Il s'énonce comme suit :

Théorème (**du point fixe de Banach**[Mic93]). *Soit* (X, d) *un espace métrique et* $f : X \to X$ *une contraction de facteur* s. *Alors,* f *possède un unique point fixe* $\mathbf{x} \in X$, *c'est-à -dire qu'il existe un unique élément* $\mathbf{x}$ *dans* X *tel que* $f(\mathbf{x}) = \mathbf{x}$. *De plus, la suite des itérées* $(f^n(\mathbf{y}))$ *converge vers le point fixe* $\mathbf{x}$, *c'est-à -dire que*

$$\lim_{n\to\infty} f^n(\mathbf{y}) = \mathbf{x}$$

quelque soit le point de départ $\mathbf{y}$.

Dans notre cas, on considère l'espace des fractales $(\mathcal{H}(\mathbb{R}^3), h)$. Cet espace est complet et donc toutes les contractions définies sur cet espace (et ses sous-espaces fermés [1] comme les blocs que l'on considère dans la deuxième étape de la méthodologie) possèdent un unique point fixe. On défini l'opérateur W comme à la section 3.1 et il s'agit d'une contraction puisque chaque w_i a un facteur de contraction de $\frac{1}{2}$. Donc, le théorème s'applique et nous permet d'affirmer que la décompression sera toujours possible.

Le point fixe est appelé l'**attracteur** de l'IFS. Ceci provient du fait que la suite des itérées avant f^n converge vers le point fixe ; elles sont attirées vers ce dernier.

1. Un ensemble est fermé s'il contient tous ses points limites.

Chapitre 4

Résultats

La machine utilisée pour compresser et décompresser les images a un processeur i7-4770 (4 coeurs cadencés à 3.4 GHz avec technologie *Hyperthread*) ainsi que 16 Go de mémoire vive RAM. L'ordinateur exécute une version 64 bits de Windows 10 Professionnel. Ces spécifications sont importantes, afin de comprendre que le temps de compression avec la méthodologie fractale est très longue.

Afin de tester notre algorithme et de vérifier la théorie mathématique, nous avons effectué des tests préliminaires avec des images en niveaux de gris. Par contre, les résultats présentés ici portent sur la compression d'images en couleurs. Voici donc les résultats obtenus lors de la compression et de la décompression fractale et quelques illustrations de la décompression en niveaux de gris et en couleurs.

Table 4.1 – Taille des fichiers

Image	R	Taille FCCI	Taille JPEG	Taille PNG	Ratio
L-64x64	8	650	3255	9030	13.89 :1
L-64x64	4	2594	3255	9030	3.48 :1
L-64x64	2	10370	3255	9030	0.87 :1
L-128x128	8	2786	8994	33101	11.88 :1
L-128x128	4	11138	8994	33101	2.97 :1
L-128x128	2	44546	8994	33101	0.74 :1
L-256x256	8	11906	27212	125271	10.52 :1
L-256x256	4	47618	27212	125271	2.63 :1
L-256x256	2	190466	27212	125271	0.66 :1
P-1920x960	8	399602	548790	7385829	18.48 :1

Le tableau 4.1 présente la taille des fichiers obtenus pour différentes images et différentes tailles de R_i. Dans ce tableau, les tailles des fichiers sont toutes en octets. La colonne R définit la taille

TABLE 4.2 – Temps d'encodage et fidélité à l'image originale

Image	R	Encodage	Déc. adouci	Déc. brut	PSNR Adouci	PSNR Brut
L-64x64	8	272 ms	9	6	8.7679	8.59116
L-64x64	4	506 ms	12	6	12.86739	15.29051
L-64x64	2	1420 ms	21	6	12.37583	NaN
L-128x128	8	5.67 s	39	24	12.64667	12.73227
L-128x128	4	9.61 s	55	24	15.40411	18.32838
L-128x128	2	24.87 s	81	23	14.37288	NaN
L-256x256	8	2m 17.2s	245	133	14.86215	15.2005
L-256x256	4	3m 17.62s	277	139	17.46362	20.96746
L-256x256	2	7m 18.73s	425	135	16.34114	NaN
P-1920x960	8	18h 43m 28.59s	7440	5027	15.50346	15.48109

des carrés R_i. La taille du fichier JPEG est obtenue en exportant le fichier PNG à partir du logiciel *Gimp* et en conservant une qualité de 90%. Le ratio de compression est calculé à partir de la taille du fichier compressé par la méthodologie fractale et du fichier PNG.

Pour ce qui est du tableau 4.2, il présente le temps d'encodage et de décodage avec la compression fractale, ainsi que la fidélité de l'image compressée à l'image originale. La colonne Déc. adouci présente le temps de décompression (en millisecondes) avec un adoucissement fait sur l'image, tandis que la colonne Déc. brut présente le même temps mais sans l'étape d'adoucissement. Le PSNR de chaque image est calculé avec l'image décompressée et l'image PNG originale. Lorsque celui-ci est égal à NaN, cela signifie que l'erreur MSE calculée est de 0 pour cette image et qu'elle est donc (presque) identique à l'image originale.

La figure 4.1 illustre la décompression de l'image de Lenna de taille 256 par 256 pixels en noir et blanc à partir d'une image en noir et blanc simple. La figure 4.2 illustre la décompression de la même image de Lenna, mais en utilisant une autre image de départ.

Les figures 4.3 et 4.4 illustrent les rendus de la décompression de l'image compressée. La taille des images est respectivement de 64 par 64 pixels et 128 et 128 pixels. La différence d'images entre l'image originale au format PNG et l'image décompressée est aussi présentée pour différents formats de « range blocks » R.

Les figures 4.5, 4.6 et 4.5 présentent les rendus de la décompression d'une image de format 256 par 256 pixels pour des choix de tailles des blocs R égaux à 8, 4 et 2. La différence d'images entre l'image originale au format PNG et l'image décompressée est aussi présentée.

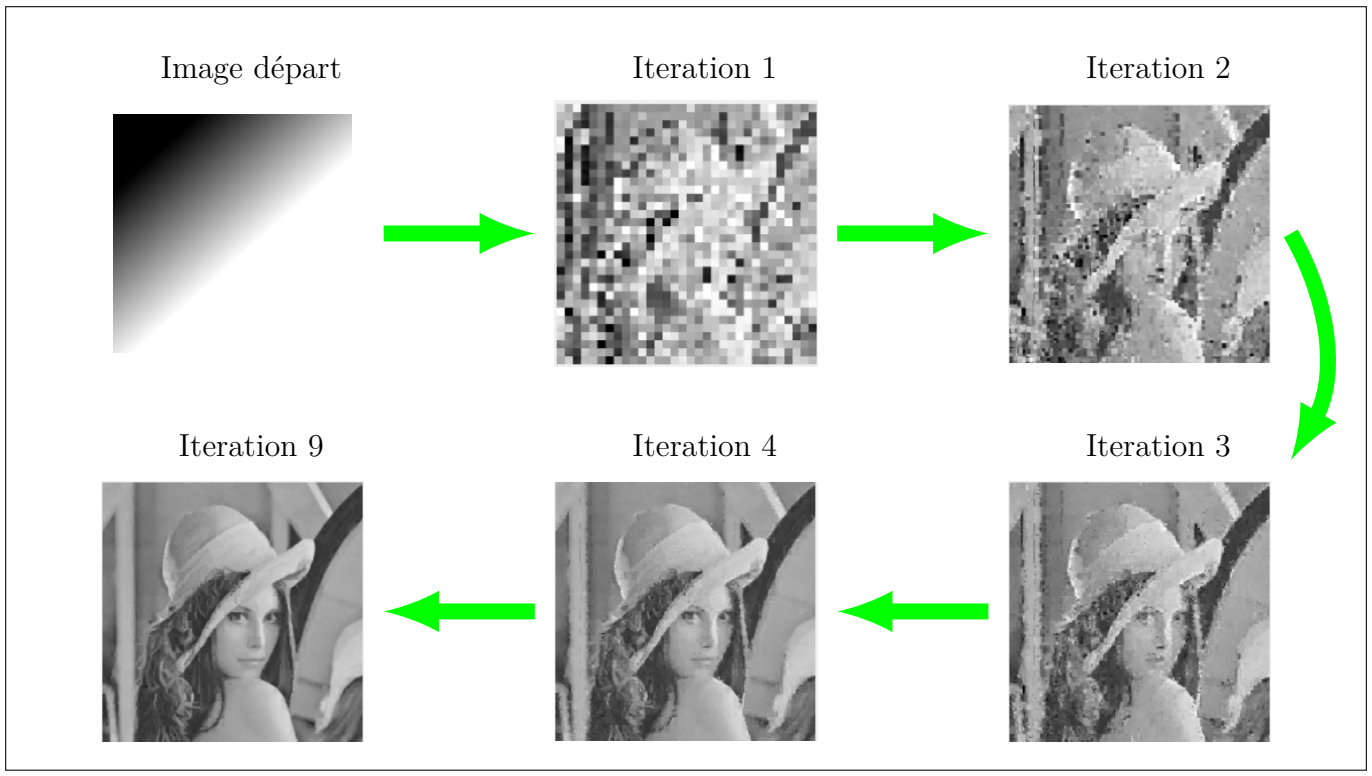


FIGURE 4.1 – Décompression de l'image compressée par la méthode de compression fractale

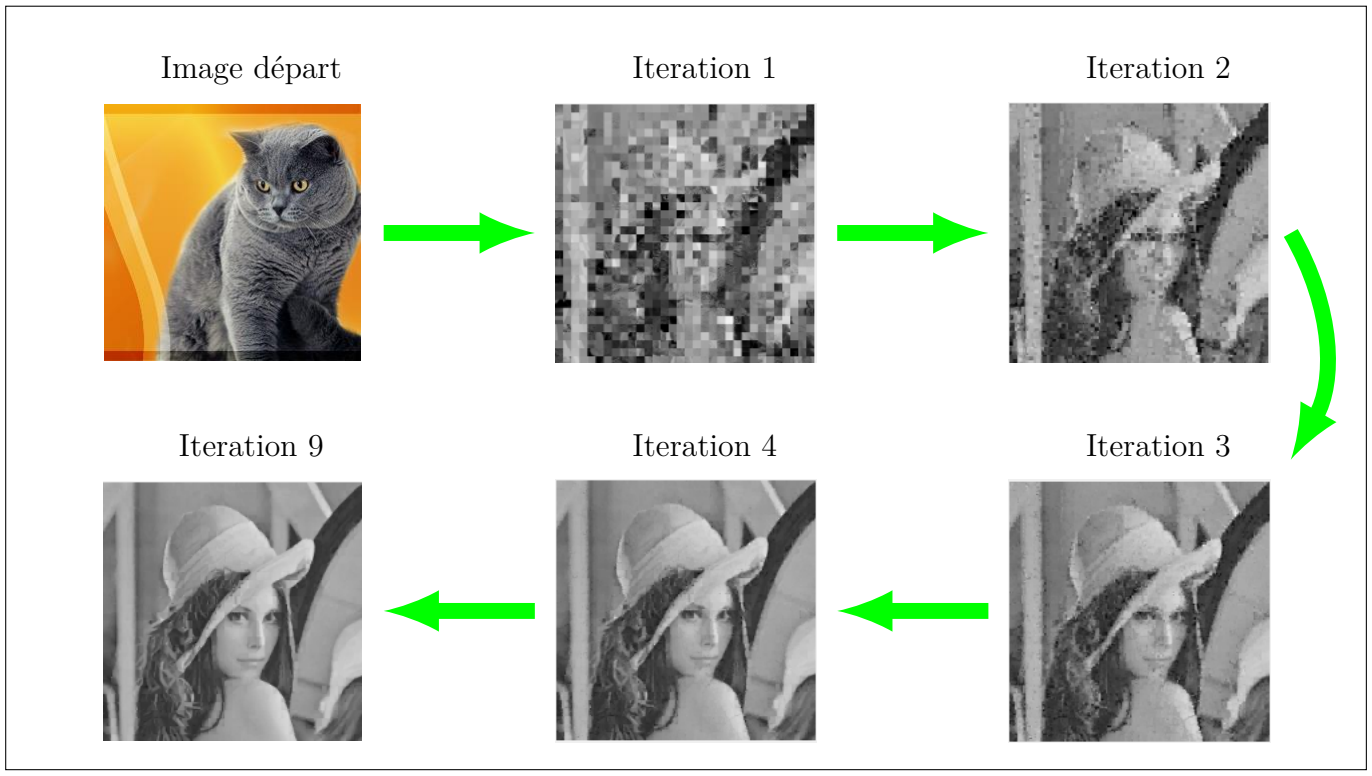
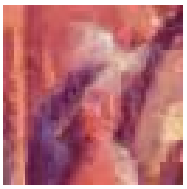


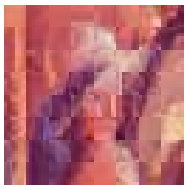
FIGURE 4.2 – Illustration de l'invariance du choix de l'image de départ sur le résultat

(a) Lenna originale

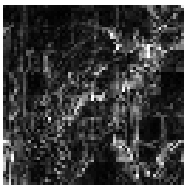


(b) Adoucie et $R = 8$

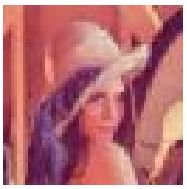
(c) Différence d'images



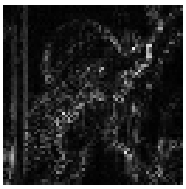
(d) Brut et $R = 8$



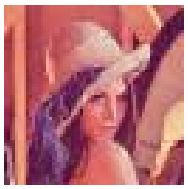
(e) Différence d'images



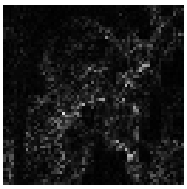
(f) Adoucie et $R = 4$



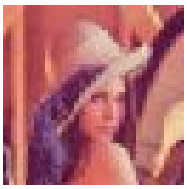
(g) Différence d'images



(h) Brut et $R = 4$

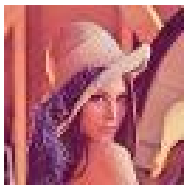


(i) Différence d'images

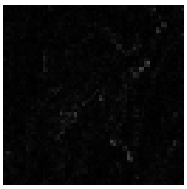


(j) Adoucie et $R = 2$

(k) Différence d'images



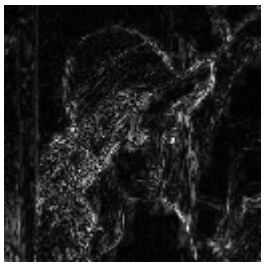
(l) Brut et $R = 2$



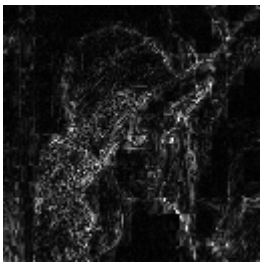
(m) Différence d'images

FIGURE 4.3 – Décompression de l'image de Lenna de taille 64×64

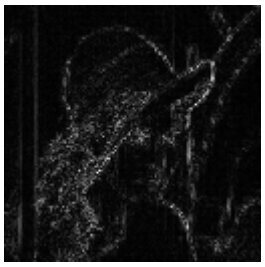
(a) Lenna originale

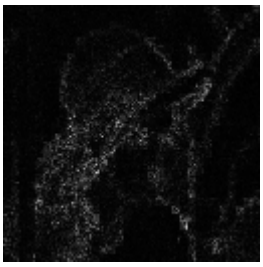
(b) Adoucie et $R = 8$ (c) Différence d'images

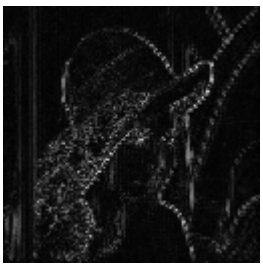
(d) Brut et $R = 8$ (e) Différence d'images

(f) Adoucie et $R = 4$ (g) Différence d'images

(h) Brut et $R = 4$ (i) Différence d'images

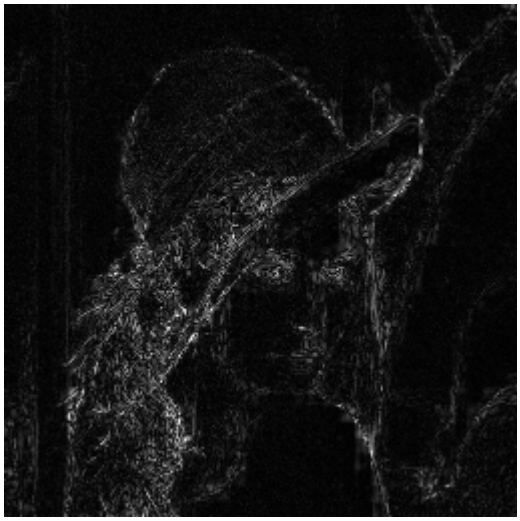
(j) Adoucie et $R = 2$ (k) Différence d'images


(l) Brut et $R = 2$ (m) Différence d'images

FIGURE 4.4 – Décompression de l'image de Lenna de taille 128×128

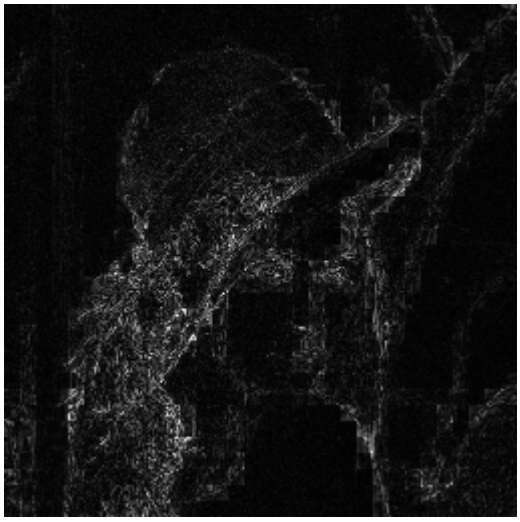
(a) Lenna originale

(b) Adoucie



(c) Différence d'images

(d) Brut



(e) Différence d'images

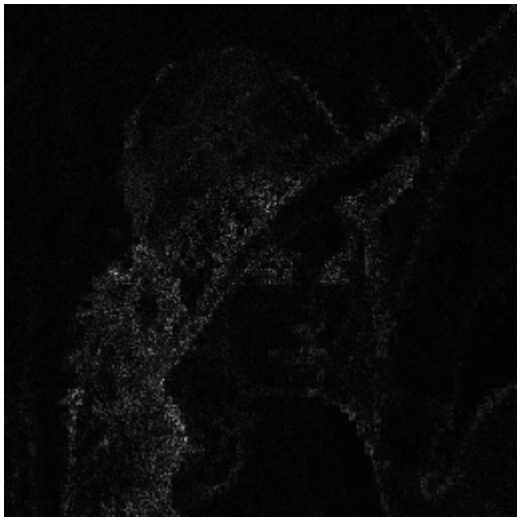
FIGURE 4.5 – Décompression de l'image de Lenna de taille 256×256 et $R = 8$

(a) Lenna originale

(b) Adoucie

(c) Différence d'images

(d) Brut



(e) Différence d'images

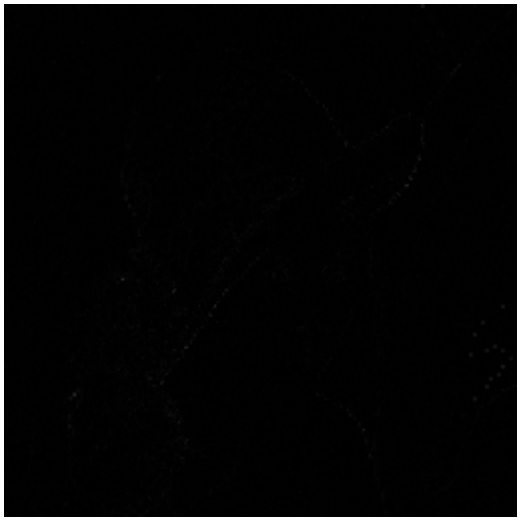
FIGURE 4.6 – Décompression de l'image de Lenna de taille 256×256 avec $R = 4$

(a) Lenna originale

(b) Adoucie

(c) Différence d'images

(d) Brut



(e) Différence d'images

FIGURE 4.7 – Décompression de l'image de Lenna de taille 256×256 avec $R = 2$

FIGURE 4.8 – Une région de l'image au format JPEG de taille 256×256

FIGURE 4.9 – Une région de l'images de format FCCI de taille 256×256 compressée avec $R = 8$

Chapitre 5

Discussion des résultats

5.1 Pour une image en niveau de gris

Les figures 4.1 et 4.2 illustrent le phénomène d'attracteur encodé dans les transformations du système de fonctions itérées associée à l'image compressée. On remarque que quelque soit l'image de départ, on retrouve, au fur et à mesure d'itérer le processus, l'image compressée. Ce phénomène provient de la propriété des systèmes de fonctions itérées. Il s'agit d'un exemple qui montre la puissance du théorème du point fixe de Banach présenté à la section 3.2.6. Il s'agit aussi d'une première remarque importante concernant l'invariance du résultat de la décompression selon l'image de départ choisie.

Puis, on remarque immédiatement que la décompression nécessite peu d'itérations afin de retrouver l'image originale. De plus, le calcul pour arriver à l'itération 9 s'effectue en moins d'une seconde. Ce qui est très rapide. C'est pour cette raison que l'application incorpore une option afin d'itérer le processus 10 fois pour obtenir l'image décompressée.

Bref, à l'aide des images en niveau de gris, on peut constater que l'algorithme de compression d'image par les fractales fonctionne bien. Nous allons maintenant centrer notre discussion sur la compression des images en couleur.

5.2 Pour une image en couleur

Tout d'abord, on remarque le même phénomène avec les images en couleurs qu'avec les images en niveaux de gris. Peu importe l'image de départ, l'image finale est la même.

Ensuite, on remarque que le temps de compression d'une image est très élevé même pour des tailles d'image qui sont de nos jours considérées comme petites. On voit que pour l'image qui est pratiquement dans un format *Full HD* (1920 par 960), le temps de compression est très long. La compression fractale est une méthode de compression très lente, spécialement notre méthode étant donné que c'est la méthode de compression fractale de base, dite de *Jacquin* [Fis95]. Ces résultats correspondent aux résultats obtenus dans les références sur le sujet.

On note également que le ratio de compression pour des R_i de taille 2 par 2 pixels n'est pas bon du tout, car la compression génère un fichier plus grand que celui de l'image initiale au format PNG. Ce phénomène peut s'expliquer par le fait suivant. D'un côté, les composantes des couleurs de l'image au format PNG nécessite au plus huit bits pour un total approximatif de $3 \cdot 8 \cdot$ TAILLE DE L'IMAGE. De l'autre côté, la compression fractale utilise un total de TAILLE DE L'IMAGE$/2$ contractions affines pour lesquelles 9 nombres sont enregistrés nécessitant un minimum de 3, 6, 6 et 8 bits respectivement pour la symétrie, les facteurs s, les translations o et les déplacements D_x et D_y. Ceci donne un total approximatif de

$$\frac{\text{TAILLE DE L'IMAGE} \cdot 3 \cdot 6 \cdot 6 \cdot 8}{2} = \text{TAILLE DE L'IMAGE} \cdot 372 > \text{TAILLE DE L'IMAGE} \cdot 24.$$

Par contre, l'image reproduite avec des R_i de cette taille est identique à l'oeil nu à l'image au format PNG lorsque celle-ci a une taille de 128 pixels et plus, d'où le PSNR élevé dans nos résultats. En effet, l'image différence entre celle compressée et l'originale montre où se cache la perte d'information (voir les figures 4.4 et 4.7). En effet, ces différences se situent davantage dans le fouleur de la jeune femme et sur les contours. Or, la compression avec des R_i de taille 4 par 4 pixels donne également de très bon résultats, autant à l'oeil nu que selon le PSNR. On obtient un taux de compression qui tourne entre 2.5 et 3.5 pour 1 avec cette taille de R_i.

Ce qui n'était pas prévu dans la littérature est le fait suivant. On remarque bien que la taille du fichier généré par la compression fractale est souvent supérieure à celui généré par la compression JPEG avec une qualité définie à 90%. De plus, la qualité de l'image JPEG est supérieure à celle de l'image générée par la compression fractale avec des carrés R_i de 4 par 4 pixels (qui elle prend plus d'espace que le JPEG). Cela s'explique par le fait que c'est l'algorithme de compression fractale de base qui a été implémenté dans notre logiciel. Cette méthode n'est pas la méthode de compression fractale la plus optimale. Si la méthode *Quadtree* avait été utilisée comme dans le livre de Fisher [Fis95], le taux de compression aurait été sans nul doute meilleur, car moins de transformations auraient été stockées dans le fichier. Cela aurait peut-être permis un meilleur temps de compression également.

Une autre remarque est que l'adoucissement de l'image lors de la décompression améliore la qualité de l'image lorsque les R_i sont de 8 par 8 pixels, mais pour des tailles de R_i plus petites,

la différence est minime. Pour des R_i de 8 par 8 pixels, malgré le fait que le PSNR est parfois meilleur sans le lissage, pour un humain, l'image semble mieux avec le lissage. Par contre, pour des R_i de taille 2 par 2 ou de 4 par 4 pixels, le lissage ne fait que flouer l'image décompressée. Ceci s'explique par le fait que les blocs R_i sont de plus petites dimensions et donc que leurs frontières sont plus rapprochées l'une de l'autre.

On remarque aussi un phénomène intéressant lors de la compression fractale. Si on compare les images carrées de Lenna de 64, 128 et 256 pixels pour une taille de R_i de 8, on remarque que l'image comporte moins d'erreur (PSNR) lorsqu'elle est plus grande. Cela s'explique par le fait que pour une image plus grande et une même taille de R_i, le nombre de D_k à analyser est plus grand et il est donc plus probable de trouver une région D_k qui correspond très bien à un R_i donné. Cela fait en sorte que l'image de taille supérieure a une erreur plus petite que l'image de taille inférieure.

Enfin, selon la littérature, une image obtenue par la méthode de décompression fractale ne doit pas subir l'effet de pixelisation lorsqu'un agrandissement est effectuée sur une région de l'image décompressée. En effet, la région ciblée n'a qu'à être calculée de nouveau à l'aide des tansformations affines enregistrées dans le fichier. Cet aspect n'a malheuresement pas été abordé en détail dans notre travail. Cependant, les figures 4.8 et 4.9 montrent que l'effet de pixellisation est moins présente dans l'image au format FCCI que dans l'image au format JPEG après l'agrandissement (sans recalcul de l'image) d'une même région. Les carrés utilisés dans l'algoritme de la compression fractale semble moins apparents que les carrés utilisés dans l'algorithme JPEG. D'ailleurs, l'image au format FCCI semble présenter moins de bruit que l'image au format JPEG.

Conclusion

Dans le cadre de notre travail, nous avons exploré en détail la méthode de base de la compression fractale, dite de *Jacquin* et nous l'avons implémenter. Afin d'accomplir cette tâche, nous avons introduit des concepts mathématiques se rapportant aux systèmes de fonctions itérées et nous avons appliqué ces outils pour compresser des images. Par contre, il existe d'autres méthodes qui améliorent les performances de la compression fractale, tant au niveau du ratio de compression que de la vitesse de compression et de sa qualité (PSNR). En effet, des développements plus poussés nous auraient permis d'obtenir de meilleurs résultats : la méthode de *quadtree*, la méthode *HV* ou la combinaison de méthodes d'analyse de Fourier (transformée de Fourier rapide [RA97]) et de la compression fractale. Ces méthodes n'ont pas été explorées dans notre travail puisqu'elles demandaient beaucoup plus d'investissement et qu'elles sont très avancées. Si la chance peut survenir, il serait intéressant de poursuivre ce travail afin d'explorer plus en profondeur les autres techniques développées afin d'améliorer la méthode de *Jacquin* pour la compression fractale. En particulier, la méthode de compression fractale *Quadtree* semble prometteuse et plusieurs documents suscitent des théories permettant un temps de compression diminué ou un taux de compression supérieur avec moins de perte de qualité.

La compression d'images avec les fractales pourrait un jour être plus utilisée, étant donné qu'elle donne de bons résultats, que la décompression des images est plus rapide qu'avec JPEG et que l'image décompressée ne subit pas l'effet de pixellisation. Bien sûr, son temps de compression élevé rend la méthode peu intéressante, mais dans des situations où la vitesse d'accès aux images est un facteur important à considérer, cette méthode pourrait être utilisée.

Bref, la compression fractale est une méthode qui a encore ces preuves à faire, mais un bon avenir lui est réservé. Elle est peut-être le futur standard que soutiendra la majorité des ordinateurs, le prochain support sur lequel les personnes du monde entier regarderont ou s'enverront leurs images personnelles ou professionnelles.

Bibliographie

Livres

[BR93] Michael Fielding Barnsley et Hawley Rising. *Fractals everywhere*. eng. 2nd ed. San Francisco, Calif. : Morgan Kaufmann, 1993, p. 531.

[Fis95] Yuval Fisher. *Fractal Image Compression Theory and Application*. New York États-Unis : Springer-Verlag, 1995, p. 341.

[Ach05] Ping-Sing Acharya, TinkuTsai. *JPEG2000 Standard for Image Compression : Concepts, Algorithms and VLSI Architectures*. N. J. : Hoboken, Wiley-Interscience, 2005, p. 274.

[Mic93] Lyman P Hurd Michael F. Barnsley. *Fractal Image Compression*. Wellesley Massachusets États-Unis : AK Peters Ltd., 1993, p. 244.

Articles

[Ah15] Eman A Al-hilo. « Comparison Fractal Color Image Compression using YIQ and YUV Color Model ». In : *IJACSA* 6.5 (2015), p. 112–116.

[AHZ14] Eman A. Al-Hilo et Rusul Zehwar. « Fractal Image Compression by YIQ Color Space ». In : *2014 International Conference on Computational Science and Computational Intelligence* (2014), p. 221–225. doi : 10.1109/CSCI.2014.45. url : http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6822112.

[GAH09] Loay E. George et Eman a. Al-Hilo. « Fractal Color Image Compression by Adaptive Zero-Mean Method ». In : *2009 International Conference on Computer Technology and Development* 1 (2009), p. 525–529. doi : 10.1109/ICCTD.2009.150. url : http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5359731.

[RA97] M. Ramkumar et G.V. Anand. « An FFT-based technique for fast fractal image compression ». In : *Elsevier, Signal Processing* 63 (1997), p. 263–268.

[SHH97] D. Saupe, R. Hamzaoui et H. Hartenstein. « Fractal Image Compression - An Introductory Overview ». In : (1997), p. 66. URL : http://dl.acm.org/citation.cfm?id=896295.

Ressources en ligne

[Mat15] MathWorks. *Compute peak signal-to-noise ratio (PSNR) between images - Simulink*. 2015. URL : http://www.mathworks.com/help/vision/ref/psnr.html.

[Pil15] Jean-François Pillou. *Le format YUV (YCrCb)*. 2015. URL : http://www.commentcamarche.net/contents/1221-le-format-yuv-ycrcb (visité le 02/01/2016).

[SAN15] Fernand LONE SANG. *La compression fractale*. 2015. URL : http://etud.insa-toulouse.fr/{~}flone{_}sa/BEmultimedia/index.php.

[SAR15] WebEvolutis SARL. *.bmp, .tiff, .gif, . jpeg, .png, ... Tout sur les formats d'image*. 2015. URL : http://www.clashinfo.com/aide-informatique/multimedia/art153-formats-image.html.

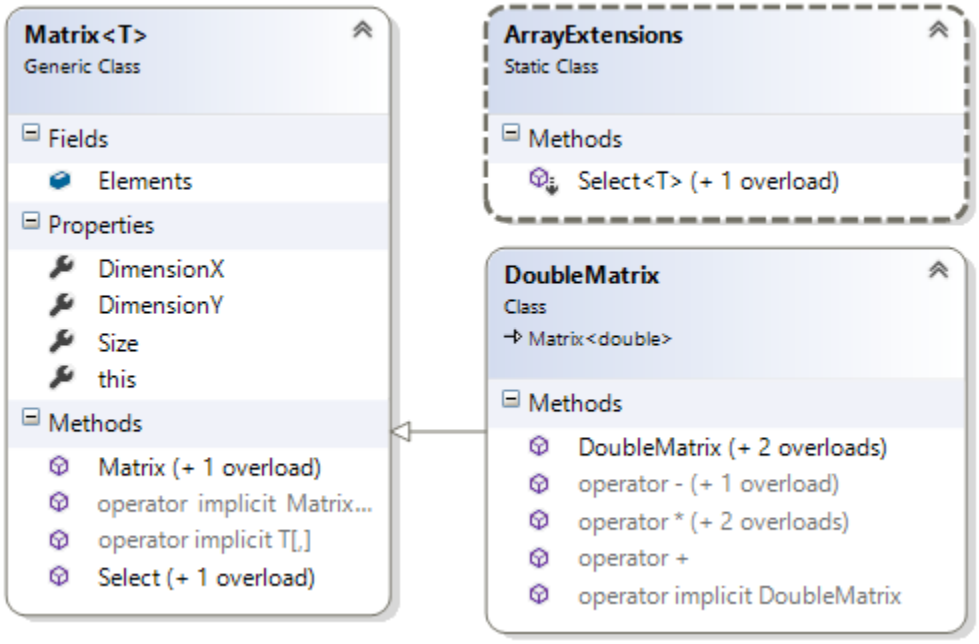
[Wik15] Wikipédia. *Compression d'image*. 2015. URL : https://fr.wikipedia.org/wiki/Compression{_}d'image.

Annexes

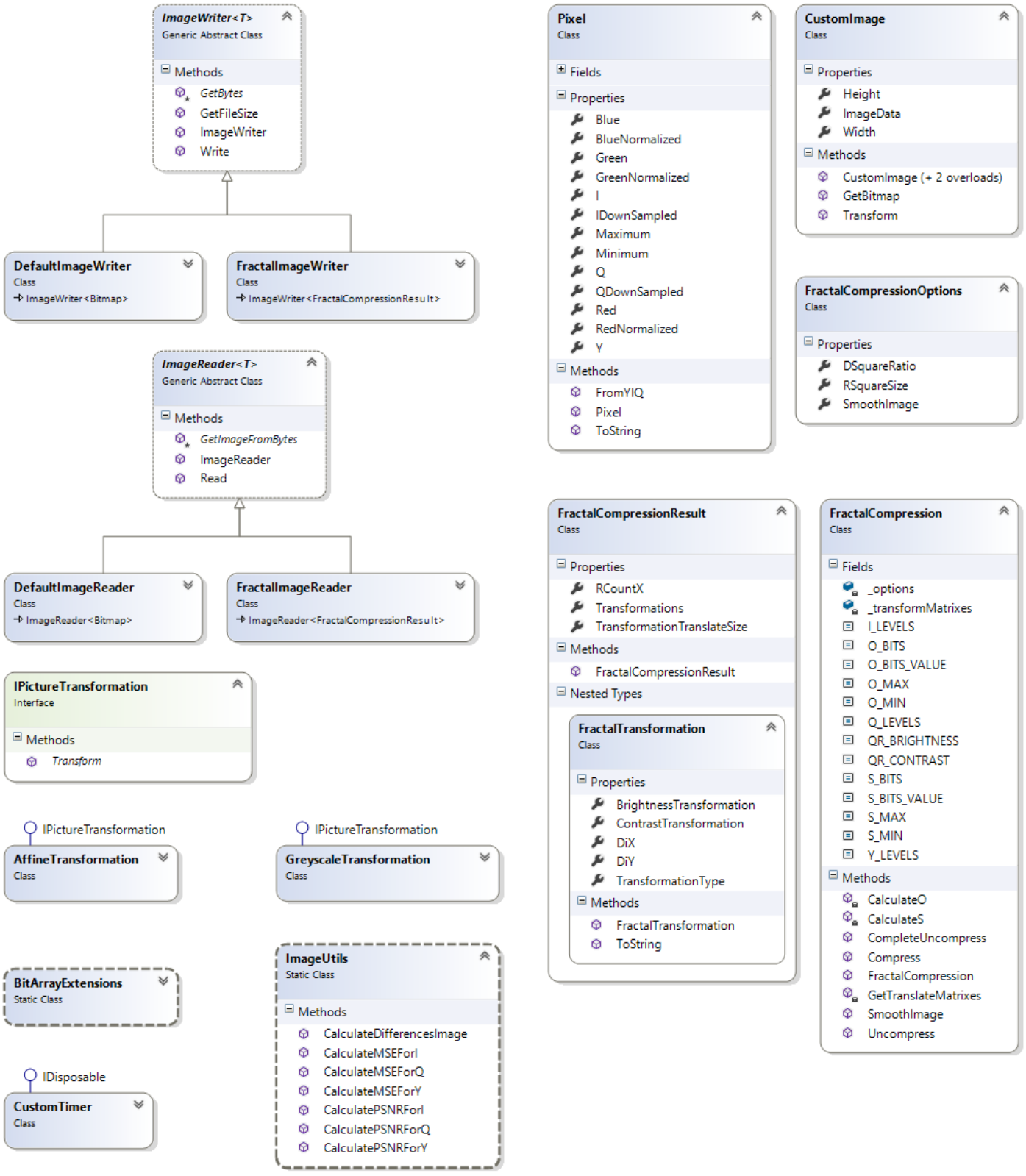
Annexe A

Diagrammes de classes

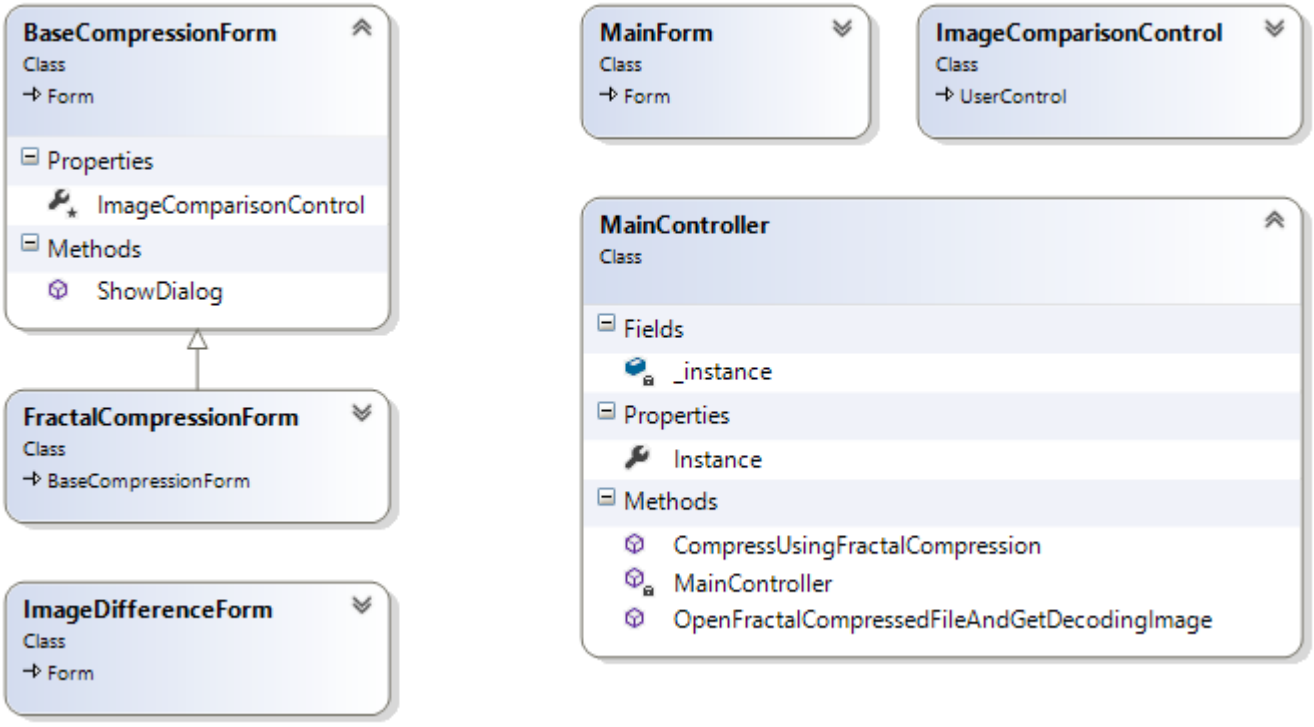
A.1 Projet MathTools



A.2 Projet ImageTools



A.3 Projet ImageCompressionTool (projet principal)



Annexe B

Description des classes importantes

Tout d'abord, il faut savoir que le logiciel est divisé en 3 projets distincts : MathTools et ImageTools sont deux projets de type librairie qui permettent la réutilisation de code pertinent à être réutilisé, puis ImageCompressionTool qui fournit l'interface graphique pour travailler avec la compression fractale.

B.1 Projet MathTools

Tout d'abord, le projet MathTools contient une classe *ArrayExtensions* contenant une méthode d'extension permettant de sélectionner et au besoin modifier les éléments d'un tableau à multiples dimensions qui est utilisée dans les 2 classes de matrices. Une classe de base de *Matrix<T>* est générique et permet donc de stocker des objets de types variés dans une matrice à 2 dimensions et fournit un accès simplifié à ceux-ci, étant donné que l'utilisation des tableaux multidimensionnels est parfois complexe en C#. La classe *DoubleMatrix* hérite de cette classe et implémente le fonctionnement d'une matrice pour des types réels (*double*). Elle permet plusieurs opérations matricielles de base, comme l'addition, la soustraction, la multiplication par un vecteur et la multiplication matricielle.

B.2 Projet ImageTools

Ensuite, le projet ImageTools utilise la librairie MathTools étant donné que plusieurs calculs sont faits avec des matrices. C'est le projet contenant le plus de code parmi les trois.

Pour commencer, cette librairie fournit une classe *ImageWriter<T>* qui permet de stocker simplement des images et obtenir leur taille sur le disque avant leur sauvegarde. Une classe *ImageReader<T>* est également fournie afin de lire les images à partir du disque. Deux implémentations concrètes de chacune de ces classes sont faites : une pour travailler avec les images de type JPEG et PNG en utilisant l'implémentation fournie par le *.NET Framework* et une pour travailler avec le format personnalisé de compression fractale.

Ensuite, une classe *Pixel* permet de représenter un pixel dans une image selon sa notation RGB (fournit également la possibilité d'obtenir les composantes YIQ du pixel). La classe CustomImage permet de simplifier le travail avec les images en permettant de transformer un objet *Bitmap* fournit par le *.NET Framework* en un tableau d'objets de type *Pixel*, ou encore de générer une image à partir d'un tableau de pixels.

Aussi, l'interface *IPictureTransformation* fournit une interface simple permettant d'effectuer un traitement sur les pixels d'une image. Une implémentation concrète de cette interface permet de transformer une image en niveaux de gris (*GreyscaleTransformation*) tandis qu'une autre permet de transformer l'image selon une transformation affine (*AffineTransformation*).

Plusieurs classes utilitaires sont également fournies dans cette librairie. La classe *BitArrayExtensions* permet de simplifier l'écriture et la lecture de bits dans un objet de type *BitArray*. Cette classe est utilisée pour l'écriture et la lecture du fichier de l'image compressée avec la méthodologie fractale. La classe *CustomTimer* permet de calculer le temps d'exécution de code, afin de calculer le temps de compression et de décompression. Ensuite, la classe *ImageUtils* permet de trouver l'image des différences entre une image A et une image B en plus de calculer les métriques *MSE* et *PSNR* pour les composantes Y, I et Q entre 2 images, afin de savoir si elles sont semblables ou non.

Enfin, la classe *FractalCompressionOptions* sert à définir les options utilisées lors de la compression fractale, soit : la taille de carrés R_i, la taille des carrés D_k et si on adoucit l'image ou non. La classe *FractalCompression* est la classe importante du logiciel, étant donné que c'est elle qui procède à la compression et à la décompression fractale. La méthode *Compress* retourne un objet de type *FractalCompressionResult*, qui contient les informations pertinentes sur la compression afin de stocker l'image sur le disque. La sauvegarde et la lecture de l'image sera faite à l'aide des classes *FractalImageWriter* et *FractalImageReader* respectivement.

B.3 Projet ImageCompressionTool

Le projet ImageCompressionTool est celui qui fournit l'interface graphique à l'utilisateur pour effectuer la compression fractale. La classe *ImageComparisonControl* définit un contrôle utilisateur utilisé pour afficher et comparer 2 images et est utilisé dans toutes les fenêtres de l'application. La classe *MainForm* définit le comportement de la fenêtre principale. La classe *FractalCompressionForm* définit le comportement de la fenêtre pour la compression et décompression fractale et celle-ci hérite de *BaseCompressionForm*, qui est une classe abstraite qui aurait été utilisée pour simplifier le développement de plusieurs algorithmes de compression différents. Seulement, seule la compression fractale a été implémentée, donc cette classe est quelque peu inutile. La classe *ImageDifferenceForm* permet d'afficher les informations sur la différence entre 2 images, ainsi que l'image des différences associée. Enfin, la classe *MainController* est un singleton permettant les interactions complexes avec les classes du logiciel.