Anne-Sophie GOBIN
Rémi DEWITTE

Production écrite pour les TPE

Les images numériques et la compression

Lycée Mongazon, TS4

Année 2001-2002

Sommaire

Rémi DEWITTE
Anne-Sophie GOBIN

Introduction

Le développement rapide des applications informatiques s'est accompagné d'un accroissement important de l'utilisation des images numériques, notamment dans le domaine des multimédias, des jeux, des transmissions satellite ou de l'imagerie médicale.

Les images numérisées, posent, par leur taille importante, de nombreux problèmes quant à leur transmission ou à leur stockage. Pour gagner aussi bien en vitesse qu'en place, il est nécessaire de "compresser" l'image.

L'utilisation d'algorithmes de compression d'image dont la complexité et la rapidité d'exécution ont grandi au cours du temps, permet de réduire la taille de mémoire occupée par une image en perdant un minimum de qualité.

Après une étude des principes généraux des images numériques nécessaires à la bonne compréhension, nous verrons quelles sont les méthodes de compression actuelles.

Rémi DEWITTE
Anne-Sophie GOBIN

I) Les images numériques

1.1) Découpage et pixel

Lors de la numérisation d'une image, l'image est découpée en une matrice de pixels (on dit aussi échantillonnage spatial). « Pixel » est l'abréviation de « picture element », c'est un élément du quadrillage de l'image numérique. Le pixel représente le plus petit point distinguable dans une image. Il est aussi le plus petit élément que peuvent manipuler les matériels et logiciels d'affichage ou d'impression, lorsqu'ils produisent un caractère ou une image. Chaque pixel possédant une teinte, c'est la juxtaposition des différents pixels qui produit une image.

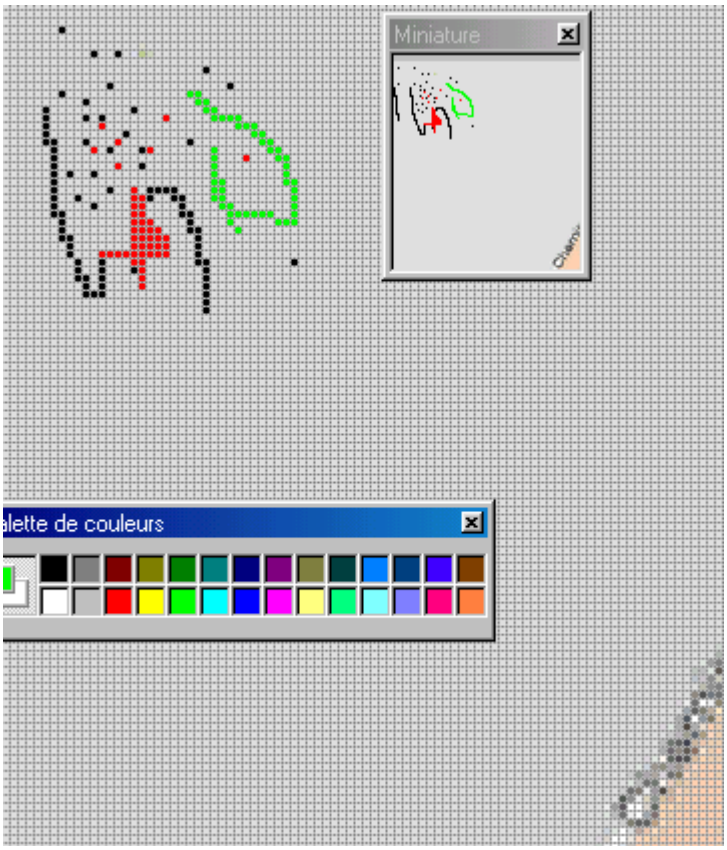


Illustration 1: Le logiciel Paint donne la possibilité de voir le quadrillage des pixels à un zoom élevé.

1.2) Résolution

Le nombre total de pixels d'une image numérique définit la résolution de l'image. Elle est exprimée en nombre de pixels par unité de longueur (souvent en ppp : pixel par pouce en français ou dpi en anglais). Cela aura une influence sur la qualité du zoom si on étire l'image.

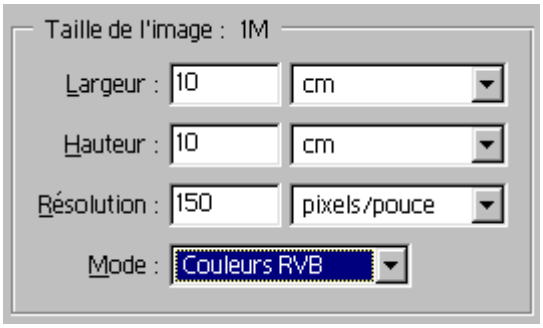


Illustration 2: Le choix de la résolution est indépendant des dimensions de l'image.

1.3) Dynamique de l'image

Les pixels possèdent une représentation binaire au sein de l'ordinateur. Le nombre de bits intervenant dans le codage d'un pixel définit le nombre maximal de teintes différentes que peut prendre le pixel. Ainsi, un pixel codé avec un bit pourra prendre une teinte parmi deux, suivant que le bit vaut 0 ou 1. Lorsque l'on parle d'images en millions de couleurs, il s'agit d'images où chaque pixel est codé sur au moins 24 bits, soit 2^{24}=16 777 216 teintes différentes.
La taille de l'image est donc définie par :

- sa résolution
- l'étendue des couleurs que peut prendre chaque pixel

Bien entendu, ces facteurs jouent plus ou moins sur la qualité de l'image. Cependant la quantité de données est très vite importante. Regardons quels sont les différents codages de la couleur…

1.4) Le codage des couleurs

Le codage des couleurs peut s'effectuer de différentes façons :

1.4.1 Noir et blanc

Un pixel prend soit la valeur noir soit la valeur blanc. C'est le codage le plus simple de la couleur qui existe. Mais il est relativement limité.

1.4.2 Le codage par synthèse additive des couleurs, le codage 24 bits

On utilise 3 composantes de couleur (rouge, vert et bleu) codées chacune sur 8 bits. L'addition des trois couleurs avec un niveau plus ou moins élevé donne une couleur. C'est comme en peinture en fin de compte.

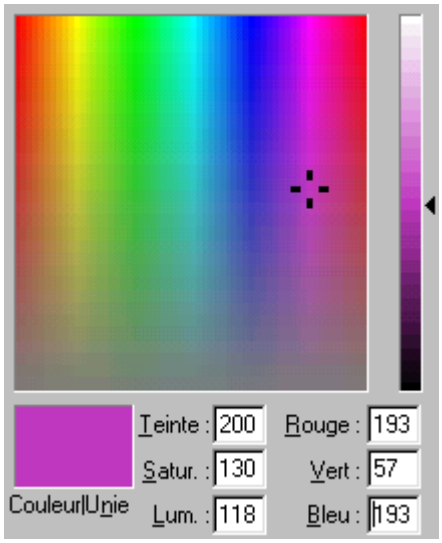


Illustration 3: Les logiciels de dessin indiquent l'indice attribué à chaque couleur au moment du choix d'une couleur particulière.

1.4.3 Le codage 8 bits

Il fonctionne en indexant 256 couleurs au départ pour l'image ($2^8 = 256$). C'est une technique pour que l'information couleur soit codée sur 1 octet (pour gagner de la place) au lieu de trois, en utilisant une palette de couleur "attachée" à l'image. A chacun de ces 256 nombres va correspondre une couleur, définie par son code RVB et stockée dans une palette avec les données de l'image (les couleurs indexées). Lors de la visualisation de l'image, la correspondance se fait entre le numéro de la couleur affecté à chaque pixel (compris entre 0 et 255) et le code couleur RVB correspondant. On divise à tous les coups la taille de l'image par trois.

Illustration 4: L'image(taille réduite).

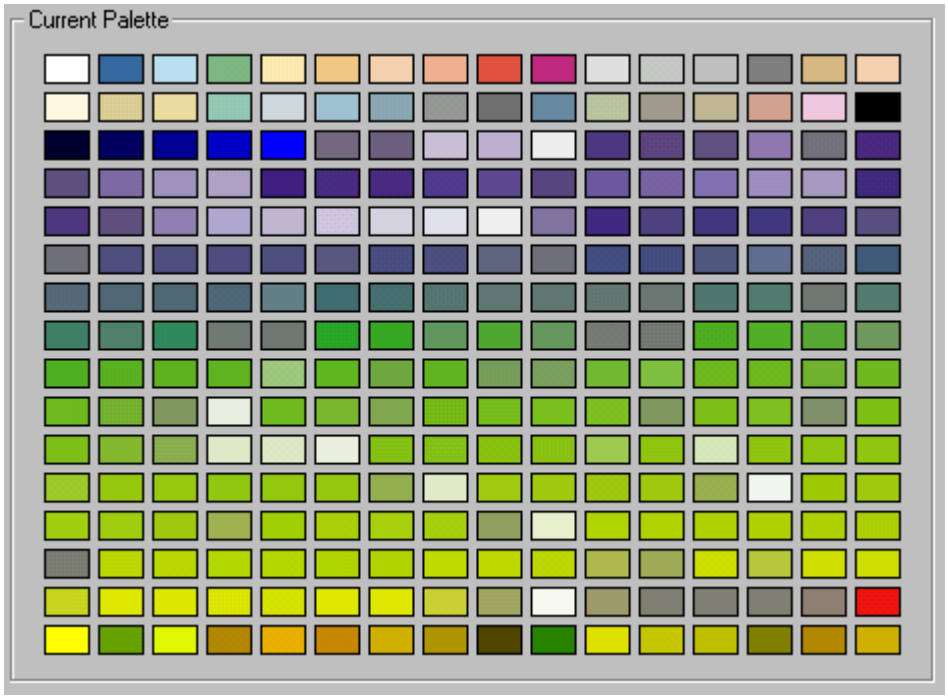


Illustration 5: La palette de couleur qui lui est associée.

1.4.4 Teintes de gris

En général, les images en niveaux de gris renferment 256 teintes de gris. C'est une image codée sur 8 bits, simplement chacune de ces 256 couleurs est définie dans la gamme des gris. Par convention la valeur zéro représente le noir (intensité lumineuse nulle) et la valeur 255 le blanc (intensité lumineuse maximale).

Illustration 6: L'image (taille réelle).

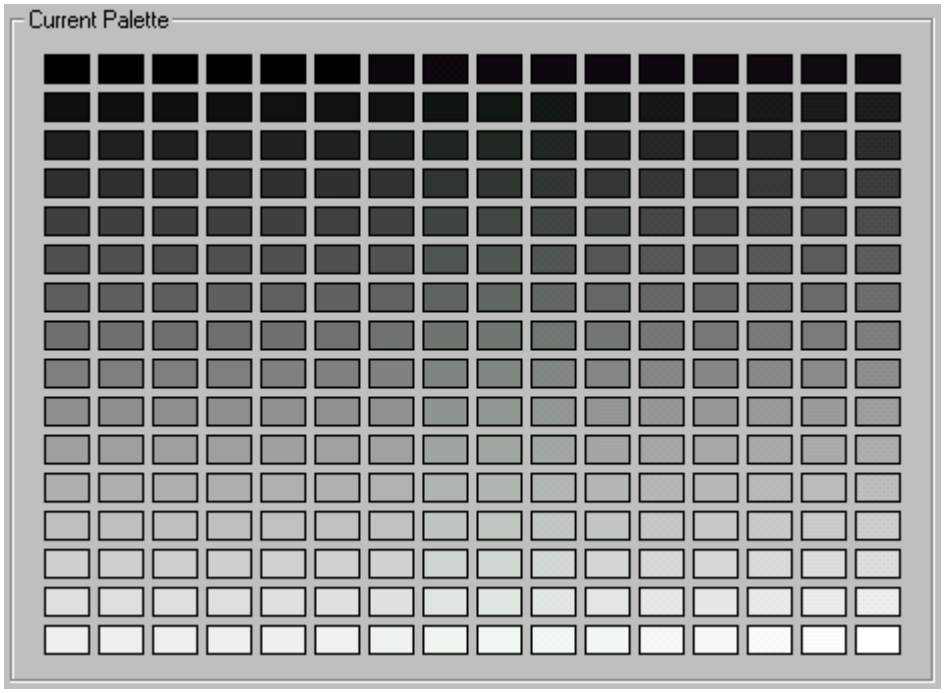


Illustration 7: La palette de teintes de gris.

1.4.5 Le codage TSL (teinte saturation luminance) ou HSL (Hue, Saturation, Luminance en anglais) ou encore YUV (Luminance (Y) - Chrominance et Saturation (U-V)) ou encore HSB (Hue, Saturation, Brightness)

La couleur a une certaine teinte à laquelle on ajoute ou soustrait du noir (saturation), et du blanc (luminosité). Ce modèle est le plus proche de la perception humaine. Une couleur est perçue par sa teinte (brun, violet, vert,...), sa saturation (couleur terne proche du gris ou couleur "pétante") et sa luminance (plus ou moins claire). Ce codage n'est pas utilisé directement en informatique (car peu pratique), mais les logiciels de dessin proposent généralement un choix de couleur à partir de cette méthode, permettant ainsi un choix plus aisé des couleurs. Les composantes TSL sont alors convertie en RGB.

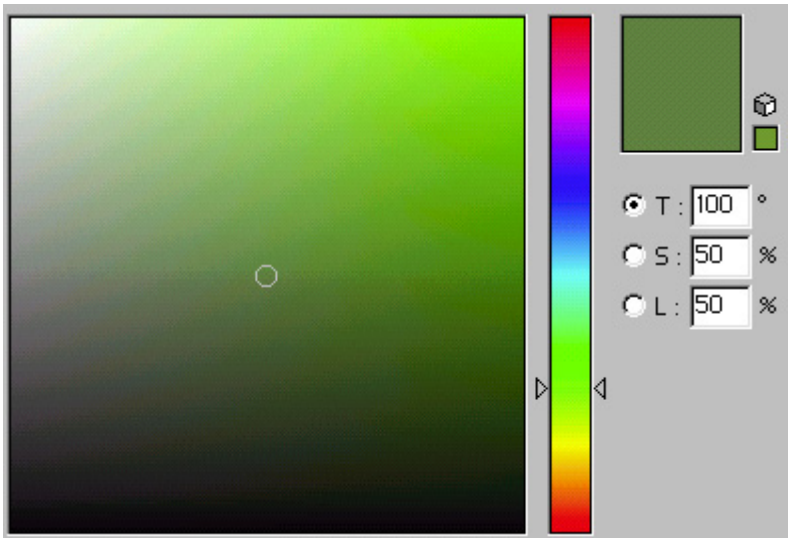


Illustration 8: Voici le premier exemple.

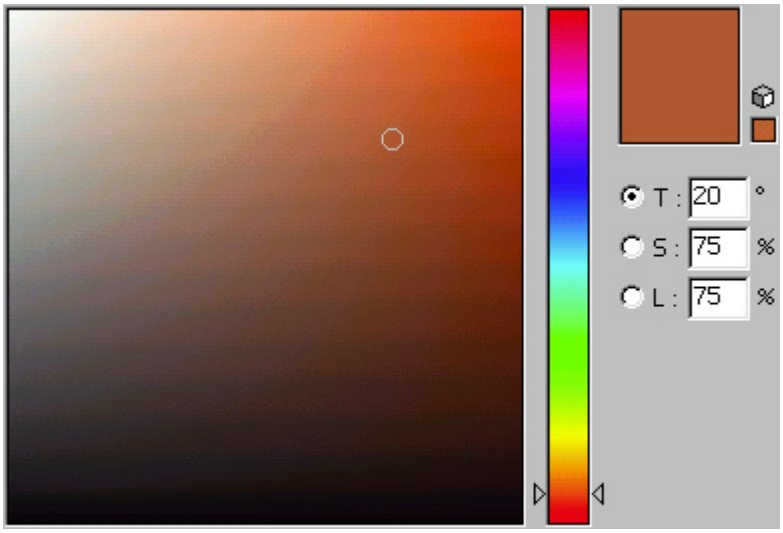


Illustration 9: Voici le deuxième.

1.4.6 Le codage CMJN

Les couleurs reproduites sont basées sur la synthèse soustractive des quatre couleurs suivantes: Cyan, Magenta, Jaune, Noir. Ceci est surtout utilisé principalement pour l'impression.

> Extrait de la documentation de PSP : « *Si une image RVB est destinée à être affichée à l'écran, il est inutile de la convertir au mode CMJN. De même, si une image numérisée CMJN doit être séparée et imprimée, il est inutile d'effectuer des corrections en mode RVB.* »

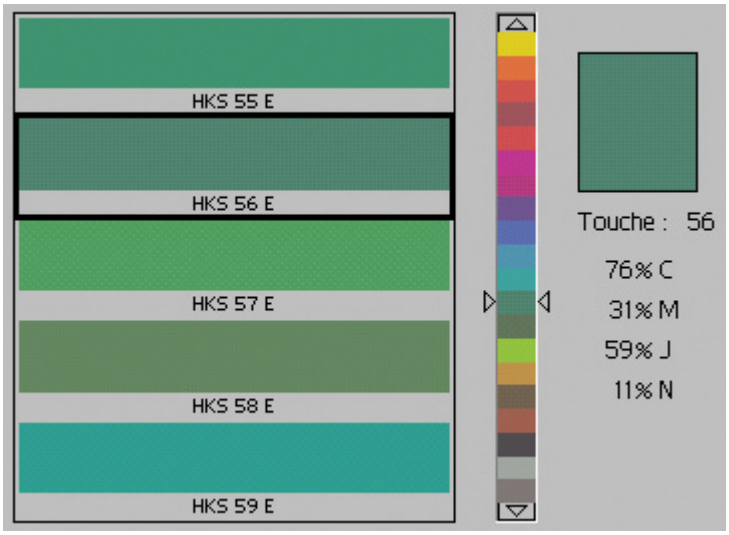


Illustration 10: Le logiciel de dessin Photoshop permet de choisir les couleurs comme on choisirait des teintes de peintures à Leroy Merlin.

1.5) Conclusion

Le problème dans la numérisation des images est qu'en augmentant la qualité d'une image, celle-ci prend rapidement beaucoup de place. Par exemple une image que l'on veut scanner à 200 dpi pour une sortie au format A4 (21 x 29,7 cm).
Hauteur : (29,7 cm/2,54) x 200 = 2338 pixels (1 pouce = 2,54 cm)
Largeur : (21 cm / 2,54) x 200 = 1653 pixels
Soit au total 3 864 714 pixels.
Si la profondeur de codage est de 24 bits (8 bits par couleur en mode RVB) : 3 864 714 x 3 = 115 944 142 octets soit 11 Mo.
Le poids du fichier est donc de 11 Mo.
Les facteurs de la dynamique de l'image sont à gérer en fonction de ce qu'on veut obtenir. Le type de compression (avec ou sans perte) utilisé va permettre de compresser plus ou moins ces données et par conséquent fournir des fichiers images dont le poids sera plus ou moins important. La dynamique des images imposée par certains formats va également influer.

II) LA COMPRESSION

2.1) L'organisation des données dans le fichier

Un fichier codant une image se décompose de la manière suivante.
Pour une image BMP, le principe de stockage est le suivant, chaque pixel est stocké avec son code couleur, la formule est très élémentaire. Le problème est qu'une image toute blanche prend la même place qu'une image contenant des millions de couleurs. Ce format permet néanmoins de conserver l'image numérisée sans perte de qualité.
Mais voyons plus en détails comment s'effectue l'encodage.
La table suivante contient une description du contenu d'un fichier BMP. Pour chaque champ, l'offset du fichier, la longueur et le contenu seront donnés. Ce tableau est donné pour être précis et montrer de quoi on parle. Le fichier codant un image contient 2 parties : la première, une entête contient les informations pour pouvoir lire la deuxième partie qui contient les informations, les données propre à l'image.

	Offset	Champs	Taille	Contenu
Entête	0000h	Identifiant	2 bytes	Le caractère identifiant le bitmap. Les entrées suivantes sont possibles : 'BM' – Windows 3.1x, 95, NT, … 'BA' – OS/2 Bitmap Array 'CI' – OS/2 Color Icon 'CP' – OS/2 Color Pointer 'IC' – OS/2 Icon 'PT' – OS/2 Pointer
	0002h	Taille du fichier	1 dword	Taille complète du fichier en bytes.
	0006h	Réservé	1 dword	Réservé pour un usage futur.
	000Ah	Bitmap Data Offset	1 dword	Offset du commencement des données graphiques.
	000Eh	Taille de l'entête du bitmap	1 dword	Longueur du Bitmap Info Header utilisé pour décrire les couleurs du bitmap, la compression, … Les tailles suivantes sont possibles: 28h – Windows 3.1x, 95, NT, … 0Ch - OS/2 1.x F0h - OS/2 2.x
	0012h	Largeur	1 dword	Largeur horizontale du bitmap en pixels.
	0016h	Longueur	1 dword	Hauteur verticale du bitmap en pixels.
	001Ah	Plans	1 word	Nombre de plans dans ce bitmap.
	001Ch	Bits Par Pixel	1 word	Bits par pixel utilisés pour stocker la palette. Ceci identifie aussi de façon indirecte le nombre possible de couleurs. Les valeurs suivantes sont possibles : 1 – Bitmap monochrome 4 – Bitmap 16 couleurs 8 – Bitmap 256 couleurs 16 – bitmap 16bits (high color) 24 – bitmap 24bits (true color) 32 – bitmap 32bits (true color)
	001Eh	Compression	1 dword	Spécifications de compression. Les valeurs suivantes sont possibles : 0 - aucune (Aussi identifié par BI_RGB) 1 - RLE 8-bits / pixel (Aussi identifié par BI_RLE4) 2 - RLE 4-bits / pixel (Aussi identifié par BI_RLE8) 3 – Bitfields (Aussi identifié par BI_BITFIELDS)

	0022h	Taille des données du bitmap en bytes	1 dword	Taille des données du bitmap en bytes. Ce nombre doit être arrondi au plus proche espace de 4 bytes.
	0026h	Hresolution	1 dword	Résolution horizontale exprimée en pixels par mètre.
	002Ah	Vresolution	1 dword	Résolution verticale exprimée en pixels par mètre.
	002Eh	Couleurs	1 dword	Nombre de couleurs utilisées par ce bitmap. Pour un bitmap de 8-bits / pixel ceci sera 100h or 256.
	0032h	Couleurs importantes	1 dword	Nombre de couleurs importantes. Ce nombre sera égal au nombre de couleurs quand chaque couleur est importante.
	0036h	Palette	N * 4 bytes	La spécification de la palette. Pour chaque entrée dans la palette 4 bytes sont utilisés pour décrire les valeurs RGB de la couleur de la façon suivante: 1 byte pour la composante bleue 1 byte pour la composante verte 1 byte pour la composante rouge 1 byte de remplissage qui est à 0
Chaîne des données	0436h	Données du bitmap	x bytes	Dépendant des spécifications de la compression, ce champs contient tous les bytes de données du bitmap qui représentent les indices dans la palette de couleurs.

Voici comment sont lues les données :

Taille	Nombres de bytes	Signe	Illustration :
Char	1	Signé	
Word	2	Non signé	
dword	4	Non signé	

Offset:	Bytes:	ANSI Text:
00000000	42 4D 3A A8 02 00 00 00 00 00	BM:¨□
0000000A	36 00 00 00 28 00 00 00 D9 00	6 (Ù
00000014	00 00 0B 01 00 00 01 00 18 00	□□ □ □
0000001E	00 00 00 00 04 A8 02 00 C4 0E	□¨□ Ä□

Illustration 11 : Voici une capture d' écran d'un éditeur hexadécimal. La modification des données provoque des effets surprenants…

L'intérêt d'utiliser la compression est de réduire au maximum la chaîne de données du bitmap qui peut rapidement devenir très longue (contrairement à ce qu'on pourrait penser au regard du tableau). Bien entendu l'entête sera automatiquement modifiée spécifiant un autre mode de lecture de la chaîne de données qui sera à interpréter différemment par le logiciel qui lit l'image. Il existe plusieurs types de compression, c'est à dire qu'il existe plusieurs manières de réécrire les données bitmap pour qu'elles prennent moins de place. Les méthodes de compression et de codage réduisent le nombre de bits par pixel à stocker ou à transmettre, en exploitant la redondance des informations dans l'image.
Les méthodes de compression opèrent différemment. Elles sont classées de la manière suivante : non-destructrices ou destructrices...

2.2) Les compressions non-destructrices

Elles ne modifient en rien l'apparence des images. Elles modifient uniquement le codage des couleurs. Elles permettent de retrouver exactement les pixels de l'image numérique originale. Voici les différentes méthodes utilisées.

2.2.1 Le RLE *(Run Length Encoding)*

Le RLE, méthode basique, consiste à repérer des répétitions de valeurs identiques (qui peuvent être des bits ou des octets). Puis il suffit d'indiquer la valeur et le nombre de répétitions consécutives. Il est assimilable à l'opérateur « fois » en mathématique. S'il repère dans le fichier une suite, il remplace la suite d'un même nombre par un couple (valeur, nombre de répétions). Les algorithmes peuvent différer sur la matrice qu'ils utilisent : on peut s'intéresser aux valeurs de bit, d'octet ou de pixel…

Exemple :

Un petit exemple pour la compréhension :
Chaîne : 000110010111111000100000000001111111
Chaîne compressée : (0,3)**(1,2)(0,2)(1,1)(0,1)***(1,6)*(0,3)**(1,1)***(0,10)(1,7)*
Les couples en gras sont peu ou pas intéressants car ils prennent plus de place que l'original ; en revanche ceux en italiques sont intéressants. L'exemple est en binaire mais il fonctionne sur un cadre de lecture plus large.
Il existe une petite optimisation pour le binaire, on omet la valeur et par convention on commence par le 0. Ce qui donne : 3 2 2 1 1 6 3 1 10 7.

2.2.2 Le LZW (Lempel-Ziv-Welch) ou LZ77

La société Unisys ayant déposé la méthode LZW, il en existe une variante non brevetée, le LZ77. Elle porte le nom de ses trois inventeurs : *Lempel* et *Ziv* qui l'ont conçue en 1977, et *Welch* qui l'a finalisée en 1984.
Elle utilise un dictionnaire qui n'est pas stocké dans le fichier compressé qu'elle construit dynamiquement, au cours de la compression et de la décompression. Plus compliqué que le RLE, il s'agit cette fois de repérer des motifs répétés composés de séries variables en longueur de bits ou d'octets. Chaque motif est copié dans le dictionnaire et se voit attribué un indice. Puis chaque motif est remplacé dans le fichier par son indice, sachant qu'une valeur isolée n'est pas codée. Elle a besoin d'un apprentissage pour être efficace, pour reconnaître des longues chaînes répétées. En effet, l'algorithme ne fonctionne pas sur un nombre fixe de motifs mais *apprend* les motifs du fichier durant la lecture du fichier à compacter. Elle est donc peu efficace sur des petits fichiers. Cette méthode est très rapide.
Voici un exemple qui va permettre de mieux comprendre :

Exemple :

La zone tampon est une fenêtre sur un nombre N de bits qui se trouve juste avant la position courante dans le fichier à compresser.
On cherche dans ces N bits si la chaîne à partir de la position courante s'y trouve. Si elle y est, on remplace la chaîne par l'offset relatif de la chaîne dans la fenêtre avec sa longueur.

Nous avons la chaîne suivante, la position courante est indiquée par le | et le nombre N est à 10 (délimiteur #) :

100110#1001110110|10011001111

Nous recherchons s'il existe plusieurs 10 : 100110#**10**01110110|**10**011001111
Plusieurs 100 : 100110#**100**1110110|**100**11001111
Plusieurs 1001 : 100110#**1001**110110|**1001**1001111
Plusieurs 100011 : 100110#**10011**10110|**10011**001111
Plusieurs 100110 : ça ne marche pas.
On copie le motif dans le dictionnaire et remplace 10011 par son indice.
On met à jour la position courante en se décalant : 100110**10011**#10110**10011**|001111
Ce qui permet de ne pas stocker le dictionnaire, c'est qu'au moins un des codes indicés du fichier n'est pas codé et les autres identiques sont une référence à celui-ci : offset ou pointeur (indiquant où) + longueur de la chaîne répétée.

2.2.3 Le codage de HUFFMAN

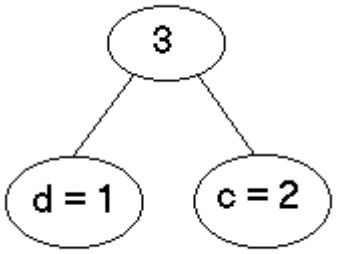
La première chose à faire est de lire complètement le fichier d'entrée pour comptabiliser le nombre d'occurrences de chaque caractère afin de créer une table de fréquences. Huffman propose de recoder les données qui ont une occurrence très faible sur une longueur binaire supérieure à la moyenne, et recoder les données très fréquentes sur une longueur binaire très courte. Pour former ces codes binaires, on utilise une structure en arbre : soit on va à gauche (1), soit on va à droite (0), en commençant en additionnant les probabilités les plus faibles. Je vous propose de bien lire l'exemple pour comprendre…
Il ne faut pas qu'une séquence binaire soit à la fois représentative d'un élément codé et constitutive du début du code d'un autre élément. Sinon ça serrait indécodable étant donné que les codes sont de longueur variable ! (on aurait pas ce problème avec un code fixe). La structure en arbre binaire permet justement de na pas générer deux codes ambigus.
Pour coder on relit le fichier puis on écrit le fichier avec les codes générés et l'arbre de codage.
Un des inconvénients est que l'on doit sauvegarder l'arbre de codage pour la décompression, il n'y a aucun moyen de connaître les codes utilisés sinon. De plus, du point de vue de la complexité en temps, cet algorithme a un inconvénient. En effet, il faut d'abord lire tout le fichier, ce qui n'est pas négligeable en raison des entrées/sorties (qui sont très gourmandes en temps CPU), surtout pour des fichiers de taille importante. La décompression aussi n'est pas très rapide.

Exemple :

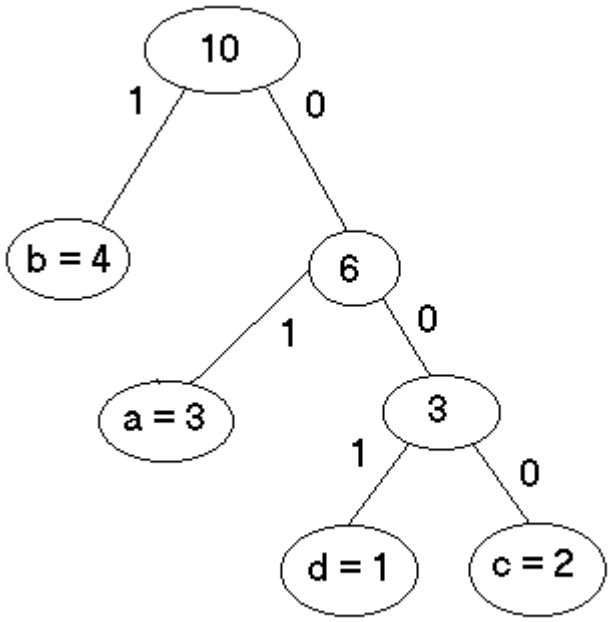
Je choisis que ma méthode de compression de huffman se base sur des mots de 4 bits (1 mot = suite de 4 chiffres), c'est la cadre de lecture si vous voulez…
Par exemple on a la chaîne suivante : 1001 1101 0101 0001 1001 1101 1101 1101 1001 0101
Que je traduis de la manière suivante : a b c d a b b b a c pour faciliter la compréhension !
On établit une table de probabilité :

1001	1101	0101	0001
a	b	c	d
3	4	2	1

On établit un arbre : on commence par les éléments de plus faible probabilité :



Puis on continue… Mon exemple ne le montre pas mais deux nœuds peuvent s'additionner pour n'en faire qu'un, et pas forcément un nœud et un mot :



Ce qui nous donne la table de correspondance suivante :

a	b	c	d
01	1	000	001

Ce qui nous donne au final une chaîne tout de même plus petite :

01 1 000 001 01 1 1 1 01 000

On note qu'il n'y a aucune ambiguïté sur la lecture, on ne peut lire un mot à la place d'un autre.

2.3) Les compressions destructrices

Ces méthodes permettent de retrouver une approximation de l'image numérique. Les pertes de qualité sont généralement invisibles à l'œil nu mais cela dépend aussi de l'image.

2.3.1 La transformation discrète en cosinus (TDC, DCT, Discrete Cosine Transform):

C'est une transformation mathématique qui transforme un ensemble de données d'un domaine spatial en un spectre de fréquence.. En fait cette transformation n'est pas destructrice mais c'est la quantification qui est associée à cette méthode qui n'est pas conservative.
Plus concrètement, on découpe l'image en matrices de taille de 8x8 pixels, l'image est donc découpée en bloc de pixels. On fait subir à ces blocs la transformée discrète en cosinus, ce qui donne ce qu'on appelle des matrices DCT. Les valeurs de la matrice DCT ont été arrondies à l'entier le plus proche.
Ensuite la quantification lit les valeurs en zigzags inclinés à 45° en commençant par le coin supérieur gauche et en finissant en bas à droite car du fait des propriétés de la matrice DCT, cela a l'avantage de regrouper les ensembles de valeurs proches.
Le but de l'étape de quantification est de diminuer la précision du stockage des entiers de la matrice DCT en divisant les valeurs de la matrice DCT et en les arrondissant, pour diminuer le nombre de bits occupés par chaque entier. C'est la partie non-conservative de la méthode.
Les données ainsi obtenues ont une forte redondance et donc présentent un terrain très favorable aux compressions conservatives.
Voici un exemple qui n'expliquera en rien comment fonctionnent ces outils mathématiques mais qui va permettre de comprendre les différentes étapes.

Exemple :

La matrice de quantification a un facteur de qualité de 2 (qui apporte des changements importants à la matrice DCT, mais des modifications mineures à l'image) : voici la matrice de pixels au départ :

3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29
17	19	21	23	25	27	29	31

L ‘application de cette table à la matrice DCT suivante aboutit à la matrice DCT suivante.

92	3	-9	-7	3	-1	0	2
-39	-58	12	17	-2	2	4	2
-84	62	1	-18	3	4	-5	5
-52	-36	-10	14	-10	4	-2	0
-86	-40	49	-7	17	-6	-2	5
-62	65	-12	-2	3	-3	-2	0
-17	14	-36	17	-11	3	3	-1
-54	32	-9	-9	22	0	1	3

Matrice DCT après quantification :

31	1	1	1	0	0	0	0
-52	-8	1	2	0	0	0	0
-12	7	0	1	0	0	0	0
-6	-3	-1	1	-1	0	0	0
-8	-3	3	0	1	0	0	0
-5	4	-1	0	0	0	0	0
-1	1	-2	1	0	0	0	0
-3	2	0	0	1	0	0	0

On comprend l’intérêt de faire ces transformations pour les méthodes conservatrices.

2.3.2 Fractale

Je n’aurais pas la prétention de donner un exemple qui montrerait exactement comment cela fonctionne mais déjà essayons de comprendre le principe. En effet un rapport de TIPE (en prépa maths-spé) montrant l’enchaînement de formules mathématiques décourage toute approche mathématique à mon explication…
Le principe de la transformation fractale repose sur le fait qu'une image est « auto-similaire par parties » : des régions de l'image peuvent être mises en correspondance avec d'autres régions de tailles plus petites leur ressemblant. Si on part de l’idée qu’une image peut être décrite à partir d'un ensemble de motifs identiques en nombre limité, transformés par translations, rotations, symétries et agrandissements ou réductions…, pour coder l'image, il suffit de décrire les motifs originaux et les transformations utilisées. C’est pour cela que l’on dit parfois que la compression fractale permet de coder une image par elle-même. Le codage d'une image devient donc indépendant de sa taille. En réalité le codage des images utilisent les fractales itératives (IFS : Iterated Function System). Voici un dessin emprunté au site http://perso.wanadoo.fr/reverrance/graf/fractal/fractal2.htm qui permet de comprendre tout de suite de quoi on parle :

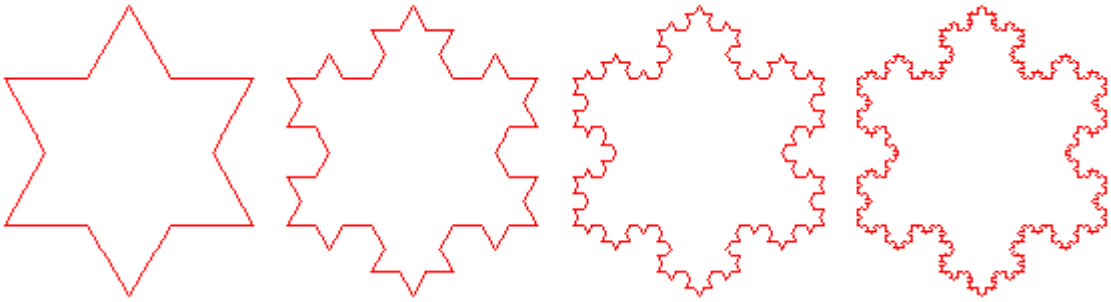


Figure 1 : le flocon de Von Koch

On procède de la manière suivante : on découpe l'image en blocs-parents carrés de 16 pixels de côté et on les découpe à leur tour en 4 blocs-fils de 8 pixels de côté. Puis on recherche un bloc attracteur : pour chaque bloc, on va calculer son attracteur de manière approximative, c'est à dire un couple de fonctions qui appliqué itérativement à un bloc quelconque permet de converger vers le bloc… On codera un bloc en indiquant la référence à l'autre bloc et les fonctions qu'il suffit d'appliquer. J'ai esquissé un exemple sur un schéma :

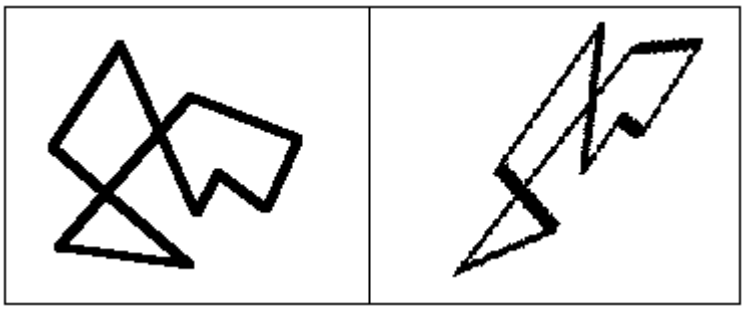


Illustration 11 : Entre le premier bloc et le deuxième bloc, il y a eu 4 transformations : une rotation de 90°, une réduction au 4/5, une inclinaison de 30° horizontale et verticale. On comprend pourquoi la méthode prend du temps à trouver les affinités entre les blocs.

Le problème est le temps de compression très élevé alors qu'à l'inverse la décompression est extrêmement rapide

2.3.3 Ondelettes

L'idée est de diviser la résolution de l'image en codant la perte d'information découlant de cette division.
La compression par ondelettes consiste à considérer les zones d'une image contenant de fortes variations du contraste comme des hautes fréquences, le reste de l'image étant de la basse fréquence. On extrait les hautes fréquences (qui sont les contours des objets) et on les garde telles quelles par une compression non destructrice, le reste étant réduit de façon destructrice puis compressé à nouveau par ondelettes.
Essayons d'aller plus loin que ce principe général. Une transformation par ondelettes consiste à décomposer un signal en une tendance grossière accompagnée de détails de plus en plus fins. Ainsi, pour reconstituer le signal avec une précision donnée, il suffira de connaître la tendance et les détails correspondant au niveau de précision recherché et de négliger les autres.
Voici l'image piqué sur :
http://www.chez.com/rpauchet/lapageTIPE/TIPEixl/TIPEondelettes/TIPEhtml/Tipe%20maths

%2099%20compression%20par%20ondelettes.htm qui montre comment le signal est décomposé.

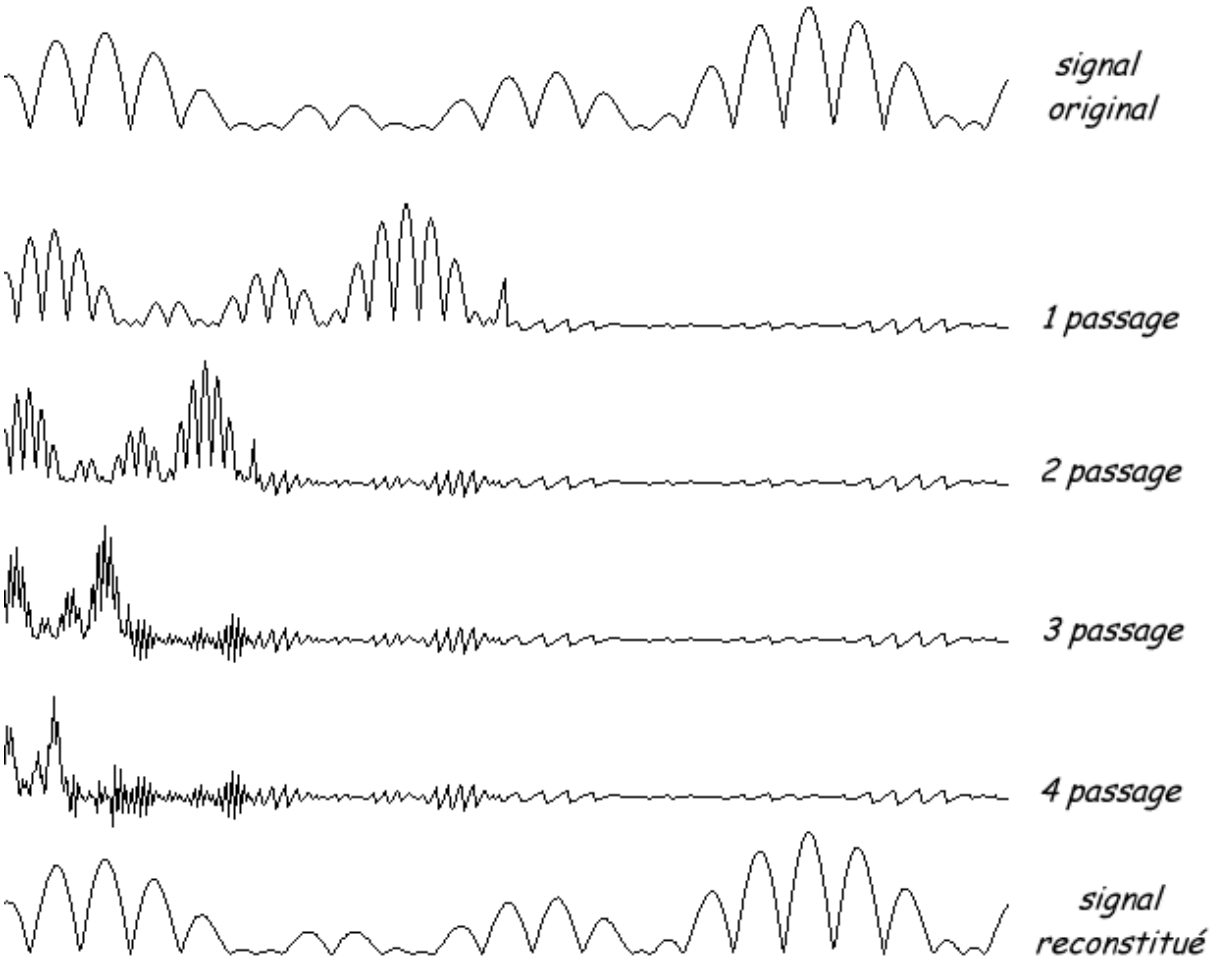
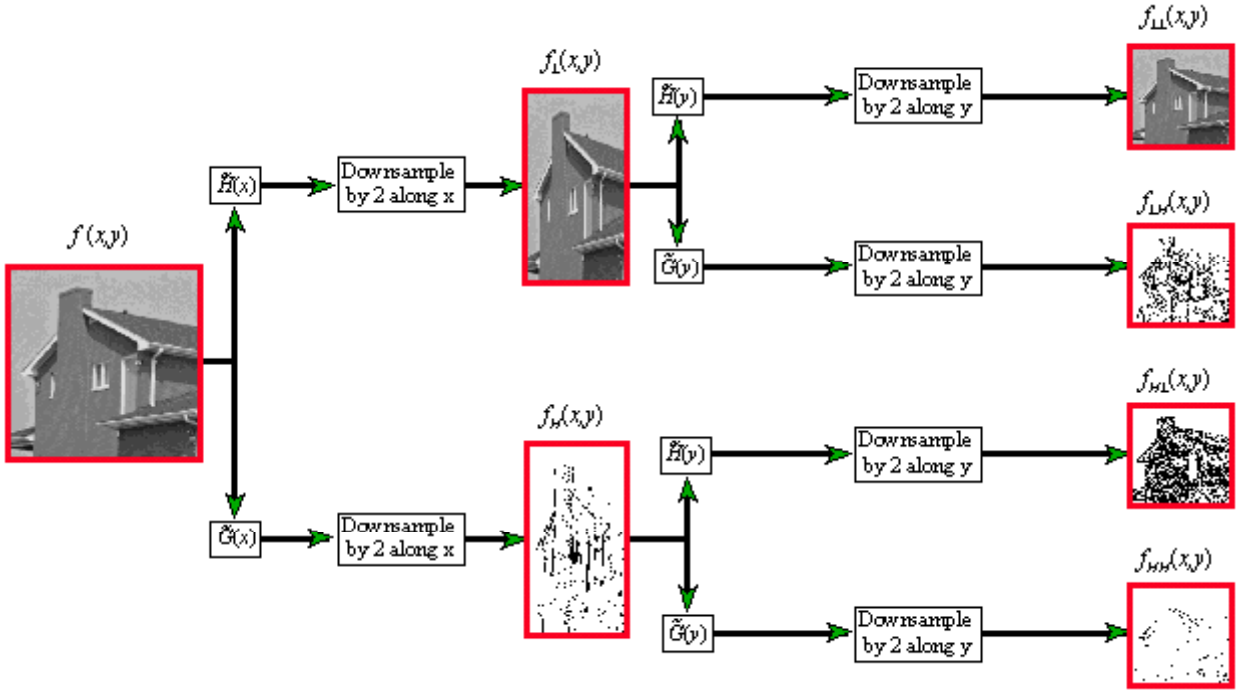


Illustration 12

On fait un sous-échantillonnage de l'image dans le sens horizontal. On calcule l'erreur entre l'image originale et l'image sous-échantillonnée dans le sens horizontal. Pour chacune des 2 images obtenues, on fait un sous-échantillonnage dans le sens vertical. Pour chacune des 2 images obtenues, on calcule l'erreur dans le sens vertical. On obtient alors 4 signaux :
Voici une illustration de mes propos du site http://www.image-etc.com/faq/wavelet/Page1.htm :



On répète cette transformation un certain nombre de fois puis on effectue une quantification. On abandonne les détails inférieurs à un certain niveau défini par l'utilisateur (si l'erreur est minime, elle est éliminée), comme ça on réduit la masse de données mais on perd des données.

2.4) Conclusion

Les formats de fichiers actuels utilisent souvent en associant plusieurs méthodes de compression. Le format le plus connu est sans doute le Jpeg qui associe la compression DCT pour linéariser les données, la quantification augmente la redondance, puis enfin une compression conservatrice. Les compressions destructrices sont récentes et très évoluées. L'avantage est qu'elles sont performantes aussi bien au niveau de la qualité que du ratio et qu'elles fonctionnent bien sur tous les types d'images (presque). Cependant la perte de données est dommageable et irréversible, c'est ce qui en font leur inconvénient majeur. Il existe des formats utilisant uniquement des méthodes conservatives qui fonctionnent très bien sur des images où les nuances de couleurs sont faibles (dessins) mais qui présentent un faible intérêt sur des images photographiques par exemple.

Ici les méthodes de compression sont décrites dans leur principe et un peu dans leur fonctionnement. Bien entendu les mises en œuvre peuvent être beaucoup plus complexes en fonction du langage de programmation utilisé ou des optimisations qu'on souhaite intégrer dans les algorithmes.