DES EXEMPLES DE TRAITEMENTS DIGITAUX

Bernard Couty

DES EXEMPLES DE TRAITEMENTS DIGITAUX

Fabrication, compression de bitmaps, manipulations.

Résumé : Ce second essai se consacre à détailler des procédés de traitement du signal numérique, principalement la constitution de fichiers bitmaps et leur compression en format JPEG. Son également abordés les procédés de conservation de l'intégrité des signaux lors de la transmission. L'ensemble de ces exemples vise à montrer comment la visée industrielle d'empirie doit tenir compte de contraintes sévères afin de préserver les conditions de création de sensations (esthématopée) tout en diminuant les coûts de transmission et de stockage des données.

Mots-clés : arborescence, bitmap, chrominance, codage de canal, codage de Hamming, codage de Hufmann, codage des couleurs, compression d'image, empirie, esthématopée, luminance, pixellisation, pixellisation, programme, redondance, stéganographie, synthèse soustractive, transformation cosinus discrète, YCbCr.

Auteur : Bernard Couty, MCF 71e section (retraité)

Chercheur associé au LiRIS, EA 7481, Université de Rennes 2

Deuxième causerie :

TRAITEMENTS NUMÉRIQUES

Cette seconde causerie est essentiellement « logicielle », afin de montrer les traitements des signaux sous un angle *théorétique* (description de l'art). Trois domaines sont concernés : le traitement des images, la stéganographie, la protection des signaux.

1 : Du numérique appliqué à l'esthématopée.

(la compression des images)

On parle de « numérique », « numérisation », « digitalisation, « digital » ... Mais aussi, pratiquement tous les usagers de l'informatique en ont entendu parler, de « compression » des images (par exemple le format JPEG), des sons (par exemple le MP3), des séquences vidéo (par exemple le MP4) etc...

1.a. Compression et pixellisation : rien de très nouveau.

Nous pourrions commencer par la « compression », car c'est une technique aussi ancienne que le dessin et la sculpture, en nous réservant d'aborder ultérieurement les aspects numériques de cette opération. Considérons la statuette ci-contre, datant de la fin du XIX[e]. A supposer qu'elle ait été modelée d'après un modèle humain en chair et en drap, il est évident que le sculpteur, outre une réduction homothétique du modèle, n'a retenu qu'une partie des traits utiles de telle manière que ne subsistent que des volumes propres à créer un certain type de sensation et de perception immédiates (esthématopée [1]) Il en irait de même pour le dessin. Une partie de « l'information » est perdue, ne subsiste que celle qui vise à produire la Gestaltung. Cette opération soustractive, mais limitée, faute de quoi il n'y aurait pas d'esthématopée, est précisément, dans son principe, ce que l'on appelle *compression*.

10 Esthématopée

Ce concept de compression nous introduit à un autre, celui de « pixellisation »[2]. Pour rester dans le domaine des Beaux-Arts, on aurait tendance à évoquer les Impressionnistes, mais à l'examen, il s'agirait plutôt chez eux de *vectorisation* -composition d'éléments simples géométriques- que de pixellisation [3]. En revanche, la photographie argentique est bel et bien une opération de pixellisation, malgré l'anachronisme de l'appellation. En effet, la pellicule (N&B +

[1] Voir Bruneau et Balut, *Artistique et Archéologie*, 1997, pp112 et sq.
[2] Rappel : « pixel » est la contraction de « **pic**tural **el**ement », élément pictural.
[3] Voir par exemple Monet *Les Nymphéas*

gris) comporte deux couches dont une composée de particules photosensibles, des grains d'halogénures d'argent. Lors de l'exposition, les photons délocalisent des électrons des ions halogénures et il se forme des atomes d'argent par capture des électrons libres par les ions argentiques Ag^+. Mais, et c'est là un point essentiel, l'image obtenue est discontinue -on le remarque au « grain » de l'image, plus ou moins dense selon le modèle de pellicule utilisée. En cela, la technique employée est à rapprocher de la mosaïque. Par conséquent, relativement au rayonnement de l'objet photographié, on a une perte d'information (compression) et une division de l'image en éléments simples (« archipels » d'atomes d'argent) ou pixels.

1.b. La couleur.

Nous avons, dans les deux cas considérés de la sculpture et de l'image 2D, des signaux au sens où l'entendait Jean Gagnepain. Par rapport à ceux dont nous avons traités lors de la première causerie, ceux-ci ne sont pas temporels mais *spatiaux*. Cela ne signifie pas pour autant que lors de la transmission ils ne revêtent pas une dimension temporelle, comme nous le verrons. Mais pour l'heure, compte tenu du fait que la plupart de ces signaux picturaux sont colorés, examinons le cas des photographies en couleur. Elles fonctionnent par synthèse soustractive de couleurs (Cyan, Magenta, Jaune). Depuis les travaux de Newton, on sait obtenir une impression de « blanc » par rotation d'un disque dont les secteurs comportent chacun, dans l'ordre, les couleurs fondamentales du spectre de la lumière blanche. Logiquement, en supprimant une partie des composantes de la lumière blanche, à l'aide de filtres colorés, on obtient un « reste » formé du rayonnement recherché. C'est sur ce principe qu'étaient fabriquées les pellicules en couleur.

Dans le domaine de la photographie numérique et de la télévision on procède par synthèse additive de trois couleurs fondamentales Rouge, Vert et Bleu. Le procédé est suffisamment connu pour que je m'y attarde, témoin l'image 11. En combinant le rouge et le vert fondamentaux, on obtient du jaune, le rouge et le bleu on obtient du magenta et le bleu avec le vert, du cyan. RVB fondamentales combinées donnent du blanc, leur absence du noir. C'est le procédé actuellement le plus employé en imagerie numérique, bien que certains logiciels de DAO[4] emploient aussi la synthèse soustractive (CMJN).

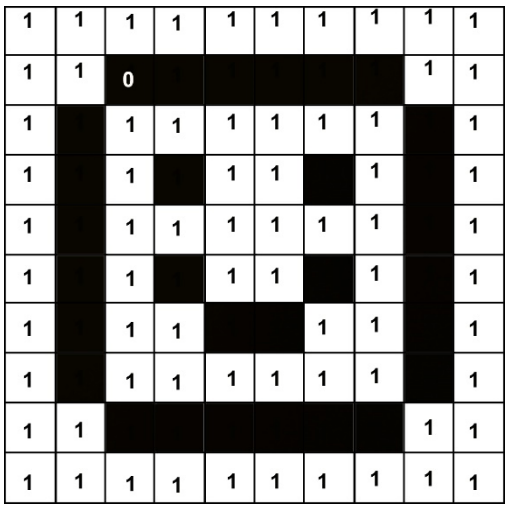
11 Synthèse additive des couleurs

Industriellement, la question est de fabriquer une image, colorée ou non.

[4] Dessin assisté par ordinateur

1.c. Le « bitmap ».

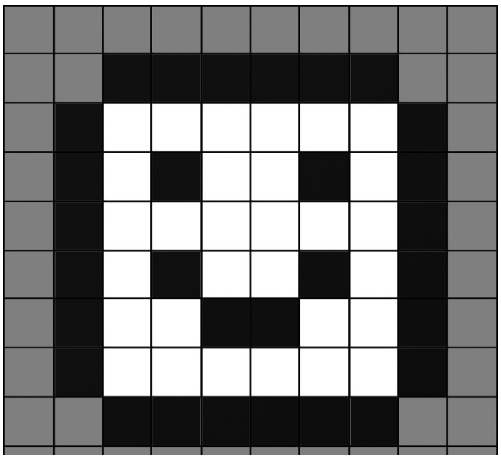
L'engin le plus simple (et peut-être le mieux approprié) est un tableau de valeurs comportant des lignes et des colonnes, un peu à la manière de la mosaïque régulière ou de la photo argentique. Ce tableau occupe des positions déterminées dans la mémoire vive (RAM) d'un ordinateur[5] et peut être exporté – par exemple sur le disque dur. Considérons donc l'exemple très simple d'un tableau de 10 x 10 soit dix lignes et dix colonnes. Chaque case correspond à un « pixel » qui, dans ce cas précis, peut prendre deux valeurs discrètes[6] : 1 et 0. Dans ce cas précis, un pixel correspond à 1 bit (**Bi**nary Dig**it**), soit une unité d'information. Par convention, l'état 1 correspond au blanc, l'état 0 au noir ; on peut ainsi tracer une « émoticône » en noir et blanc.



12a Codage 1 bit par pixel : 2 possibilités

Ce mode général de fabrication où chaque pixel du tableau porte une valeur de couleur (ici seulement N ou B) porte le nom de « BitMap[7] » ; ce n'est pas le seul procédé, mais il est très majoritairement employé. Cependant, si le codage 1bbp (1 bit par pixel) pourrait convenir à la fabrication de caractères (lettres), il est très limité.

Supposons que l'on veuille remplacer les blancs extérieurs à l'émoticône par une nuance de gris, par exemple un gris moyen. Il est évident que l'on ne peut plus fonctionner en « tout ou rien ». Le pixel devra alors prendre une valeur entre N et B ! Autrement dit, nous sortons du système discret initial pour entrer dans un intervalle [0..1] comportant théoriquement une infinité de nuances. En fait, industriellement parlant, on limite à 256 ou 65536 nuances dans l'intervalle [N..B] soit 2^8 ou 2^{16}. Chaque valeur est cependant discrète, puisqu'elle s'oppose à toutes les autres. Ces valeurs sont exprimables non plus en bits, mais en ***octets***. J'assume que tout le monde sait qu'un octet est une mesure



12b gris moyen

[5] Pour avoir une idée de la matérialité du tableau en RAM, nous pouvons imaginer une « grille » de transistors ou de micro-circuits intégrés capables de prendre des états déterminés.

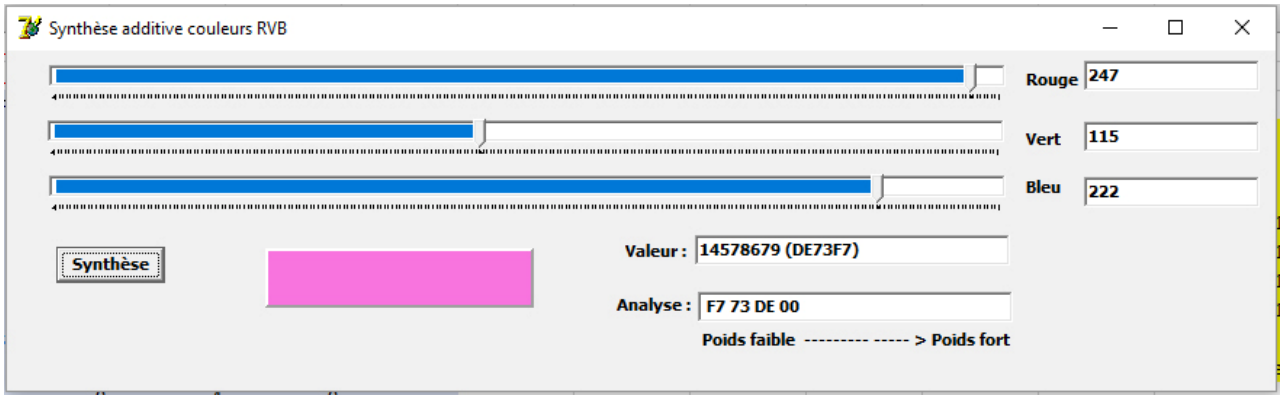
[6] Valeurs discrètes : ici, valeurs booléennes qui s'excluent mutuellement. Le domaine du « discret » est semble-t-il typiquement humain, voir par exemple l'existence formelle d'un phonème dont la définition est de ne pas être les autres phonèmes. Attention : un intervalle [0...1] n'est pas discret, car une infinité de valeurs existent entre 0 et 1. Mais en logique booléenne, on a soit la valeur 1, soit la valeur zéro (OU exclusif), et aucune valeur intermédiaire entre les deux.

[7] Ou image matricielle.

de 8 bits[8]. Dès lors, chacun des pixels concernés par le gris moyen sera codé non plus sur 1 bit, mais sur 8 ou 16 bits, soit [0..255] ou [0..65535]. Dans la figure 12b, je prends la valeur 127 pour le gris moyen Notre tableau contient donc désormais, outre les valeurs N et B, des valeurs de 127 ; les octets ont donc ces valeurs binaires : 00000000 (noir), 111111111 (blanc), 01111111(gris). Ce nouveau codage a des conséquences sur le « poids » de l'image. En effet, on doit tenir compte du nombre de pixels (ici 10 x 10) mais aussi du type de codage de la couleur : bit, 1 octet, 2 octets etc... Par exemple l'image 12a, codée sur un bit, pèse (10*10*(1/8))/1024 = 0,012207031 Ko, l'image 12b codée sur 1 octet pèse (10*10*8)/1024 = 0,78125 Ko[9]. Ce « poids » correspond en fait à la quantité de mémoire qu'il faut allouer pour stocker et/ou produire l'image.

Cette notion de « poids » témoigne bien sûr de contraintes techniques (industrielles) -capacité de la RAM ou des unités de stockage- mais aussi d'autres contraintes, relevant, elles, des plans III et IV, à savoir le coût de production et d'acheminement de l'image ; notamment, plus le poids est élevé, plus il mobilise de ressources sur les canaux (payants !) de transmission. On peut limiter les contraintes en choisissant, par exemple, un codage sur 8 bits à l'aide d'une « palette » prédéfinie de 256 couleurs codées de 0 à 255, mais cela implique que le logiciel chargé de *re-produire* l'image connaisse cette palette ; en outre certains dégradés risquent d'être assez mal rendus pour l'œil.

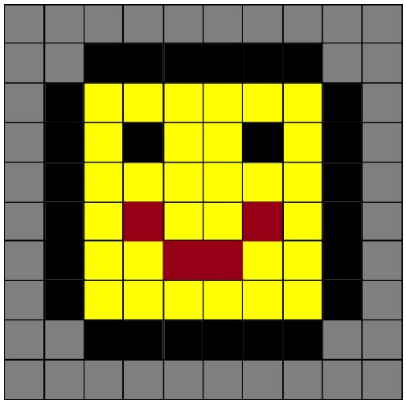
Car, ne l'oublions pas, nous sommes dans l'esthématopée, production de sensations. En théorie, l'œil humain perçoit jusqu'à 200 nuances de chaque couleur fondamentale (RVB), ce qui donnerait quelque 8 millions de nuances par synthèse additive. Il semble que la réalité soit plus restrictive. Néanmoins, la synthèse additive RVB sous 24 bits (3 octets, un octet par couleur fondamentale) autorise 16777216 couleurs allant du noir au blanc. La copie d'écran d'un petit logiciel que j'ai fabriqué *ad hoc* montre un exemple de nuancier fonctionnant par synthèse additive :



[8] L'affaire se complique à peine si l'on se rend compte que la valeur maximale d'un octet est de 255 bits (11111111 en binaire), mais il faut ajouter la valeur 0 (00000000), ce qui donne bien 256 valeurs. Par contre, le **nombre** 256 nécessite deux octets pour s'écrire en binaire : 00000001 11111111.

[9] 1 kilooctet = 1024 octets soit 8192 bits

Dans cet exemple, la valeur de la couleur est 14 578 679 par synthèse additive de R=247, V=115, B=222, ce qui correspond à un nombre binaire codé sur trois octets (24 bits) : 110111100111001111110111[10].



12c codage couleur sur 24 bits

Je peux donc coder les couleurs de mon bitmap-exemple en 24 bits. Imaginons que je veuille un émoticône en jaune : les pixels encore blancs prendront la valeur 65535 (R=256, V=256, B=0 soit, en hexadécimal, 00 FF FF). Si je veux teinter la bouche en carmin plutôt qu'en noir, les pixels correspondants prendront la valeur 1573014 (R=150, V=0, B=24, soit en hexadécimal 18 00 96). Mais puisque j'ai changé de mode, passant de 256 niveaux de gris (1 octet) au RVB (3 octets), le gris devra lui aussi être codé sur 24 bits, ce qui est facile : R=V=B=127, le noir également (00 00 00). Le « poids du bitmap en sera augmenté : (10*10*24)/1024 = 2,34375 Ko.

L'émoticône de cet exemple est minuscule, toutefois pas très éloigné(e ?) en taille des émoticônes habituelles. Mais plus la taille des images grandit, plus leur poids s'accroît. Si je veux afficher une image en « plein écran », ça se complique, car il faut tenir compte de la « résolution » de l'écran (ou de l'imprimante). Les formats d'affichage modernes, pour les ordinateurs standards, sont de l'ordre de 1280 x 1024 en RVB ; la *résolution* d'une image est le nombre de points (pixels) sur une ligne, rapporté à une unité de longueur, l'inch, qui vaut ≈2,54 cm. Par exemple, une image de 200 pixels en largeur, mesurant 2 pouces de côté aura une résolution de 200/2 = 100 pixels par inch (on dit 100 dpi, « dots per inch). Un écran de 17 pouces de diagonale en format 1280 x 1024 aura une résolution de 9297 PPI² (pixels par inch carré). On comprend qu'une grosse image devra subir, si besoin est, une diminution, tandis qu'une petite image devra subir une extrapolation ; il s'agit toujours, cependant, d'une transformation *homothétique*[11] opérée par logiciel.

Considérant le tableau (en RAM ou sur un support constant type disque dur), on pourrait avoir l'impression que l'image existe « virtuellement » ; il n'en est rien, bien sûr, car le résultat des opérations précédentes n'est qu'une étape dans un processus industriel, en termes d'ergologie, où la saisie a autant d'importance que la restitution. Qu'avons-nous ? un tableau de chiffres (plutôt de nombres) qui, du début à la fin, sont graphies par 1 et 0 (même si on peut les lire en hexadécimal) de *commandes*. Voici

---

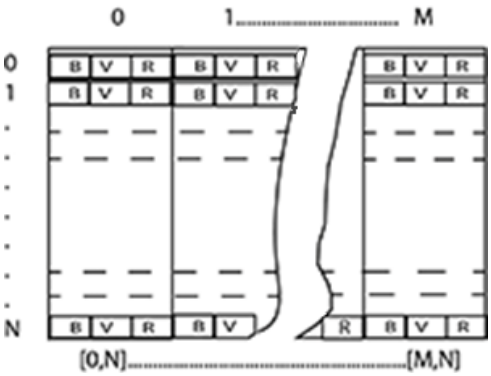
[10] La lecture d'un tel nombre binaire est difficile ; on la simplifie en découpant le nombre en paquets de quatre bits à partir de la droite, que l'on code en numération hexadécimale (base seize : 10 vaut seize, F vaut 15, E vaut 14,..,A vaut 10 etc...) ; par exemple le second groupe de 4 bits à droite (« poids faible'), dans l'exemple, est 1111, soit 15 en décimal, F en hexadécimal ; notre nombre est donc, en hexadécimal, DE73F7. C'est une commodité pour les ingénieurs, mais *in fine* les processeurs ne connaissent que le binaire. Je fournirai l'algorithme de décodage hexadécimal→binaire si besoin est.

[11] Rappel : l'image d'un point M par l'homothétie de centre 0 et de rapport k positif est le point M' tel que M' appartient à la demi-droite OM et OM'=k*OM

qui semble a priori paradoxal, car on considère d'habitude ces valeurs tabulées comme des *données*. Mais, outre qu'elles ne sont pas données mais obtenues (« *adeptas* »), elles imposent au logiciel de restitution des procédures bien spécifiques. C'est littéralement un *programme* (écriture *pour*, coalescence ordonnée de machines à fabriquer de la sensation) obtenu par action d'un dispositif et destiné à commander un autre dispositif en synergie avec le précédent, aussi éloignés soient-ils dans l'espace... et dans le temps. 1 signifie l'envoi d'une impulsion, 0 son absence. Plus exactement, l'analogie avec l'interrupteur ne s'imposant plus depuis que les transistors ont remplacé les relais électromagnétiques qui, effectivement, ouvraient ou fermaient un circuit, c'est une différence entre un état « bas » (par exemple 0,25 V=0) et un état « haut » (par exemple 2 V=1).

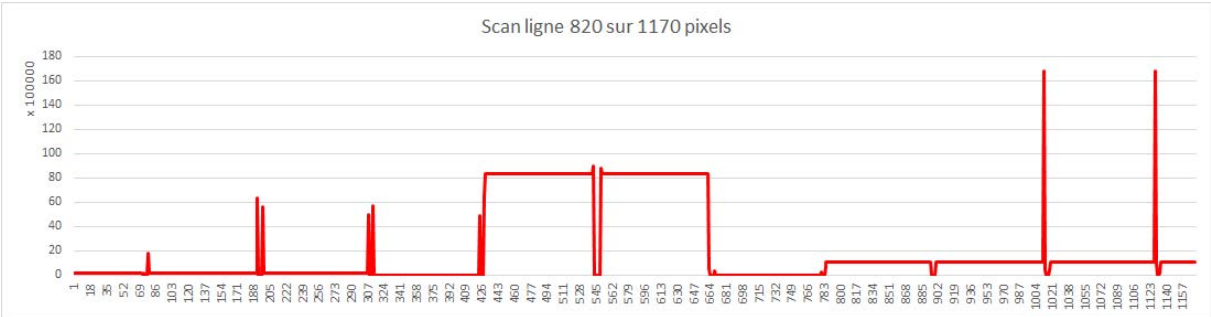
1.d. Redondance, fréquence, contrastes.

Jusqu'ici, avec l'image, nous sommes demeurés au niveau d'un signal *spatial*. Il n'échappera cependant à personne que le tableau se lit ligne par ligne, colonne par colonne et qu'à chaque fois l'information y est inscrite ou lue de manière séquentielle. On pourrait dire que l'information est contenue dans un *tableau de tableaux* : la structure correspondant à un pixel est elle-même constituée de trois champs numériques (B, V, R ou R, V, B), et arrangée séquentiellement en lignes et colonnes.



14 Tableau des valeurs numériques

Il s'ensuit qu'une ligne est une succession régulière de blocs de 24 octets, la synthèse des valeurs de chaque bloc donnant la valeur de la couleur du pixel. Si nous prenons maintenant ligne par ligne, nous constatons donc qu'apparaît à chaque fois un signal linéaire, dont l'élongation[12] est la valeur de couleur du pixel. Ce qui signifie que l'on peut parfaitement traiter l'image comme on traite un signal de type vibratoire ; le calcul est un peu plus compliqué, puisque l'on travaille non plus en une dimension, mais en deux dimensions. L'image 15 montre la courbe obtenue en lisant la ligne 820 d'un bitmap de 1170 x 1162 pixels, prise au hasard :



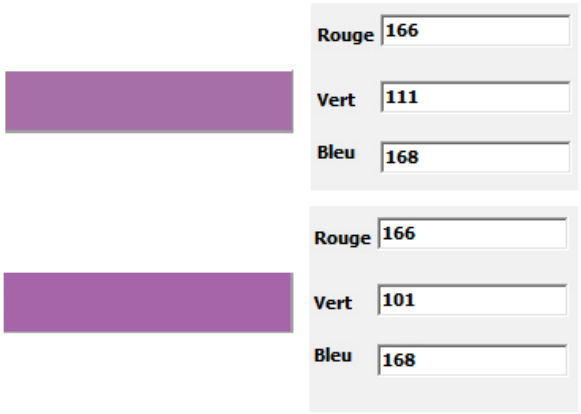
15 Courbe des couleurs de pixels sur une ligne

[12] C'est-à-dire la distance entre 0 et l'ordonnée y pour un pixel *m* de l'espace [0..M]

Ce signal se caractérise par de brusques changements de fréquence[13], constatables aux brusques pics ou dépressions interrompant une succession de valeurs égales. On imagine facilement que cela pourrait correspondre à des délimitations de surfaces ou de volumes. Si l'on intègre le calcul pour chaque ligne sur la totalité des lignes, on obtient une représentation mathématique de l'image. On pourra ainsi effectuer sur l'image une transformée de Fourier en 2D, utile notamment pour analyser et transformer l'image. Nous en reparlerons. Pour le moment, disons que dans le cas d'une image (signal 2D), les basses fréquences représentent les grandes surfaces homogènes présentant une forte **redondance** de teinte, et les parties floues alors que les hautes fréquences représentent les contours, plus généralement les changements brusques d'intensité et enfin le bruit.

1.e. La compression d'images.

La compression vise à réduire la quantité de mémoire nécessaire pour le stockage d'une image ou/et de réduire le temps de transmission de celle-ci. Ceci vaut pour l'image fixe et a fortiori pour les images de vidéo. Toutefois, comme nous l'avons dit, la compression est une réduction, une perte d'information ; il faut donc qu'elle puisse demeurer dans des limites acceptables, au-delà desquelles l'esthématopée serait compromise, ou du moins rendue incertaine. Le résultat de la compression devra par conséquent ne pas comporter trop de changements, hiatus, ruptures, afin que l'œil (ou l'oreille dans le cas de la compression de sons) ne perçoive pas les distorsions introduites par l'artifice. Tant que, comme le montre l'image 16, le contraste est suffisamment réduit pour que l'œil s'y trompe, tout va bien. Nous avons bien physiquement deux couleurs distinctes, puisque que vert est passé de 111 à 101, mais esthésiquement on ne fait pas de différence.



16 Deux couleurs réduites à une seule perception

La compression JPEG (la plus fréquente actuellement) des images numériques fait appel à des algorithmes complexes dont je me contenterai de donner ici un aperçu.

[13] Dans un signal périodique (voir première causerie), la fréquence F est l'inverse de la période T : F=1/T. Ici, on parle de *fréquence spatiale* : une dimension unitaire de l'espace (ici le pixel) prend la place du temps. « La **fréquence spatiale** est une grandeur caractéristique d'une structure qui se reproduit identiquement à des positions régulièrement espacées. Elle est la mesure du nombre de répétitions par unité de longueur ou par unité d'angle. » Cette Wikipédie en vaut bien d'autres.

1.e.1. Le codage RVB → YCbCr.

Considérons donc une image dont les caractéristiques sont stockées dans un tableau tri-dimensionnel en y, x, z (y lignes, x colonnes par ligne, z=3 couches de couleur (R, V, B)[14]. Et accrochons-nous !

Une première manipulation, appelons-la A, consiste à changer le codage RVB en un codage quantitatif et qualitatif, autrement dit : *luminance* et *chrominance*[15]. En fait, c'est sur ce codage que fonctionnent les images de télévision, vidéo incluse. En RVB, il n'y a pas de signal noir ni de signal blanc, il faudrait donc un quatrième octet qui aurait les proportions adéquates des trois couleurs fondamentales pour obtenir des pixels en N et B. Mais c'est ennuyeux, car on aurait désormais quatre octets à transmettre. On réservera donc un octet pour la luminance. Cet octet est noté Y tel que Y = 30%R+ 60%V +10%B[16]. Comme Y est synthèse proportionnelle aux valeurs des trois couleurs primaires, on peut n'en transmettre que deux en plus de Y, ce qui nous donne toujours trois octets.

Je peux par exemple escamoter V et transmettre Y, R, B ; je retrouverai V en résolvant une simple équation : Y = 30%R+ x +10%B. Techniquement, on transmet Y, R-Y, B-Y, c'est le codage noté YCrCb (ou YCbCr). Exemple numérique : une teinte est R=166, V=111, B=169. Nous aurons :

$$Y = 166 \times 30\% + 111 \times 60\% + 169 \times 10\% = 49 + 66 + 16 = 131$$

$$B\text{-}Y = 169 - 131 = 38$$

$$R\text{-}Y = 166 - 131 = 35$$

C'est ce que je transmets. Je récupère au décodage :

$$B = B+Y = 38 + 131 = 169$$

$$R = R+Y = 35 + 131 = 166$$

Selon la formule de Y, il faut récupérer 60% de V, soit Y = 49 + x = 60%V + 16, donc 60% V = 131- (49 +16) =66 ; 1% = 66/60=1,1 ; 100% = 110 donc V = 110. Ah ! Mais au départ V = 111 : il y a une petite perte d'information, mais comme nous l'avons vu avec l'exemple de la figure 16, ce n'est pas perceptible.

On utilise donc volontiers le codage YCbCr, dont la représentation matricielle est[17] :

---

[14] Pour simplifier, nous prendrons des images de dimension 16*16 pixels et 3*8 bits pour coder la couleur.

[15] Notre œil est sensible d'abord à la quantité de lumière, car il dispose de quelque 92 à 100 millions (en moyenne) de photorécepteurs fonctionnant en Noir, Blanc et Gris. Ce sont les récepteurs de *luminance*. Il est aussi sensible à la couleur grâce aux quelque 3 à 4millions de cônes sensibles aux signaux R, V, B, récepteurs de la *chrominance*.

[16] Blanc sera Y=255 et Noir Y=0.

[17] I est le numéro de ligne, J le numéro de colonne, le dernier chiffre représente le numéro de l'octet de couleur fondamentale (R,G,B R,V,B en Français).

$$\begin{pmatrix} Y[I,J,O] \\ Cb[I,J,1] \\ Cr[I,J,2] \end{pmatrix} = \begin{pmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}1687 & -0{,}3313 & 0{,}5 \\ 0{,}5 & -0{,}41874 & -0{,}0813 \end{pmatrix} \begin{pmatrix} R[I,J,O \\ G[I,J,1] \\ B[I,J,2] \end{pmatrix}$$

Ce codage vaut pour la vidéo-TV, mais il est aussi employé pour les images fixes car, pour des raisons que je ne détaillerai pas ici, il permet d'obtenir un meilleur taux de compression. Voici les instructions en code Pascal qui effectuent ce codage YCbCr :

```
For I := 0 to Longueur - 1 do
 begin
                for J := 0 to Largeur - 1 do
      begin
                R := TPixels[I, J, 0];
                G := TPixels[I, J, 1];
                B := TPixels[I, J, 2];

                TPixels[I, J, 0] := Round(0.299 * R + 0.587 * G + 0.114 * B);
                TPixels[I, J, 1] := Round(-0.1687 * R - 0.3313 * G + 0.5 * B + 128);
                 TPixels[I, J, 2] := Round(0.5 * R - 0.41874 * G - 0.0813 * B + 128);
       end;
 end;
{Round signifie « arrondi » à la valeur entière la plus proche}
```

Comparons 5 pixels (séparés par « | | ») de la ligne 2 d'un bitmap quelconque, représentés par leurs octets (séparés par « ; ») d'abord dans l'état originel :

`203;199;177||203;199;177||203;199;177||207;177;220||207;177;220||`

puis après codage YCrCb (l'octet le plus à gauche, dans chaque cas, est la valeur de Y, les autres respectivement de Cr et Cb) :

`198;116;132||198;116;132||198;116;132||191;144;139||191;144;139||`

Les valeurs entières ne correspondent pas exactement aux pourcentages réels, calculés avec des décimales. Nous enregistrons encore une perte d'information de l'ordre de quelques dixièmes, encore tolérable.

1.e.2. Sous-échantillonnage de la chrominance.

Mais ce n'est pas fini ! La manipulation B consiste en un sous-échantillonnage de la chrominance.

La chrominance paraît avoir un impact moindre sur la perception finale des couleurs que la luminance. L'idée est donc de considérer des blocs adjacents de pixels

dont on conserve la luminance mais dont les chrominances Cb, Cr sont remplacées par une moyenne. Exemple pris sur 4 pixels codés YCrCb :

Ligne	Pixels col. a			Pixels col. b		
	R	G	B	R	G	B
m	203	145	123	199	139	87
m+1	210	112	100	222	93	138

Codés selon la manipulation B :

Ligne	Pixels Col. a			Pixels Col.b		
	Y	Cr	Cb	Y	Cr	Cb
m	203	122	112	199	122	112
m+1	210	122	112	222	122	112

Sans toucher à la luminance, on peut regrouper Cr et Cb qui occupent 8 cases en seulement deux cases :

Ligne	Pixels Col. a	Pixels Col.b	Chrominances a et b	
	Y	Y	Cr	Cb
m	203	199	122	112
m+1	210	222		

L'opération de sous-échantillonnage (« Downsampling ») divise par quatre la taille des matrices contenant la chrominance et par deux la taille de l'image[18]. Mais cela se fait encore au prix d'une perte d'information. Et ce n'est pas fini !

1.e.3. La transformation cosinus discrète, passage à la fréquence.

La manipulation C est appelée Transformation Cosinus Discrète (DCT en Anglais). On cherche à transformer l'image traitée en un ensemble de valeurs numériques pour lesquelles l'algorithme de compression peut être le plus efficace, et cela marche mieux lorsqu'il y a peu de valeurs distinctes. En fait, comme je l'ai évoqué dans le commentaire *supra* de l'image 15, une image ordinaire présente une grande continuité entre les valeurs des pixels adjacents. Il est alors possible d'envisager de

[18] En fait, ce regroupement est calculé lors de cette phase du traitement, mais ne deviendra effectif qu'ultérieurement, sinon les opérations suivantes seraient perturbées.

transférer l'information du domaine spatial dans le domaine fréquentiel. Comment s'y prendre ?

L'idée du groupe JPEG[19] a considéré qu'il était simple et efficace de traiter l'image par blocs de 8x8 pixels, étant observé que les changements au sein du bloc pourraient être de peu d'ampleur, comme nous l'avons dit. Qui dit « fréquence » dit π [20] et Cosinus. On peut ainsi de donner une table TCos[I, J] telle que celle de la figure 17 :

$$U = \frac{1}{2}\begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \cos\frac{\pi}{16} & \cos\frac{3\pi}{16} & \cos\frac{5\pi}{16} & \cos\frac{7\pi}{16} & \cos\frac{9\pi}{16} & \cos\frac{11\pi}{16} & \cos\frac{13\pi}{16} & \cos\frac{15\pi}{16} \\ \cos\frac{2\pi}{16} & \cos\frac{6\pi}{16} & \cos\frac{10\pi}{16} & \cos\frac{14\pi}{16} & \cos\frac{18\pi}{16} & \cos\frac{22\pi}{16} & \cos\frac{26\pi}{16} & \cos\frac{30\pi}{16} \\ \cos\frac{3\pi}{16} & \cos\frac{9\pi}{16} & \cos\frac{15\pi}{16} & \cos\frac{21\pi}{16} & \cos\frac{27\pi}{16} & \cos\frac{33\pi}{16} & \cos\frac{39\pi}{16} & \cos\frac{45\pi}{16} \\ \cos\frac{4\pi}{16} & \cos\frac{12\pi}{16} & \cos\frac{20\pi}{16} & \cos\frac{28\pi}{16} & \cos\frac{36\pi}{16} & \cos\frac{44\pi}{16} & \cos\frac{52\pi}{16} & \cos\frac{60\pi}{16} \\ \cos\frac{5\pi}{16} & \cos\frac{15\pi}{16} & \cos\frac{25\pi}{16} & \cos\frac{35\pi}{16} & \cos\frac{45\pi}{16} & \cos\frac{55\pi}{16} & \cos\frac{65\pi}{16} & \cos\frac{75\pi}{16} \\ \cos\frac{6\pi}{16} & \cos\frac{18\pi}{16} & \cos\frac{30\pi}{16} & \cos\frac{42\pi}{16} & \cos\frac{54\pi}{16} & \cos\frac{66\pi}{16} & \cos\frac{78\pi}{16} & \cos\frac{90\pi}{16} \\ \cos\frac{7\pi}{16} & \cos\frac{21\pi}{16} & \cos\frac{35\pi}{16} & \cos\frac{49\pi}{16} & \cos\frac{63\pi}{16} & \cos\frac{77\pi}{16} & \cos\frac{91\pi}{16} & \cos\frac{105\pi}{16} \end{bmatrix}$$

17 Table des cosinus

Il s'agit d'une matrice carrée de 8x8 valeurs calculées informatiquement de la manière suivante :

Pour I de 0 à 7 :

Si I = 0 alors Pour J de 0 à 7 : TCos[0, J]=$0{,}5 * \frac{\sqrt{2}}{2}$

Sinon Pour J de 0à 7 : TCos[I, J]=$0{,}5 * Cos((2 * J + 1) * I * \frac{\pi}{16})$

(On comprend que c'est le calcul automatique du multiplicateur de π qui est opéré par (2*J+1)*I). Concrètement, on obtient la matrice TCos :

0,3535; 0,3535; 0,3535; 0,3535; 0,3535; 0,35355; 0,35355; 0,3535;

0,4903; 0,4157; 0,2777; 0,0975; -0,0975 ; -0,2777; -0,4157; -0,4903;

0,4619; 0,1913; -0,1913; -0,4619; -0,4619; -0,1913; 0,1913; 0,4619 ;

0,4157; -0,0975; -0,4903; -0,2777; 0,2777; 0,4903; 0,0975; -0,4157 ;

0,3535; -0,3535; -0,3535; 0,3535; 0,3535 ; -0,3535; -0,3535; 0,3535;

0,2777; -0,4903; 0,0975 ; 0,4157; -0,4157; -0,0975; 0,4903; -0,2777;

0,1913; -0,4619; 0,4619 ; -0,1913; -0,1913 ; 0,4619; -0,4619; 0,1913 ;

0,0975; -0,2777; 0,4157 ; -0,4903; 0,4903; -0,4157; 0,2777; -0,0975;

[19] Joint Photographic Experts Group

[20] En électricité, par exemple, la pulsation d'un courant est $\omega=2\pi F$.

La suite de la TCD est informatiquement simple : si l'on a un tableau TPixels de dimension Longueur x Largeur, on se donne trois matrices carrées de 8x8, respectivement Mat, Mat1 et MatT, cette dernière recevant les résultats de la TCD.

1- Pour I de 0 à Largeur div 8 -1 :

2- Pour J de 0 à Longueur div 8 -1 : {*div est l'opérateur de la division euclidienne*}

3- Pour Z de 0 à 2 :

4- Pour Y de 0 à 7 :

5- Pour x de 0 à 7 : Mat[Y,x] = Tpixels[I*8+Y, J*8+x, Z] ([21])

Fin 5

Fin 4

6- Pour U de 0 à 7 :

7- Pour V de 0 à 7 :

S = 0

8- Pour N de 0 à 7 :

S = S+ TCos[U,N]*Mat[N,V]

Fin 8

Mat1[U,V] = S ([22])

Fin 7

Fin 6

9- Pour U de 0 à 7 :

10- Pour V de 0 à 7 :

S = 0

11- Pour N de 0 à 7 :

S = S + Mat1[U,N] * TCos[V,N]

Fin 11

MatT[U,V]= Valeur entière de S ([23])

Fin 10

Fin 9

12- Pour N de 0 à 7 :

13- Pour M de 0 à 7 : Tpixels[I*8+N, ,J*8+M,Z] = MatT[N,M] [24]

Fin 13

Fin 12

Fin 3

---

[21] On remplit la matrice par les valeurs x,y,z de chaque pixel du bloc de 8x8.
[22] On remplit la matrice intermédiaire Mat1 avec le résultat de la première transformation.
[23] On remplit la matrice MatT avec le résultat ultime de la TCD.
[24] On transfère les résultats dans TPixels.

Fin 2

Fin 1

Les boucles 6 → fin 6 et 9 → fin 9 correspondent très exactement à la TCD. Pour des raisons de lisibilité, ces boucles imbriquées dans l'algorithme sont de préférence renvoyées en procédures séparées. Quoi qu'il en soit, on constate que de 6 à fin 9, les valeurs sont soumises à deux fois la multiplication par le cosinus. Voilà qui (me) paraît plus lisible que la formule générale :

$$TCD[\mathrm{I},\mathrm{J}] = \frac{1}{\sqrt{2}}\, C[\mathrm{I}]C[\,\mathrm{J}] \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} TPixels[x,y]Cos\left(\frac{(2\mathrm{x}+1)\mathrm{I}\pi}{2\mathrm{N}}\right) Cos\left(\frac{(2\mathrm{y}+1)\mathrm{J}\pi}{2\mathrm{N}}\right)$$

Petite remarque : On travaille en trois dimensions : I, J, Z ; Par conséquent la boucle 4-fin 4 traite trois fonctions. À chaque pas des boucles 1, 2, 3, la matrice Mat contient successivement, pour les traiter : 8x8 valeurs de Y, 8x8 valeurs de Cr, 8x8 valeurs de Cb, par exemple :

État 1 :Valeurs 8x8 Y	État 2 :Valeurs 8x8 Cr	État 3 : Valeurs 8x8 Cb
36;89;65;89;107;71;23;81;	1;1;1;1;1;1;1;1;	5;5;5;5;8;8;8;8;
56;106;39;68;24;26;58;56;	1;1;1;1;1;1;1;1;	5;5;5;5;8;8;8;8;
76;26;14;38;16;45;66;23;	1;1;1;1;1;1;1;1;	5;5;5;5;8;8;8;8;
58;78;65;15;100;47;75;48;	1;1;1;1;1;1;1;1;	5;5;5;5;8;8;8;8;
76;75;74;45;81;36;101;74;	-4;-4;-4;-4;-1;-1;-1;-1;	-2;-2;-2;-2;2;2;2;2;
75;37;89;65;31;67;65;70;	-4;-4;-4;-4;-1;-1;-1;-1;	-2;-2;-2;-2;2;2;2;2;
80;60;119;81;47;54;67;77;	-4;-4;-4;-4;-1;-1;-1;-1;	-2;-2;-2;-2;2;2;2;2;
58;42;85;51;48;84;59;22;	-4;-4;-4;-4;-1;-1;-1;-1;	-2;-2;-2;-2;2;2;2;2;

Juste un aperçu des résultats partiels :

1ere sommation Σ 6-fin 6	S = 0 TCos[1,0]*Mat[0,0] dans Mat1[1,0] S:=S+TCos[U,I]*[Mat[I,V] et Mat1[U,V]:=S = 17,6541347503662 TCos[1,1]*Mat[1,0] dans Mat1[1,0] S:=S+TCos[U,I]*[Mat[I,V] et Mat1[U,V]:=S = 40,9352836608887 TCos[1,2]*Mat[2,0] dans Mat1[1,0] S:=S+TCos[U,I]*[Mat[I,V] et Mat1[U,V]:=S = 62,0469512939453 TCos[1,3]*Mat[3,0] dans Mat1[1,0] S:=S+TCos[U,I]*[Mat[I,V] et Mat1[U,V]:=S = 67,7045669555664 TCos[1,4]*Mat[4,0] dans Mat1[1,0] S:=S+TCos[U,I]*[Mat[I,V] et Mat1[U,V]:=S = 60,2911338806152 TCos[1,5]*Mat[5,0] dans Mat1[1,0] S:=S+TCos[U,I]*[Mat[I,V] et Mat1[U,V]:=S = 39,4572486877441 TCos[1,6]*Mat[6,0] dans Mat1[1,0] S:=S+TCos[U,I]*[Mat[I,V] et Mat1[U,V]:=S = 6,19846487045288 TCos[1,7]*Mat[7,0] dans Mat1[1,0] S:=S+TCos[U,I]*[Mat[I,V] et Mat1[U,V]:=S = **-22,2443065643311**
2eme sommation Σ 9-fin 9	S = 0 Mat1[3,0]*TCos[3,0] dans MatT[3,3] S:= S+ Mat1[U,I]*TableCos[V,I] et MatT[U,V]:= Round(S) = -6 Mat1[3,1]*TCos[3,1] dans MatT[3,3] S:= S+ Mat1[U,I]*TableCos[V,I] et MatT[U,V]:= Round(S) = -7 Mat1[3,2]*TCos[3,2] dans MatT[3,3] S:= S+ Mat1[U,I]*TableCos[V,I] et MatT[U,V]:= Round(S) = -17 Mat1[3,3]*TCos[3,3] dans MatT[3,3] S:= S+ Mat1[U,I]*TableCos[V,I] et MatT[U,V]:= Round(S) = -12 Mat1[3,4]*TCos[3,4] dans MatT[3,3] S:= S+ Mat1[U,I]*TableCos[V,I] et MatT[U,V]:= Round(S) = -8 Mat1[3,5]*TCos[3,5] dans MatT[3,3] S:= S+ Mat1[U,I]*TableCos[V,I] et MatT[U,V]:= Round(S) = -9 Mat1[3,6]*TCos[3,6] dans MatT[3,3] S:= S+ Mat1[U,I]*TableCos[V,I] et MatT[U,V]:= Round(S) = -6 Mat1[3,7]*TCos[3,7] dans MatT[3,3] S:= S+ Mat1[U,I]*TableCos[V,I] et MatT[U,V]:= Round(S) = **3**

L'état final du tableau pour une image 16x16 pixels (avec valeurs initiales aléatoires pour la démonstration) est par exemple :

485;-6;26| |23;-5;-13| |13;0;0| |-11;2;4| |-17;0;0| |10;-1;-3| |-4;0;0| |20;1;3| |494;10;10| |-21;20;13| |-4;0;0| |12;-7;-4| |12;0;0| |-39;5;3| |26;0;0| |-28;-4;-3| | Ligne 0

-25;13;24| |0;5;2| |3;0;0| |7;-2;-1| |26;0;0| |-48;1;0| |-55;0;0| |-30;-1;0| |-56;16;-13| |15;-8;-8| |10;0;0| |30;3;3| |-15;0;0| |34;-2;-2| |53;0;0| |-

52;-5;-8||8;-4;2||38;0;0||3;1;-1||10;0;0||-3;-1;0||24;0;0||-14;1;0||-8;1;5||54;0;-1||-44;0;0||32;0;0||8;0;0||17;0;0||-6;0;0||7;0;0|| Ligne 13

14;0;0||21;0;0||-63;0;0||22;0;0||-3;0;0||52;0;0||25;0;0||-7;0;0||21;0;0||7;0;0||21;0;0||21;0;0||-8;0;0||-28;0;0||-8;0;0||-21;0;0|| Ligne 14

3;4;6||2;3;-1||-2;0;0||-19;-1;0||-16;0;0||10;1;0||-15;0;0||-11;-1;0||9;-1;-4||-14;0;1||-1;0;0||21;0;0||18;0;0||-33;0;0||-19;0;0||-7;0;0|| Ligne 15

On remarquera que cette fois encore la TCD entraîne une petite perte d'information, notamment du fait que les résultats décimaux sont convertis en valeurs entières dans la boucle 9.

Il va de soi qu'il existe une transformation inverse qui permet de restaurer les valeurs initiales (entières) du TPixels lors de la reconstitution de l'image pour affichage.

1.e.4. Le facteur de qualité.

Faisant suite à la TCD, la manipulation D est plus simple, malheureusement elle est la seule de l'algorithme JPEG qui occasionne des pertes d'informations parfois vraiment sensibles. On cherche à réduire le nombre de valeurs dans le tableau TPixels, sans pour autant perdre les valeurs importantes. On applique donc à chaque triplet de valeurs correspondant à un pixel, un coefficient A dépendant d'un facteur de qualité Q tel que :

```
Pour I de 0 à longueur-1
  Pour J de 0 à largeur -1
    Pour K de 0 à 2
        A = 1+(1+ I mod 8 + J mod 8) *Q
        TPixels[I,J,K] =  Valeur entière (TPixels[I,J,K] / A
```

Il va de soi que l'on perdra d'autant plus d'informations que A sera grand, donc que Q sera grand.

Voici un exemple comparant 3 lignes de Tpixels sorties de la TCD avec ces lignes après application des coefficients (Q=6) :

72;-1359104;-84912||-6;-1272916;-79557||-30;-3600;-225||10;446852;27928||-26;3957;247||-3;-298491;-18655||0;-4470;-279||-9;252786;15799||1;-8;37||15;-20;-5||-15;0;0||14;7;2||-14;0;0||11;-5;-1||-8;0;0||4;4;1|| ligne 3

112;-11943;-746||-10;-11185;-699||-11;-32;-2||-9;3926;245||-13;35;2||3;-2623;-164||-16;-39;-2||6;2221;139||87;0;0||17;0;0||-16;0;0||13;0;0||-11;0;0||8;0;0||-5;0;0||2;0;0|| ligne 4

66;907703;56710||39;850141;53134||-38;2405;150||2;-298438;-18652||-5;-2643;-165||-10;199353;12459||-1;2986;187||-2;-168828;-10552||-25;6;-24||-13;13;3||12;0;0||-10;-5;-1||7;0;0||-5;3;1||3;0;0||-1;-3;-1|| ligne 5

Avec application des coefs :

7;-123555;-7719||0;-106076;-6630||-2;-277;-17||1;31918;1995||-2;264;16||0;-18656;-1166||0;-263;-16||0;14044;878||0;-1;3||1;-2;0||-1;0;0||1;0;0||-1;0;0||1;0;0||0;0;0||0;0;0|| ligne 3

9;-995;-62||-1;-860;-54||-1;-2;0||-1;262;16||-1;2;0||0;-154;-10||-1;-2;0||0;117;7||7;0;0||1;0;0||-1;0;0||1;0;0||-1;0;0||0;0;0||0;0;0||0;0;0|| ligne 4

5;69823;4362||3;60724;3795||-3;160;10||0;-18652;-1166||0;-155;-10||-1;11075;692||0;157;10||0;-8441;-528||-2;0;-2||-1;1;0||1;0;0||-1;0;0||0;0;0||0;0;0||0;0;0||0;0;0|| ligne 5

On constate qu'est apparu un grand nombre de 0, qui croît au fur et à mesure de l'accroissement des lignes du tableau, par exemple :

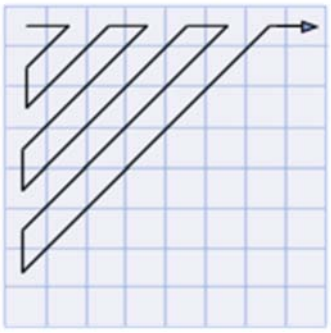
-6;0;0||-2;0;0||1;0;0||-1;0;0||1;0;0||0;0;0||0;0;0||0;0;0||8;0;0||0;0;0||0;0;0||0;0;0||0;0;0||0;0;0||0;0;0||0;0;0|| ligne 14

-1;1;-2||-1;0;0||1;0;0||-1;0;0||1;0;0||0;0;0||0;0;0||0;0;0||-2;1;-2||0;0;0||0;0;0||0;0;0||0;0;0||0;0;0||0;0;0||0;0;0|| ligne 15

Plus généralement, on constate une **redondance** de certaines valeurs.

1.e.5. Codage RLE.

C'est ce phénomène qui motive la manipulation E dite « codage RLE » (Anglais Run Length Encoding), le but étant d'obtenir un fichier nettement plus compressé. Le principe est théoriquement simple : prenons par exemple une chaîne abbbaaccc, il suffit de compter séquentiellement le nombre de lettres et de représenter la chaîne par une séquence Nombre+ lettre, soit 1a3b2a3c. Ce n'est pas très efficace, puisqu'on substitue à une chaîne de 9 caractères une autre de seulement 8 caractères. On obtiendrait un résultat meilleur avec aaadddbbbbcc (12 lettres) codé 3a3d4b2c (8 lettres), mais ce serait catastrophique avec « anticonstitutionnel » dont le codage dépasserait nettement la longueur de la chaîne[25] ! Donc, le codage RLE fonctionne bien lorsqu'il se trouve des redondances séquentielles importantes – les ingénieurs considèrent que 3 est le seuil minimum de redondance. Voilà qui s'applique bien aux images dont nous avons signalé à plusieurs reprises -et traité- le caractère redondant des valeurs de pixels consécutifs.



18 parcours en diagonales

Ainsi pouvons-nous explorer le tableau TPixels de manière linéaire par blocs de pixels, soit en lignes, soit en colonnes, soit en diagonales. C'est cette dernière méthode qui permet de récupérer le plus de 0 possibles ensemble – la TCD et la quantification D faisant que la plupart des valeurs utiles se trouvent en haut à gauche du tableau. Il est donc utile de compter les zéros en laissant les autres nombres. Cela fonctionne sur des matrices de dimension paire (une dimension impaire tronquerait l'image en escalier). Mais comment coder les séquences de zéros ? On peut admettre (c'est une astuce de programmeur) de les différencier des autres nombres en les ajoutant à 10 000 -cette quantité ne figure pas dans les tableaux, on ne peut donc pas la confondre avec un autre nombre. Par exemple, une séquence redondante de trois zéros sera notée 10003. Exemple :

1,0,5,0,0,0,6,3,0,0,0,0,0,0,4 sera noté 1,10000,5,10003,6,3,100005,4. [26]

Voici des aperçus des résultats du codage RLE :

---

[25] 1u 1s 1l 1e 1c 1a 2o 3i 4n 4t = 20. Ce n'est pas du tout économique !

[26] Pour le moment, cette astuce semble augmenter la quantification, mais l'étape suivante (Huffman) compensera efficacement cet inconvénient.

(0,0,2) (1,0,2) (0,1,2) (0,2,2) (1,1,2) (2,0,2) (3,0,2) (2,1,2)	51285
(1,2,2) (0,3,2) (0,4,2) (1,3,2) (2,2,2) (3,1,2) (4,0,2) (5,0,2)	40331
(4,1,2) (3,2,2) (2,3,2) (1,4,2) (0,5,2) (0,6,2) (1,5,2) (2,4,2)	40018
(3,3,2) (4,2,2) (5,1,2) (6,0,2) (7,0,2) (6,1,2) (5,2,2) (4,3,2)	97
(3,4,2) (2,5,2) (1,6,2) (0,7,2) (0,8,2) (1,7,2) (2,6,2) (3,5,2)	32372
(4,4,2) (5,3,2) (6,2,2) (7,1,2) (8,0,2) (9,0,2) (8,1,2) (7,2,2)	99
(6,3,2) (5,4,2) (4,5,2) (3,6,2) (2,7,2) (1,8,2) (0,9,2) (0,10,2)	-10619
(1,9,2) (2,8,2) (3,7,2) (4,6,2) (5,5,2) (6,4,2) (7,3,2) (8,2,2)	81
(9,1,2) (10,0,2) (11,0,2) (10,1,2) (9,2,2) (8,3,2) (7,4,2)	80
(6,5,2) (5,6,2) (4,7,2) (3,8,2) (2,9,2) (1,10,2) (0,11,2)	-10536
(0,12,2) (1,11,2) (2,10,2) (3,9,2) (4,8,2) (5,7,2) (6,6,2)	-83
(7,5,2) (8,4,2) (9,3,2) (10,2,2) (11,1,2) (12,0,2) (13,0,2)	-8839
(12,1,2) (11,2,2) (10,3,2) (9,4,2) (8,5,2) (7,6,2) (6,7,2)	10001
(5,8,2) (4,9,2) (3,10,2) (2,11,2) (1,12,2) (0,13,2) (0,14,2)	-8840
(1,13,2) (2,12,2) (3,11,2) (4,10,2) (5,9,2) (6,8,2) (7,7,2)	-83
(8,6,2) (9,5,2) (10,4,2) (11,3,2) (12,2,2) (13,1,2) (14,0,2)	5674
(15,0,2) (14,1,2) (13,2,2) (12,3,2) (11,4,2) (10,5,2) (9,6,2)	-70
19 Exemple de parcours en diagonale pour les valeurs de Cb	*20 Valeurs Cb recueillies dans le parcours diagonale*

1.e.6. L'élégant codage de Huffman.

Il reste encore deux manipulations. La manipulation F est appelée « Codage de Huffman ». N'oublions pas que l'on cherche à minimiser la taille des fichiers sans trop dégrader la qualité de l'image afin de préserver les conditions de l'esthématopée tout en économisant l'encombrement des canaux de transmission. L'algorithme de codage de Huffman est relativement simple dans son principe, et ne s'applique pas qu'aux images, la compression .LZH, par exemple, en est une application répandue. On peut ainsi réduire de 20 à 90% la taille d'un fichier.

Le plus simple est d'expliquer le principe à l'aide d'un exemple alphanumérique. Supposons que nous disposions d'un fichier-source contenant cette phrase idiote :

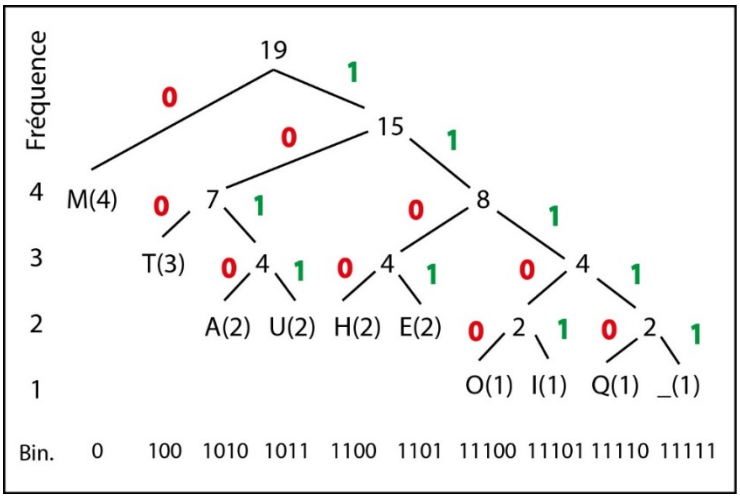
MAMMOUTH_THEMATIQUE

Nous avons 19 lettres, que l'on peut classer selon leur fréquence respective :

Lettre	M	T	A	U	H	E	O	I	Q	_
Fréq	4	3	2	2	2	2	1	1	1	1

Si l'on devait transmettre ce « MAMMOUTH_THEMATIQUE » en codage ANSI, chaque caractère demandant un octet, il faudrait 19 x 8 = 152 bits. L'astuce consiste à coder chaque lettre avec un nombre minimal de bits, les plus fréquentes ayant le code le plus court, les moins fréquentes le code le plus long ; potentiellement, on devrait donc obtenir un nombre inférieur à 152 bits. Comment s'y prend-on ?

En construisant un arbre d'Huffman. On considère que les lettres du tableau *supra* constituent les « feuilles » de l'arbre, et le tout est de remonter à la racine d'un *arbre binaire* (ainsi appelé parce qu'un « nœud », ou embranchement, ne comporte que deux « branches »). L'idée est de partir d'une liste de valeurs de fréquence classées de manière croissante (ou décroissante, tout dépend de la direction de lecture). L'algorithme procède à chaque étape en comparant, de droite à gauche, deux « feuilles » ou deux nœuds de valeur minimale, par exemple _ (1) et Q(1)[27] et les remplace par la fourche de l'arborescence qui aura pour étiquette la somme des valeurs, soit _(1)+Q(1)=2. Et ainsi de suite. On finit, par cumulation successives, par atteindre la racine (19, soit le nombre de lettres de l'exemple). Ensuite, la codification est facile : on affecte la valeur 0 à toutes les branches à gauche, la valeur 1 à toutes les branches à droite, puis on lit l'arbre en sens inverse (racine→feuilles) et l'on reporte séquentiellement les « poids » (ou valeurs) de chaque branche rencontrée jusqu'à atteindre une feuille. Ainsi, dans l'exemple, M de fréquence 4 sera atteint par la seule branche 19→M(4), soit code 0 ; à l'opposé, le parcours 19→_(1) ne rencontrera que des poids de 1, soit code 11111. Notre « MAMMOUTH_THEMATIQUE » sera donc codable :



231 Arbre de Huffman

M	A	M	M	O	U	T	H	_	T	H	E	M	A	T	I	Q	U	E
0	1010	0	0	11100	1011	100	1100	11111	100	1100	1101	0	1010	100	11101	11110	1011	1101

soit sur 65 bits seulement. Comparé au codage ANSI, le rapport est donc de 65/152 ≈ 0,42 (42%). On pourrait peut-être faire mieux... Mais ce n'est pas mal : la longueur moyenne du code est de 64/19 = 3,36 alors que l'entropie de la source[28] est de 3,15, donc, légèrement inférieure. On pourrait encore s'approcher de la valeur de l'entropie en codant des blocs de lettres. On peut donc construire une table :

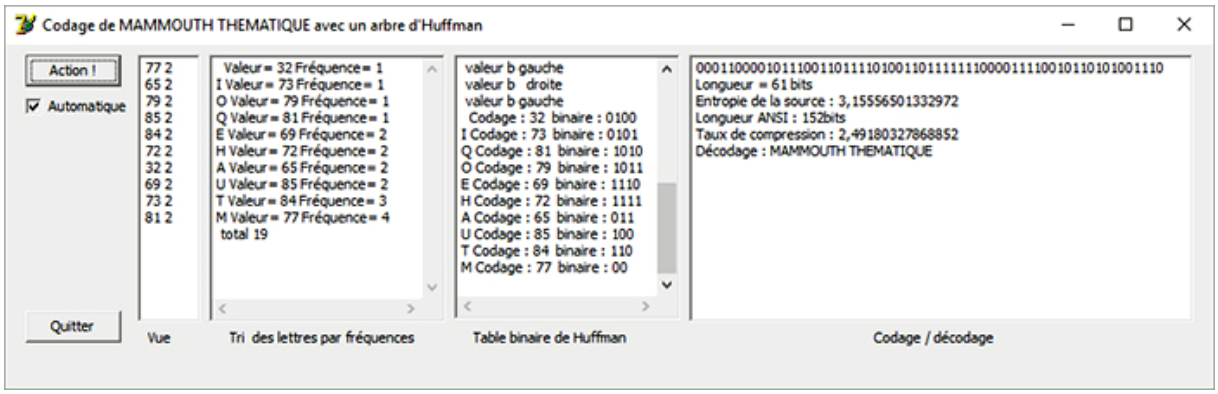
M	T	A	U	H	E	O	I	Q	_
0	100	1010	1011	1100	1101	11100	11101	11110	11111

pouvant servir au décodage. Il n'est alors pas besoin de fractionner la chaîne des 0 et 1, il suffit de parcourir l'arbre dans le sens racine → feuilles pour obtenir une valeur binaire pertinente et de comparer avec la table pour retrouver les lettres. On peut optimiser l'algorithme de construction de l'arbre, car celui que je donne en exemple

[27] Entre parenthèses : le dividende de la fréquence ; Q(1) équivaut à 1/19

[28] L'entropie d'une source composée de N éléments se calcule, connaissant pour chaque élément n de N sa probabilité P(n), selon la sommation $E = -\sum_{n=1}^{N} P(n).log_2\ P(n)$ (nb : log_2 = logarithme de base 2). Voir mon bouquin page 23.

est plus intuitif qu'opérationnel. Une copie d'écran des résultats d'un logiciel que j'ai créé *ad hoc* le montre :



22 Sorties du logiciel Arborescence.exe

Cette fois, le taux de compression, 2.49, est nettement inférieur à l'entropie de la source. L'optimisation de l'algorithme permet de coder chaque caractère sur seulement 4 bits au maximum. La restitution de la « phrase » initiale (décodage) se fait par relecture de la chaîne binaire bit par bit, en augmentant le contenu d'un curseur que l'on compare, à chaque pas, avec la table. L'arborescence d'Huffman sert donc à construire la table, laquelle sert ensuite au codage et au décodage.

L'algorithme convient également aux valeurs numériques, et compte tenu de la redondance considérable du zéro après le codage RLE, on peut espérer une forte compression.

1.e.7. Enregistrement en fichier exploitable.

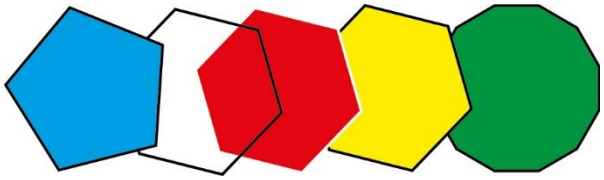
Codages et compression sont maintenant terminés. La dernière étape, G, de l'algorithme JPEG pris en exemple est la constitution d'un fichier exploitable par l'algorithme de restitution. On enregistre les informations utiles dans l'en-tête du fichier : longueur, largeur de l'image, taille de la table de conversion, table de conversion d'Huffman (soit : valeur décimale et équivalent binaire, par exemple : 82 |1010010). On code ensuite le tableau (ou la liste des valeurs Y, Cr, Cb des pixels du tableau) selon la table d'Huffman. et puisque tout se transmet en octets, on découpe l'immense chaine de ces valeurs mises bout à bout en octets (exemple : octet 11011011 est un fragment de la chaîne-code) et on enregistre chaque octet.

À la réception (restauration de l'image), on récupèrera les entêtes et on rétablira la chaîne de code ($\bar{\bar{G}}$), que l'on décodera avec la table d'Huffman ($\bar{F}$) Ensuite, les opérations s'enchaîneront : décodage RLE ($\bar{\bar{E}}$) qui rétablit le tableau de pixels, désapplication des cœfficients ($\bar{\bar{D}}$), transformation cosinus inverse ($\bar{\bar{C}}$avec, encore, un arrondi des valeurs !), rééchantillonnage ($\bar{\bar{B}}$), conversion YCrCb en RGB ($\bar{\bar{A}}$)... et afficher le bitmap ! Les deux étapes de production et de restitution sont symétriques. Avec des pertes d'information.

Les fichiers vidéo de type MPEG sont constitués d'images compressées JPEG encapsulées dans un conteneur comportant aussi la bande son compressée en MP3 ou AAC (advanced audio coding), mais à la différence des films 16 ou 32 mm, l'affichage se fait par comparaison des différences d'une image n-1 avec une image n ; on n'affiche alors que les différences sur fond d'image n-1, à moins que le fond ne soit plus exploitable. Je n'entrerai pas dans le détail, les procédés de compression différant peu de ceux que j'ai expliqués pour le JPEG.

1.f. Les images vectorielles.

Le fichier (JPEG ou MP3 ou MPEG) n'est donc pas un objet « virtuel », mais simplement une série d'éléments codées qui nécessitent un appareillage récepteur pour (re)produire l'image, le son ou la vidéo. Le « monde numérique » n'a rien de virtuel et tout de technique. Bien sûr, la technique peut donner à voir ce qui n'existe pas, les animations numériques (vidéos ou png, par exemple) ne s'en privent pas. Dans le cas d'images dites « de synthèse » (en réalité, c'est du dessin !) on utilise plutôt des techniques de vectorisation[29]. L'image vectorielle est constituée d'objets « primitifs » géométriques (polygones, courbes, segments de droite...) héritant des propriétés générales de leur catégorie (par exemple, un polygone est une figure fermée avec au minimum trois côtés, mais il peut en avoir presque autant qu'on veut jusqu'à ressembler... à un cercle !). L'objet se définit aussi par sa position dans la composition, sa couleur etc... Cette fois, on ne travaille plus avec des pixels, mais simplement avec des formules mathématiques que l'on transmet via le processeur à la carte graphique. Il s'ensuit que le fichier d'une image vectorielle est beaucoup moins encombrant que celui d'un bitmap. On peut transmettre et redimensionner l'image vectorielle sans perte. L'inconvénient majeur tient au fait que l'on ne peut pas coder vectoriellement une image analogique (comme une photographie).

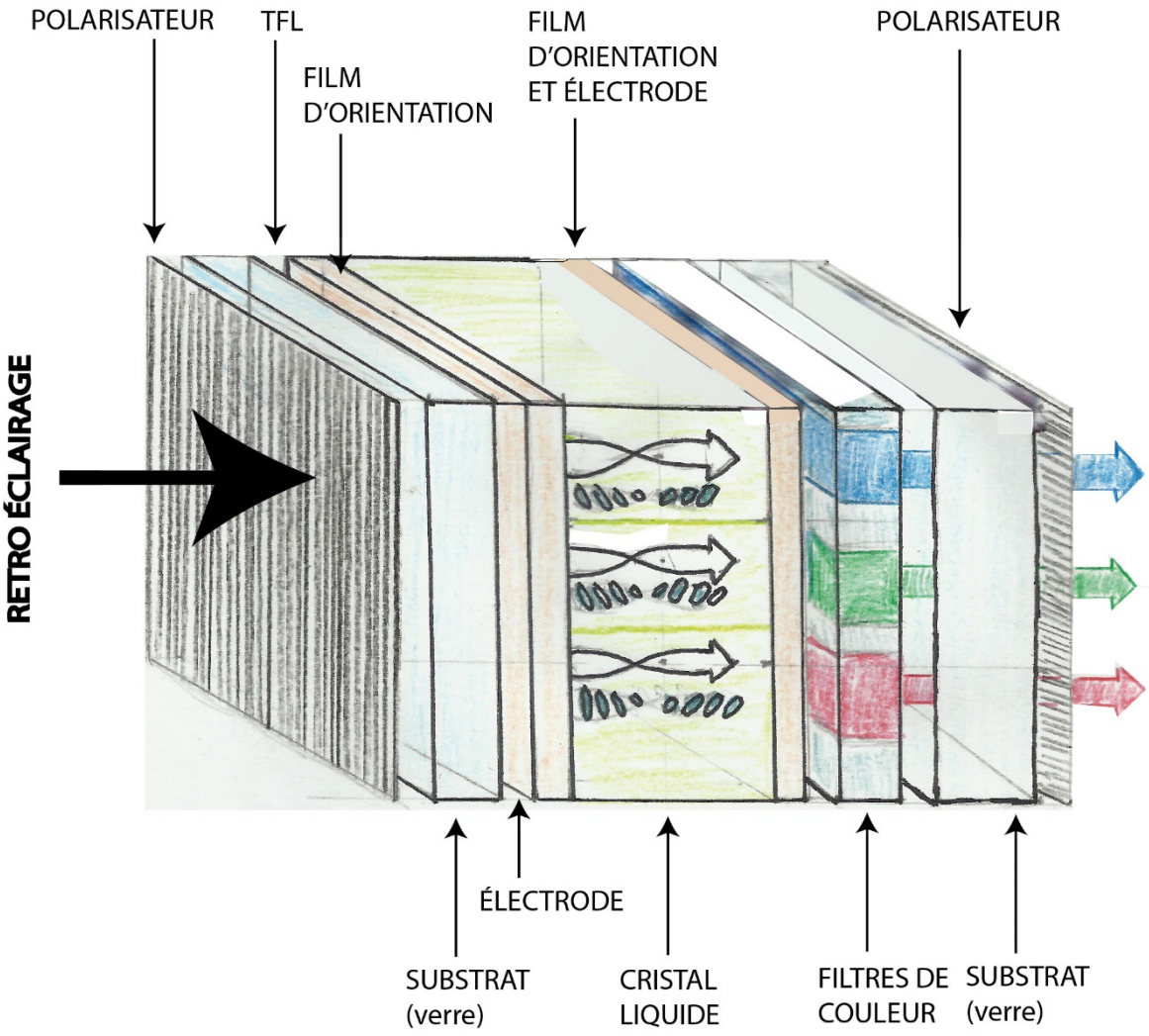


23 Variations sur un polygone vectoriel

1.g. Restitution sur un écran : le matériel.

Examinons rapidement le dernier stade de l'esthématopée. C'est bien sûr l'écran ; nous nous intéressons ici aux écrans à cristaux liquides (ECL, en Anglais LCD). Les cristaux liquides sont constitués de molécules de forme allongée, dont la position est fluctuante. Mais sous l'effet d'un champ électrique, les molécules peuvent s'ordonner et d'exercer une influence sur la lumière. Examinons le schéma ci-dessous :

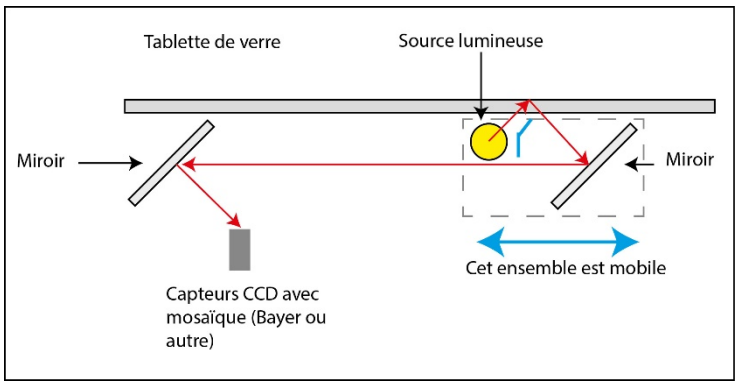
[29] Des logiciels comme Adobe Illustrator permettent de fabriquer de telles images.



24 Schéma d'un pixel LCD

Les deux électrodes situées de part et d'autre du cristal liquide produisent le champ électrique. Les photons lumineux (issus de rétro-éclairage) oscillent de part et d'autre de la direction de propagation de la lumière. Mais on peut les discipliner en polarisant la lumière : la direction de l'oscillation photonique est alors orientée, soit verticalement (polarisateur vertical V), soit horizontalement (polarisateur horizontal H). Si l'on se contentait de faire passer la lumière à travers deux polarisateurs, l'un V l'autre H, rien ne sortirait. Mais si l'on intercale entre ces deux polarisateurs un cristal liquide, les molécules modifieront la polarisation de la lumière en fonction de leur orientation. Une partie de la lumière pourra ainsi traverser le dispositif. En fonction d'un signal électrique, la lumière pourra donc ne pas passer (noir), passer entièrement (blanc) ou partiellement (gris). Trois cellules de cristaux liquides pourront donc être pilotés par trois signaux de polarisation correspondant aux couleurs primaires R, V, B, et la lumière obtenue passera par un filtre de couleur correspondante. Ces filtres étant extrêmement proches les uns des autres, c'est l'œil et les zones cérébrales adéquates qui réaliseront la synthèse additive des couleurs. Voilà donc les rayonnements techniquement produits qui produisent la sensation (esthématopée). On imagine assez facilement qu'à l'origine, une caméra munie d'une grille d'éléments

photosensibles (les *photosites)* ou un scanner ont une structure symétrique de celle d'un écran, et qu'entre ces deux dispositifs se produisent toutes les manipulations que nous avons évoquées plus haut. Les scanners à plat connectés aux ordinateurs individuels analysent ligne par ligne les documents, le pas de lecture déterminant la densité des pixels par pouce (par exemple : 600 dpi). Voilà qui n'est pas sans rappeler les débuts de la transmission par signaux électriques analogiques des documents par des appareils tels le bélinographe évoqué plus haut. La barre de CCD (Charge Couple Devices) économise une lecture colonne par colonne, mais d'autres appareils procèdent par balayage [ligne, colonne], comme lorsque nous lisons les tableaux de pixels.

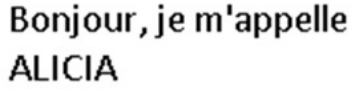


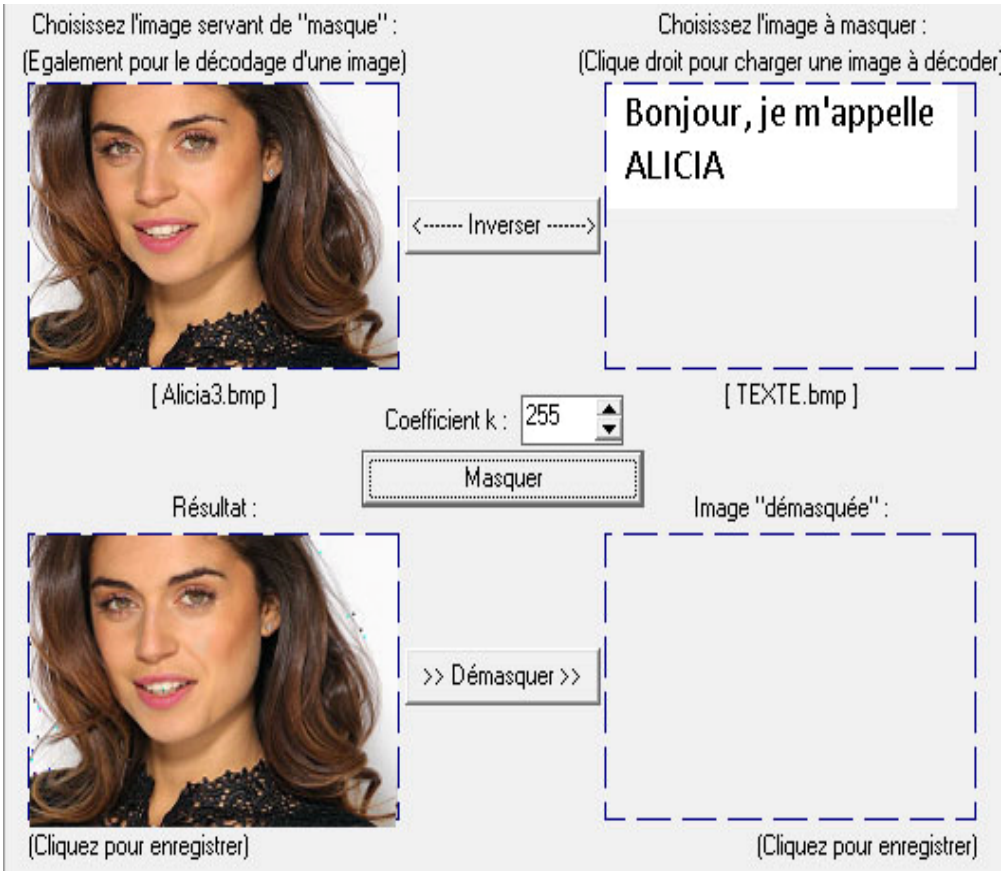
25 Schéma de principe d'un scanner à plat

2. La manipulation stéganographique.

Avant de conclure, notons que les techniques numériques permettent bien évidemment le montage (tout utilisateur de Photoshop le sait), mais prêtent le flanc à des manipulations plus ou moins licites. On songe naturellement aux images subliminales incluses dans une séquence vidéo (par exemple une sur 24 dans un affichage au 1/24 de seconde : avec cela, un fait un « film » au ralenti perçu inconsciemment par le cerveau), mais je préfère évoquer la *stéganographie* permettant de cacher une image ou un message dans une image « truquée ». À titre d'exemple, les images bitmap type .bmp sont faciles à truquer dans la mesure où elles ne sont pas compressées. De fait, la compression JPEG donnerait un résultat plutôt imprévisible, étant donné toutes les manipulations évoquées plus haut. Une image bitmap codée en couleur sur 24 bits offre des possibilités intéressantes d'insertion.

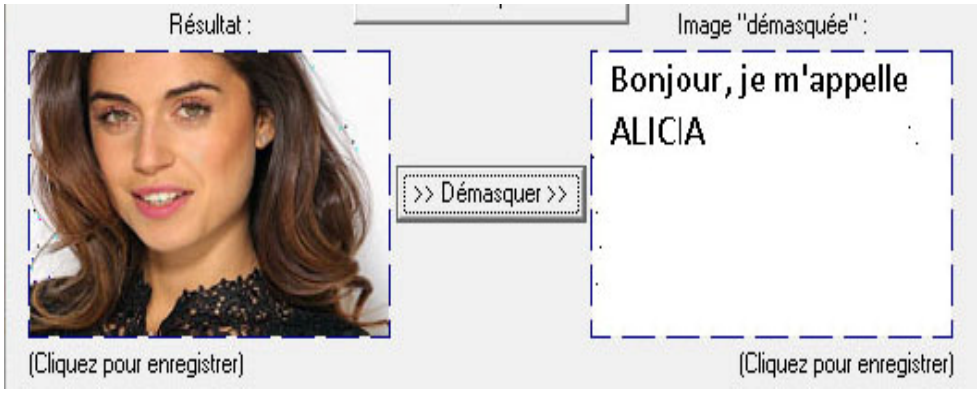
Voulez-vous connaître le prénom de cette personne dont voici la photo originale à gauche ? Il suffit d'y ajouter un autre bitmap contenant le texte à droite. Un logiciel simple permet ce tour de passe-passe. Dans un premier temps, on fusionne les deux images :





26 Séquence de codage et résultat

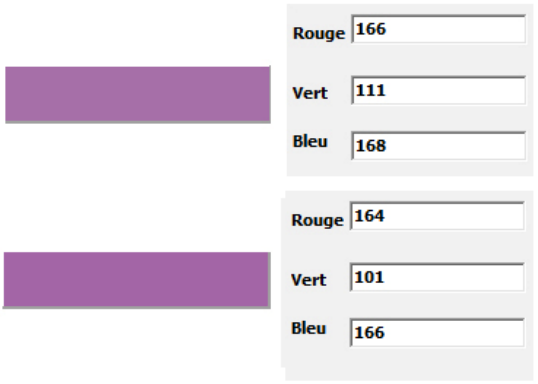
On distingue à peine une différence de luminosité (le résultat paraît légèrement plus pâle que l'original), mais c'est tout. La lecture du message caché se fait en cliquant le bouton « Démasquer » :



27 Séquence de décodage

Comment obtenir cela ? Nous avons noté le fait que l'œil humain ne distingue pas des couleurs très proches, pourtant physiquement distinctes ; voir la figure 16, montrant que, R et B étant égales, une différence de 10 dans le dosage du V n'entraînait pas de différence perceptible entre les deux couleurs produites. C'est précisément là-dessus que jouent les encrypteurs. 111 ou 101 pour V ne font pas de différence perçue. Mon petit programme nuancier me donne encore une valeur très

voisine, peu discernable, avec R=164 (contre 166), V=101 (contre 111), B=166 (contre 168). Évidemment, si l'on introduisait des écarts plus importants, la différence deviendrait perceptible. L'idée est donc d'introduire les différences dans les bits dits « de poids faible » des octets codant la couleur des pixels. Pourquoi « poids faible » ? Un octet organise les bits par puissances de deux, de la manière suivante :



28 Peu de différence physique, pas de différence esthésique

2^7, 2^6, 2^5, 2^4, 2^3, 2^2, 2^1, 2^0. Les bits de poids fort sont affectés d'exposants élevés, les bits de poids faible d'exposants bas. Un bit de 2^0 sera égal à 0 s'il est à zéro, à 1 (l'unité) s'il est à l'état 1. L'exemple de la figure 28 montre qu'en modifiant les bits 0 et 1 (on augmente ou on diminue la valeur de l'octet de 3 (11 en binaire), on n'introduit pas de différence esthésiquement perçue dans la couleur résultante. Ainsi, dans le cas d'une image de 196*127 pixels en RVB, comme c'est le cas des images traitées par le logiciel de la figure 26, on dispose de 74 676 octets soit 597 408 bits, et puisqu'on modifie au plus deux bits par octet, 149 352 bits modifiables, 18 669 octets pour l'image RVB masquée. Le rapport le meilleur entre l'image-hôte et l'image parasite est de ¼. Ce qui, dans notre exemple, autoriserait une image-parasite de 6223 pixels tout de même. Un rapport plus grand, par exemple 1/3, introduirait des différences visibles entre l'image-source et l'image-hôte.

Comment s'y prend-on pour introduire l'image-parasite ? Considérons le A de « ALICIA ». Ici, ma démo sera théorique, car au lieu d'introduire un bitmap dans l'image-hôte, j'introduis une chaîne de caractères[30]. A a pour code ASCII la valeur 65, soit en binaire 1000001 (l'octet sera en fait 01000001). Je découpe ma chaîne en quatre : 01 00 00 01, chaque tronçon remplaçant les bits de poids faible d'un octet dans l'image-hôte. On voit qu'il me faudra quatre octets de l'hôte pour introduire A. Supposons que ces octets aient les valeurs 188, 127, 220, 220, soit en binaire 10111100, 0 1111111, 11011100, 11011100. Le remplacement donnera 10111101, 0 1111100, 11011100, 11011101, soit en décimal 189, 124, 220, 221. Le tour est joué... et l'œil aussi ! Mais on voit bien que le même subterfuge peut être utilisé pour introduire un octet de codage d'une couleur de bitmap-parasite, il faudra toujours quatre octets de l'image-hôte pour introduire un octet du parasite.

Comment démasquer l'image-parasite ? On fait l'inverse. Prenant un à un les octets de l'image-hôte, on accède aux bits de poids faible d'un octet en prenant le résultat de la division entière de la valeur de l'octet par 4 (V_{octet} mod 4). Voyons :

[30] En fait les caractères sont souvent eux-mêmes affichés en bitmaps.

nb binaire	décimal	décimal mod 4	binaire
10111101	189	1	01
1111100	124	0	00
11011100	220	0	00
11011101	221	1	01

On retrouve exactement les valeurs obtenues par découpage de 01000001, soit 65 en binaire, donc « A ». On procède ainsi jusqu'à la fin du code de l'image-parasite, fin qu'il faut connaître en introduisant un code d'arrêt lors du codage. Si l'on a pris soin de stocker les octets reconstitués dans un tableau [I, J, Z], on peut facilement afficher l'image-parasite.

On voit que le numérique offre maintes occasions de duper son monde[31]. Tant qu'il s'agit de protéger le copyright d'une image contre des utilisations abusives, le procédé est acceptable. Mais on devine les emplois illicites voire pervers que l'on peut faire du procédé !

3. Protéger les « adeptas » : le codage du canal.

3.a. Le problème.

Conclusion ? Pas encore ! J'ai décrit le codage JPEG, un exemple de travail industriel. Nous avons vu ce qui le motivait : réduire l'encombrement et obtenir une vitesse de transmission compatible avec les canaux. Ces opérations constituent le « codage de source ». Mais... Qui n'a jamais rencontré des messages d'erreur au chargement d'un MP4 ou d'une image JPEG (ou tout autre) ? Parfois le DOS nous signale un fichier corrompu, d'autres fois on nous dit qu'on ne sait pas lire la vidéo, et l'on doit télécharger un *codec* pour ce faire. Remarquons que « codec » nous met sur la voie : il s'agit de décoder un code, qui n'est pas celui du document-source, mais très exactement ce que l'on appelle le *codage du canal.* Appelons « canal » tout moyen permettant d'acheminer le document-source.

Selon le schéma de la *transduction* [32] appliqué à notre propos, nous avons :

Source → Codage de source→ Émission → Réception → Décodage → Restitution

S1 ----- vecteur ---------- S2

Le vecteur, c'est le signal transmis et reçu via un canal. Tous les « info-com » vous diront que c'est lui qui porte l'*information*, mais en fait il n'est que modulation

[31] Je disais plus haut que le cryptage stéganographique ne convient pas aux JPEG : mais certains travaillent à contourner la difficulté !

[32] Voir mon bouquin pages 86-95

permettant de reconstituer l'information. Ce vecteur, ce peut être un disque dur, une clef, un DVD-Rom, un flux d'impulsions électromagnétiques ou lumineuses portées par un câble ou une onde électromagnétique type WiFi et ainsi de suite. Le vecteur peut subir des perturbations, par exemple des coupures, inductions parasites, transmodulation etc... C'est ce fameux « bruit » qui hante les esprits des ingénieurs. Bon : on comprend que si le signal est perturbé, le décodage est compromis. C'est pourquoi il faut se donner un moyen de contrôler la validité des codes reçus. D'où le codage de canal. Jusque-là, nous avions cherché à minimiser la taille des signaux... mais le contrôle de validité impose son augmentation ! Comment cela ?

3.b. des solutions peu satisfaisantes.

Imaginons un octet transmis de valeur 142, soit 10001110 en binaire. à la réception, on reçoit 10101110 soit 174 ; la différence est importante, mais comment savoir si c'est le bon code ? Une idée serait de coder en plus de l'octet un bit de parité : si la somme des bits de l'octet est un nombre pair, on peut mettre le bit de parité à 1, zéro sinon. Comme ceci : 10101110 0000000, le second octet étant codé par 142 mod 2 = 0. Si, au lieu de cela, je reçois un nombre pair comme 174, l'erreur n'est pas détectée ; en revanche, si je reçois 10001111, avec le bit de parité à zéro, je me rends compte que ça ne va pas, car 10001111 = 142 et 142 mod 2 =1. Une erreur est détectée. J'expose comme cela juste pour le principe, en fait le codage de parité est bien basé sur ce procédé, mais en réalité on enverrait non pas deux octets, mais un « novet » de neuf bits codé 100011110, le bit de poids faible étant le bit de parité. En fait, on s'arrange pour transmettre un codage sur 7 bits, le huitième étant le bit de parité. De toute façon, ce ne serait pas un système bien fiable, puisque l'erreur pourrait passer inaperçue (voir 174 au lieu de 142) et, de toute façon, rien ne me dirait ni où est l'erreur ni comment la corriger !

Alors on a inventé des systèmes codant à la fois la parité et offrant la possibilité d'une correction éventuelle. On peut, par exemple, coder par répétition triple de chaque bit : 10001110 devient 111000000000111111111000. Si je reçois 110000 etc.., j'en déduis qu'il y a une erreur sur le bit 2^7 car je devrais avoir 111 et non pas 110... d'où correction possible. Mais cela prend de la place.

3.c. Le codage de Hamming.

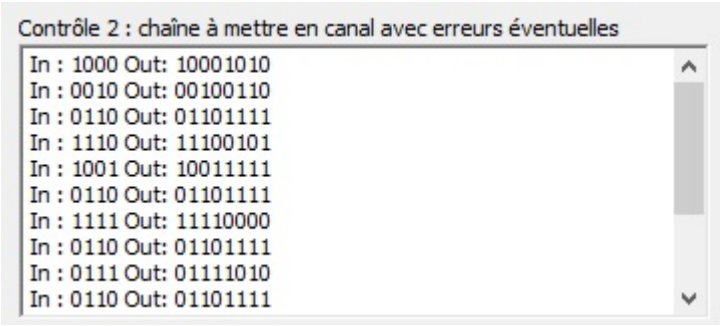
L'une des méthodes les plus efficaces est celle de Hamming. Passons sur l'aspect théorique, et regardons rapidement (!!!) comment ça se passe. Supposons que je veuille transmettre le mot « Avion ». Il se compose de 5 octets, soit 40 bits. Prenons la lettre A (la manœuvre sera la même pour toutes les lettres), code ANSI = 65 soit 01000001 en binaire. On va d'abord inverser le code binaire (« anti-dump ») soit 10000010 et diviser l'octet en deux quartets (4 bits) : 1000|0010. Occupons-nous du premier quartet, la manœuvre sera la même pour le second.

Je vais « signer » le quartet 1000, de la manière suivante : je prends chaque bit du quartet que je place dans un enregistrement (appelons-le HI) de quatre champs booléens. Si le bit n du quartet est à 1, le champ HI[n] sera à « vrai », sinon à « faux ».

Maintenant, je vais transférer les valeurs de HI dans les quatre premiers champs d'un enregistrement (appelons-le HO) de huit champs booléens. Donc pour n de 1 à 4, je fais HO[n] := HI[n]. Il reste les champs de 5 à 8 de HO. Là, je ne peux pas faire plus simple que de donner le code (en Pascal) :

```
HO[5]:=HI[1] xor HI[2];
HO[6]:=HI[3] xor HI[4];
HO[7]:=HI[1] xor HI[3];
HO[8]:=HI[2] xor HI[4];
```

La fonction logique Xor est le "ou exclusif" [33]. On peut vérifier avec l'exemple du quartet 1000 que la signature (booléennes de 5 à 8) est vrai, faux, vrai, faux, ce qui se traduit facilement en 1010. J'ai donc obtenu, après conversion des champs booléens de HO en écriture {1,0}, un octet 10001010. La figure 29 montre le résultat du traitement de plusieurs quartets mis en octets avec la signature. L'ensemble à transmettre passe donc d'une chaîne de 40 bits (5 octets) à une chaîne de 80 bits (10 octets), et telle est la chaîne que l'on va transmettre. C'était bien la peine de chercher à compresser ! Si, pourtant, car une image non compressée traitée par le code Hamming prendrait une place démesurée !



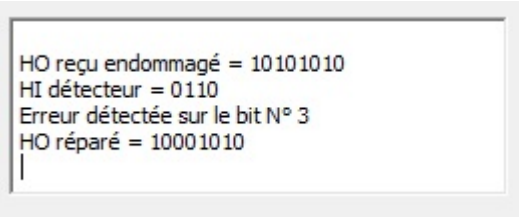
29 Résultat de codage de Hamming

Maintenant, comment le récepteur peut-il déceler une erreur ? Facile : puisque la signature a été obtenue à l'aide des bits du quartet, si elle n'est pas compatible avec le premier quartet d'un octet reçu, c'est qu'il y a erreur. L'avantage de la méthode est qu'il n'y a pas besoin d'envoyer de table de correspondance, le calcul suffit. Reprenons l'étude de notre A. Au lieu de 10001010, supposons que j'aie reçu 10101010 par altération du bit n°3. Un dispositif de détection d'erreur bien pensé ne se contente pas de signaler une erreur, mais il la trouve et la corrige. La série d'instructions :

```
HI[1]:=HO[1] xor HO[2] xor HO[5];

HI[2]:=HO[3] xor HO[4] xor HO[6];

HI[3]:=HO[1] xor HO[3] xor HO[7];

HI[4]:=HO[2] xor HO[4] xor HO[8];
```

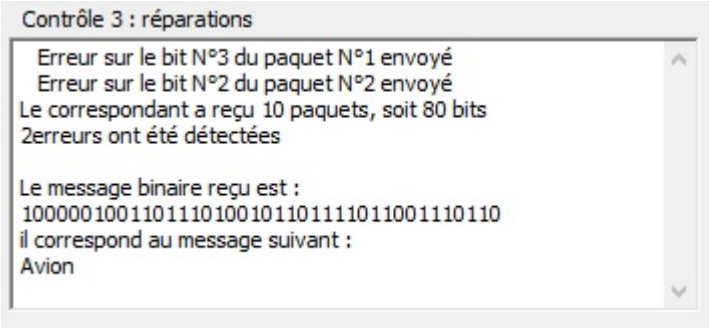
va produire 0110, qui au moyen d'un traitement mathématique adéquat indiquera quel est le bit fautif. Il suffira alors simplement d'inverser sa valeur booléenne pour récupérer le quartet d'origine.

[33] Le Ou exclusif comparant deux valeurs booléennes a et b est « vrai » que si et seulement si a=vrai et b=faux ou a=faux et b=vrai ; donc il est faux si a=faux et b= faux ou a=vrai et b=vrai.



30 Réultat de la correction Hamming

Ce code appelé (8,4) permet de relever plusieurs erreurs et de les corriger :



31 Détection et corrections de plusieurs erreurs. Hamming

Certes, tout cela encombre les canaux, exige des programmes de codage/décodage-correction, mais que l'on songe que dans certains dispositifs une erreur portant sur un seul bit peut provoquer des catastrophes (par exemple perdre une fusée pleine de coûteux satellites). Sans parler des effets quantiques qui commencent à se faire sentir sur les ordinateurs très rapides où le doute peut porter sur l'état d'un bit : WANTED, SCHRÖDINGER'S CAT DEAD AND ALIVE !

Index	*Symbole (8 bits)*	*Code (14 bits)*
100	01100100	01000100100010
101	01100101	00000000100010
102	01100110	01000000100010
103	01100111	00100100100010
104	01101000	01001001000010
105	01101001	10000001000010
106	01101010	10010001000010
107	01101011	10001001000010
108	01101100	01000001000010
109	01101101	00000001000010

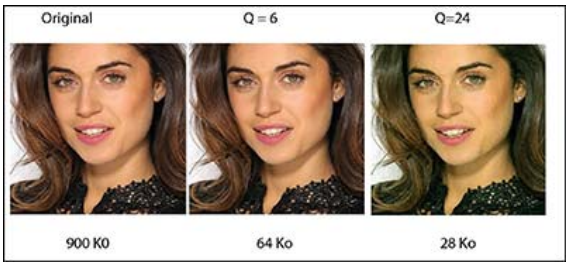
32 Extrait de la table de codage EFM

Encore le codage de Hamming est-il élégant par rapport à celui utilisé pour les CD et DVD, comme le montre la table de codage EFM pour quelques nombres (voir ci-contre figure 32). Car ces engins-là (les CD et DVD) posent des tas de problèmes. Je n'en parlerai pas ici, ayant déjà bien trop parlé, mais si le besoin s'en faisait sentir...

Conclusion :

Les techniques ici exposées se transposent assez facilement aux codages (puisqu'il y en a plusieurs en cascade) des sons, des écrits etc... La compression a des limites physiques au-delà desquelles l'esthématopée recherchée est compromise. Par exemple, l'algorithme JPEG fonctionne bien si l'on se contente d'une qualité de

compression raisonnable (en moyenne de Q= 6) mais le résultat se dégrade rapidement :



au-delà de Q= 40, on décèle les défauts non seulement dans le rendu des couleurs, mais aussi dans celui des traits.

En fait, ce qui se joue, c'est le respect des limites physiologiques de la perception (dans le cas des images comme dans celui des sons, voire du mouvement). Au passage, remarquons que les ingénieurs bénéficient des connaissances de la neurologie en matière de Gestaltung, c'est-à-dire de fonctions animales de représentation, proprioperception etc... On sait qu'ils savent aussi commander des prothèses par numérisation de l'influx des nerfs-moteurs. Bref : on est strictement dans le domaine des fonctions -que les neurologues connaissent assez bien maintenant- mais nullement dans le domaine des *facultés* proprement humaines. C'est une des raisons qui me font douter (avec soulagement !) des croyances au « transhumanisme » : on peut accroître certaines fonctions humaines (en fait, ce sont les industries dynamiques vieilles comme notre capacité à analyser de l'Outil), je ne pense pas que nous soyons même à l'aube d'une amplification de nos facultés par la grâce du numérique, même si un ordinateur calcule plus vite que nous. En d'autres termes, je ne crois pas que l'on soit pour le moment en mesure d'améliorer notre cerveau en lui implantant un processeur.

Revenons un instant sur cette idée du *virtuel*. Tout ce qui précède tend à montrer qu'il n'y a jamais rien de virtuel dans toutes ces opérations. Il y a un matériau de base, que ce soit une tension électrique, un flux de photons, un arrangement de spins électroniques, des creux et des bosses dans un CD, voire -c'est une recherche en cours chez IBM- des séquences A, G, C, T sur des chaînes d'ADN ; matériau choisi comme tel parce qu'analysé comme tel. Ce qui fait qu'aucune image numérisée, aucun son numérisé, n'est virtuel. J'insiste : il s'agit de vecteurs, modulés par un traitement de l'information, exploitables pour restituer de l'information, mais ne véhiculant pas un contenu d'information. Car s'il faut à un bout des dispositifs de capture, de codage, de compression, cela ne donne rien s'il n'y a pas à l'autre bout des dispositifs symétriques de réception, décompression, décodage pour refaire de l'information. Laquelle n'est jamais celle du départ, nous avons bien assez rencontré au cours de ce long exposé des pertes et distorsions. Tout est là. Le fichier, le flux numérique, ne sont que des pièces *matérielles* d'une machine

Maintenant, il est clair, je pense, que toutes ces manipulations techniques ont quelque chose à voir avec du savoir scientifique. Nous ne devons pas tomber dans le piège de la « *pensée technique* », cependant. Il est vrai que, par exemple en présentant le codage de Hamming, j'aurais pu développer la théorie mathématique des *matrices génériques* nécessaires au codage : j'ai préféré l'algorithme, comme je l'ai fait plus haut pour la TDC ; s'il y a une parenté entre l'équation et l'algorithme, ils ne relèvent pas du même plan de rationalité (encore l'écriture fait-elle de l'équation une « machine à

calculer » !). Nous verrons que si les algèbres jouent un rôle *théorétique* (représentation de l'art) et *épitactique* (« recette »), c'est finalement l'outil qui impose son propre mode d'emploi.

Je ne sais pas ce qu'un ergologue pourra faire de tout ce que j'ai raconté. Pour le moment, tout ce savoir résiste à l'analyse ergologique, mais tôt ou tard une faille se révèlera.

Dans la prochaine causerie, j'aborderai la numérisation des signaux analogiques et, à proprement parler, « l'outil » numérique.