Transformée en cosinus discrète et Compression d'images

L'étude que nous allons faire ici est purement expérimentale. Elle ne fait pas la théorie de la transformation en cosinus discrète (*DCT*) ni celle de son lien avec la transformée de Fourier discrète (pour cela consulter par exemple [VAS]). Elle va se faire en deux temps : d'abord l'étude de la *DCT* en une dimension, en la faisant jouer sur un tableau de nombres (ou vecteur à N composantes), puis ensuite l'étude de la *DCT* en deux dimensions.

1) Courbes et sinusoïdes

Une sinusoïde est une courbe dont l'équation est de la forme $f(t) = A \sin(\omega t)$.[1] Sa période est T telle que $\omega = 2\pi / T$ ou $T = 2\pi / \omega$, et sa fréquence est F telle que $\omega = 2\pi F$ ou $F = \omega / (2\pi)$. La courbe oscille entre A et $-A$, A étant appelée amplitude, et ce phénomène se répète sur chaque longueur de période T. Plus la période est petite, plus la fréquence est grande, plus les oscillations sont resserrées.

Evidemment, la somme de plusieurs sinusoïdes donne une courbe périodique. Par exemple prenons ces trois sinusoïdes :

$f_1(t)= \sin(2\pi\, t\, /64)$, $f_2(t)= \sin(2\pi\, 5\, t\, /64)$, $f_3(t)= \sin(2\pi\, 20\, t\, /64)$, puis leur somme :
$f(t) = \sin(2\pi\, t\, /64) + \sin(2\pi\, 5\, t\, /64) + \sin(2\pi\, 20\, t\, /64)$

Pour les tracer, nous avons pris les valeurs entières de t allant de 0 à 63, ce qui donne 64 points de chaque courbe, puis ces points successifs sont joints par de petits traits. Le fait de remplacer une courbe présentant un tracé continu par une succession de points régulièrement espacés est appelé « échantillonnage ». On obtient ainsi une période de la première courbe, cinq périodes pour la deuxième, et 20 pour la troisième.

Puis ajoutons les trois fonctions précédentes. On trouve une nouvelle courbe d'équation $y = f(t)$, elle aussi périodique (*figure 1 à droite*, avec l'ajout d'une constante 2 supplémentaire, ce qui fait simplement monter la courbe de 2 unités verticalement).

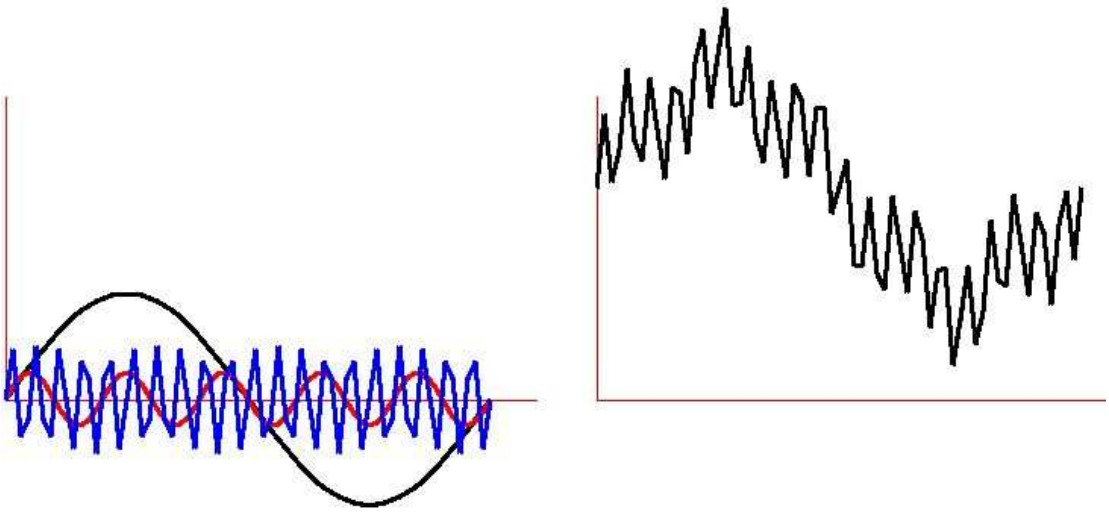


Figure 1 : *A gauche*, les trois sinusoïdes dessinées à partir de 64 points, *à droite* leur addition, avec une constante (égale à 2) ajoutée en plus, ce qui donne des y tous positifs.

[1] En fait l'équation générale est $f(t) = A \sin(\omega t + \varphi)$ où φ est la phase, mais celle-ci n'intervient pas dans notre problème.

Mais le plus important est le phénomène inverse : Une courbe périodique peut être remplacée par une somme de sinusoïdes. Et l'on a les moyens de connaître ces sinusoïdes. C'est le fondement de ce que l'on appelle la transformée de Fourier. Mais cette transformation utilise des nombres complexes et présente parfois des effets de bord gênants. Ces derniers sont provoqués par le fait que l'on prend une courbe dans une zone finie et qui n'est pas périodique, et que la transformée de Fourier utilise la répétition périodique infinie de cette courbe, ce qui provoque des discontinuités (*figure 2*).

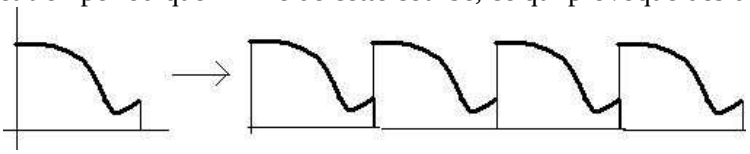


Figure 2 : La courbe initiale *à gauche* est remplacée par sa répétition périodique (*à droite*) lorsque sa transformée de Fourier est calculée

Aussi préfère-t-on utiliser une variante, la transformée en cosinus discrète (ou *Discrete Cosine Transform*, *DCT* en anglais).

2) DCT, ou transformée en cosinus discrète

Partons d'une courbe d'équation $y = f(t)$, que l'on remplace par une succession « discrète » de points de coordonnées $(n, f(n))$ avec n variant de 0 à $N - 1$, ce qui donne N points de cette courbe. A partir de là, on calcule sa transformée en cosinus, par la formule :

$$DCT(k) = \frac{2c(k)}{N} \sum_{n=0}^{N-1} f(n) \cos\frac{(2n+1)k\pi}{2N}$$ avec $c(0) = 1/\sqrt{2}$ et $c(k) = 1$ pour $k \neq 0$.[2]

Avec k variant de 0 à $N - 1$, on trouve ainsi N nombres.

Cette formule fait intervenir N sinusoïdes de la forme $\cos((2n + 1)k\pi/(2N))$, une pour chaque valeur de k. Sur chacune d'elles, on prend N points, lorsque n va de 0 à $N - 1$ (*figure 3*). C'est à partir de ces points que les calculs sont faits, et l'on obtient une nouvelle courbe « discrétisée », avec ses points $(k, DCT(k))$.

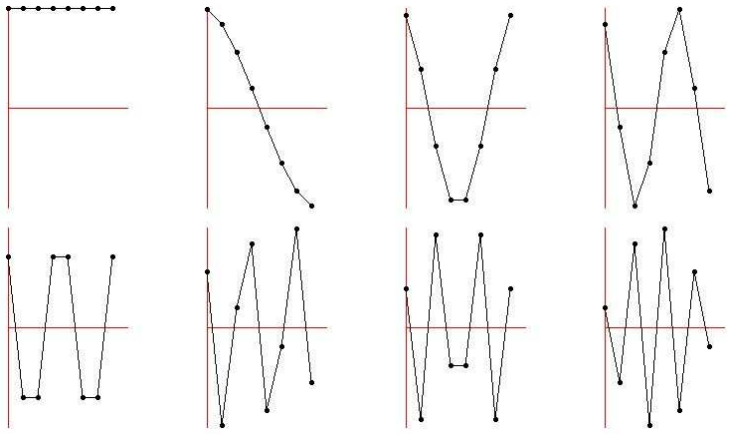


Figure 3 : Pour $N = 8$, les huit sinusoïdes simplifiées en une succession de 8 points joints par des traits

On est ainsi passée de la « zone temporelle » (la variable t pouvant désigner le temps) avec le signal initial et ses points $(n, f(n))$, à la « zone des fréquences », avec ses points $(k, DCT(k))$. En effet, la *DCT* fait ressortir les fréquences des sinusoïdes sous-jacentes au signal initial. Constatons-le en

[2] Il existe plusieurs variantes de cette formule. Nous avons choisi ici la formule donnée dans le document de référence [WEN1977]

reprenant la courbe telle que $f(t) = \sin(2\pi\, t\, /64) + \sin(2\pi\, 5\, t\, /64) + \sin(2\pi\, 20\, t\, /64)$ avec ici $N = 64$. La « courbe » des points $(k, DCT(k))$ fait apparaître les trois fréquences des sinusoïdes avec leurs amplitudes respectives, sous la forme de variations verticales pour les trois abscisses correspondant aux fréquences (*figure 4*).

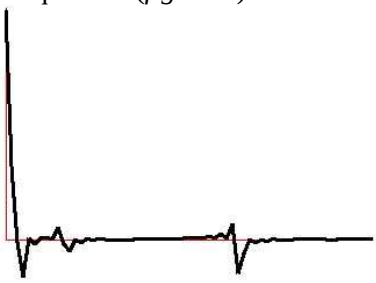


Figure 4 : La *DCT* fait ressortir les trois fréquences du signal initial avec leurs amplitudes respectives

L'intérêt de cette transformation est qu'elle est inversible, et qu'après l'avoir inversée, on retrouve le signal initial, grâce à cette formule :

$$f(n) = \sum_{k=0}^{N-1} c(k)\ DCT(k) \cos\frac{(2n+1)k\pi}{2N}$$

Vérifiions-le sur l'exemple précédent (*figure 5*).

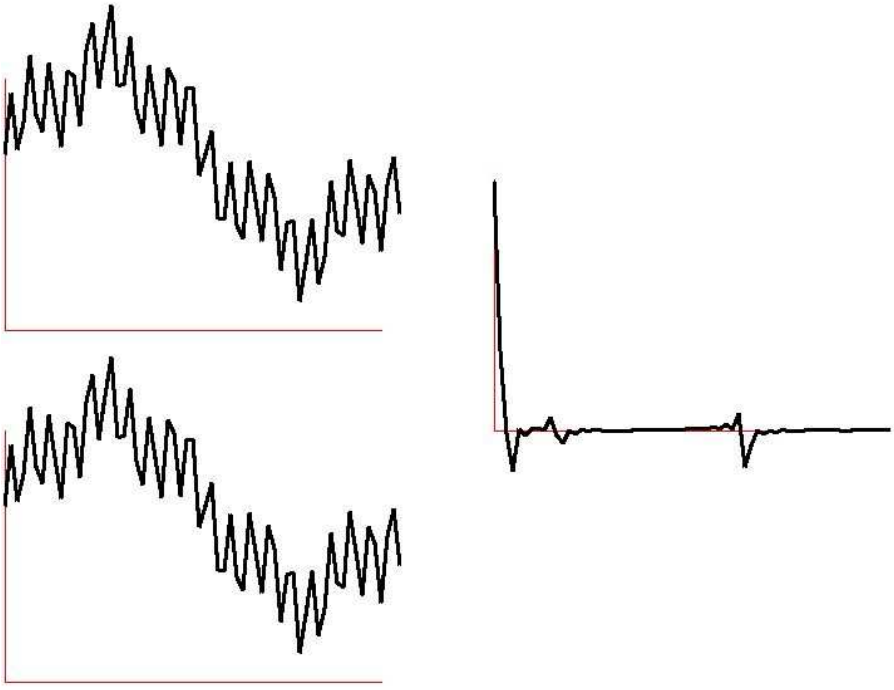


Figure 5 : *En haut* le signal initial, *à droite* la transformée en cosinus *DCT*, et *au-dessous* la reconstitution du signal initial à partir de la *DCT*

Cela s'applique à n'importe quel signal initial, notamment à des courbes qui, contrairement à l'exemple précédent, ne possède pas de petites oscillations multiples (correspondant à de grandes fréquences), comme celle de la *figure 6*. Un phénomène remarquable apparaît alors. Pour ce genre de courbes, seules les basses fréquences observées sur la *DCT* sont importantes, les hautes fréquences étant quasiment négligeables. Là intervient la notion de corrélation entre les points successifs de la courbe.

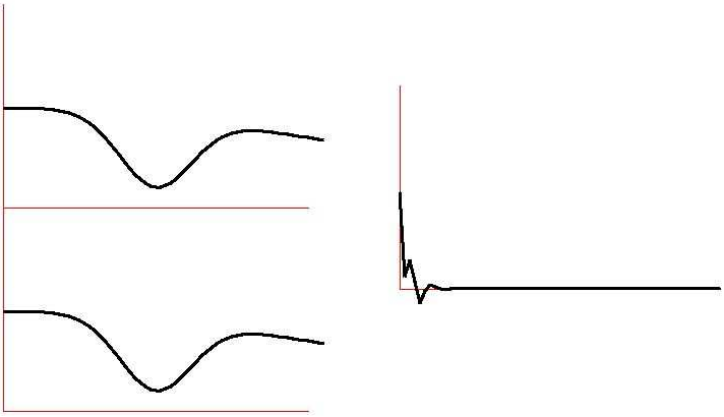


Figure 6 : Le signal initial, sa DCT, puis sa reconstitution

La notion de corrélation

Cette notion fait référence à l'intensité de la liaison qui existe entre les nombres successifs $f(n)$ de la fonction f que l'on a choisie. Comme on vient de le voir, plus cette corrélation est forte, avec des variations faibles entre points successifs, plus les basses fréquences ressortent, au détriment des hautes fréquences. Or c'est en général le cas lorsque les nombres $f(n)$ sont les indices des couleurs des points successifs sur une ligne prise sur une image. C'est cela qui va être mis à profit pour la compression d'images.

Exercice

Prendre N = 8 et pratiquer la DCT sur les trois cas suivants des valeurs de f(n) (pour n allant de 0 à 8)

- $f(n) = 20$. Comme c'est une constante, la période de la fonction est infinie et la fréquence est 0, avec une forte amplitude. On trouve effectivement comme transformée par la *DCT* :
  28,3 0 0 0 0 0 0 0.
- $f(n) = 4 \times n$. On obtient après transformation : 19,8 -12,9 0 -1,3 0 -0,4 0 -0,1
- $f(n)$ est une suite de nombres pris au hasard : 62 5 17 5 83 7 28 25. Après transformation, cela donne : 40,5 2,5 0,3 17 21,4 -5,7 -1,5 25.

Le premier cas présente la meilleure corrélation possible entre les données, d'où la présence de 0 partout lors de la *DCT*, sauf le premier nombre. Le deuxième cas traite une suite de nombres en bonne corrélation, d'où la petitesse des hautes fréquences lors la *DCT*, les quatre derniers termes étant proches de 0. Le dernier cas, avec sa faible corrélation entre les nombres successifs, fait apparaître des basses fréquences aussi bien que des hautes.

3) Compression d'images ligne par ligne

Prenons une image en niveaux de gris, avec les couleurs numérotées de 0 à 255, du blanc au noir (et non l'inverse).[3] Chaque ligne de l'image est formée d'une suite de nombres (de 0 à 255) correspondant aux couleurs des pixels successifs. C'est cela qui constitue notre signal initial, et il y en a un par ligne. Grâce à la *DCT*, on peut le transformer en succession de fréquences, dans l'ordre croissant. Comme certains résultats de la *DCT* sont négatifs, on les met en valeur absolue, de façon à associer à ces nombres positifs ou nuls des niveaux de gris pour les dessiner. Les lignes successives

[3] Si l'image est en couleurs, on traite séparément chacune des trois composantes RGB comme on le fait ici pour les niveaux de gris. Et si l'on a pris dans le cas présent des niveaux de gris allant du blanc au noir (indices de 0 à 255) c'est simplement pour que la concentration des basses fréquences soit mise en noir, et donc mieux visible (*figure 7 à droite*).

des fréquences, ainsi dessinées l'une après l'autre, forment un tableau rectangulaire de la même dimension que l'image. On constate une forte concentration du côté des basses fréquences (*figure 7*).

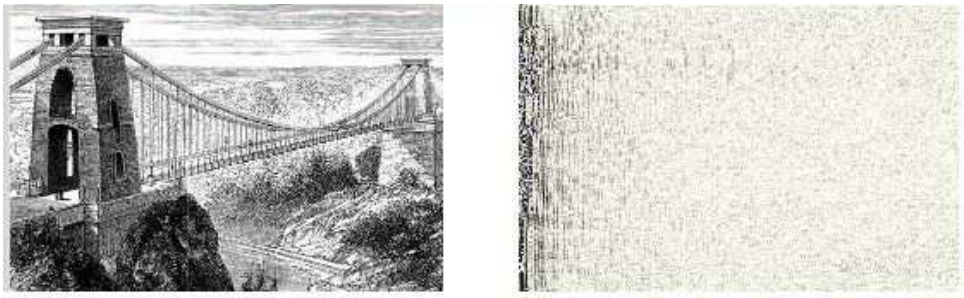


Figure 7 : *A gauche* l'image de dimensions 256x170 en niveaux de gris, avec *N* = 256, *à droite* le tableau des fréquences ligne par ligne donné par la *DCT*

Programme pour visualiser la DCT de l'image

On commence par prendre une image sur Internet ou ailleurs, grâce à une capture d'écran. Cette image est au format JPEG, mais pour se faciliter la tâche en SDL, on la convertit au format BMP, ce qui évite d'utiliser la bibliothèque *SDL_image*.

```
/** image récupérée avec son nom « imageniveauxgris» et placée en haut à gauche de l'écran */
   SDL_Surface* bmp = SDL_LoadBMP("imageniveauxgris.bmp");
   SDL_Rect dstrect;
   dstrect.x = 0 ; dstrect.y = 0;
    SDL_BlitSurface(bmp, 0, screen, &dstrect);

/** image mise dans un rectangle de 256 sur 170, avec récupération des niveaux de gris */
    for(i=0;i<N;i++) for(j=0;j<170;j++)
      { color=getpixel(i,j);
        SDL_GetRGB(color,screen->format,&indgris[i][j],&indgris[i][j],&indgris[i][j]);
        indgris[i][j]=255-indgris[i][j]; /*inversion de l'ordre des couleurs, le blanc mis à 0 */
      }
    SDL_FillRect(screen,0,white);
  /** dessin de l'image sur l'écran */
    for(i=0;i<N;i++) for(j=0;j<170;j++)
    putpixel(i,j,col[indgris[i][j]]);
/** les N = 256 sinusoïdes de base */
    for(k=0;k<N;k++) for(n=0; n<N;n++)
    co[k][n]=cos(M_PI*k*(float)(2*n+1.)/(float)(2.*N));
/* DCT */
    for(ligne=0;ligne<170; ligne++)
     {for(k=0;k<N; k++)
       { cumul=0.;
         for(n=0; n<N;n++)  { cumul+=(float)indgris[ligne][n]*co[k][n];  }
         if (k==0) cc[k]=1/sqrt(2.); else cc[k]=1.;
         DCT[k]=2.*cc[k]*cumul/(float)(N);
        }
      /* pour le spectre des fréquences, on prend la valeur absolue et on multiplie par 10
         afin d'accentuer les couleurs */
      for(k=0;k<N;k++) putpixel(300+k, ligne, 10*col[abs(DCT[k])] );
     }
```

Le principe de la compression découle de ce qui précède. Il suffit d'éliminer les hautes fréquences dont le rôle est très faible. On profite du fait que l'œil humain n'est pas très sensible aux hautes fréquences. Le fait de les supprimer ne modifie pas la perception que l'on a de l'image et celle-ci

prend beaucoup moins de place en mémoire. Pour la reconstituer il suffit d'appliquer l'inverse de la *DCT*, ce qui donne une image très proche de l'image d'origine.

Mais comment éliminer ces hautes fréquences proches de 0 ? On utilise pour cela le procédé de la « quantification ».

La quantification

Une méthode consiste à diviser les résultats de la *DCT*, qui sont des nombres flottants, en les divisant par un nombre q (par exemple $q = 5$ ou 10, voire plus) puis à prendre l'entier le plus proche. En procédant ainsi, on trouve des nombres entiers dont une partie relativement importante est tombée à 0. Puis on reconstitue l'image à partir de là, en multipliant les niveaux de gris par q.

Résultats

En prenant $N = 256$, ce qui est la longueur de l'image, on trouve une image reconstituée assez proche pour $q = 5$, après avoir éliminé à peu près les deux tiers des nombres –ceux mis à 0 (*figure 8*). Comme l'image initiale présente beaucoup de détails et peu de zones d'un blanc uniforme, la compression est limitée, et le dessin final un peu flou.

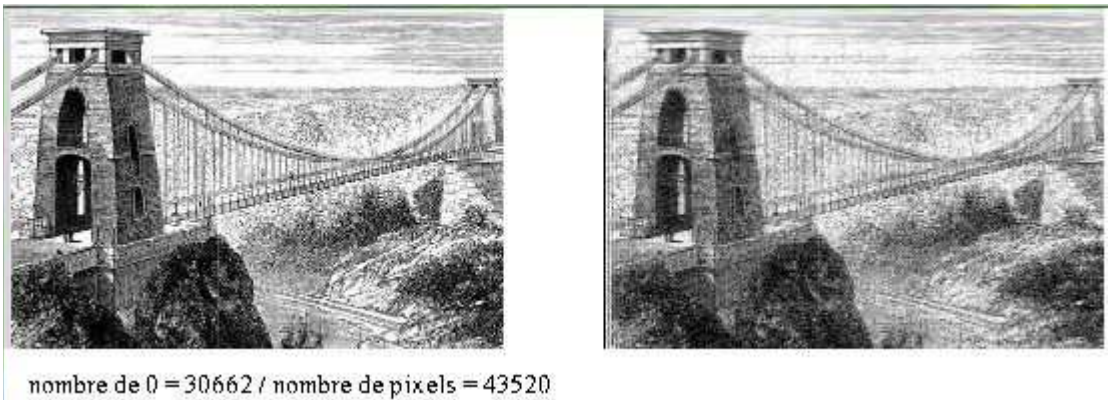


Figure 8 : L'image originelle à gauche, et sa reconstitution approchée à droite.

Programme de la DCT et de son inverse

Au début du programme est placée la récupération de l'image BMP et de ses niveaux de gris, comme on l'a fait dans le programme précédent. Puis on a pris un carré 8 sur 8 de cette image que l'on dessine en remplaçant les pixels par de petits carrés pour la voir plus clairement. Le tableau des indices de ses niveaux de gris est affiché. Puis le programme se charge de traiter l'évolution :

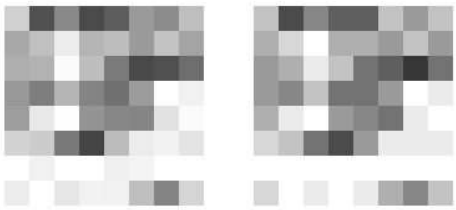
image initiale → DCT avec quantification → DCT inverse et restitution de l'image

```
/******   les sinusoïdes de base pour DCT    *****/
for(k=0;k<N;k++)  for(n=0; n<N;n++) co[k][n]=cos(M_PI*k*(float)(2*n+1.)/(float)(2.*N));
/****  DCT et IDCT  ****/
q=20; nombre0=0;
  for(ligne=0;ligne<8; ligne++)
 {   /**** DCT ****/
   for(k=0;k<N; k++)
     {  cumul=0.;
        for(n=0; n<N;n++)  { cumul+=(float)indgris[n][ligne]*co[k][n]; }
        if (k==0) cc[k]=1/sqrt(2.); else cc[k]=1.;
        DCT[k]=2.*cc[k]*cumul/(float)(N);
        if (DCT[k]>=0.) DCT[k]=(float)((int)( DCT[k]/(float)q+0.5));   /* quantification avec q = 20 ici */
        else DCT[k]=(float)((int)( DCT[k]/(float)q-0.5));
```

```
  }
  /**** inverseDCT (avec multiplication par q)****/
 for(n=0;n<N; n++)
  { cumul=0.;
    for(k=0; k<N;k++)
       { if (k==0) cc[k]=1./sqrt(2.); else cc[k]=1.;
         cumul+=cc[k]*DCT[k]*co[k][n];
       }
     IDCT[n]=cumul;
  }
 for(n=0;n<N;n++) carre(n+10,ligne, col[ q*(int)(IDCT[n]+0.5)] ); /* dessin du carré reconstitué */
}
```

Résultats sur un carré de 8 sur 8

Nous avons pris un morceau de l'image précédente, dans une zone où la corrélation des niveaux de gris n'est pas excellente, ce qui va limiter la portée de la *DCT* (*figure 9*).



Pour q = 20 nombre de 0 : 24 / nombre de pixels : 64

47	174	118	180	162	101	115	64
86	63	19	74	65	100	61	91
80	77	4	65	133	182	175	145
103	129	74	123	142	99	0	14
97	22	1	107	120	121	29	4
44	51	134	186	67	19	13	27
0	14	0	0	15	11	0	0
19	0	24	13	14	64	123	42

60	180	120	160	160	60	100	60
80	40	0	80	80	100	60	100
100	80	20	60	140	160	200	140
100	120	60	140	140	100	0	20
80	20	0	100	120	140	20	0
40	60	140	180	100	20	20	20
0	0	0	0	0	0	0	0
40	0	20	0	20	80	120	60

8	1	-2	-1	0	-1	-2	0
5	-1	0	1	1	0	0	-1
8	-3	1	2	0	0	-1	0
6	2	-2	1	1	-1	0	-1
4	0	-2	3	1	0	0	-1
5	2	-3	-2	1	1	0	0
0	0	0	0	0	0	0	0
3	-2	1	1	-1	1	0	0

Figure 9 : *En haut à gauche*, l'image 8 sur 8 initiale, *à droite* l'image reconstituée. *Au-dessous à gauche*, le tableau des niveaux de gris de l'image initiale et *à droite* celui de l'image finale. On notera des différences.[4] *En bas*, le tableau intermédiaire fourni par la *DCT* après quantification, avec un certain nombre de 0 présents.

C'est le tableau de la *DCT* avec quantification qui constitue l'image compressée, à partir de laquelle l'image sera ensuite reconstruite. Mais comment s'y prend-on pour la compresser? On ne peut pas supprimer tous les 0, même s'ils ne servent à rien, sinon toute reconstitution future de l'image serait impossible. On procède alors à une compression par plages sur les nombres obtenus par la *DCT*. Après avoir placé ses 64 nombres dans un tableau en ligne, chaque bloc formé de 0 successifs est remplacé par le nombre 0 suivi du nombre de fois où il apparaît. D'où un gain appréciable de place.[5] Par exemple la séquence :

[4] C'est seulement pour $q = 1$ que l'image finale est la recopie exacte de l'image initiale, à une unité près, à cause du remplacement des nombre flottants par des entiers.

[5] Pour renforcer la compression, on peut ajouter à la compression par plages un codage de Huffman.

54 57 27 **0 0 0 0 0 0** 15 20 **0 0 0 0 0** 2 **0** 20 50 40 12 **0 0 0 0 0 0 0 0 0 0**
devient
54 57 27 **0 6** 15 20 **0 5** 2 **0 1** 20 50 40 12 **0 10** [6]

Au stade où nous en sommes, force est de constater que les résultats sont mitigés, l'image reconstituée n'étant pas extrêmement proche de l'image originelle, et le taux de compression pas très élevé. Comment faire mieux ? Au lieu d'utiliser seulement les lignes indépendamment les unes des autres, en jouant sur la bonne corrélation entre pixels successifs sur chaque ligne, nous allons en plus faire intervenir les colonnes, avec leur propre corrélation entre pixels verticaux successifs. Il s'agit là de la méthode propre à la compression JPEG. Au lieu de travailler sur des vecteurs lignes, en une dimension, on opère sur les vecteurs lignes puis sur les vecteurs colonnes, en deux dimensions.

4) Compression d'images en deux dimensions

Dorénavant, nous allons prendre $N = 8$, car l'image à comprimer sera découpée en blocs de 8 sur 8 (*figure 10*).[7]

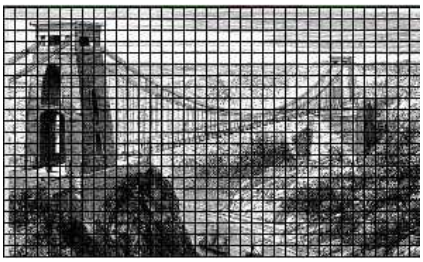


Figure 10 : Découpage de l'image initiale (296 sur 176) en blocs de 8 sur 8, ici 814 blocs

4.1) Passage de la *DCT* d'une dimension à deux dimensions

Rappelons la formule de la *DCT* en une dimension que nous avons prise précédemment :

$DCT(k) = \frac{c(k)}{4} \sum_{n=0}^{N-1} f(n) \cos\frac{(2n+1)k\pi}{16}$. Dans ce qui suit nous prenons une variante de cette formule, plus précisément le double, soit

$DCT(k) = \frac{c(k)}{2} \sum_{n=0}^{N-1} f(n) \cos\frac{(2n+1)k\pi}{16}$ avec $c(0) = 1/\sqrt{2}$ et $c(k) = 1$ pour $k \neq 0$.

Lorsque l'on passe en deux dimensions, la *DCT* devient une matrice carrée 8 sur 8 (un tableau à deux dimensions), dont les coefficients sont :

$$DCT(i,j) = \frac{1}{4} c(i) c(j) \sum_{x=0}^{7} \sum_{y=0}^{7} I(x,y) \cos\frac{i(2x+1)\pi}{16} \cos\frac{j(2y+1)\pi}{16}$$

où $I(i,j)$ est le niveau de gris du point de la ligne i et de la colonne j sur un bloc 8 sur 8 de l'image originelle. Cette formule peut aussi s'écrire :

---

[6] La compression n'intervient vraiment que pour les gros blocs de 0. Si l'on a des 0 isolés, chacun d'eux est remplacé par deux nombres, et le tableau est plus long, au lieu de raccourcir. Le nombre de 0 ne suffit pas pour indiquer le taux de compression.

[7] Si l'on fait cela, c'est essentiellement pour pouvoir obtenir des formules spécifiques à ces petites dimensions, afin de limiter les multiplications et les additions.

$$DCT(i,j) = \sum_{x=0}^{7}\sum_{y=0}^{7} I(x,y)\ \frac{c(i)}{2}\cos\frac{i(2x+1)\pi}{16}\ \ \frac{c(j)}{2}\cos\frac{j(2y+1)\pi}{16}$$

Prenons la matrice M des cosinus, dont les éléments sont $M(i,j)$ tels que :

$M(i,j) = \frac{c(i)}{2}\cos\frac{i(2j+1)}{16}$. Voici cette matrice M :

0.35	0.35	0.35	0.35	0.35	0.35	0.35	0.35
0.49	0.42	0.28	0.10	-0.10	-0.28	-0.42	-0.49
0.46	0.19	-0.19	-0.46	-0.46	-0.19	0.19	0.46
0.42	-0.10	-0.49	-0.28	0.28	0.49	0.10	-0.42
0.35	-0.35	-0.35	0.35	0.35	-0.35	-0.35	0.35
0.28	-0.49	0.10	0.42	-0.42	-0.10	0.49	-0.28
0.19	-0.46	0.46	-0.19	-0.19	0.46	-0.46	0.19
0.10	-0.28	0.42	-0.49	0.49	-0.42	0.28	-0.10

En utilisant cette matrice M, la formule de la *DCT* se réécrit :

$$DCT(i,j) = \sum_{x=0}^{7}\sum_{y=0}^{7} I(x,y)\ M(i,x)\ M(j,y)$$

$$= \sum_{x=0}^{7}\sum_{y=0}^{7} I(x,y)\ M(i,x)\ M^T(y,j)$$

où M^T est la transposée de M, c'est-à-dire sa symétrique par rapport à la diagonale

$$= \sum_{x=0}^{7} M(i,x)\sum_{y=0}^{7} I(x,y)\ M^T(y,j)$$

$$= \sum_{x=0}^{7} M(i,x)\ IMT(x,j), \text{ où la matrice } IMT \text{ est égale au produit } I\times M^T$$

$$= MIMT(i,j), \text{ où la matrice } MIMT \text{ est égale au produit } M\times IMT$$

Ayant ainsi utilisé à deux reprises la définition du produit de deux matrices,[8] on arrive au résultat matriciel suivant :

$$DCT = M\times I\times M^{\mathrm{T}}$$

Commençons par prendre le produit $I\times MT = IMT$, et appliquons-le sur une ligne x_0 de l'image, les coordonnées des éléments de cette ligne étant (x_0, j) avec j de 0 à 7. On constate que :

$$IMT(x_0,j) = \sum_{y=0}^{7} I(x_0,y)M^T(y,j) = \sum_{y=0}^{7} I(x_0,y)M(j,y) = \sum_{y=0}^{7} I(x_0,y)\frac{c(j)}{2}\cos\frac{j(2y+1)\pi}{16}$$

$$= \frac{c(j)}{2}\sum_{y=0}^{7} I(x_0,y)\cos\frac{j(2y+1)\pi}{16}$$

On retrouve la *DCT* à une dimension sur la ligne x_0. La matrice *IMT* correspond à un parcours ligne par ligne de l'image, comme on l'a fait précédemment. De façon analogue, le fait de faire le produit final $MIMT = M\times IMT$ consiste à parcourir l'image colonne par colonne. Ainsi, on fait jouer non seulement la corrélation entre les éléments de chaque ligne, mais aussi celle entre les éléments des colonnes de niveaux de gris de l'image. Les résultats s'en trouvent grandement améliorés, comme on va le constater.

Ayant ainsi obtenu la *DCT* en deux dimensions, on procède à sa quantification pour faire apparaître les blocs de 0.

[8] Rappelons qu'avec $C = AB$, la multiplication des matrices s'écrit $C_{ij} = \sum_k a_{ik}b_{kj}$

4.2) La quantification de la *DCT*

Au lieu de faire comme précédemment, en divisant par un nombre *q*, on utilise une matrice *Q* qui se présente ainsi :

```
Matrice Q de quantification
 16  11  10  16  24  40  51  61
 12  12  14  19  26  58  60  55
 14  13  16  24  40  57  69  56
 14  17  22  29  61  87  80  62
 18  22  37  56  68 109 103  77
 24  35  55  64  81 104 113  92
 49  64  78  87 103 121 120 101
 72  92  95  98 112 100 103  90
```

Cette matrice a été déterminée expérimentalement, après divers essais. Elle présente la particularité d'avoir ses éléments qui ont tendance à augmenter lorsque l'on descend dans son tableau suivant une direction diagonale sud-est. Pourquoi avoir fait cela ? Parce que c'est aussi la direction privilégiée lorsque l'on parcourt l'image initiale ligne par ligne puis colonne par colonne.

La quantification consiste à remplacer les éléments $DCT(i, j)$ par $DCTQ(i, j) = DCT(i,j) / Q(i, j)$ puis à arrondir ce résultat en prenant l'entier le plus proche. Grâce à la direction privilégiée de *Q* et de l'image, les 0 qui apparaissent s'accumulent dans le coin sud-est (*figure 11*).

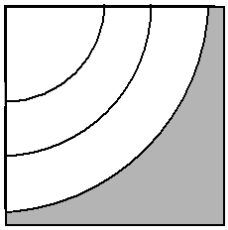


Figure 11 : Tableau de la matrice *DCTQ* après quantification, avec en gris la zone où ont tendance à se concentrer les hautes fréquences mises à 0

Entrée de la matrice *Q* dans le programme

```
Q[0][0]=16; Q[0][1]=11; Q[0][2]=10; Q[0][3]=16;
Q[0][4]=24; Q[0][5]=40; Q[0][6]=51; Q[0][7]=61;
Q[1][0]=12; Q[1][1]=12; Q[1][2]=14; Q[1][3]=19;
Q[1][4]=26; Q[1][5]=58; Q[1][6]=60; Q[1][7]=55;
Q[2][0]=14; Q[2][1]=13; Q[2][2]=16; Q[2][3]=24;
Q[2][4]=40; Q[2][5]=57; Q[2][6]=69; Q[2][7]=56;
Q[3][0]=14; Q[3][1]=17; Q[3][2]=22; Q[3][3]=29;
Q[3][4]=61; Q[3][5]=87; Q[3][6]=80; Q[3][7]=62;
Q[4][0]=18; Q[4][1]=22; Q[4][2]=37; Q[4][3]=56;
Q[4][4]=68; Q[4][5]=109; Q[4][6]=103; Q[4][7]=77;
Q[5][0]=24; Q[5][1]=35; Q[5][2]=55; Q[5][3]=64;
Q[5][4]=81; Q[5][5]=104; Q[5][6]=113; Q[5][7]=92;
Q[6][0]=49; Q[6][1]=64; Q[6][2]=78; Q[6][3]=87;
Q[6][4]=103; Q[6][5]=121; Q[6][6]=120; Q[6][7]=101;
Q[7][0]=72; Q[7][1]=92; Q[7][2]=95; Q[7][3]=98;
Q[7][4]=112; Q[7][5]=100; Q[7][6]=103; Q[7][7]=90;
/* ou bien pour une plus forte compression*/
for(ligne=0;ligne<8;ligne++) for(col=0; col<8;col++)
{Q[ligne][col]=10*Q[ligne][col];  /* ici multiplication par 10 */
if (Q[ligne][col]>255) Q[ligne][col]=255;
}
```

Pour augmenter la compression, il est possible de multiplier *Q* par un nombre (5, 10, 15 par exemple), les éléments de la matrice sont tous multipliés par ce nombre, mais les résultats qui dépassent 255 sont ramenés à 255.

4.3) Compression de l'image

Au lieu de parcourir la matrice *DCTQ* (après quantification) ligne après ligne, on procède différemment, en faisant un parcours diagonal en zig-zag, comme indiqué sur la *figure 12*.[9] Ce procédé a l'avantage de donner de longs blocs de 0, beaucoup plus que si l'on faisait un parcours en ligne. Ce sont ces blocs qui sont ensuite comprimés grâce à la compression par plages.

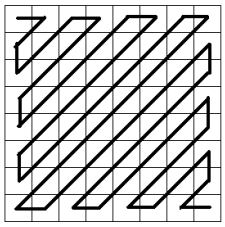


Figure 12 : Parcours en zig-zag de la matrice *DCTQ*

Programme

```
/* Mise dans le tableau a[] de la matrice DCTQ avec son parcours en zig-zags */
 a[0]=DCTQ[0][0];  k=1;
 for(i=1;i<8;i++)
   { if (i%2==1) { col=i; ligne=0; a[k++]=DCTQ[ligne][col];
                     while (col!=0) {col--; ligne++; a[k++]=DCTQ[ligne][col]; }
                   }
     else  { col=0; ligne=i; a[k++]=DCTQ[ligne][col];
              while  (ligne!=0) {col++; ligne--; a[k++]=DCTQ[ligne][col]; }
            }
   }
 for(i=1;i<8;i++)
   { if (i%2==1) { col=i; ligne=7; a[k++]=DCTQ[ligne][col];
                     while (col!=7) {col++; ligne--; a[k++]=DCTQ[ligne][col]; }
                   }
     else  { col=7; ligne=i; a[k++]=DCTQ[ligne][col];
              while  (ligne!=7) {col--; ligne++; a[k++]=DCTQ[ligne][col]; }
            }
   }
/* passage de a[] à aa[] après codage par plages de 0 */
a[64]=10; nbfois =0;k=0;  /* on a rajouté une base butoir a[64] */
for(i=0;i<64;i++)
if (a[i]==0 && a[i+1]!=0)   {nbfois++;aa[k]=0;aa[k+1]=nbfois; k+=2; nbfois=0;}
 else if (a[i]==0) nbfois++;
 else aa[k++]=a[i];
/* on fait cela pour chaque bloc 8x8, et l'on cumule à chaque fois les longueurs du bloc compressé */
 cumullongueur+=k;
```

4.4) Reconstruction de l'image

Il suffit de pratiquer les opérations inverses des précédentes. A partir de la matrice *DCT* quantifiée précédente, on procède à sa multiplication terme à terme par les éléments de la matrice *Q* de quantification, pour supprimer la quantification. Puis on inverse la formule $DCT = M \times I \times M^T$ en

[9] J'ai utilisé ce même type de parcours dans le document *Chemin en lacets d'un robot, ou encore Q est dénombrable* (cf . *Travaux complémentaires, Algorithmes, 10*).

profitant du fait que la matrice M est orthogonale, d'où $M^{-1} = M^T$. On arrive ainsi à $I = M^T \times I \times M$. Il suffit de faire ces produits de matrices pour trouver l'image I finale, que l'on pourra comparer avec l'image initiale.

Résultats

Matrice *DCTQ* après quantification grâce à la matrice *Q*

```
-29  -1 -10   4   1   0  -1  -1
 19  -1  -1   2   0  -1  -1  -1
 -3  -5   3  -3  -2   0  -1   0
  0   7  -5  -3   1   0   0   0
  3   2  -1   1   0   0   0   0
  1   2  -1  -1   0  -1   0   0
  3  -1   0  -1   0   0   0   0
  0  -1   0   1   0   0   0   0
```

```
Image initiale
 47  174  118  180  162  101  115   64
 86   63   19   74   65  100   61   91
 80   77    4   65  133  182  175  145
103  129   74  123  142   99    0   14
 97   22    1  107  120  121   29    4
 44   51  134  186   67   19   13   27
  0   14    0    0   15   11    0    0
 19    0   24   13   14   64  123   42

Image finale
 39  196  102  190  167  108   87   85
 73   82    0   78   50   93  118   46
 66   91   18   86  132  188  165  160
116  126   79  113  133  104    0   19
129    0    0  108  126  130   44    0
 42   52  138  169   69    0    5   65
  0   24   17   19   39   10    0    0
 17    8   15   12    0   65  140   52
```

On pourra comparer l'image originelle et sa reconstruction après compression

Programme (ici en mode texte)

Ce programme ne s'applique qu'à un bloc 8 sur 8. Il conviendra ensuite de découper l'image globale en blocs puis à répéter ce programme pour chaque bloc.

Entrer la matrice I[8][8] du bloc image, avec ses niveaux de gris de 0 à 255, puis enlever 128 pour les ramener entre – 128 et 127. Entrer aussi la matrice Q de quantification

```
/* Matrice M des cosinus et sa transposée MT*/
 for(ligne=0;ligne<8;ligne++)
   {  printf("\n");
      for(col=0;col<8;col++)
        { if (ligne==0) M[0][col]=1./sqrt(8.);
          else  M[ligne][col]=0.5*cos( (2.*(float)col+1.)*ligne*M_PI/16.);
          printf(" %5.2f  ",M[ligne][col]);
        }
   }
 for(ligne=0;ligne<8;ligne++) for(col=0;col<8;col++)    MT[ligne][col]=M[col][ligne];
/* Produit I  MT donnant la matrice IMT */
for(ligne=0;ligne<8;ligne++)
for(col=0;col<8;col++)
  { cumul=0.;
    for(k=0;k<8;k++)  cumul+=I[ligne][k]*MT[k][col];
    IMT[ligne][col]=cumul;
  }
/* Produit M  IMT donnant la matrice MIMT */
for(ligne=0;ligne<8;ligne++)
for(col=0;col<8;col++)
  { cumul=0.;
    for(k=0;k<8;k++)  cumul+=M[ligne][k]*IMT[k][col];
```

```
      MIMT[ligne][col]=cumul;
   }
/**** Affichage de la matrice DCT 2D, c'est-à-dire MIMT ****/
 printf("\n\n");
 for(ligne=0;ligne<8;ligne++)
    {  printf("\n");
       for(col=0;col<8;col++)  printf(" %5.1f  ",MIMT[ligne][col]);
    }
/****   DCT quantifiée en divisant terme à terme par Q, soit la  matrice DCTQ  ****/
for(ligne=0;ligne<8;ligne++)
for(col=0;col<8;col++)
   { quotient=MIMT[ligne][col]/Q[ligne][col];
     if (quotient >=0.) DCTQ[ligne][col]=(int)(quotient +0.5);  /* arrondi au plus proche entier */
     else DCTQ[ligne][col]=(int)(quotient -0.5);
   }
printf("\n\n");
for(ligne=0;ligne<8;ligne++)
   { printf("\n");
     for(col=0;col<8;col++)
     printf(" %3.1d ", DCTQ[ligne][col]);
   }
/***  Reconstruction de DCT  puis son inversion pour arriver à l'image finale reconstruite ****/
/* Multiplication de DCTQ par Q terme à terme, soit RDCT */
printf("\n\n");
for(ligne=0;ligne<8;ligne++)
   { printf("\n");
     for(col=0;col<8;col++)
        { RDCT[ligne][col] =DCTQ[ligne][col]*Q[ligne][col];
          printf(" %3.1d ", RDCT[ligne][col]);
        }
   }
/* Produit RDCT  M, soit RDCTM */
for(ligne=0;ligne<8;ligne++)
for(col=0;col<8;col++)
   { cumul=0.;
     for(k=0;k<8;k++)  cumul+=(float)RDCT[ligne][k]*M[k][col];
     RDCTM[ligne][col]=cumul;
   }
/* Produit MT  RDCTM, soit MTRDCTM  qui est l'image finale IFin reconstituée */
for(ligne=0;ligne<8;ligne++)
for(col=0;col<8;col++)
   { cumul=0.;   for(k=0;k<8;k++)  cumul+=MT[ligne][k]*RDCTM[k][col];
     MTRDCTM[ligne][col]=cumul;
     if (MTRDCTM[ligne][col]>=0)
     IFin[ligne][col]=(int) (MTRDCTM[ligne][col]+0.5);
     else IFin[ligne][col]=(int) (MTRDCTM[ligne][col]-0.5);
     if (IFin[ligne][col]<-128) IFin[ligne][col]=-128;
     if (IFin[ligne][col]>=128) IFin[ligne][col]=127;
   }
/**** Comparatif de l'image initiale et de sa reconstitution ****/
 printf("\n\n  Image initiale");
for(ligne=0;ligne<8;ligne++)
   { printf("\n");   for(col=0;col<8;col++)   printf(" %3.1d ", I[ligne][col]+128);
   }
 printf("\n\n  Image finale");
for(ligne=0;ligne<8;ligne++)
   { printf("\n");  for(col=0;col<8;col++) printf(" %3.1d ", IFin[ligne][col]+128);
   }
```

Résultats (*figures 13 et 14*)

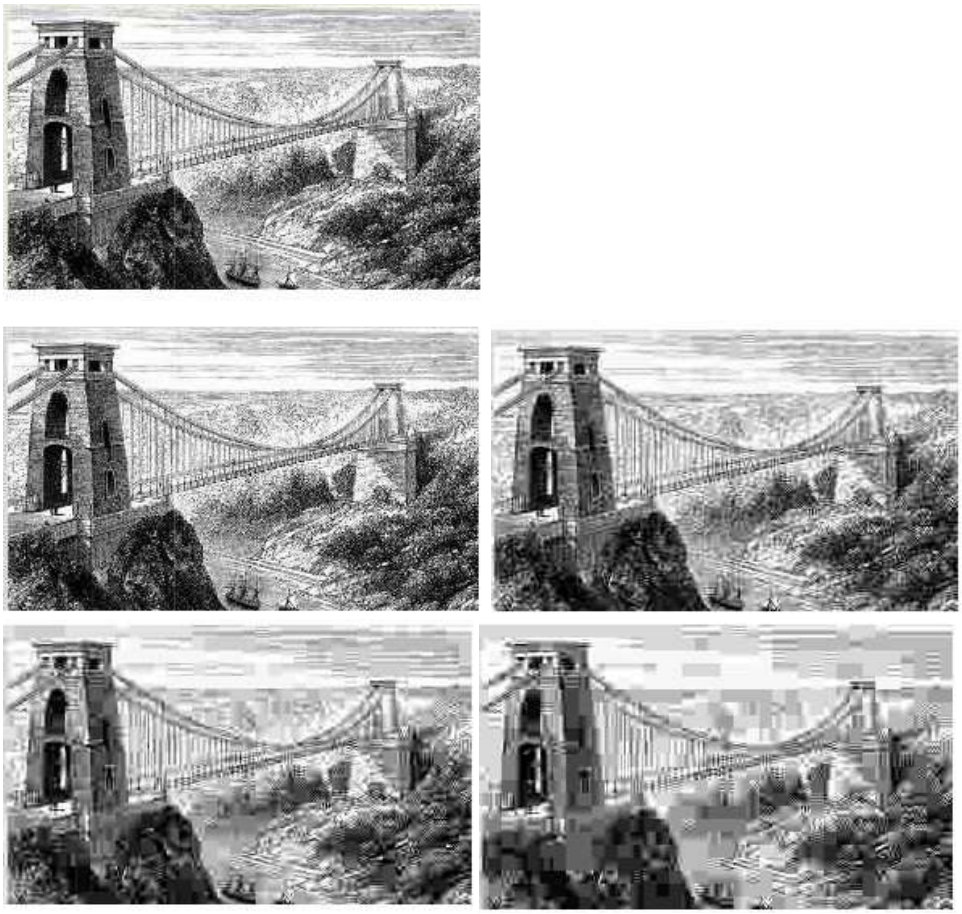


Figure 13 : *En haut* l'image originelle, puis *en dessous* quatre images reconstituées, la première occupant une place de 75% par rapport à celle de l'original (avec la matrice *Q*), la deuxième avec 25% (*Q* multiplié par 5), la troisième avec 15% (*Q* multiplié par 10), la dernière avec 10% (*Q* multiplié par 15). Seule la dernière est vraiment dénaturée.

Figure 14 : un bloc 8x8 de l'image, *en haut* à l'origine, et *au-dessous* dans l'image reconstruite avec *Q*, 5*Q*, 10*Q* et 15*Q*. Malgré des changements prononcés pour 5*Q* et 10*Q*, les répercussions sur l'image globale sont peu visibles. Elles ne sont vraiment importantes que pour 15*Q*

Nous en restons là dans notre étude de la compression d'images. L'étape suivante consisterait à dégager des formules qui diminuent le nombre des calculs de la *DCT* sur les blocs 8 sur 8. Telle que nous l'avons utilisée, la *DCT* à une dimension nécessite en gros $N^2 = 64$ multiplications et autant d'additions, ce qui en donne N^4 pour la *DCT* en deux dimensions. Cela peut être amélioré en combinant les calculs, un peu comme on le fait pour la transformée de Fourier rapide (voir [SHA2007].

Références bibliographiques

[CAB] K. Cabeen, P. Gent, *Image Compression and the Discrete Cosine Transform*, Math, College of the Redwoods, www.lokminglui.com/dct.pdf.

[SHA2007] Shao-Yi Chien, *DCT Algorithm*, g.osie.org/uict/slide/DCT

[VAS] N. Vasconcelos, *Discrete Cosine Transform, slides*, www.svcl.ucsd.edu/courses/ece161c/

[WEN1977] Wen-Hsiung Chen, C. Harrison Smith, S.C. Fralick, *A Fast Computational Algorithm for the Discrete Cosine Transform*, IEEE Transactions on communications, vol.25, no. 9, septembre 1977.