La DCT 3D revisitée par l'exemple

Par Michel J. DUMONTIER

Introduction : Comme je le faisais remarquer à la fin de l'article sur l'an 2000, même si on donne des exemples d'utilisation, il y en a encore qui restent en rade...

Ici, je m'aperçois que, non seulement il faut donner des exemples mais il faut donner aussi des explications à ceux qui manquent de culture dans ce domaine (et ce n'est pas une honte, je tiens à le préciser: tout le monde ne fait pas du traitement d'images, pour ma part, ce n'est pas non plus ma spécialité…).

Il faut quand même savoir que la FFT (c'est vieux comme tout! et j'en ai écrit une très performante il y a plus de 15 ans), n'a rien à voir avec la DCT, encore moins la DCT 3D. La FFT fait intervenir en général des nombres complexes contrairement à la DCT.

Pour l'exemple de traitement d'images que je donne ici, ce n'est pas la peine de parler de la FFT, ni de perdre son temps à faire la comparaison.

Explications:

C'est la première fois que je fais intervenir des amis non APListes dans ce journal et on aurait tort de leur en vouloir, ils ont fait un bon travail qui mérite d'être cité dans ces colonnes et en voici la preuve.
D'abord, rendons à César ce qui est à César: il y avait 3 auteurs cités pour l'article précédent.
Pascal Leray qui avait besoin d'écrire en APL une DCT 3D, m'avait demandé de lui transmettre ma fonction APL FFT citée plus haut, ce que j'ai fait.
Le problème de la permutation des axes s'étant posée, il a demandé à Patrick Gerlier des conseils. Ce dernier m'a reposé la même question et je lui ai donné la réponse, à savoir la transposition dyadique (je me rallie à la majorité pour l'écriture de dyadique!) avec, à la clé, la fourniture de la norme APL ISO 8485. (La référence en la matière pour les bonnes définitions formelles de la notation). Je lui ai laissé le soin de trouver la bonne permutation à utiliser (ici: 3 1 2).
J'ai demandé à Pascal Leray de m'envoyer les fonctions qu'il avait dû réaliser et il en a sûrement profité pour le publier ailleurs, ce qu'on ne peut lui reprocher, et c'est pourquoi l'article est en anglais.
Une bonne fois pour toute, je tiens à préciser que je ne traduirai pas (et je ne l'ai jamais fait) les articles qu'on m'envoie, (sauf demande expresse) que ce soit la traduction d'une langue parlée au français, ou d'un Fortran APLisé en APL pratiqué par les APListes.
Par contre, je me laisse la liberté de valoriser des travaux qui en vaillent la peine et la lettre de Claude Henriod à la fin de cette revue apporte de l'eau à mon moulin.
Dès que j'ai reçu les fonctions de Pascal Leray, je les ai réécrites pour les tester.
Voici les fonctions revues et corrigées par mes soins (en février 1999).

```
    ∇ Z←MCOS DCT3D T;N;⎕IO
[1]   ⎕IO←1 ⋄ N←3 1 2
[2]   Z←0.015625×MCOS+.×N⍉(MCOS+.×N⍉(MCOS+.×N⍉T))
    ∇

    ∇ Z←MCOS IDCT3D T;N;⎕IO
[1]   ⎕IO←1 ⋄ N←3 1 2
[2]   Z←(⍉MCOS)+.×N⍉((⍉MCOS)+.×N⍉((⍉MCOS)+.×N⍉T))
    ∇
```

Il est préférable d'appeler un tableau T et non V!

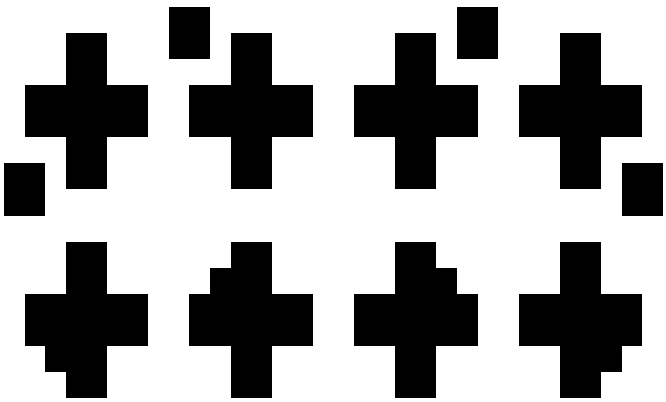
Il est préférable de localiser ⎕IO sinon, gare aux surprises! et j'en ai été moi-même victime quand j'ai commencé à tester les fonctions.

On peut éviter que MCOS soit une variable globale.

```
    ∇ M←COEFFCOS;⎕IO;FAC;A;B;C;D;E;F;G
[1]   ⎕IO←0 ⋄ FAC←1
[2]   A←FAC×2○○÷4
[3]   B←FAC×2○○÷16
[4]   C←FAC×2○○3÷16
[5]   D←FAC×2○○5÷16
[6]   E←FAC×2○○7÷16
[7]   F←FAC×2○○÷8
[8]   G←FAC×2○○3÷8
[9]   M←8 8⍴0
[10]  M[0;]←8⍴A
[11]  M[1;]←8⍴B,C,D,E,(-E),(-D),(-C),(-B)
[12]  M[2;]←8⍴F,G,(-G),(-F),(-F),(-G),G,F
[13]  M[3;]←8⍴C,(-E),(-B),(-D),D,B,E,(-C)
[14]  M[4;]←8⍴A,(-A),(-A),A,A,(-A),(-A),A
[15]  M[5;]←8⍴D,(-B),E,C,(-C),(-E),B,(-D)
[16]  M[6;]←8⍴G,(-F),F,(-G),(-G),F,(-F),G
[17]  M[7;]←8⍴E,(-D),C,(-B),B,(-C),D,(-E)
    ∇
```

Encore ici, il faut localiser les variables. On n'a pas besoin de PI puisqu'on a effacé la ligne (PI÷16)×⍳8 qui devait appartenir à la mise au point je suppose et qu'on a intégré PI par sa fonction APL dans les calculs de cosinus.
Du coup, la fonction cos disparaît.
Important: Il faut savoir que dans les normes de compression (d'après les spécialistes) il est courant de traiter des blocs d'images 8x8 ces images évoluant dans le temps, on prend des blocs spatio-temporels de 8x8x8.

Exemple 1:
Pour cet exemple, j'ai choisi de faire 8 images représentant une croix avec un carré noir se déplaçant dans chaque image.



Voici les 8 images:

On utilise le test suivant:

```
    ∇ Z←TST3D;⎕PP
[1]   ⎕IO←1 ⋄ ⎕PP←6
[2]   MCOS←COEFFCOS
[3]   A←MCOS DCT3D MT
[4]   B←0.001×⌊A×1000
```

```
[5]   Z←⌈8 8 8⍴⍳6 1⍉MCOS IDCT3D B
    ∇
```

MT est la matrice 8x8x8 qui contient 8 images de 64 pixels.

Le test est le suivant:

```
      TIME 'MC←TST3D'
0
```

Déjà, on s'aperçoit qu'on a mis 0 millisecondes pour le faire. C'est pourtant le traitement exact qui était donné dans l'article, à savoir: traiter DCT et inverse DCT sur un bloc 8x8x8. Comme cela ne satisfera pas certains, j'ai fait un test sur une image réelle

On vérifie aussi qu'on a bien les 8 images identiques: (512 pixels).

```
      +/(,MC)=,MT
512
```

Exemple 2:

Nous allons traiter une image bmp en noir et blanc toujours par blocs de 8x8x8.
Pour cela, nous utilisons la fonction suivante de lecture de fichier.

```
    ∇ R←LIRFIC F;L
[1]   ⎕nuntie L←¯1⋄F ⎕ntie L⋄ R←⎕nread L,82,⎕nsize L⋄ ⎕nuntie L
    ∇
```

NB: Ne pas utiliser la fonction `depuisfic` de Gérard Langlet située dans le N° 18 de la revue page 51, qui supprime le caractère de fin de fichier s'il existe et c'est ici le cas, sinon voici ce que cela donne lorsqu'on réécrit un fichier lu de cette manière

avec LIRFIC

avec depuisfic 

Surprise, elle a plutôt la g.. de traver

Voici la fonction qui va traiter cette image (qui occupe 1614 octets en bmp) par blocs de 8x8x8 et nous faisons en même temps un benchmark 'à la Joseph'…pour contenter notre (nos) lecteur(s).

```
    ∇ Z←TSTDCT3D MT;⎕IO;R;X;I
[1]   ⎕IO←0  ◇MJ←⎕AV⍳MT ◇ R←⍴MT
[2]    X←512×⍳20 ◇I←X[(X≥R)⍳1] ◇ MJ←I↑MJ
[3]    M1←8 8 8⍴512↑MJ
[4]    M2←8 8 8⍴512↑512↓MJ
[5]    M3←8 8 8⍴512↑1024↓MJ
[6]    M4←8 8 8⍴512↑1536↓MJ
[7]   MCOS←COEFFCOS
[8]   A1←MCOS DCT3D M1
[9]   B1←0.001×⌊A1×1000
[10]  N1←⌈8 8 8⍴⍬,6 1⍉MCOS IDCT3D B1
[11]  A2←MCOS DCT3D M2
[12]  B2←0.001×⌊A2×1000
[13]  N2←⌈8 8 8⍴⍬,6 1⍉MCOS IDCT3D B2
[14]  A3←MCOS DCT3D M3
[15]  B3←0.001×⌊A3×1000
[16]  N3←⌈8 8 8⍴⍬,6 1⍉MCOS IDCT3D B3
[17]  A4←MCOS DCT3D M4
[18]  B4←0.001×⌊A4×1000
[19]  N4←⌈8 8 8⍴⍬,6 1⍉MCOS IDCT3D B4
[20]  Z←R↑(,N1),(,N2),(,N3),,N4 ◇ Z←⎕AV[Z]
    ∇
```

Voici des explications que j'espère assez complètes pour comprendre son fonctionnement.

On commence par lire une image par exemple par:

```
      FO←'C:\FIC.BMP'
      MT←LIRFIC FO
```

Puis on fait le test suivant:

```
      ⍴MT
1614
      TIME 'MC←TSTDCT3D MT'
110
      ⍴MC
1614
      +/(,MC)=,MT
1614
```

On constate qu'il nous a fallu 110 millisecondes pour faire la DCT et l'inverse de la DCT sur 4 blocs de 8x8x8 mais aussi en fabricant les blocs et en reconstituant l'image de départ. Qui dit mieux?

Fonctionnement du test:

```
    ∇ Z←TSTDCT3D MT;⎕IO;R;X;I
[1]   ⎕IO←0  ◇MJ←⎕AV⍳MT ◇ R←⍴MT
[2]    X←512×⍳20 ◇I←X[(X≥R)⍳1] ◇ MJ←I↑MJ
```

`MJ←⎕AV⍳MT` on transforme le vecteur `MT` de caractères en un vecteur de nombres entiers variant de 0 à 255.
`R←⍴MT` on calcule sa taille
`X←512×⍳20` on fabrique les multiples de 512

`I←X[(X≥R)⍳1]` et on cherche quel est le multiple de 512 le plus proche de cette taille calculée.
`MJ←I↑MJ` on complète `MJ` par des 0 en queue jusqu'à la taille calculée (ici 2048, ce qui fait 4 blocs de 512).
On reconstituera le vecteur à sa taille d'origine à la ligne 20. (`Z←R↑`). On reconstituera les caractères à la fin de cette ligne 20 `(Z←⎕AV[Z])`

Ces 2 premières lignes sont la préparation d'une gestion automatique quelle que soit la taille de l'image lue.
Nous laisserons le lecteur écrire, à titre d'exercice, cette gestion automatique.

Lignes 3 à 6: fabrication des 4 blocs.

```
[7]    MCOS←COEFFCOS   calcul de MCOS
```

Les lignes 8 à 19 sont bâties sur le modèles suivant:

```
[8]    A1←MCOS DCT3D M1
[9]    B1←0.001×⌊A1×1000
[10]   N1←⌈8 8 8⍴⍉,6 1⍤MCOS IDCT3D B1
```

`A1 :` calcul de la DCT3D du bloc `M1`
`B1:` arrondi de `A1` (désolé, `0.01×⌊A1×100` n'était pas assez précis! et c'est là que le bât blesse: si le calcul aboutit à des nombres qui occupent plus de mémoire à quoi la transformée sert-elle?…).
`MCOS IDCT3D B1` calcul de la DCT inverse (pour reconstituer le tableau).
`⍉,6 1⍤` artifice pour avoir des nombres sans trop de décimales.
`⌈` autre artifice pour avoir cette fois-ci des nombres entiers.
On en fait de même pour A2, A3, A4 et le tour est joué.

Ce que je soupçonnais:

Nous avons vu que nous avons dû allonger l'arrondi avec des nombres variant de 0 à 255, or cela fonctionnait pour des nombres compris entre 0 et 63.
J'ai récrit une fonction avec l'arrondi précédent `(0.01×⌊A1×100)` et j'ai testé sur d'autres nombres.
Est-ce qu'on a bien le même résultat qu'avant:

```
      MT←⎕AV[2048⍴⍳64]
      TIME 'MC←TSTDCT3D1 MT'
110
      +/(,MC)=MT
2048
```

Oui, on a bien le même résultat.
Et si on les mettait dans n'importe quel ordre?

```
      MT←⎕AV[2048⍴64?64]
      TIME 'MC←TSTDCT3D1 MT'
110
      +/(,MC)=MT
2048
```

C'est rassurant:
Et si on prenait des nombres compris entre 0 et 127?

```
      MT←⎕AV[2048⍴128?128]
      TIME 'MC←TSTDCT3D1 MT'
110
      +/(,MC)=MT
1992
```

Effectivement, cela ne marche plus bien!
Et avec des nombres compris entre 0 et 255?

```
      MT←⎕AV[2048⍴?512⍴256]
      TIME 'MC←TSTDCT3D1 MT'
110
      +/(,MC)=MT
```

Cela se dégrade un peu.
Après avoir fait ces constatations, j'ai téléphoné à Pascal Leray, l'auteur des fonctions de l'article précédent, spécialiste en compression d'images. Résultat de la conversation: effectivement, il avait constaté que cet arrondi convenait à des nombres variant de 0 à 63.
Il avait donné cet exemple pour prouver que cela fonctionne, mais en réalité c'est beaucoup plus compliqué: on ne prend pas le même arrondi pour toutes les valeurs du tableau (c'était une simplification).
Il m'a dit: "on dit que JPEG c'est une DCT, ce n'est pas que cela: n'importe quel mathématicien est capable d'écrire une DCT, mais pour écrire un JPEG ou autre, c'est autre chose, tout le monde se heurte au problème que j'ai rencontré et tout l'art de la compression c'est de savoir adapter aux valeurs, la conservation du minimum d'information".
Peut-être des spécialistes pourront-ils nous en dire plus et nous donner le résultat de leurs travaux?
Nous espérons que les lecteurs seront un peu plus satisfaits avec ces quelques explications d'un APListe non spécialiste de la compression.