

Exemple d'encodage d'une MCU

Cette annexe donne un exemple d'encodage d'une MCU, de la compression des pixels RGB initiaux jusqu'à l'encodage bit par bit dans le flux de données JPEG. À étudier en parallèle du chapitre 2.

On travaille sur la version couleur de l'image étudiée dans l'[annexe sur l'entête](#) au format ppm, `poupoupidou.ppm` (élargie ici pour l'affichage):



C'est une image 16x16 que l'on souhaite encoder avec sous-échantillonnage horizontal (4:2:2 horizontal). On considère donc deux MCU de taille 16x8, et on s'intéresse ici à l'encodage de la première.

MCU en RGB

La première MCU en RGB est:

d83d67	d6407d	c0407b	d88170	fcc35c	e4b357	e4bf4c	e7c365	e6bd53	f4b25b	c45685	d83e87	d34378	d244
d5407a	c34b78	d06e6d	ffc745	dcbe44	f8dd5c	b68945	9e8437	e5d84b	f5d13d	fb06c	ce4980	cb4486	cb40
c73a7c	d78e65	ffc6b4	eec643	756335	a79435	e2bf47	ae9f40	eac866	f3d460	eacd43	ffcd52	f8c454	d885
c64271	934b56	e4c44d	cdae44	7f6849	dcc867	f1cc64	e5cd6b	eecf63	b7974c	f5df55	f9d65e	f7d557	ffce
c94574	b16a4e	ebd462	d5bf47	deadb0	e7b6af	e7b696	eeb3ab	f7b3c0	e9b7ac	e6bd9d	d3b764	d6be52	e7ca
ca4a63	82344e	8a6f42	be9b75	eaaba6	f0b6ab	edb5b6	f3adab	fba5b0	f6b5cb	e7bbb2	a29046	d4c66e	a190
c7437d	b6875d	7a693e	a77a81	e7acae	edb2b6	edb3a8	edadc5	efb0cb	e5b8b3	e7b2b8	ba965c	d4b753	b490
dc6470	b8a556	2b1c1f	b18386	ad7d8b	e2b1a3	f1afcb	cb9f96	bb828b	bd988f	a1767d	836751	bdab47	9a89

Représentation YCbCr

La représentation de cette MCU en YCbCr est:

[Y]:

70 74 6d 99 c8 b7 bd c3	bd bc 7c 74 74 74 77 72
73 74 8b c9 b9 d6 8f 83	cc cb bf 77 74 77 6d 76
6c 9f cf c3 63 8f bc 99	c7 d0 c6 ce c7 9c 7b 73
6f 62 c0 ab 6b c3 cb c9	cc 98 d6 d3 d1 d2 90 6f
72 7c ce b8 bc c4 c1 c4	c9 c5 c6 b6 b9 c5 82 7e
73 4e 72 a1 bd c6 c6 c2	c0 cb c7 8d c0 8b 4b 79
71 90 69 88 be c4 c3 c3	c6 c5 c3 9a b4 9c 66 8a
89 a2 21 91 8d be c6 ab	94 a2 84 6d a5 88 5b 8e

[Cb]:

7b 85 88 69 43 4a 40 4b	44 49 85 8b 82 7f 84 81
84 82 6f 36 3e 3b 56 55	37 30 51 85 8a 7e 81 87
89 5f 44 38 66 4d 3e 4e	49 41 36 3a 3f 6a 7c 8c
81 79 3f 46 6d 4c 46 4b	45 55 37 3e 3b 48 75 85
81 66 43 40 79 74 68 72	7b 72 69 52 46 41 63 88
77 80 65 67 73 71 77 73	77 80 74 58 52 51 7f 7f
87 63 68 7c 77 78 71 81	83 76 7a 5d 49 5b 6e 77
72 55 7f 7a 7f 71 83 74	7b 75 7c 70 4b 63 70 77

[Cr]:

ca c6 bb ad a5 a0 9c 9a	9d a8 b3 c7 c4 c3 d0 bf
c6 b8 b1 a7 99 98 9c 93	92 9e ab be be bc c8 bb
c1 a8 a2 9f 8d 91 9b 8f	99 99 9a a3 a3 ab c2 c6
be a3 9a 98 8e 92 9b 94	98 96 96 9b 9b a0 b4 c1
be a6 95 95 98 99 9b 9e	a1 9a 97 95 95 98 a7 bf
be a5 91 95 a0 9e 9c a3	aa 9f 97 8f 8e 90 9e c1
bd 9b 8c 96 9d 9d 9e 9e	9d 97 9a 97 97 91 91 b4
bb 90 87 97 97 9a 9f 97	9c 93 95 90 91 8d 8e b7

Sous échantillonnage

Cette MCU est sous-échantillonnée horizontalement, pour ne conserver qu'un seul bloc 8x8 par composante de chrominance.

[Y]:

70 74 6d 99 c8 b7 bd c3	bd bc 7c 74 74 74 77 72
73 74 8b c9 b9 d6 8f 83	cc cb bf 77 74 77 6d 76
6c 9f cf c3 63 8f bc 99	c7 d0 c6 ce c7 9c 7b 73
6f 62 c0 ab 6b c3 cb c9	cc 98 d6 d3 d1 d2 90 6f
72 7c ce b8 bc c4 c1 c4	c9 c5 c6 b6 b9 c5 82 7e
73 4e 72 a1 bd c6 c6 c2	c0 cb c7 8d c0 8b 4b 79
71 90 69 88 be c4 c3 c3	c6 c5 c3 9a b4 9c 66 8a
89 a2 21 91 8d be c6 ab	94 a2 84 6d a5 88 5b 8e

[Cb]:

80 78 46 45 46 88 80 82
83 52 3c 55 33 6b 84 84
74 3e 59 46 45 38 54 84
7d 42 5c 48 4d 3a 41 7d
73 41 76 6d 76 5d 43 75
7b 66 72 75 7b 66 51 7f
75 72 77 79 7c 6b 52 72
63 7c 78 7b 78 76 57 73

[Cr]:

c8 b4 a2 9b a2 bd c3 c7
bf ac 98 97 98 b4 bd c1
b4 a0 8f 95 99 9e a7 c4
b0 99 90 97 97 98 9d ba
b2 95 98 9c 9d 96 96 b3
b1 93 9f 9f a4 93 8f af
ac 91 9d 9e 9a 98 94 a2
a5 8f 98 9b 97 92 8f a2

DCT : passage au domaine fréquentiel

La DCT est ensuite appliquée à chacun des blocs de données, après avoir soustrait 128¹ aux valeurs. Les basses fréquences sont en haut à gauche et les hautes fréquences en bas à droite. On représente maintenant des *entiers signés 16 bits*. Ainsi, `0xffff` représente la valeur `-1` ici, et pas `65535` !

[Y]:

00f2 ff36 ffca 0008 fff2 001f 0000 ffd	00fb 00cb ffe1 0021 0004 ffc1 0027 000c
0005 0028 ffc b fcd fff6 0017 0023 000b	fffb 001a 001d 0000 ffec 0030 ffe8 fff5
ffda fff5 0015 005e 0027 ffca ffb b fddf	ff82 fff5 005a fff2 0000 ffe0 ffe4 0000
0018 ffc2 ffe0 0023 0031 ffde 0010 0030	ffe8 0018 0024 0000 fff6 fff0 0000 0000
000c ffe8 0022 0000 0000 0000 0000 0000	fff1 ffc8 fffa 0021 0000 0015 0000 fff1
fffc ffe4 002c ffe6 fff1 0000 0000 0011	0022 fff9 ffe0 000d 000f 0014 0000 0000
ffed 0000 001f ffe f 0000 0017 ffe8 0000	ffec 0000 fff0 0000 0014 ffe9 ffe8 0000
001b 0023 0000 ffd9 ffeb ffed ffeb 0013	001b 0000 0000 0000 0000 0000 ffec ffed

[Cb]:

ff2b fffc 003f fff8 0037 fffd 002c fffd
ffcd ffd9 005d 001c fff0 0023 fffb fff7
0034 fff1 0007 001d ffc b fff9 ffe2 ffd8
002c ffff fff1 0017 fff7 fff7 0000 fff5
0000 0003 fff6 0004 fff7 fff2 0008 0000
fff6 000f 000b fffd 0000 fff0 0000 0008
0000 0001 0000 fff7 0000 fff2 0000 0007
0000 0000 ffed fffb 0001 fffd 0007 000a

[Cr]:

0114 fff2 0046 0009 0029 0002 0014 fffe
003d ffe8 0029 0011 ffec ffff fff7 fff7
0016 0000 0000 000a ffec 0006 0000 ffff
0016 0007 fffa 0003 0000 0001 0005 0005
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 fffb 0008 fffa 0000 0003
0000 0000 0000 0000 0000 0000 0000 0000

Zig-zag

On réordonne ensuite les blocs en zig-zag. Cette étape permettra de regrouper les coefficients des plus haute-fréquences, souvent nuls après la phase de quantification, en "fin" de bloc. Pour des questions pratiques, on effectue cette réorganisation ZZ avant la quantification, puisque *les tables de quantification sont stockées au format zig-zag dans l'en-tête JPEG*. On pourrait bien entendu fusionner ces deux étapes (réordonnancement zig-zag + quantification) en une seule opération, comme indiqué sur la figure présentée en fin de section [2.1](#).

[Y]:

00f2 ff36 0005 ffda 0028 ffca 0008 ffc b	00fb 00cb fffb ff82 001a ffe1 0021 001d
fff5 0018 000c ffc2 0015 ffc d fff2 001f	fff5 ffe8 fff1 0018 005a 0000 0004 ffc1
fff6 005e ffe0 ffe8 fffc ffed ffe4 0022	ffec fff2 0024 ffc8 0022 ffec fff9 fffa
0023 0027 0017 0000 ffd d 0023 ffca 0031	0000 0000 0030 0027 000c ffe8 ffe0 fff6
0000 002c 0000 001b 0023 001f ffe6 0000	0021 ffe0 0000 001b 0000 fff0 000d 0000
ffde ffb b 000b ffd f 0010 0000 fff1 ffe f	fff0 ffe4 fff5 0000 0000 0015 000f 0000
0000 ffd9 0000 0000 0000 0030 0000 0000	0000 0000 0014 0014 0000 0000 fff1 0000
0017 ffeb ffed ffe8 0011 0000 ffeb 0013	ffe9 0000 0000 ffe8 0000 0000 ffec ffed

[Cb]:

ff2b fffc ffc d 0034 ffd9 003f fff8 005d
fff1 002c 0000 ffff 0007 001c 0037 fffd
fff0 001d fff1 0003 fff6 0000 000f fff6
0017 ffc b 0023 002c fffd fffb fff9 fff7
0004 000b 0001 0000 0000 0000 fffd fff7
fff7 ffe2 fff7 ffd8 0000 fff2 0000 fff7
ffed fffb 0000 fff0 0008 fff5 0000 0000
fff2 0001 fffd 0000 0008 0007 0007 000a

[Cr]:

0114 fff2 003d 0016 ffe8 0046 0009 0029
0000 0016 0000 0007 0000 0011 0029 0002
ffec 000a fffa 0000 0000 0000 0000 0000
0003 ffec ffff 0014 fffe fff7 0006 0000
0000 0000 0000 0000 0000 0000 0000 0000
0001 0000 fff7 ffff 0005 0000 0000 fffb
0000 0000 0008 0000 0000 0005 0000 0000
ffa 0000 0000 0000 0000 0003 0000 0000

Quantification

La quantification consiste à diviser les blocs par les tables de quantification, une pour la luminance et une pour les deux chrominances. Dans cet exemple, les deux tables sont issues du projet *The Gimp*. Lors du décodage, les tables utilisées sont bien sûr lues dans l'entête du fichier.

[table que quantification Y]

05	03	03	05	07	0c	0f	12
04	04	04	06	08	11	12	11
04	04	05	07	0c	11	15	11
04	05	07	09	0f	1a	18	13
05	07	0b	11	14	21	1f	17
07	0b	11	13	18	1f	22	1c
0f	13	17	1a	1f	24	24	1e
16	1c	1d	1d	22	1e	1f	1e

[table que quantification chrominances Cb/Cr]

05	05	07	0e	1e	1e	1e	1e
05	06	08	14	1e	1e	1e	1e
07	08	11	1e	1e	1e	1e	1e
0e	14	1e	1e	1e	1e	1e	1e
1e	1e	1e	1e	1e	1e	1e	1e
1e	1e	1e	1e	1e	1e	1e	1e
1e	1e	1e	1e	1e	1e	1e	1e
1e	1e	1e	1e	1e	1e	1e	1e

Après quantification, les blocs de la MCU sont finalement :

[Y]:

0030 ffb0 0001 fff9 0005 fffc 0000 fffe	0032 0043 ffff ffe7 0003 fffe 0002 0001
fffe 0006 0003 fff6 0002 fffd 0000 0001	fffe fffa fffd 0004 000b 0000 0000 fffd
fffe 0017 fffa fffd 0000 ffff ffff 0002	fffb fffd 0007 fff8 0002 ffff 0000 0000
0008 0007 0003 0000 fffe 0001 fffe 0002	0000 0000 0006 0004 0000 0000 ffff 0000
0000 0006 0000 0001 0001 0000 0000 0000	0006 fffc 0000 0001 0000 0000 0000 0000
fffc fffa 0000 ffff 0000 0000 0000 0000	fffe fffe 0000 0000 0000 0000 0000 0000
0000 fffe 0000 0000 0000 0001 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000
0001 0000 0000 0000 0000 0000 0000 0000	ffff 0000 0000 0000 0000 0000 0000 0000

[Cb]:

ffd6 0000 fff9 0003 ffff 0002 0000 0003
fffd 0007 0000 0000 0000 0000 0001 0000
fffe 0003 0000 0000 0000 0000 0000 0000
0001 fffe 0001 0001 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 ffff 0000 ffff 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000

[Cr]:

0037 fffe 0008 0001 0000 0002 0000 0001
0000 0003 0000 0000 0000 0000 0001 0000
fffe 0001 0000 0000 0000 0000 0000 0000
0000 ffff 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000

Codage différentiel DC

La composante continue DC est la première valeur d'un bloc.

Premier bloc

la MCU de cet exemple est la première de l'image (en haut à gauche), la valeur `0x0030` (= 48, l'âge du capitaine) doit être encodée en premier. Elle appartient à la classe de magnitude 6 :

`-63, ..., -32, 32, ..., 63`. Dans cette classe, l'indice de 48 est `110000`

Pour continuer il faut connaître les tables de Huffman à utiliser pour l'encodage, qui seront ensuite incluses dans l'entête et lues par le décodeur. On considère ici les tables "statistiques standard" définies dans la norme [ISO/IEC IS 10918-1 | ITU-T Recommendation T.81](#) (section K.3) et aussi accessibles via `./jpeg2blabla -vh poupoupidou.jpg`

Le code de Huffman de la magnitude 6 est `1110` (elle est portée par une feuille de profondeur 4). Finalement, la suite de bits à inclure dans le flux de l'image compressée est le code de la magnitude puis l'indice dans la classe de magnitude, soit ici : `1110110000`

Bloc suivant

La valeur DC du bloc suivant de la composante Y est `0x0032` (= 50). Par contre, la valeur à encoder est la différence par rapport à la valeur DC du bloc précédent (de la même composante de lumière), soit ici $50 - 48 = 2$. Avec `011` le code de Huffman de la magnitude 2, l'encodage de ce coefficient DC dans le flux de bits est `01110`.

Codage AC avec RLE

On s'intéresse au codage des 63 coefficients AC du bloc de Cr. La composante continue `0x0037` a déjà été encodée dans le flux. Il reste donc la séquence :

```
0xfffe 0x0008 0x0001 0x0000 0x0002 0x0000 0x0001 ... 0x0000
```

- Un premier symbole RLE sur un octet est calculé pour le premier coefficient `0xfffe` (-2, de magnitude 2 et d'indice 2):
 - les quatre bits de poids forts sont nuls, car aucun coefficient nul ne précède ce coefficient.
 - les quatre bits de poids faible contiennent la magnitude de -2, qui est 2. Le symbole RLE est donc `0x02` ;
 - on n'insère pas directement ce symbole dans le flux mais plutôt son code de Huffman, ici `100` (3 bits). Attention à bien lire dans le bon arbre, celui des AC / chrominances!
 - finalement l'indice du coefficient sera inséré, ici `10` (2 bits)
- Le prochain coefficient non nul est `0x0008`, de magnitude 4 et d'indice 8.
 - Aucun coefficient nul ne précède ce coefficient, donc le symbole RLE correspondant est `0x04` ;
 - le flux contiendra le code de Huffman du symbole, `11000` (5 bits), puis l'indice `1000` (4 bits).
- Le prochain coefficient non nul est `0x0001`, de magnitude 1 et d'indice 1.
 - Aucun coefficient nul ne précède ce coefficient, donc le symbole RLE correspondant est `0x01` ;
 - le flux contiendra le code de Huffman du symbole, `01` (2 bits), puis l'indice `1` (1 bits).
- Le prochain coefficient non nul est `0x0002`, de magnitude 2 et d'indice 2.
 - Un coefficient le précède, donc le symbole RLE correspondant est `0x12` ;
 - le flux contiendra le code de Huffman du symbole, `111001` (6 bits), puis l'indice `10` (2 bits).
- Le code des six prochains coefficients non nuls est:
 - `0x0001` -> symbole RLE `0x11`, encodé `1011` puis `1`

- `0x0003` -> symbole RLE `0x12`, encodé `111001` puis `11`
- `0x0001` -> symbole RLE `0x41`, encodé `111010` puis `1`
- `0xffffe` -> symbole RLE `0x12`, encodé `111001` puis `01`
- `0x0001` -> symbole RLE `0x01`, encodé `01` puis `1`
- `0xfffff` -> symbole RLE `0x71`, encodé `1111010` puis `0`
- Tous les coefficients suivants étant nuls, il suffit de mettre une balise EOB `0x00` pour terminer le codage du bloc; son code de Huffman est `00`.

Un gain de place?

Au final l'ensemble des 63 coefficients AC du bloc Cr est totalement encodé dans le flux par les 66 bits: `100 10 11000 1000 01 1 111001 10 1011 1 111001 11 111010 1 111001 01 01 1 1111010 0 00`.

Sans codage de Huffman, la séquence aurait été `00000002 10 00000004 1000 00000001 1 00000012 10 00000011 1 00000012 11 00000041 1 00000012 01 00000001 1 00000071 0 00`, soit 99 bits.

Et brutalement, sans RLE ni magnitude, il aurait fallu 63 octets soit 504 bits... Et nous n'avons parlé que du stockage des données compressées, mais par rapport à l'image PPM, il faut en plus considérer le gain lié à l'encodage du signal lui-même.

Y'a plus qu'à!

Voilà, ça pique un peu les yeux mais cette annexe devrait bien vous aider à comprendre l'encodage.

Ah oui, mais nous on doit faire un décodeur! Ok, ben faites dans l'autre sens alors.

Mais si en testant sur `poupoupidou.jpg` vous obtenez une image plus proche de `pompompidou.jpg`, ça vaut le coup de relire encore une fois cette annexe!



1. en fait 2^{P-1} , avec ici une précision $P = 8$ ↩