

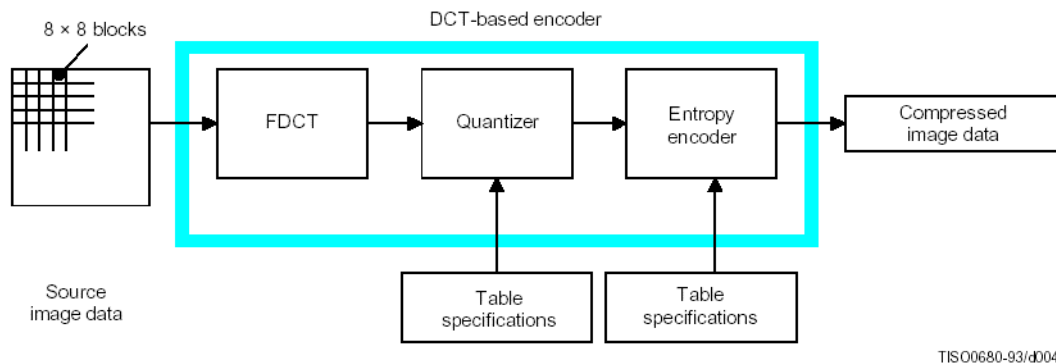
嵌入式視覺

JPEG 演算法實作

陳慶瀚

2014.10.08

JPEG 壓縮流程



TISO0680-93/d004

Figure 1 JPEG 流程

JPEG 壓縮流程

1. 色系變換
2. 正向離散餘弦轉換
3. 量化
4. ZigZag 編碼
5. RLE 編碼
6. Huffman 編碼

色彩轉換

所以 JPEG 壓縮的第一步，就是要把平常用來表示顏色的 RGB 格式，轉為亮度以及色度的表現方法，本次實驗是用 CCIR601 的格式，也就是 Y、Cb、Cr 格式，Y 代表亮度，Cb、Cr 則是代表色度 (藍色以及紅色的色度差)，而一般的轉換公式如下：

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ Cb &= 0.1687R - 0.3313G + 0.5B \\ Cr &= 0.5R - 0.4187G - 0.0813B \end{aligned}$$

但為了增快效率，因此把公式化為

$$\begin{aligned} Y &= 0.2990(R-G)+G+0.1140(B-G) \\ Cb &= 0.5643(B-Y) \\ Cr &= 0.7133(R-Y) \end{aligned}$$

這是一個不會失真的轉換，因此代表這個公式是可逆的。而 JPEG 轉化成 Y、Cb、Cr 的主要目的是，在壓縮的時候，可以用份量比較多的 Y(基於上述所說人類眼睛的習性)，而減少 Cb、Cr 的份量。以下是反轉的公式：

$$\begin{aligned} R &= Y+1.402(Cr-128) \\ G &= Y-0.34414(Cb-128)-0.71414(Cr-128) \\ B &= Y+1.772(Cb-128) \end{aligned}$$

正向離散餘弦轉換(Forward Discrete Cosine transform)

轉化為 Y、Cb、Cr 的格式，圖像檔案仍然以圖點的格式儲存，因此要合併鄰近的點，這時就必須透過離散餘弦轉換，將圖點儲存的方式轉為”變化率”的儲存方式，不過這裡就是 JPEG 造成圖像檔案失真的地方所在，數位化時所定的係數決定了資料流失量的多寡，以及影像品質的好壞。

JPEG 將整個 Y 矩陣與 Cb 矩陣與 Cr 矩陣，視為一個基本單元稱作 MCU，而每個 MCU 包含不超過 10 個矩陣，將圖像數據分成多個 8*8 矩陣，先將每個數值減去 128，然後代入 FDCT 的公式，減去 128 是因為 FDCT 所接受的，使得其數字的範圍在於-128~127 間，以下是 FDCT 的公式

$$\text{FDCT 轉換公式: } F(u, v) = \frac{2C(u)C(v)}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \cos \frac{(2x+1)u\pi}{2N} \cos \frac{(2y+1)v\pi}{2N}$$

$$\text{FDCT 反轉公式: } f(x, y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u, v) \cos \frac{(2x+1)u\pi}{2N} \cos \frac{(2y+1)v\pi}{2N}$$

其中

x,y 代表圖像數據矩陣內的某個數值的座標位置

f(x,y)代表圖像數據舉鎮內的某個數值

u,v 代表 FDCT 變換後矩陣內某個數值的座標位置

$F(u,v)$ 代表 FDCT 變換後舉鎮內的某個數值

$u=0$ 且 $v=0$ $c(u)c(v)=1/1.414$

$u>0$ 或 $v>0$ $c(u)c(v)=1$ 因為大小為 $8*8$ ，因此 $N=8$

經過 FDCT 後，矩陣數據自然數為頻率系數，這些係數又以 $F[0,0]$ 的值最大，稱為 DC，其餘 63 個頻率係數則大部分接近 0，稱為 AC。

量化

圖像數據轉換回頻率係數後，還要接受一項量化程序，才能進入編碼階段。量化需要兩個 $8*8$ 的矩陣，一個處理亮度係數，一個處理色度係數，將頻率係數除以量化矩陣的值後，取得與商數最近的整數，我們稱作量化。目的是將頻率係數由浮點數轉變為整數，這才方便最後的編碼階段，不過因為取整數的這個步驟，也損失了一些數據內容，以下是這次實驗所用的量化表：

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Figure 2 色度量化表

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	51	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	99

Figure 3 亮度量化表

這兩張表是依據心理視覺製作，也是控制 JPEG 壓縮比的所在處，這個步驟除掉了一些高頻的能量，但事實上人眼對高空間頻率沒有低頻敏感，所以處理後視覺的損失很小。

ZigZag 編碼

量化後的所有數據都是線性存放的，如果我們一行行處理這 64 個數字，每行的結尾的點和下行開始的點就沒有什麼關係了，所以 JPEG 規定按照下表來整理這 64 個數：

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	68	63

Figure 4 ZigZag 表

ZigZag 重排順序，由於圖像中相鄰的點值比較接近，重複出現的機率比較高，重排後的數據對後面的 RLE 編碼才有意義。

RLE 編碼

64 個變換數經量化後，左上角係數是直流分量（DC 係數），即空間域中 64 個圖像採樣值的均值。相鄰 8*8 塊之間的 DC 係數一般有很強的相關性，JPEG 標準對 DC 係數採用 DPCM（差分脈衝碼調制）方法，即對相鄰區塊之間的 L 係數的差值進行編碼。其餘 63 個交流分量（AC 係數）使用游程編碼，從左上角開始沿對角線方向，以 Z 字型（zigzag）進行掃描直到結束。

1、DC 係數編碼

使用 DPCM 對直流係數進行編碼，8*8 圖像塊經過 DCT 編碼之後得到的 DC 係數有兩個特點：一是係數的數值比較大；二是相鄰的 8*8 圖像塊的 DC 系數值變化不大。根據這個特點，JPEG 算法使用了 DPCM 技術，對相鄰圖像塊之間量化 DC 係數的差值（Diff）進行編碼。

$$\text{Diff} = \text{DC}(i) - \text{DC}(i-1)$$

2、AC 係數編碼

使用 RLE 對 AC 係數進行編碼。量化 AC 係數的特點是 1-64 向

量中包含有許多“0”係數，並且許多的“0”係數都是連續的，因此使用非常簡單和直觀的 RLE 編碼對它們編碼。

現在向量中有許多連續的“0”，可以使用 RLE 來壓縮掉這些“0”。跳過第一個 DC 向量，假設有一組 C 的 AC 向量（64 個的後 63 個）是：57,45,0,0,0,0,23,0,-30,-16,0,0,1,0,0,0,0,0,0,0,...

經過 RLE 壓縮後如下：

(0,57);(0,45);(4,23);(1,-30);(0,-16);(2,1);EOB。

EOB 是一個結束標記，表示後面都是“0”了。實際上，我們用 (0,0) 表示 EOB。但是，如果這組數字不以 0 結束，那麼就不需要 EOB。零行程長度超過 15 個時，有一個符號(15, 0)。

Huffman 編碼

Huffman 編碼器可以使用查表方法進行編碼。壓縮資料符號時，Huffman 編碼器對出現頻率較高的符號分配比較短的代碼，而對出現頻率比較高低的符號分配比較長的代碼。這種可變長度的 Huffman 表可以事先定義。

編碼時，每個矩陣資料的 DC 值與 63 個 AC 值，將分別使用不同的 Huffman 編碼表，而亮度與色度也需要不同的 Huffman 編碼表，所以一共需要四個編碼表，才能順利地完成 JPEG 編碼工作。

1、DC 編碼

JPEG 指出連續的 DC 之間有很緊密的聯繫，因此決定對 8*8 塊的 DC 值的差別進行編碼（Y，Cb，Cr 分別有自己的 DC）。

$Diff = DC(i) - DC(i-1)$ 所以當前塊的 $DC(i) = DC(i-1) + Diff$

DC 是採用差值脈衝碼調制的差值編碼法，也就是在同一個圖像分量

中取得每個 DC 值與前一個 DC 值的差值來編碼。採用這種編碼方

式的主要原因是由於在連續色調的圖像中，其差值多半比原值小，

對差值進行編碼所需的位元數比對原值編碼所需的位元數要少許

多。例如：差值為 5，它的 2 進制表示為 0b101，如果差值為 -5，則

先取其絕對值，再按位求反，即為 0b010。差值所需位元數與差值

內容的對照表如下：

差值位數	DC 差值內容
0	0
1	-1, 1
2	-3, -2, 2, 3
3	-7, ..., -4, 4, ..., 7
4	-15, ..., -8, 8, ..., 15
5	-31, ..., -16, 16, ..., 31
6	-63, ..., -32, 32, ..., 63
7	-127, ..., -64, 64, ..., 127
8	-255, ..., -128, 128, ..., 255
9	-511, ..., -256, 256, ..., 511
10	-1023, ..., -512, 512, ..., 1023
11	-2047, ..., -1024, 1024, ..., 2047

Figure 5 DC 差值對照表

在差值前端需要另外加入一些差值的 Huffman 碼值，例如亮度

差值為 5 (101)，則 Huffman 碼值為 100，兩者連接在一起為 100101。下列的兩個表分別是亮度和色度的 Huffman 碼表。根據這兩個 Huffman 碼表，就可以完成 DC 的 Huffman 編碼過程。

差值位數	編碼位數	Huffman 編碼
亮度 DC 差值的 Huffman 編碼表		
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110
色度 DC 差值的 Huffman 編碼表		
0	2	00
1	2	01
2	2	10
3	3	110
4	4	1110
5	5	11110
6	6	111110
7	7	1111110
8	8	11111110
9	9	111111110
10	10	1111111110
11	11	11111111110

Figure 6 DC Huffman 編碼表

JPEG 從“0”開始對 DC 編碼，所以 DC (0) = 0，然後再將當前

的 Diff 值加上上一個值得到當前值。

例如 Y 矩陣中的一個 8*8 塊的 DC 值為 10，其 2 進制值為 0b1010，碼長為 4，查表得 Huffman 碼長為 3，Huffman 碼值為 101，則 Huffman 編碼為 1011010。再如 DC 值為-10，那麼其反碼為 0101，因此其 Huffman 編碼為 1010101。

2. AC 編碼

AC 編碼方式與 DC 略有不同，在 AC 編碼之前，首先將 AC 值做 Z 重排，將 AC 係數轉為中間符號，中間符號表示為 RRRR/SSSS，RRRR 是指第非零的 AC 之前，其值為 0 的 AC 個數，SSSS 是指 AC 值所需的位數，AC 係數的範圍與 SSSS 的對應關係與 DC 差值 Bits 數與差值內容對照表相似。

如果連續為 0 的 AC 個數大於 15，則用 15/0 來表示連續的 16 個 0，15/0 稱為 ZRL (Zero Run-Length)，而 (0/0) 稱為 EOB

(End of Block) 用來表示其後所剩餘的 AC 係數皆等於 0，以中間符號值作為索引值，從相應的 AC 編碼表中找出適當的霍夫曼碼值，再與 AC 值相連即可。

為了提高儲存效率，JPEG 裡並不直接保存數值，而是將數值按位數分成 16 組：

數值	SSSS 組	實際保存值
0	0	不保存
-1, 1	1	0, 1
-3, -2, 2, 3	2	00, 01, 10, 11
-7, ..., -4, 4, ..., 7	3	000, 001, 010, 011, 100, 101, 110, 111
-15, ..., -8, 8, ..., 15	4	0000, ..., 0111, 1000, ..., 1111
-31, ..., -16, 16, ..., 31	5	...
-63, ..., -32, 32, ..., 63	6	...
-127, ..., -64, 64, ..., 127	7	...
-255, ..., -128, 128, ..., 255	8	...
-511, ..., -256, 256, ..., 511	9	...
-1023, ..., -512, 512, ..., 1023	10	...
-2047, ..., -1024, 1024, ..., 2047	11	...
-4095, ..., -2048, 2048, ..., 4095	12	...
-8191, ..., -4096, 4096, ..., 8191	13	...
-16383, ..., -8192, 8192, ..., 16383	14	...
-32767, ..., -16384, 16384, ..., 32767	15	...

Figure 7 SSSS 組對照表

繼續前面的例子：

(0,57);(0,45);(4,23);(1,-30);(0,-16);(2,1); (0, 0) 只處理每對數右邊的

數：

57 是第 6 組的，實際保存值為 111001，所以被編碼為 (6,111001)

45 -> (6,101101); 23 -> (5,10111); -30 -> (5,00001); -16 -> (5,01111); 1 -

> (1,1) 前面的那串數字就變成了：

(0,6),111001;(0,6),101101;(4,5),10111;(1,5),00001;(0,5),01111;(2,1),1;(0,0) 括弧裡的數值正好合成一個位元組。後面被編碼的數位表示範圍是 $-32767 \dots 32767$ 。合成的位元組裡，高 4 位是前續 0 的個數，低 4 位元描述了後面數字的位元數。

亮度 AC 表

Run/Size	Code length	Code word
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
2/1	5	11100
2/2	8	11111001
2/3	10	1111110111
2/4	12	111111110100
2/5	16	1111111110001001
2/6	16	1111111110001010
2/7	16	1111111110001011
2/8	16	1111111110001100
2/9	16	1111111110001101
2/A	16	1111111110001110
3/1	6	111010
3/2	9	111110111
3/3	12	111111110101
3/4	16	1111111110001111
3/5	16	1111111110010000
3/6	16	1111111110010001
3/7	16	1111111110010010
3/8	16	1111111110010011
3/9	16	1111111110010100
3/A	16	1111111110010101

Figure 8 亮度 AC 表(4/1)

Run/Size	Code length	Code word
4/1	6	111011
4/2	10	1111111000
4/3	16	111111110010110
4/4	16	111111110010111
4/5	16	111111110011000
4/6	16	111111110011001
4/7	16	111111110011010
4/8	16	111111110011011
4/9	16	111111110011100
4/A	16	111111110011101
5/1	7	1111010
5/2	11	1111110111
5/3	16	111111110011110
5/4	16	111111110011111
5/5	16	111111110100000
5/6	16	111111110100001
5/7	16	111111110100010
5/8	16	111111110100011
5/9	16	111111110100100
5/A	16	111111110100101
6/1	7	1111011
6/2	12	11111110110
6/3	16	111111110100110
6/4	16	111111110100111
6/5	16	111111110101000
6/6	16	111111110101001
6/7	16	111111110101010
6/8	16	111111110101011
6/9	16	111111110101100
6/A	16	111111110101101
7/1	8	11111010
7/2	12	11111110111
7/3	16	111111110101110
7/4	16	111111110101111
7/5	16	111111110110000
7/6	16	111111110110001
7/7	16	111111110110010
7/8	16	111111110110011
7/9	16	111111110110100
7/A	16	111111110110101
8/1	9	111111000
8/2	15	11111111000000

Figure 8 亮度 AC 表(4/2)

Run/Size	Code length	Code word
8/3	16	111111110110110
8/4	16	111111110110111
8/5	16	111111110111000
8/6	16	111111110111001
8/7	16	111111110111010
8/8	16	111111110111011
8/9	16	111111110111100
8/A	16	111111110111101
9/1	9	111111001
9/2	16	111111110111110
9/3	16	111111110111111
9/4	16	111111111000000
9/5	16	111111111000001
9/6	16	111111111000010
9/7	16	111111111000011
9/8	16	111111111000100
9/9	16	111111111000101
9/A	16	111111111000110
A/1	9	111111010
A/2	16	111111111000111
A/3	16	111111111001000
A/4	16	111111111001001
A/5	16	111111111001010
A/6	16	111111111001011
A/7	16	111111111001100
A/8	16	111111111001101
A/9	16	111111111001110
A/A	16	111111111001111
B/1	10	1111111001
B/2	16	111111111010000
B/3	16	111111111010001
B/4	16	111111111010010
B/5	16	111111111010011
B/6	16	111111111010100
B/7	16	111111111010101
B/8	16	111111111010110
B/9	16	111111111010111
B/A	16	111111111011000
C/1	10	1111111010
C/2	16	111111111011001
C/3	16	111111111011010
C/4	16	111111111011011

Figure 8 亮度 AC 表(4/3)

Run/Size	Code length	Code word
C/5	16	111111111011100
C/6	16	111111111011101
C/7	16	111111111011110
C/8	16	111111111011111
C/9	16	111111111100000
C/A	16	111111111100001
D/1	11	1111111000
D/2	16	111111111100010
D/3	16	111111111100011
D/4	16	111111111100100
D/5	16	111111111100101
D/6	16	111111111100110
D/7	16	111111111100111
D/8	16	111111111101000
D/9	16	111111111101001
D/A	16	111111111101010
E/1	16	111111111101011
E/2	16	111111111101100
E/3	16	111111111101101
E/4	16	111111111101110
E/5	16	111111111101111
E/6	16	111111111110000
E/7	16	111111111110001
E/8	16	111111111110010
E/9	16	111111111110011
E/A	16	111111111110100
F/0 (ZRL)	11	1111111001
F/1	16	111111111110101
F/2	16	111111111110110
F/3	16	111111111110111
F/4	16	111111111111000
F/5	16	111111111111001
F/6	16	111111111111010
F/7	16	111111111111011
F/8	16	111111111111100
F/9	16	111111111111101
F/A	16	111111111111110

Figure 8 亮度 AC 表(4/4)

色度 AC 表

Run/Size	Code length	Code word
0/0 (EOB)	2	00
0/1	2	01
0/2	3	100
0/3	4	1010
0/4	5	11000
0/5	5	11001
0/6	6	111000
0/7	7	1111000
0/8	9	111110100
0/9	10	1111110110
0/A	12	11111110100
1/1	4	1011
1/2	6	111001
1/3	8	11110110
1/4	9	111110101
1/5	11	11111110110
1/6	12	111111110101
1/7	16	1111111110001000
1/8	16	1111111110001001
1/9	16	1111111110001010
1/A	16	1111111110001011
2/1	5	11010
2/2	8	11110111
2/3	10	1111110111
2/4	12	111111110110
2/5	15	111111111000010
2/6	16	1111111110001100
2/7	16	1111111110001101
2/8	16	1111111110001110
2/9	16	1111111110001111
2/A	16	1111111110010000
3/1	5	11011
3/2	8	11111000
3/3	10	1111111000
3/4	12	111111110111
3/5	16	1111111110010001
3/6	16	1111111110010010
3/7	16	1111111110010011
3/8	16	1111111110010100
3/9	16	1111111110010101
3/A	16	1111111110010110
4/1	6	111010

Figure 9 色度 AC 表(4/1)

Run/Size	Code length	Code word
4/2	9	111110110
4/3	16	1111111110010111
4/4	16	1111111110011000
4/5	16	1111111110011001
4/6	16	1111111110011010
4/7	16	1111111110011011
4/8	16	1111111110011100
4/9	16	1111111110011101
4/A	16	1111111110011110
5/1	6	111011
5/2	10	1111111001
5/3	16	1111111110011111
5/4	16	1111111110100000
5/5	16	1111111110100001
5/6	16	1111111110100010
5/7	16	1111111110100011
5/8	16	1111111110100100
5/9	16	1111111110100101
5/A	16	1111111110100110
6/1	7	1111001
6/2	11	11111110111
6/3	16	1111111110100111
6/4	16	1111111110101000
6/5	16	1111111110101001
6/6	16	1111111110101010
6/7	16	1111111110101011
6/8	16	1111111110101100
6/9	16	1111111110101101
6/A	16	1111111110101110
7/1	7	1111010
7/2	11	11111111000
7/3	16	1111111110101111
7/4	16	1111111110110000
7/5	16	1111111110110001
7/6	16	1111111110110010
7/7	16	1111111110110011
7/8	16	1111111110110100
7/9	16	1111111110110101
7/A	16	1111111110110110
8/1	8	11111001
8/2	16	1111111110110111
8/3	16	1111111110111000

Figure 9 色度 AC 表(4/2)

Run/Size	Code length	Code word
8/4	16	111111110111001
8/5	16	111111110111010
8/6	16	111111110111011
8/7	16	111111110111100
8/8	16	111111110111101
8/9	16	111111110111110
8/A	16	111111110111111
9/1	9	111110111
9/2	16	111111111000000
9/3	16	111111111000001
9/4	16	111111111000010
9/5	16	111111111000011
9/6	16	111111111000100
9/7	16	111111111000101
9/8	16	111111111000110
9/9	16	111111111000111
9/A	16	111111111001000
A/1	9	111111000
A/2	16	111111111001001
A/3	16	111111111001010
A/4	16	111111111001011
A/5	16	111111111001100
A/6	16	111111111001101
A/7	16	111111111001110
A/8	16	111111111001111
A/9	16	111111111010000
A/A	16	111111111010001
B/1	9	111111001
B/2	16	111111111010010
B/3	16	111111111010011
B/4	16	111111111010100
B/5	16	111111111010101
B/6	16	111111111010110
B/7	16	111111111010111
B/8	16	111111111011000
B/9	16	111111111011001
B/A	16	111111111011010
C/1	9	111111010
C/2	16	111111111011011
C/3	16	111111111011100
C/4	16	111111111011101
C/5	16	111111111011110

Figure 9 色度 AC 表(4/3)

Run/Size	Code length	Code word
C/6	16	111111111011111
C/7	16	111111111100000
C/8	16	111111111100001
C/9	16	111111111100010
C/A	16	111111111100011
D/1	11	1111111001
D/2	16	111111111100100
D/3	16	111111111100101
D/4	16	111111111100110
D/5	16	111111111100111
D/6	16	111111111101000
D/7	16	111111111101001
D/8	16	111111111101010
D/9	16	111111111101011
D/A	16	111111111101100
E/1	14	1111111100000
E/2	16	111111111101101
E/3	16	111111111101110
E/4	16	111111111101111
E/5	16	111111111110000
E/6	16	111111111110001
E/7	16	111111111110010
E/8	16	111111111110011
E/9	16	111111111110100
E/A	16	111111111110101
F/0 (ZRL)	10	111111010
F/1	15	11111111000011
F/2	16	111111111110110
F/3	16	111111111110111
F/4	16	111111111111000
F/5	16	111111111111001
F/6	16	111111111111010
F/7	16	111111111111011
F/8	16	111111111111100
F/9	16	111111111111101
F/A	16	111111111111110

Figure 9 色度 AC 表(4/4)

(0,6),111001;(0,6),101101;(4,5),10111;(1,5),00001;(0,5),01111;(2,1),1;(0,0)

那麼最後對於前面的例子表示的C的63個AC係數按位流寫入JPEG檔

中是這樣的：111000 111001 111000 101101 1111111110011001

10111 11111110110 00001 11001 01111 11010 1 00

到此為止，JPEG 壓縮完成。

3-2 JPEG 實作

本次實驗採用簡單的 8*8 的圖檔實作，用階段性的 JPEG 實作方法來測試實驗板，因為 8*8 的圖檔太小，以下是將測試圖檔放大 500 倍的樣子：

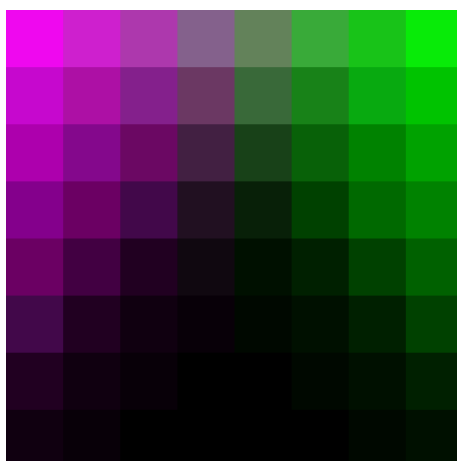


Figure 10 測試檔圖

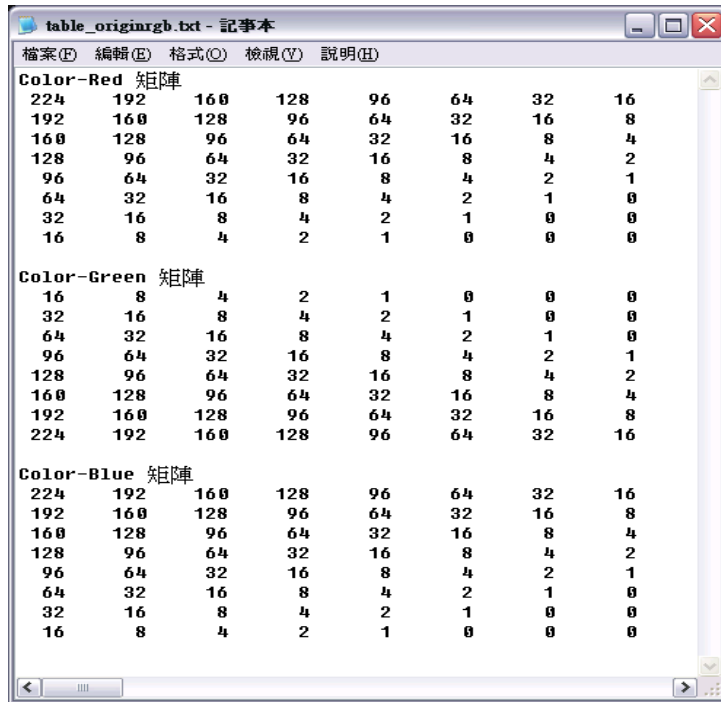


Figure 11 測試檔圖 RGB 資料矩陣

色系變換

將原圖的 RGB 轉成 Y Cb Cr。

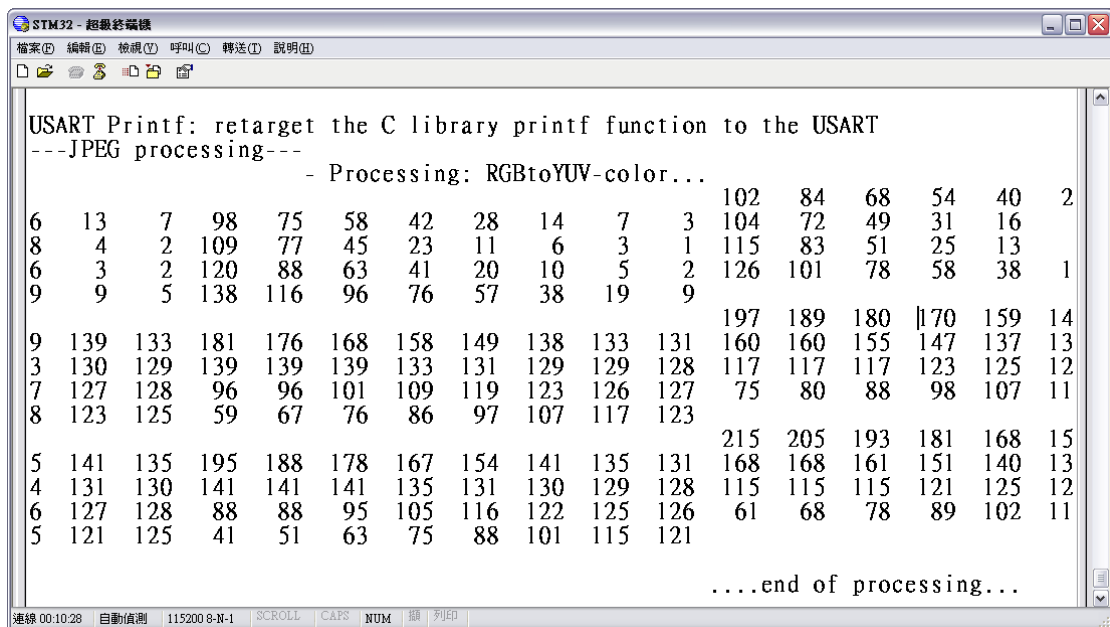


Figure 12 Y Cb Cr 圖檔資料

正向離散餘弦轉換 (FDCT)

將 Y Cb Cr 用 FDCT 轉換。

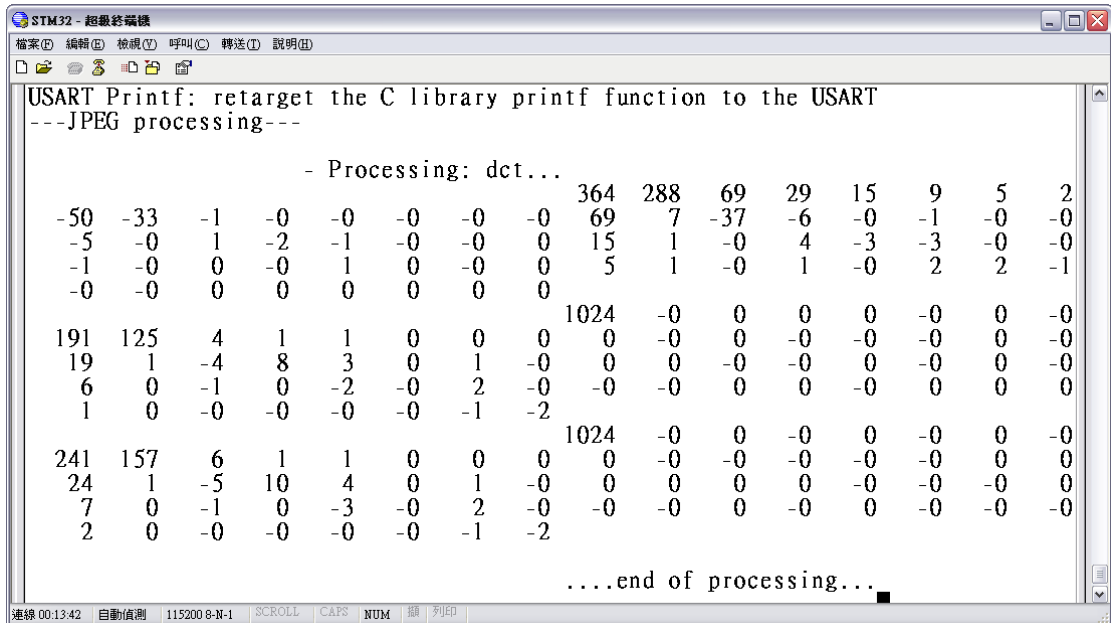


Figure 13 FDCT 轉換後圖檔資料

量化 (Quantize)

FDCT 後的數值，量化其數值。

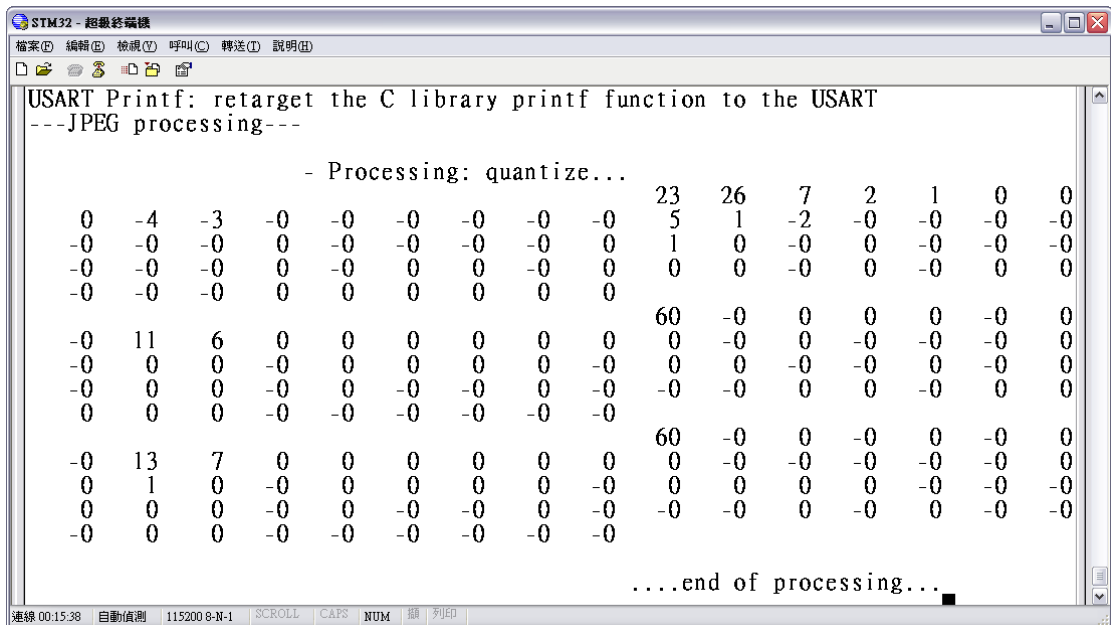
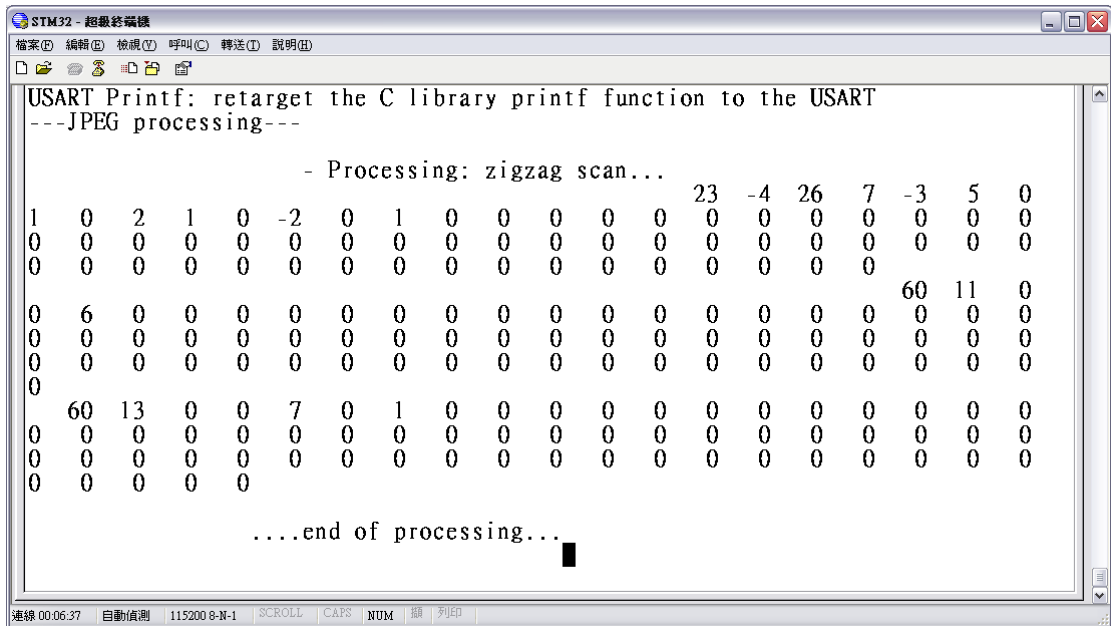


Figure 14 量化後的圖檔資料

ZigZag 編碼

量化後將其中的資料經過 ZigZag 編碼。



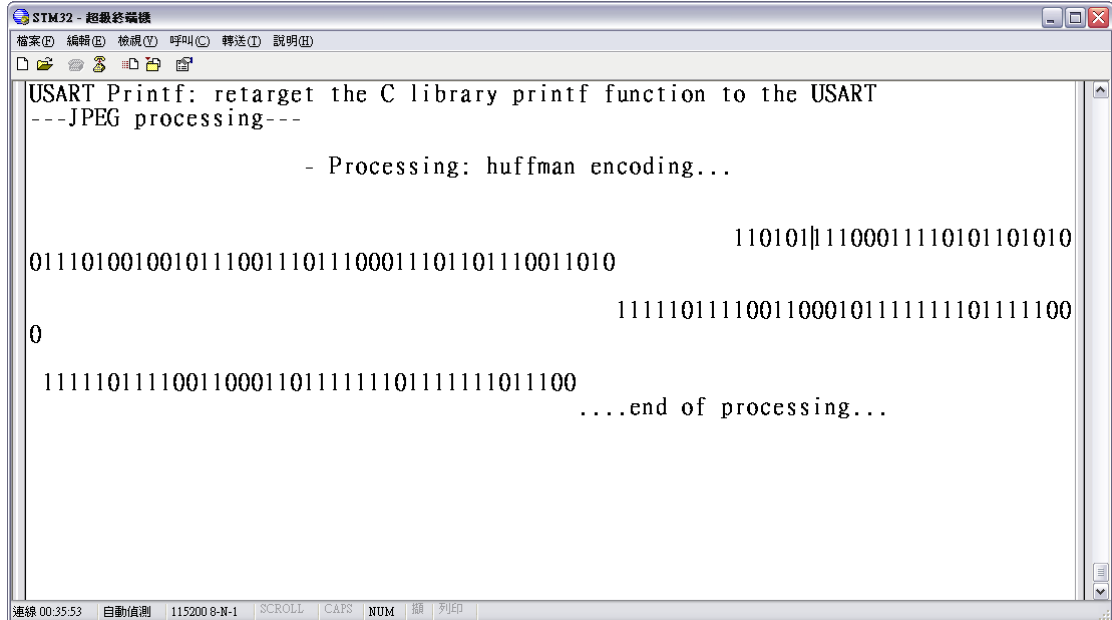
```
STM32 - 超級終端機
檔案(F) 編輯(E) 檢視(V) 呼叫(C) 轉送(T) 說明(H)
USART Printf: retarget the C library printf function to the USART
---JPEG processing---
      - Processing: zigzag scan...
1  0  2  1  0 -2  0  1  0  0  0  0  23 -4 26  7 -3  5  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  60 11  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0
60 13  0  0  7  0  1  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0
      ....end of processing...

```

Figure 15 ZigZag 編碼後的圖檔資料

RLE 編碼以及 Huffman 編碼

把最後出來的資料經過編碼所產生的 Bit String。



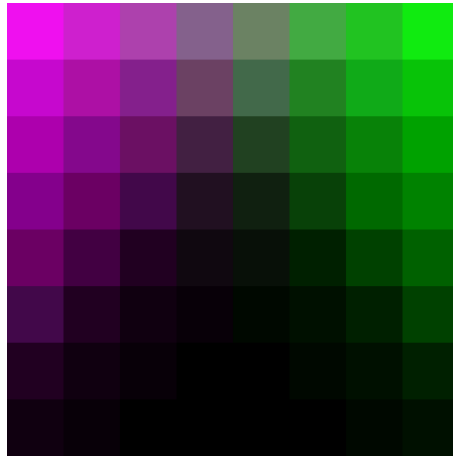
```
STM32 - 超級終端機
檔案(F) 編輯(E) 檢視(V) 呼叫(C) 轉送(T) 說明(H)
011101001001011100111011100011101101110011010
110101|1100011110101101010
0
1111101111001100010111111101111100
....end of processing...
```

Figure 16 Huffman 編碼後的圖檔資料

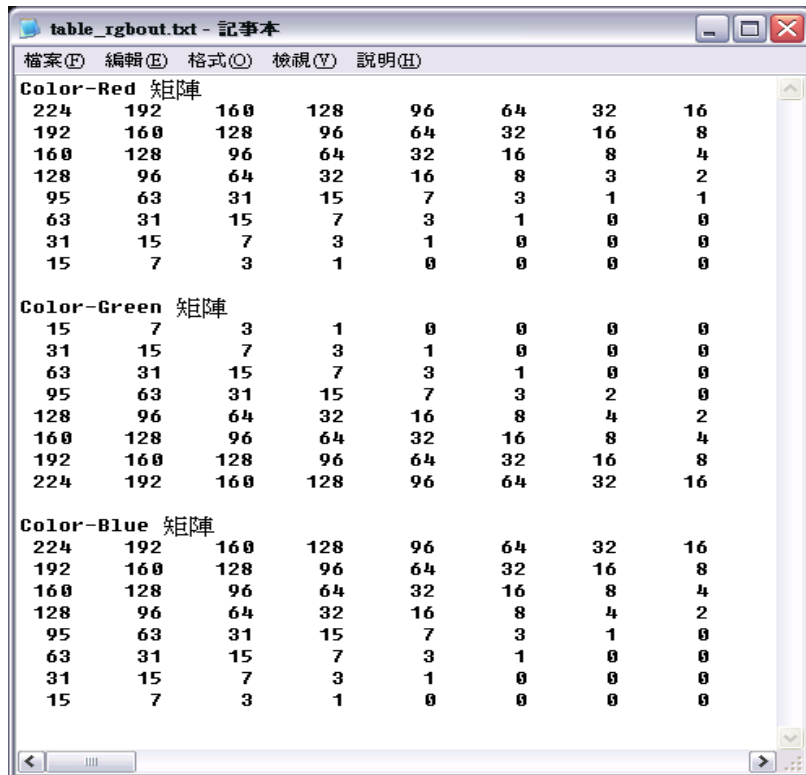
實驗

實驗結果

將 bit String 還原成圖檔，以下為經過這次實驗後產生的 JPEG 壓縮結果。



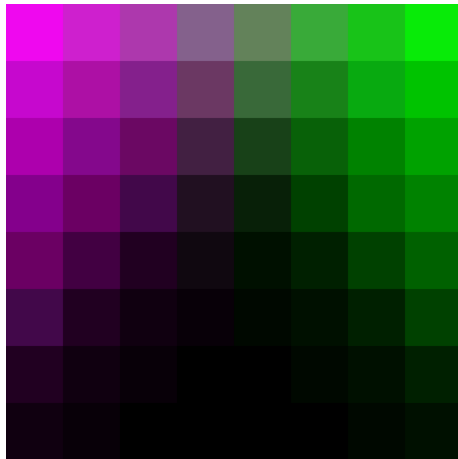
經 JPEG 壓縮後圖檔



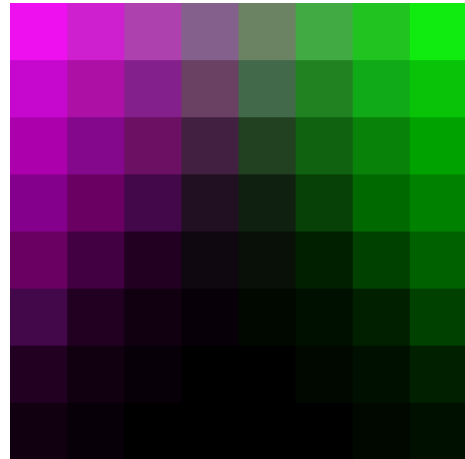
經 JPEG 壓縮後圖檔資料

1. 本次實驗將 8*8 的圖檔，透過 STM32F429 實作 JPEG 壓縮，因為只有 8*8 的原圖測試，所以其中只有極少部分的失真，以下是失真的部分。

原圖



JPEG 壓縮後



原圖

table_origrgb.txt - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)

Color-Red 矩陣

224	192	160	128	96	64	32	16
192	160	128	96	64	32	16	8
160	128	96	64	32	16	8	4
128	96	64	32	16	8	4	2
96	64	32	16	8	4	2	1
64	32	16	8	4	2	1	0
32	16	8	4	2	1	0	0
16	8	4	2	1	0	0	0

Color-Green 矩陣

16	8	4	2	1	0	0	0
32	16	8	4	2	1	0	0
64	32	16	8	4	2	1	0
96	64	32	16	8	4	2	1
128	96	64	32	16	8	4	2
160	128	96	64	32	16	8	4
192	160	128	96	64	32	16	8
224	192	160	128	96	64	32	16

Color-Blue 矩陣

224	192	160	128	96	64	32	16
192	160	128	96	64	32	16	8
160	128	96	64	32	16	8	4
128	96	64	32	16	8	4	2
96	64	32	16	8	4	2	1
64	32	16	8	4	2	1	0
32	16	8	4	2	1	0	0
16	8	4	2	1	0	0	0

JPEG 壓縮後

table_rgbout.txt - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)

Color-Red 矩陣

224	192	160	128	96	64	32	16
192	160	128	96	64	32	16	8
160	128	96	64	32	16	8	4
128	96	64	32	16	8	3	2
95	63	31	15	7	3	1	1
63	31	15	7	3	1	0	0
31	15	7	3	1	0	0	0
15	7	3	1	0	0	0	0

Color-Green 矩陣

15	7	3	1	0	0	0	0
31	15	7	3	1	0	0	0
63	31	15	7	3	1	0	0
95	63	31	15	7	3	2	0
128	96	64	32	16	8	4	2
160	128	96	64	32	16	8	4
192	160	128	96	64	32	16	8
224	192	160	128	96	64	32	16

Color-Blue 矩陣

224	192	160	128	96	64	32	16
192	160	128	96	64	32	16	8
160	128	96	64	32	16	8	4
128	96	64	32	16	8	4	2
95	63	31	15	7	3	1	0
63	31	15	7	3	1	0	0
31	15	7	3	1	0	0	0
15	7	3	1	0	0	0	0

原圖與 JPEG 壓縮後失真比較

JPEG source code

```
#include <iostream.h>
```

```
#include <stdio.h>
```

```
#include <math.h>
```

```
/*===== Test data =====*/
```

```
float r_in[8][8] = { { 224, 192, 160, 128, 96, 64, 32, 16},  
                    { 192, 160, 128, 96, 64, 32, 16, 8},  
                    { 160, 128, 96, 64, 32, 16, 8, 4},  
                    { 128, 96, 64, 32, 16, 8, 4, 2},  
                    { 96, 64, 32, 16, 8, 4, 2, 1},  
                    { 64, 32, 16, 8, 4, 2, 1, 0},  
                    { 32, 16, 8, 4, 2, 1, 0, 0},  
                    { 16, 8, 4, 2, 1, 0, 0, 0}};
```

```
float g_in[8][8] = { { 16, 8, 4, 2, 1, 0, 0, 0},  
                    { 32, 16, 8, 4, 2, 1, 0, 0},  
                    { 64, 32, 16, 8, 4, 2, 1, 0},  
                    { 96, 64, 32, 16, 8, 4, 2, 1},  
                    { 128, 96, 64, 32, 16, 8, 4, 2},  
                    { 160, 128, 96, 64, 32, 16, 8, 4},  
                    { 192, 160, 128, 96, 64, 32, 16, 8},  
                    { 224, 192, 160, 128, 96, 64, 32, 16}};
```

```
float b_in[8][8] = { { 224, 192, 160, 128, 96, 64, 32, 16},  
                    { 192, 160, 128, 96, 64, 32, 16, 8},  
                    { 160, 128, 96, 64, 32, 16, 8, 4},  
                    { 128, 96, 64, 32, 16, 8, 4, 2},  
                    { 96, 64, 32, 16, 8, 4, 2, 1},  
                    { 64, 32, 16, 8, 4, 2, 1, 0},  
                    { 32, 16, 8, 4, 2, 1, 0, 0},  
                    { 16, 8, 4, 2, 1, 0, 0, 0}};
```

```
float q0[8][8] = { { 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, 51, 87, 80, 82},
        { 18, 22, 37, 56, 68, 109, 103, 77},
        { 24, 35, 55, 64, 81, 104, 113, 92},
        { 99, 64, 78, 87, 103, 121, 120, 101},
        { 72, 92, 95, 98, 112, 100, 103, 99}};

float q1[8][8] = {{ 17, 18, 24, 47, 99, 99, 99, 99},
                 { 18, 21, 26, 66, 99, 99, 99, 99},
                 { 24, 26, 56, 99, 99, 99, 99, 99},
                 { 47, 66, 99, 99, 99, 99, 99, 99},
                 { 99, 99, 99, 99, 99, 99, 99, 99},
                 { 99, 99, 99, 99, 99, 99, 99, 99},
                 { 99, 99, 99, 99, 99, 99, 99, 99},
                 { 99, 99, 99, 99, 99, 99, 99, 99}};

/*===== Test data-End =====*/

/*===== RGB-YUV =====*/
void rgb_to_yuv(float r[8][8],float g[8][8],float b[8][8],float y[8][8],float u[8][8],float v[8][8])
{
int i,j;

for ( i=0 ; i<8 ; i++){
    for ( j=0 ; j<8 ; j++){
        //2013--05--31--MARK WHY ALGORIM????
        y[i][j]= 0.299 *r[i][j]+0.587 *g[i][j]+0.114 *b[i][j]      ;
        u[i][j]=-0.1687*r[i][j]-0.3313*g[i][j]+0.5      *b[i][j]+128;
        v[i][j]= 0.5      *r[i][j]-0.4187*g[i][j]-0.0813*b[i][j]+128;
    }
}
}

/*===== RGB-YUV-End =====*/

/*===== YUV-RGB =====*/
void yuv_to_rgb(float y[8][8],float u[8][8],float v[8][8],float r[8][8],float g[8][8],float b[8][8])
{

```

```

int i,j;

for ( i=0 ; i<8 ; i++){
  for ( j=0 ; j<8 ; j++){
    r[i][j]=y[i][j]          +1.402 *(v[i][j]-128);
    g[i][j]=y[i][j]-0.34414*(u[i][j]-128)-0.71414*(v[i][j]-128);
    b[i][j]=y[i][j]+1.772 *(u[i][j]-128)          ;
  }
}
}

/*===== YUV-RGB-End =====*/

/*===== DCT =====*/

//2013--05--31--MARK WHAT IS DCT??
void dct (float pic_in[8][8], float enc_out[8][8])
{
  int      u,v,x,y;
  float u_cs,v_cs,Pi;

  Pi=3.1415927;
  for ( u=0 ; u<8 ; u++){
    for ( v=0 ; v<8 ; v++){
      for ( x=0 ; x<8 ; x++){
        for ( y=0 ; y<8 ; y++){
          u_cs=cos(((2*x+1)*u*Pi)/16);//WHY?
          if (u==0) u_cs=(1/(sqrt(2)));
          v_cs=cos(((2*y+1)*v*Pi)/16);
          if (v==0) v_cs=(1/(sqrt(2)));
          enc_out[v][u]+=0.25*pic_in[y][x]*u_cs*v_cs;
        }
      }
    }
  }
}

/*===== DCT-End =====*/

```

```

/*===== DCT^-1 =====*/
void inv_dct (float enc_in[8][8], float rec_out[8][8])
{
    int      u,v,x,y;
    float u_cs,v_cs,Pi;

    Pi=3.1415927;
    for ( x=0 ; x<8 ; x++){
        for ( y=0 ; y<8 ; y++){
            for ( u=0 ; u<8 ; u++){
                for ( v=0 ; v<8 ; v++){
                    u_cs=cos(((2*x+1)*u*Pi)/16);
                    if (u==0) u_cs=(1/(sqrt(2)));
                    v_cs=cos(((2*y+1)*v*Pi)/16);
                    if (v==0) v_cs=(1/(sqrt(2)));
                    rec_out[y][x]+=0.25*enc_in[v][u]*u_cs*v_cs;
                }
            }
        }
    }
}
/*===== DCT^-1-End =====*/

```

```

/*===== Quant =====*/
void quantize(float dctb[8][8],float qb[8][8],int n)
{
    int      u,v;

    for ( v=0 ; v<8 ; v++){
        for ( u=0 ; u<8 ; u++){
            if (n==0)
                qb[v][u]=dctb[v][u]/q0[v][u];
            else
                qb[v][u]=dctb[v][u]/q1[v][u];
        }
    }
}

```

```

}

}

/*===== Quant-End =====*/

/*===== Dequant =====*/
void dequantize(float qb[8][8],float dctb[8][8],int n)
{

int    u,v;

for ( v=0 ; v<8 ; v++){
    for ( u=0 ; u<8 ; u++){
        if (n==0)
            dctb[v][u]=qb[v][u]*q0[v][u];
        else
            dctb[v][u]=qb[v][u]*q1[v][u];
    }
}

}

/*===== Dequant-End =====*/

/*===== ZigZag =====*/
void zigzag(float quant_in[8][8], float zigzaged_out[64])
{
int i;
int u[64]={1,2,1,1,2,3,4,3,2,1,1,2,3,4,5,6,
           5,4,3,2,1,1,2,3,4,5,6,7,8,7,6,5,
           4,3,2,1,2,3,4,5,6,7,8,8,7,6,5,4,
           3,4,5,6,7,8,8,7,6,5,6,7,8,8,7,8
           };
int v[64]={1,1,2,3,2,1,1,2,3,4,5,4,3,2,1,1,
           2,3,4,5,6,7,6,5,4,3,2,1,1,2,3,4,
           5,6,7,8,8,7,6,5,4,3,2,3,4,5,6,7,

```



```

        8,8,7,6,5,4,5,6,7,8,8,7,6,7,8,8
    };

    for ( i=0 ; i<64 ; i++){
        zigzaged_out[i]=quant_in[u[i]][v[i]];
    }
}
/*===== ZigZag-End =====*/

/*===== Inv-ZigZag =====*/
void inv_zigzag(float zigzaged_in[64], float quant_out[8][8])
{
    int i;
    int u[64]={1,2,1,1,2,3,4,3,2,1,1,2,3,4,5,6,
                5,4,3,2,1,1,2,3,4,5,6,7,8,7,6,5,
                4,3,2,1,2,3,4,5,6,7,8,8,7,6,5,4,
                3,4,5,6,7,8,8,7,6,5,6,7,8,8,7,8
    };
    int v[64]={1,1,2,3,2,1,1,2,3,4,5,4,3,2,1,1,
                2,3,4,5,6,7,6,5,4,3,2,1,1,2,3,4,
                5,6,7,8,8,7,6,5,4,3,2,3,4,5,6,7,
                8,8,7,6,5,4,5,6,7,8,8,7,6,7,8,8
    };

    for ( i=0 ; i<64 ; i++){
        quant_out[u[i]][v[i]]=zigzaged_in[i];
    }
}
/*===== Inv-ZigZag-End =====*/

//*****

/*===== Rll =====*/
void rll(float quant_in[8][8],float rll_out[96],int *buffsize)
{
    int buffcnt,u,v,zcount;
    int waszero;

```

```
buffcnt=0; u=0; v=0; zcount=0; waszero=0; // initialize variables
```

```
for (u=0 ; u<8 ; u++){
```

```
  for (v=0 ; v<8 ; v++){
```

```
    if ((int)quant_in[u][v]==0){
```

```
      if ((u==7) && (v==7)){
```

```
        zcount++;
```

```
        rll_out[buffcnt]=0;
```

```
        buffcnt++;
```

```
        rll_out[buffcnt]=zcount;
```

```
    } //end-if (u,v)==7
```

```
    else{
```

```
      if (zcount<14){
```

```
        zcount++;
```

```
        rll_out[buffcnt]=0;
```

```
        rll_out[buffcnt+1]=zcount;
```

```
        waszero=1;
```

```
      } //end-if zcount<14
```

```
    else{
```

```
      rll_out[buffcnt]=0;
```

```
      buffcnt++;
```

```
      rll_out[buffcnt]=zcount;
```

```
      buffcnt++;
```

```
      zcount=0;
```

```
      waszero=0;
```

```
    } //end-else zcount<14
```

```
  } //end-else (u,v)==7
```

```
} //end-if quant_in==0?
```

```
else{
```

```
  zcount=0;
```

```
  if (waszero==1){
```

```
    buffcnt+=2;
```

```
    rll_out[buffcnt]=quant_in[u][v];
```

```
    buffcnt++;
```

```
    waszero=0;
```

```
  } //end-if waszero=1?
```

```
  else{
```

```
    rll_out[buffcnt]=quant_in[u][v];
```

```

        buffcnt++;
    } //end-else waszero==1?
} //end-else quant==0?
} //end-for v
} //end-for u
*buffsize=buffcnt;
} //end rll
/*===== Rll-End =====*/

/*===== Inv_Rll =====*/
void inv_rll(int rll_in[8][8],float quant_out[8][8])
{

}
/*===== Inv_Rll-End =====*/

//*****

/*===== Trace_Rll =====*/
void trace_rll(float tracebuff[96],int maxcount)
{
int i,j,k;

cout << maxcount << endl;
i=0; j=0;
for( i=0 ; i<(maxcount+1) ; i++){
    printf("i= %i. Buffer= %4.0f ||",i,tracebuff[i]);
    j++; if (j==4) { printf("\n"); j=0; }
}
cout << endl;
}
/*===== Trace_Rll-End =====*/

/*===== Trace Mode =====*/
void trace3 (float data_to_dump1[8][8], float data_to_dump2[8][8], float data_to_dump3[8][8])
{

```

```

int u,v;

for ( u=0 ; u<8 ; u++ ){
    for ( v=0 ; v<8 ; v++ ){
        printf("%4.0f ",data_to_dump1[u][v]);
    }
    printf("      ");
    for ( v=0 ; v<8 ; v++ ){
        printf("%4.0f ",data_to_dump2[u][v]);
    }
    printf("      ");
    for ( v=0 ; v<8 ; v++ ) {
        printf("%4.0f ",data_to_dump3[u][v]);
    }
    printf("      ");
    printf("\n");
}
printf("\n\n");
}
/*===== Trace Mode-End =====*/

```

```

float      y[8][8],      u[8][8],      v[8][8];
float      dct_y[8][8],  dct_u[8][8],  dct_v[8][8];
float      quant_y[8][8], quant_u[8][8], quant_v[8][8];
float      iquant_y[8][8], iquant_u[8][8], iquant_v[8][8];
float      idct_y[8][8], idct_u[8][8], idct_v[8][8];
float      iy[8][8],      iu[8][8],      iv[8][8];
float      r_out[8][8],   g_out[8][8],   b_out[8][8];
float      o_rl_y[96],   o_rl_u[96],   o_rl_v[96];
int         cnt_y,        cnt_u,          cnt_v;

```

```

int main()
{

```

```

printf("\n\n\n\n=====
=====");
printf("\n\n\n**** Starting JPEG-Compression **** \n\n");

printf("- Processing: show_source-color...\n");
trace3(r_in,g_in,b_in);

rgb_to_yuv(r_in,g_in,b_in,y,u,v);
printf("- Processing: RGBtoYUV-color...\n");
trace3(y,u,v);

dct(y,dct_y); dct(u,dct_u); dct(v,dct_v);
printf("\n\n- Processing: dct...\n");
trace3(dct_y,dct_u,dct_v);

quantize(dct_y,quant_y,0); quantize(dct_u,quant_u,1); quantize(dct_v,quant_v,1);
printf("\n\n- Processing: quantize...\n");
trace3(quant_y,quant_u,quant_v);

for(int i=0;i<8;i++)for(int j=0;j<8;j++)
{
    iquant_y[i][j]=quant_y[i][j]; iquant_u[i][j]=quant_u[i][j]; iquant_v[i][j]=quant_v[i][j];
}

rll(quant_y,o_rll_y,&cnt_y); rll(quant_u,o_rll_u,&cnt_u); rll(quant_v,o_rll_v,&cnt_v);
trace_rll(o_rll_y,cnt_y); trace_rll(o_rll_u,cnt_u); trace_rll(o_rll_v,cnt_v);

dequantize(iquant_y,idct_y,0); dequantize(iquant_u,idct_u,1); dequantize(iquant_v,idct_v,1);
printf("\n\n- Processing: dequantize...\n");
trace3(idct_y,idct_u,idct_v);

inv_dct(idct_y,iy); inv_dct(idct_u,iu); inv_dct(idct_v,iv);
printf("\n\n- Processing: inv_dct...\n");
trace3(iy,iu,iv);

yuv_to_rgb(iy,iu,iv,r_out,g_out,b_out);
printf("\n\n- Processing: YUVtoRGB-color...\n");

```

```
    trace3(r_out,g_out,b_out);
    printf("****      ... finished!      ****\n\n");

}
/*===== Main-End =====*/
```