第二章　影像像素的熵編碼

在第一章動態影像壓縮概論中，我們提到了視訊壓縮的三大過程：預測、轉換與熵編碼。畫面方格內的像素經由轉換，產生 Transformed Coefficients（轉換後的影像像素）。Transformed Coefficients 的統計特性如下：低頻到高頻的大小呈現遞減，並且有許多 Transformed Coefficients 大小是零。利用上述統計特性，以特別設計的熵編碼方式將位元總數降到最低。

本章我們介紹並比較 H.263 與 H.264 的 Transformed Coefficients 的熵編碼〔12〕〔13〕。兩個視訊壓縮標準的轉換不同（H.263 採用 8x8 離散餘弦轉換，H.264 採用 4x4 核心轉換），因此 Transformed Coefficients 統計特性不同，熵編碼也不同。我們先介紹 H.263 的熵編碼，也就是 DC Fixed Length Coding，Run – Level Coding 與 Escape Coding；再介紹 H.264 的 CAVLC。接著我們介紹 H.263 與 H.264 的區塊層級語法及標頭欄位。我們並且模擬 H.263 與 H.264 熵編碼的壓縮表現，最後以兩者的比較以及定性的優劣分析做爲本章的結束。

模擬結果顯示 H.264 比 H.263 有更佳的壓縮品質。

2.1 H.263 影像像素熵編碼

H.263 區塊內像素經由 8x8 離散餘弦轉換，量化與掃描，每個方格產生由 64 個整數所構成的數列：數列的第一個整數是 DC 而後續的 63 個是 AC。在 I 畫面中，H.263 的 DC 使用 Fixed Length Coding，AC 則使用 Run – Level Coding（簡稱 R – L Coding）或 Escape Coding；在 P 畫面中，則使用 Run – Level Coding 或 Escape Coding。〔12〕

2.1.1 Fixed Length Coding，Run – Level Coding 與 Escape Coding

- **Fixed Length Coding**：

I 畫面中的 DC 使用使用 8 個位元 Fixed Length Code 描述 DC 值，範圍從-127 ~ +128。

- **Run – Level Coding**：

Run – Level Coding 紀錄 I 畫面中的 AC Coefficient 以及 P 畫面中的 DC 以及 AC Coefficient。Run – Level Coding 以紀錄 AC Coefficients 之間零的數目（**RUN**）與個別 AC Coefficient 大小（**LEVEL**）的方式進行編碼。每個 Non Zero Coefficient 與前一個 Non Zero Coefficient 之間零的數量定義爲 **RUN**；Coefficient 大小定義爲 **LEVEL**。**LAST** 是單一位元的旗標，顯示 Coefficient 是否數列中最後一個 Coefficient。根據 **RUN**，**LEVEL** 與 **LAST** 查表（請參考附錄表 A-1）得到 Codeword。以下面的數列爲例：（7,0,0,4,0,-3,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,……..,0）63 個 AC 值中，有四個 Non Zero Coefficient，7，4，-3 與 1。這四個 Coefficient 的 Run 與 Level 如表 2-1：

表 2-1 Run – Level Coding 範例

Coefficient	RUN	LEVEL	SIGN	LAST	Codeword	Bit Number
7	0	7	0	0	0001001000	**10**
4	2	4	0	0	0000010100010	**13**
-3	1	3	1	0	000111101	**9**
1	0	1	0	1	01110	**5**

- **Escape Coding**：

附錄表 A-1 內所能涵蓋的 Run 與 Level 皆有限。當 Non Zero Coefficient 的 Run 或 Level 不在表內，使用 Escape Coding。**ESCAPE CODE** 的使用方式是 **ESC**（7 bits，0000011）＋**LAST**（1 bit）＋**RUN**（6 bits）＋**LEVEL**（8 bits），共 22 個位元。**LEVEL** 有八個位元，所以 **Escape Coding** 可描述的範圍從 -255 到 +256。

2.1.2 模擬

以下的模擬，我們觀察 DC Code，Run – Level Code，與 Escape Code 所佔用的位元數。模擬的編碼／解碼器採用 TMN8〔17〕。以 Coastguard 的第一張畫面（QCIF）進行壓縮編碼，QP 值設爲 14，觀察三個編碼方式所佔用的位元比例。

圖 2-1 Coastguard 第一張畫面

表 2-2 顯示畫面中 99 個區塊的 Coefficients 經由 H.263 編碼後的個別位元數目（不包含方格標頭位元）。表 2-2 橫向編號 1 至 11，與縱向編號 1 至 9 分別表示區塊在畫面裡的橫向縱向幾何位置。每個區塊都有四個 8x8 明度（Luminance）方格；每個方格經由轉換與 Zigzag 掃描產生包含 64 個整數的數列。數列的第一個整數是 DC 值，使用八個位元 Fixed Length Coding。因此 DC 的編碼佔用每個區塊 32 個位元。數列中其餘的 63 個 AC 值使用 Run – Level Code 或 Escape Code 編碼。

表 2-2 各區塊位元總數

	1	2	3	4	5	6	7	8	9	10	11
1	165	137	145	222	184	56	68	79	95	88	120
2	180	119	194	205	149	207	145	152	118	189	177
3	162	119	146	182	150	202	174	118	126	155	150
4	171	73	86	99	117	103	64	107	108	65	172
5	135	72	82	275	314	302	247	201	291	238	325
6	60	86	70	85	92	88	79	59	47	45	151
7	72	113	73	64	47	57	60	59	37	37	163
8	107	73	115	84	99	80	96	115	77	56	173
9	100	66	66	82	94	106	101	67	57	45	173

表 2-3 顯示 DC Fixed Length Code 在每個區塊裡佔用的位元數百分比。

表 2-3 各區塊內 DC Code 佔用的百分比

	1	2	3	4	5	6	7	8	9	10	11
1	19.3	23.3	22	14.4	17.4	57	47	40	33.6	36.3	26.6
2	17.7	26	16	15.6	21.4	15	22	21	27	16.9	18
3	19.7	26.8	22	17	21.3	15.8	18	27	25	20.6	21.3
4	18.7	43.8	37	32	27.3	31	50	29.9	29.6	49	18.6
5	23.7	44	39	11.6	10.2	10.6	12.9	15.9	10.9	13.4	9
6	53	37	45.7	37.6	34.7	36	40.5	54.2	68	71	21.1
7	44	28.3	43	50	68	56	53	54	86	86	19
8	29	43.8	27.8	38	32.3	40	33	27.8	41.5	57	18.4
9	32	48	48	39	34	30	31.6	47.7	56	71	18.4

表 2-4 顯示每個區塊使用 Escape Code 的位元數。

表 2-4 各區塊內 Escape Code 數量

	1	2	3	4	5	6	7	8	9	10	11
1	44	0	0	44	0	0	0	0	22	0	0
2	0	0	0	0	0	0	0	0	0	0	22
3	22	0	22	44	22	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	44
5	22	0	0	0	88	0	0	44	66	66	132
6	0	0	0	0	0	0	0	0	0	0	44
7	0	22	0	0	0	0	0	0	0	0	66
8	0	0	0	22	0	22	0	22	0	0	88
9	0	0	0	0	0	0	0	0	0	0	88

表 2-5 顯示 Escape Code 在每個區塊的百分比。

表 2-5 各區塊內 Escape Code 百分比

	1	2	3	4	5	6	7	8	9	10	11
1	26	0	0	19.8	0	0	0	0	23	0	0
2	0	0	0	0	0	0	0	0	0	0	12.4
3	13.5	0	15	24	14.6	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	25.5
5	16.3	0	0	0	28	0	0	21.9	27	27.7	40.6
6	0	0	0	0	0	0	0	0	0	0	29.1
7	0	19.4	0	0	0	0	0	0	0	0	40.5
8	0	0	0	26.2	0	27.5	0	19.1	0	0	50.8
9	0	0	0	0	0	0	0	0	0	0	50.8

圖 2-2 顯示單一畫面中（Coastguard，Frame No.1）DC， Run – Level Code，Escape Code 所佔用的位元數百分比。

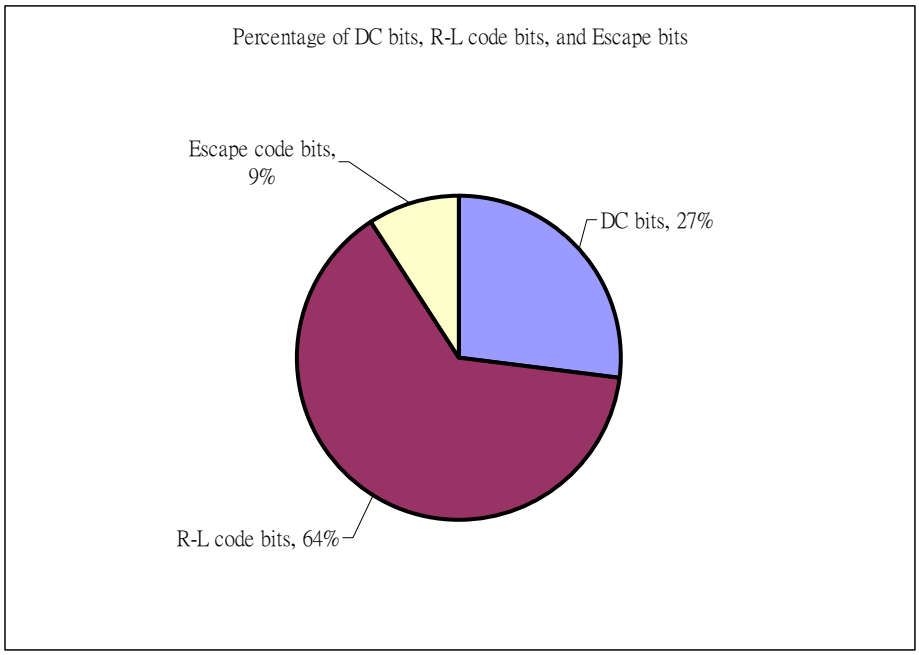


圖 2-2 DC Code, R-L Code, Escape Code 百分比

模擬結果顯示，最重要的 DC Code 占了總位元數的四分之一。Run – Level Code 是 AC 的主要編碼方式，占了六成。

根據表 2-4，總共有 23 個區塊用到 Escape Code，其中六個區塊用了兩次，三個區塊用了三次，三個區塊用了四次，一個區塊用了六次。每用一次 Escape Code 就要花費 22 個位元，描述超越 Run – Level Code 範圍的大小。由圖 2-2，Escape Code 占了整張畫面總位元數的十分之一。

2.2 H.264 影像像素熵編碼

2.2.1 CAVLC 彈性語法

H.264 區塊裡的 4x4 方格像素，經由 4x4 核心轉換以及量化，再以 Zigzag 的順序將資料掃描成爲由低頻至高頻排列的數列。這個數列有以下三項特性：〔6〕

- **分布稀疏**
  16 個整數只有少數非零整數（Non Zero Coefficients），而且分布稀疏（Non Zero Coefficients 之間多數爲零）。

- **越往高頻非零整數的值越小，而且多為正負一**
  核心轉換後能量集中在低頻。16 個整數所排成的數列中，越往高頻值越小，而且高頻值多爲正一或負一。

- **方格之間非零整數的數量有極高的相關性**
  核心轉換後，相鄰方格內非零整數的數量（定義爲 total_coeff）有極高的相關性。

H.264 利用以上三項特性，設計了 CAVLC（Context – based Adaptive VLC）〔13〕。”Context – based Adaptive”的編碼方式，特定標記（Symbol）所對應的二進位代號（Codeword）隨著前後文而改變。也就是說，相同的方格內容，隨著鄰近方格的改變會對應到不同的二進位代號；相同的 Coefficient，位在數列的不同位置上也會對應到不同的二進位代號。”**Context – based Adaptive**”的目的，在於二進位代號可隨著前後標記之間的相關性而調整；CAVLC 的編碼表也根據前後標記之間的相關性而特別設計，務必使得編碼後整體的位元總數降到最少。

2.2.2 CAVLC 裡的三個區段

經由 CAVLC 編碼，16 個整數資料的編碼結果分成三個區段；區段 A，Non Zero Coefficient 的數量（Number Coding）。區段 B，Non Zero Coefficient 的大小（Level Coding）。區段 C，Non Zero Coefficient 的分布（Distribution Coding）。解碼器根據 CAVLC 在區段 A，區段 B 與區段 C 的編碼結果得知 16 個整數的大小與分布，並得以還原成原來的數列（見圖 2-3）。

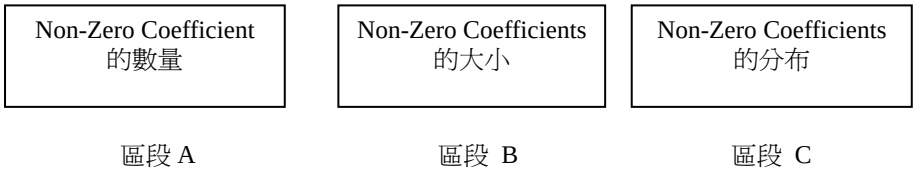


圖 2-3 CAVLC 的三個區段

2.2.2.1 區段 A，Number Coding

區段 A 描述 16 個整數資料中，Non Zero Coefficient 與”T1s”的數量。”T1s”指的是 16 個整數資料中，高頻部分正負一的總數。以下列的數列爲例，低頻至高頻的排序如下：”6，8，5，4，-3，0，2，1，1，3，1，-1，0，0，0，0”。此數列的高頻有兩個一（2 與 3 之間的 1 不算在內），所以 T1s 的值爲 2。至於 total_coeff，由於數列中有 9 個 Non Zero Coefficient，所以 total_coeff 等於 9。根據 total_coeff，T1s 以及鄰近方格 Non Zero Coefficient 數量（nC，左側與上側方格 Non Zero Coefficient 的平均）查表 A-2 得到區段 A 的二進位代號。

2.2.2.2 區段 B，Level Coding

區段 B 描述 Non Zero Coefficient 的值，因此稱爲 Level Coding。由高頻至低頻依序爲每個 Non Zero Coefficients 進行編碼。

CAVLC Level Coding 的做法，是將 Non Zero Coefficient 進行除法運算；完成運算後，商數以及餘數分別查表得到所對應的二進位代號，並且將之輸入至位元流中。Level Coder 爲每一個 Non Zero Coefficient 指定一個特定的除數作爲除法運算的依據。解碼器的做法，則是從位元流中讀取二進位代號，分別查表還原商數以及餘數的大小；接著依據判斷得到除數的大小，將除數乘以商數加上餘數得到原來的 Coefficient。也就是說，CAVLC Level Coding 靠著商數以及餘數顯示 Non Zero Coefficient 的值。至於除數並沒有顯示在位元流中，我們會在下面的編碼過程中描述解碼器如何從前後的位元資料判斷除數的大小。

Level Coding 的編碼過程，需要計算四個變數：**suffixLength**、**levelPrefix**、**levelSuffix**、與 **levelCode**。CAVLC 定義四者的關係如下：

$$\textbf{levelCode} = (\ \textbf{levelPrefix} \ll \textbf{suffixLength}\ ) + \textbf{levelSuffix} \qquad (2.1)$$

levelCode 是一個正整數，他的大小代表著 Non Zero Coeffcient 值，兩者的轉換關係見下列編碼步驟二。從**(2.1)**我們得知 **levelPrefix** 左移 **suffixLength**，然後加上 **levelSuffix** 等於 **levelCode**；而根據 CAVLC 定義，**levelSuffix** 的長度等於 **suffixLength** 的值。因此我們可以將**(2.1)**轉換成除法的關係，**levelCode** 是被除數，（1 << **suffixLength**）是除數，**levelPrefix** 是商數，而 **levelSuffix** 是餘數。如**(2.2)**與**(2.3)**所示。

$$\textbf{levelPrefix} = \textbf{levelCode} \,/\, （1 \ll \textbf{suffixLength}）。 \qquad (2.2)$$

$$\textbf{levelSuffix} = \textbf{levelCode} \,\%\, （1 \ll \textbf{suffixLength}）。 \qquad (2.3)$$

根據**(2.2)**與**(2.3)**，編碼器從 **levelCode** 與 **suffixLength** 計算得到 **levelPrefix** 與 **levelSuffix**。Level Coder 爲每一個 Non Zero Coefficient 設定特定的 **suffixLength**，以達到改變每一個 Non Zero Coefficient 的除數的目的。**suffixLength** 的初始值以

及變更方式見下面的編碼步驟一與步驟七。

編碼的步驟如下：

步驟一：初始值設定。
suffixLength 與 **i** 的初始值設爲 **0**，**level [i]**爲 Non Zero Coefficient 值。

步驟二：將有號（signed）的 **level [i]**轉換成無號（unsigned）的 **levelCode**。
如果 **level [i]**是正的，**levelCode =（level [i] << 1） - 2**;
如果 **level [i]**是負的，**levelCode = -（level [i] << 1） - 1**

步驟三：計算 **levelPrefix**。
levelPrefix = levelCode / （1 << **suffixLength**）。

步驟四：計算 **levelSuffix**。
levelSuffix = levelCode %（1 << **suffixLength**）。

步驟五：**levelPrefix** 查表 A-3 得到二進位代碼，然後輸入至位元流。

步驟六：**levelSuffix** 的大小以 **suffixLength** 的位元長度表示。以 **suffixLength** 長度的位元數目將 **levelSuffix** 的大小輸入至位元流。

步驟七：如果 **suffixLength** 等於 0，**suffixLength** 加 1；如果 **levelCode** 大於 3<<(**suffixLength**-1)並且 **suffixLength** 小於 6，**suffixLength** 再加 1。

步驟八：如果 **i** 小於（total_coeff – T1s），**i** 加 1；回到步驟二。

解說：

i 代表著 Non Zero Coefficient 的編號，**i** 從 0 到（total_coeff – T1s - 1），編號的順序從高頻至低頻。**level [i]**爲 Non Zero Coefficient 值，步驟一至步驟八顯示 Level Coder 爲每一個 **level [i]**編碼的執行流程。步驟一是初始值設定，**suffixLength** 設爲 0，表示 **level [0]**的編碼除數一定是 1。由於 **level [i]**有正有負，步驟二將有號的 **level [i]**轉換成無號的 **levelCode**。根據步驟二的方式轉換，如果 **level [i]**是正，轉換後 **levelCode** 是偶數；如果是 **level [i]**負，轉換後 **levelCode** 是奇數。步驟三與步驟四根據**(2.2)**與**(2.3)**計算 **levelPrefix** 與 **levelSuffix**。步驟五與步驟六將 **levelPrefix** 與 **levelSuffix** 的二進位代碼輸入至位元流。步驟七則重新設定 **suffixLength** 的值，也重新設定下一個 Non Zero Coefficient 的除數。**suffixLength** 是否加一的判斷準則來自 **levelCode** 的大小；如果 **levelCode** 大於 3<<(**suffixLength**-1)，**suffixLength** 加 1；反之 **suffixLength** 維持其原值。

從步驟七 **suffixLength** 的重新設定，**levelCode** 只要大於門檻值 3<<(**suffixLength**-1)，**suffixLength** 就會加 1，Level Coder 以加大一倍的除數替下一個 Non Zero Coefficient 進行編碼。下一個 Non Zero Coefficient 的除數增大一倍意味著 Level Coder 對下一個 Non Zero Coefficient 的大小範圍描述能力增大一倍。因此 **suffixLength** 在這裡好比位階的觀念，不同的位階有不同的動態範圍；Level Coder 視目前的 **levelCode** 大小判斷是否**晉級**到下一個位階，是否以增

大一倍的除數描述下一個 Non Zero Coefficient。如此的語法設計合乎 Non Zero Coefficient 從高頻至低頻遞增的自然特性。根據步驟七對門檻值大小的定義以及 **(2.1)**，各 **suffixLength** 所對應的動態範圍與門檻值見表 2-6。

表 2-6 **suffixLength**、動態範圍、與門檻值

suffixLength	動態範圍	門檻值
0	-15 ~ -1, +1 ~ +15	3
1	-15 ~ -1, +1 ~ +15	6
2	-30 ~ -1, +1 ~ +30	12
3	-60 ~ -1, +1 ~ +60	24
4	-120 ~ -1, +1 ~ +120	48
5	-240 ~ -1, +1 ~ +240	NA

大 levelCode 的編碼方式

當 **levelCode** 大小超過表 2-6 的動態範圍，CAVLC 以特別的方式描述 **levelCode**。Level Coder 同樣根據**(2.1)**的關係計算 **levelSuffix**；特別的是，**levelPrefix** 此時定爲 15（二進位代碼是 0000000000000001），而 **levelSuffix** 的長度設爲 12。**levelSuffix** 的長度爲 12，所以 **levelSuffix** 的大小從（1 << 0 - 1）到（1 << 12 - 1）。以 **suffixLength** 等於 0 爲例，根據**(2.1)**的關係，**levelCode** 的最小值是 15 << 0 +（1 << 0 - 1），最大值是 15 << 0 +（1 << 12 - 1）。**levelCode** 的範圍從 15 ~ (15 + 4095)。我們將 **suffixLength** 分別爲 0,1,2,3,4,5 時的 **levelCode** 範圍列在表 2-7。

表 2-7 levelCode 範圍（當 levelPrefix 等於 15）

suffixLength	**levelPrefix**	**levelCode 範圍**
0	**15**	**15 ~ (15 + 4095)**
1	**15**	**30 ~ (30 + 4095)**
2	**15**	**60 ~ (60 + 4095)**
3	**15**	**120 ~ (120 + 4095)**
4	**15**	**240 ~ (240 + 4095)**
5	**15**	**480 ~ (480 + 4095)**

這種做法很像是 H.263 的 Escape Coding，**levelPrefix** 設爲 15 如同 Escape Coding 中 **ESC** 設爲 0000011。CAVLC 解碼器讀到 **levelPrefix** 二進位代碼是 0000000000000001 便啓動特殊解碼。

2.2.2.3 區段 C，Distribution Coding

CAVLC 已經從區段 A 與區段 B 告知 Non Zero Coefficients 的數量與個別大小。在區段 C 中，CAVLC 以 16 個整數中 0 的位置告知解碼器 Non Zero Coefficients 的分布情形。

區段 C 的編碼，定義了三個變數：**total_zero**、**run_before** 與 **zero_left**。**total_zero** 是 Coefficient 內 0 的總數，編碼表見附錄表 A-4，A-5。**run_before** 指的是 Non Zero Coefficient 與鄰近一個 Non Zero Coefficient 之間 0 的數量；**zero_left** 指的是 Non Zero Coefficient 與 **DC** 之間 0 的總數。**run_before** 與 **zero_left** 所對應的代號見表附錄表 A-6。區段 C 中記錄了 **total_zero**；同時記錄了每一個 Non Zero Coefficient 的 **run_before** 與 **zero_left**。從區段 C 的內容，解碼器得以知道 16 個整數中 0 的總數與個別位置；從而還原 16 個整數所構成的數列的原貌。

以下列的數列爲例，低頻至高頻的排序如下：”6，8，5，0，-3，0，2，0，0，0，0，0，0，0，0，0”。在此數列中，**total_zero = 2** （**DC** 值 6 到高頻 2 共有兩個 0）。2 的 **run_before = 1**，**zero_left = 2**（2 與前一個 coefficient -3 之間有一個 0，所以 **run_before = 1**；DC 值 6 到 2 共有兩個 0，所以 **zero_left = 2**）。-3 的 **run_before = 1**，**zero_left = 1**。（-3 與前一個 coefficient 5 之間有一個 0，所以 **run_before = 1**；**DC** 值 6 到-3 有一個 0，所以 **zero_left = 2**）。將 **total_zero = 2**、**run_before = 1**，**zero_left = 2**、**run_before = 1**，**zero_left = 1** 依序查表 A-4、A-6，得到區段 C 的二進位代碼。

以下是區段 C 的解碼：
解碼器從 **total_zero = 2** 的二進位代碼知道 **DC** 值 6 到高頻 2 之間共有兩個 0。從 **run_before = 1**，**zero_left = 2** 的二進位代碼知道 coefficient 2 與前一個 coefficient -3 之間有一個 0；而 coefficient 2 與 DC 之間有兩個 0。從 **run_before = 1**，**zero_left = 1** 的二進位代碼知道 coefficient -3 與前一個 coefficient 5 之間有一個 0；而 coefficient -3 與 DC 之間有一個 0。**total_zero = 2** 的兩個 0，一個在 coefficient 2 與 coefficient -3 之間，另一個在 coefficient -3 與 coefficient 5 之間；表示 coefficient 5 與 **DC** 值 6 之間沒有 0。解碼器掌握了所有 0 的位置，完成了區段 C 的解碼。

2.2.3 H.264 區塊層級語法

圖 2-4 顯示 H.264 裡，I 畫面中的區塊層級語法〔14〕〔15〕。至於 P 畫面中的區塊層級語法請參考 4.3.2 節。如圖 2-4 所示，H.264 區塊層級（Macroblock Layer）語法中，有 **MB_ TYPE**、**prev_intra4x4_pred_mode_flag**、**rem_intra4x4_pred_mode**、**mb_qp_delta**、**coded_block_pattern** 等欄位。H.264 區塊層級編碼方式，是根據所屬欄位的屬性、語意找到其索引號碼（Index Number）；然後根據索引號碼計算 Exp-Golomb Code，得到二進位代號（Codeword），完成編碼〔16〕。以下我們分別介紹 H.264 區塊層級裡各欄位的索引號碼語意以及 Exp-Golomb Code 與欄位索引號碼之間的對應關係。

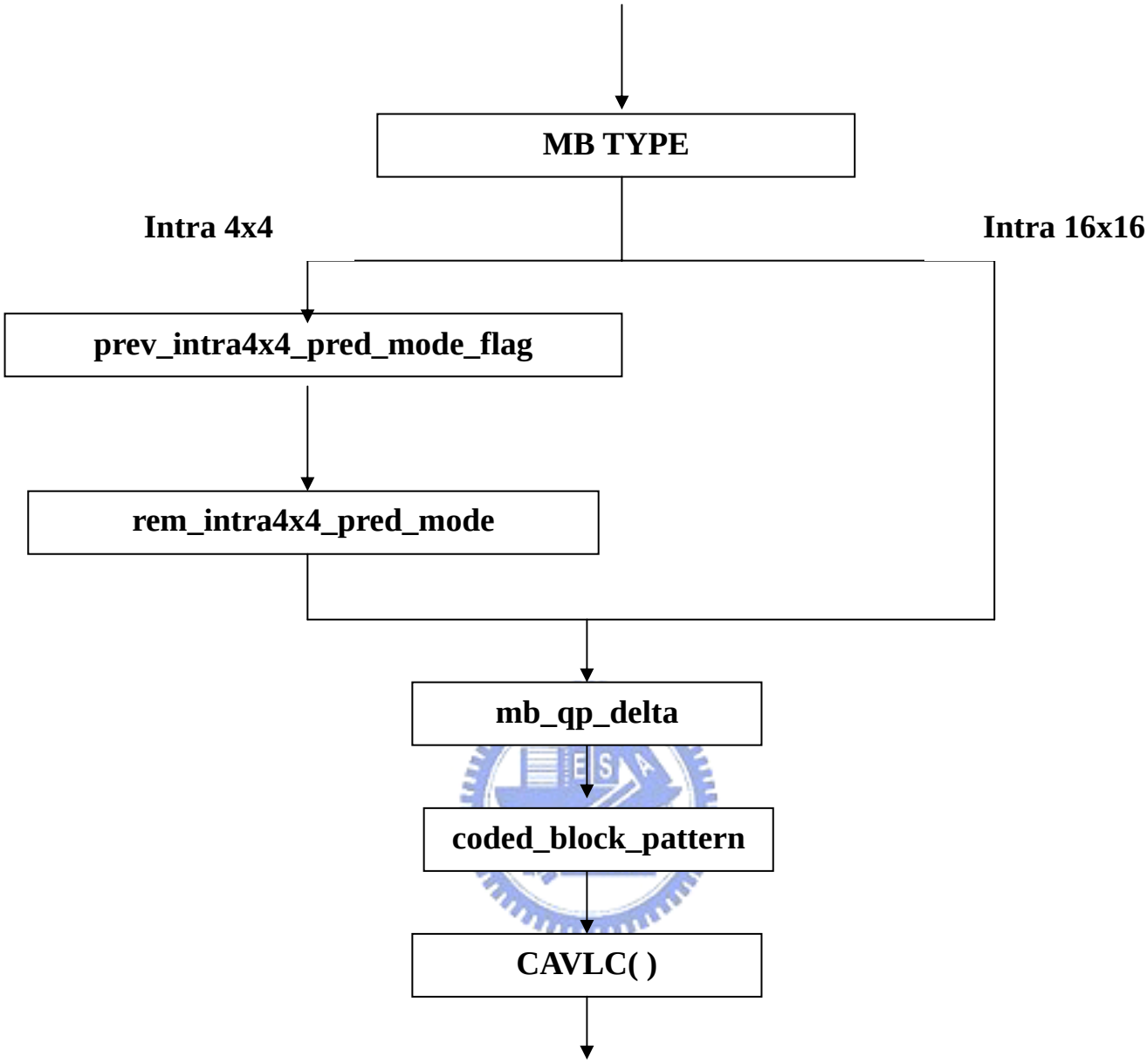


圖 2-4 H.264 區塊層級語法

2.2.3.1 區塊層級欄位的語意

MB_ TYPE：顯示區塊型態、區塊預測方式以及明度色度方格的型態。區塊所在畫面的不同（I 畫面或 P 畫面），以及區塊使用預測方式的不同（Intra 預測或 Inter 預測），都影響到了 **MB_ TYPE** 的編號索引。

I 畫面裡的 Intra 預測區塊：**MB_ TYPE** 的索引號碼從 0 至 24。當區塊採用 Intra4x4 預測，索引號碼是 0。當區塊採用 Intra16x16 預測，索引號碼是 1 至 24。Intra4x4 與 Intra16x16 的作法與差別請參考本文 3.2 節。當區塊採用 Intra16x16 預測，索引號碼同時顯示了明度與色度方格的型態（**Coded Block Pattern Luma** and **Coded Block Pattern Chroma**）。H.264

將區塊切割成四個 8x8 明度方格以及兩個 8x8 色度方格，並且以分別以 **Coded Block Pattern Luma** 以及 **Coded Block Pattern Chroma** 描述明度與色度方格的型態。**Coded Block Pattern Luma** 有兩個值，0 或 1。**Coded Block Pattern Luma** 是 0，表示核心轉換過後，四個明度方格裡都沒有 AC；**Coded Block Pattern Luma** 是 1，表示明度方格裡有 AC。**Coded Block Pattern Chroma** 有三個值，0、1、或 2。**Coded Block Pattern Chroma** 是 0，表示色度方格裡沒有任何值；**Coded Block Pattern Chroma** 是 1，表示兩個 8x8 色度方格裡有 DC 無 AC； **Coded Block Pattern Chroma** 是 2 表示兩個 8x8 色度方格裡有 AC。Intra16x16 的四種模式、**Coded Block Pattern Luma** 的兩個值、**Coded Block Pattern Chroma** 的三個值，總共 24 種型別構成了索引號碼 1 至 24。Intra16x16 模式、**Coded Block Pattern Luma**、**Coded Block Pattern Chroma** 與 **MB_ TYPE** 的索引號碼之間的關係請參考表 3-2。

P 畫面裡的 Inter 預測區塊：**MB_ TYPE** 的索引號碼從 0 至 4。索引號碼 0 至 3 分別表示區塊以 16x16、16x8、8x16、8x8 大小的明度方格進行 Inter 預測。索引號碼 4 表示區塊被切割四個 8x8 明度方格，每個明度方格有可能再被切成更小的方格進行 Inter 預測。有關 P 畫面裡的 Inter 預測區塊的詳細語法請見本文 4.3.2 節。

P 畫面裡的 Intra 預測區塊：**MB_ TYPE** 的索引號碼從 5 至 30。P 畫面裡 Intra 預測區塊與 I 畫面裡的 Intra 預測區塊，**MB_ TYPE** 的索引號碼語意相同，只是號碼增加 5。有關 P 畫面裡的 Intra 預測區塊的詳細語法請見本文 4.3.2 節。

prev_intra4x4_pred_mode_flag 與 **rem_intra4x4_pred_mode**：無論 I 畫面或 P 畫面，都使用這兩個欄位顯示區塊內明度方格的 Intra4x4 模式。當區塊內明度方格使用 Intra4x4 進行預測，將會產生 16 個 Intra4x4 模式。H.264 採用 DPCM 方式爲這 16 個 Intra4x4 模式進行編碼。Intra4x4 的 DPCM 編碼方式請見 3.1.3 節。

mb_qp_delta：顯示目前區塊與前一個區塊的量化參數差。量化參數差有正，有負，透過函數 se( )找到對應的 codeNum，再以此對應 Exp – Golomb Codeword。

coded_block_pattern：如果目前區塊不是 Intra16x16 預測，區塊以此欄位顯示明度色度方格的型態。**coded_block_pattern** 的值從 0 至 47，**coded_block_pattern** 除以 16 的餘數得到 **Coded Block Pattern Luma**，**coded_block_pattern** 除以 16 的商數得到 **Coded Block Pattern Chroma**。

Coded Block Pattern Luma = **coded_block_pattern** % 16 **(2.4)**

Coded Block Pattern Chroma = **coded_block_pattern** / 16 **(2.5)**

由**(2.4)**可知 **Coded Block Pattern Luma** 的大小由 0 至 15。0 至 15，代表了區塊裡四個明度方格是否有 AC 值。由**(2.5)**可知 **Coded Block Pattern**

Chroma 的大小由 0 至 2。**Coded Block Pattern Chroma** 的值，代表了區塊裡兩個色度方格型態。**Coded Block Pattern Chroma** 等於 0，表示兩個色度方格都沒有值；**Coded Block Pattern Chroma** 等於 1，表示色度方格裡有 DC 沒有 AC；**Coded Block Pattern Chroma** 等於 2，表示色度方格裡有 AC。

2.2.3.2 Exp – Golomb Code

H.264 的壓縮編碼器根據區塊的型態、預測方式等特徵查表找尋各欄位對應的索引號碼（Index Number）。索引號碼請參考 2.2.3.1 的介紹。索引號碼再找出對應的 codeNum。codeNum 再查詢 Exp – Golomb Code 的表格（表 2-8），得到二進位代碼（Codeword），完成區塊層級標頭的熵編碼。因此編碼的過程是索引號碼 -> codeNum -> Exp – Golomb Codeword。〔16〕

表 2-8 Exp – Golomb Codeword 與 codeNum 的關係

Exp – Golomb Codeword	codeNum
1	0
010	1
011	2
00100	3
00101	4
00110	5
00111	6
0001000	7
0001001	8
0001010	9
……	…….

依照語法的定義，索引號碼對應 codeNum 有三類關係：ue( )、se( )、me( )。

ue( )：Unsigned Exp – Golomb Code，在這層關係下 codeNum 等於索引號碼。大部分欄位索引號碼與 codeNum 之間的對應屬於此類關係，**MB TYPE** 屬於此類。

se( )：Signed Exp – Golomb Code，這層關係將有號的索引號碼對應到無號的 codeNum。**mb_qp_delta**、移動向量屬於此類關係。se( )中，codeNum 與有號索引的對應關係見表 2-9。

表 2-9 codeNum 與有號索引的關係（se( )）

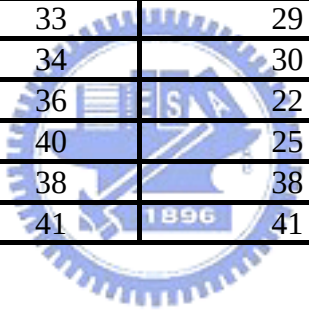
codeNum	有號的索引
0	0
1	1
2	-1
3	2
4	-2
5	3
6	-3
7	4
8	-4
……	……

me( )：Mapped Exp – Golomb Code，這個關係是特別爲 Intra4x4 或 Inter 區塊所定義的。當區塊是 Intra4x4 或 Inter 區塊，欄位 **coded_block_pattern** 的索引號碼所對應到的 codeNum，根據 me( )的定義，參考表 2-10。

表 2-10 codeNum 與 coded_block_pattern 索引號碼的關係（me( )）

codeNum	**coded_block_pattern 索引號碼**	
	Intra4x4	Inter
0	47	0
1	31	16
2	15	1
3	0	2
4	23	4
5	27	8
6	29	32
7	30	3
8	7	5
9	11	10
10	13	12
11	14	15
12	39	47
13	43	7
14	45	11
15	46	13
16	16	14
17	3	6
18	5	9
19	10	31
20	12	35
21	19	37

22	21	42
23	26	44
24	28	33
25	35	34
26	37	36
27	42	40
28	44	39
29	1	43
30	2	45
31	4	46
32	8	17
33	17	18
34	18	20
35	20	24
36	24	19
37	6	21
38	9	26
39	22	28
40	25	23
41	32	27
42	33	29
43	34	30
44	36	22
45	40	25
46	38	38
47	41	41



2.2.4 模擬

本節的模擬，我們觀察相同畫質表現下（也就是相同的訊噪比），H.264 比起 H.263 的壓縮增益。在 2.1.2 節中，我們以第一張 Coastguard 模擬 H.263 的編碼方式，量化參數 14 下得到了 30.49 的訊噪比。現在我們同樣使用 Coastguard 的第一張作爲模擬依據，模擬測試 H.264 的壓縮結果。爲了使 H.264 的壓縮結果有近似於 H.263 的訊噪比，在模擬進行時，我們嘗試爲 H.264 輸入不同的量化參數；最後我們選擇量化參數爲 35，產生的訊噪比爲 30.6809；以下表格內的數據便是根據此量化參數所得到的模擬結果。模擬的編碼／解碼器採用 JM7.1〔18〕。

表 2-11 H.264 編碼後區塊內的位元數

	1	2	3	4	5	6	7	8	9	10	11
1	154	124	146	168	148	62	68	103	53	105	75
2	165	171	206	198	163	185	157	160	132	224	181
3	161	100	118	172	168	202	164	140	110	156	152
4	185	66	77	103	122	92	76	110	128	70	83
5	133	54	82	221	248	282	262	207	204	171	150
6	48	48	60	83	73	64	71	71	38	10	52
7	67	106	69	73	46	51	83	42	30	5	8
8	92	84	114	89	97	91	91	119	82	43	5
9	103	81	97	85	115	98	98	102	37	8	38

表 2-11 顯示第一張 Coastguard 畫面裡的 99 個區塊（水平 11 個，垂直 9 個）經由 H.264 壓縮的位元數目。表格內的數字代表了壓縮後每一個區塊內的位元數目，包含了區塊層級標頭位元，以及 CAVLC 編碼產生的位元。

在 2.1.2 節中，我們將 H.263 的編碼結果記錄在表 2-2。將表 2-2 裡的數字減掉表 2-11 裡的數字，也就是 H.263 與 H.264 編碼後區塊內的位元數差，得到表 2-12。表 2-12 裡的數字大都是正數，也就是說，相同的畫質下，H.263 的位元數比 H.264 多，而壓縮表現比較差。

表 2-12 H.263 與 H.264 區塊內的位元數差

	1	2	3	4	5	6	7	8	9	10	11
1	9	13	-1	54	36	-6	0	-4	42	-17	45
2	15	9	-12	7	-24	22	-12	-8	-14	-35	-4
3	1	19	28	10	-18	0	10	-22	16	-1	-2
4	-14	7	9	-4	28	11	-12	-3	20	-5	89
5	2	18	0	54	17	20	-15	-6	87	67	175
6	12	38	10	2	19	22	8	-12	9	35	99
7	5	7	4	-9	1	6	-24	17	7	32	155
8	15	-11	1	-5	2	-11	5	-4	-5	13	168
9	-3	-15	-11	-3	-21	8	3	-35	20	37	125

2.3 H.263 與 H.264 像素熵編碼模擬

我們以測試序列 Akiyo，Coastguard，Container 以及 Foreman 模擬測試 H.263 與 H.264 的編碼方式。這個模擬純粹比較兩者 Transformed Coefficients 的熵編碼，沒有使用 Intra 或 Inter 預測。每個測試序列取一百張連續的 4：2：0 QCIF 畫面，也就是每張 QCIF 畫面有 176x144 個明度像素，2x88x72 個彩度像素。每個像素佔用一個位元組（八個位元）的記憶體空間，則一百張 4：2：0 QCIF 畫面的總共位元數：

$$8 \times 176 \times 144 \times 100 + 8 \times 2 \times 88 \times 72 \times 100 = 30412800$$

模擬時的編碼／解碼器（Source Codec）：H.263 用 TMN8〔17〕，H.264 用 JM7.1〔18〕。H.263 的 DC 使用 8 bits Fixed Legth Coding，AC 使用 R – L Coding 與 Escape Coding。H.264 使用 CAVLC。將模擬結果繪製在下列四張圖，橫軸是編碼位元總數，縱軸是平均訊噪比。

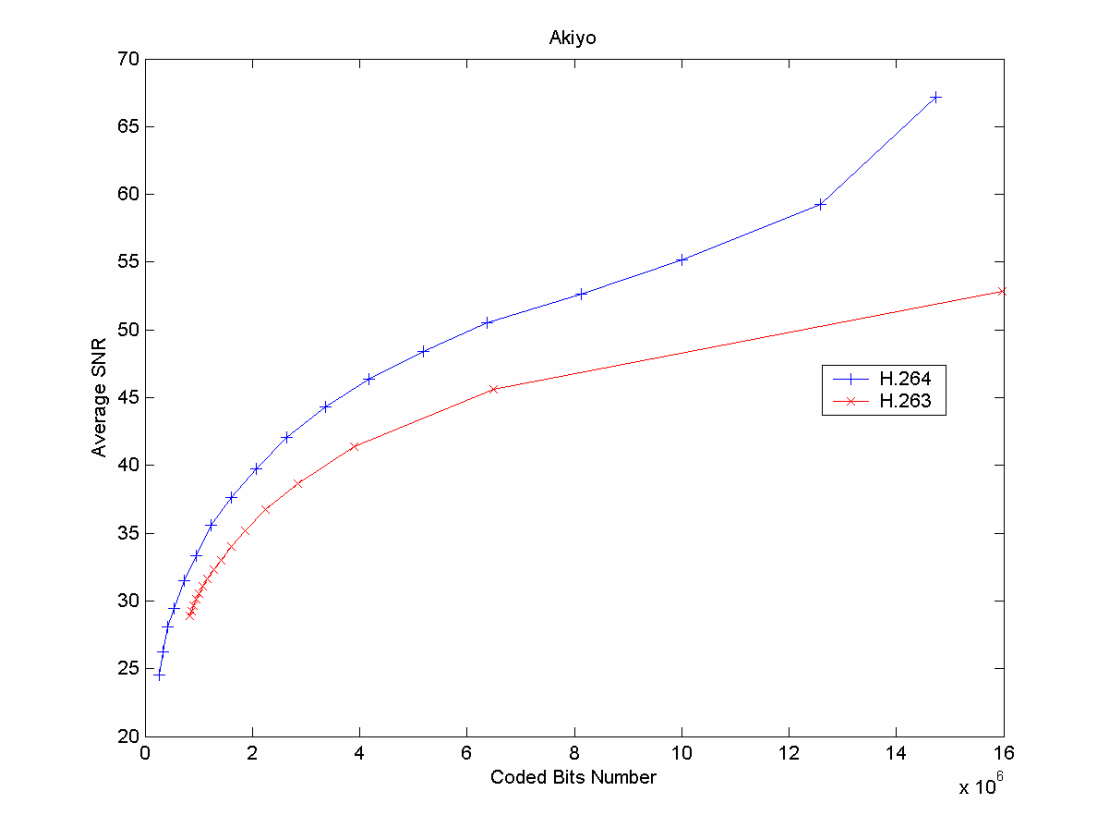


圖 2-5 Akiyo；H.263 與 H.264 編碼結果

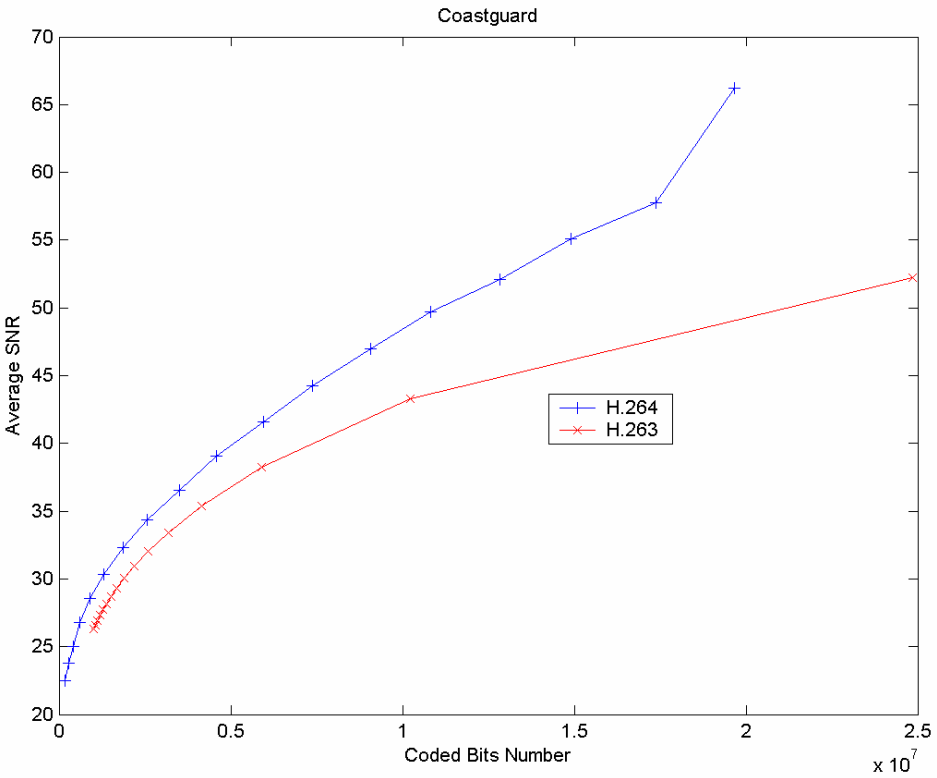


圖 2-6 Coastguard；H.263 與 H.264 編碼結果

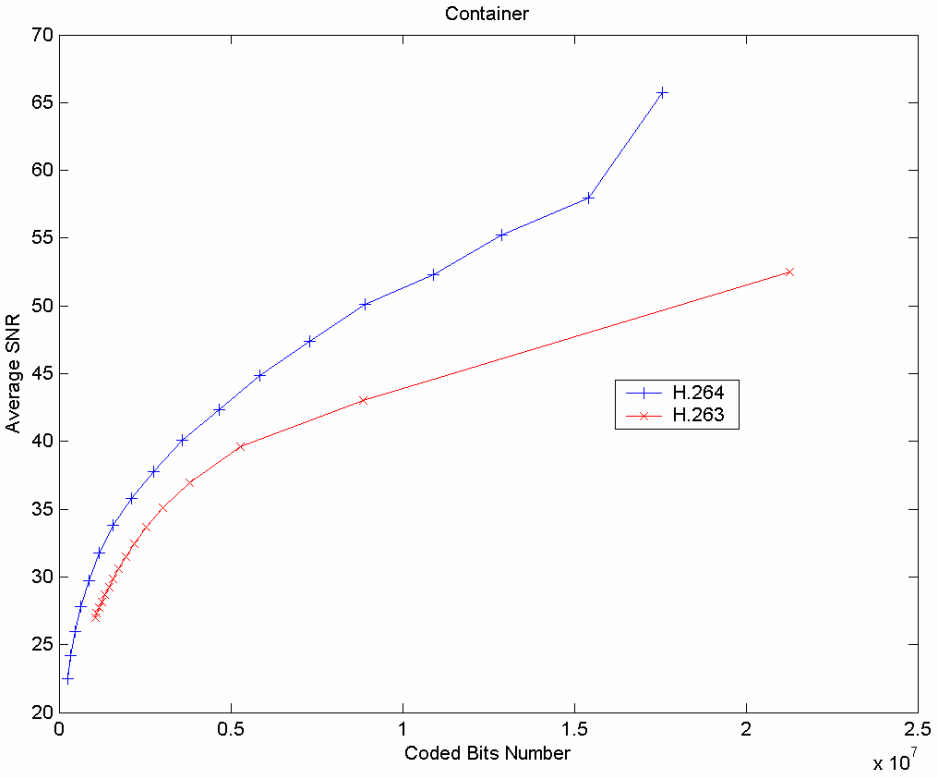


圖 2-7 Container；H.263 與 H.264 編碼結果

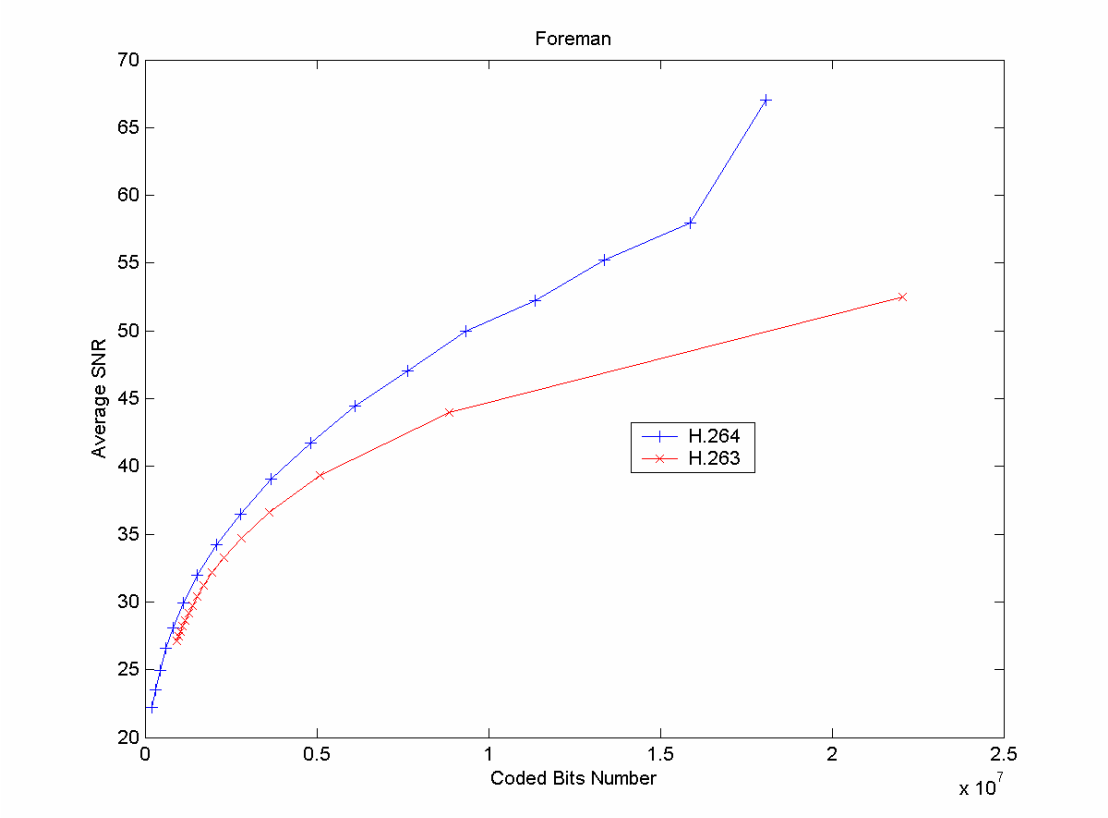


圖 2-8 Foreman；H.263 與 H.264 編碼結果

模擬結果分析：

四個測試序列的結果皆顯示 H.264 在相同訊噪比下產生較少的位元數，壓縮表現較佳。高訊噪比下的差別特別大。

編碼產生的結果包含了畫面標頭位元（Frame Header），方格標頭位元（MB Header）以及 Transformed Coefficients 位元。三個位元總數共同影響了最後的壓縮表現。在沒有分析比較過兩者的畫面標頭與方格標頭的位元數量前，我們還不能片面作出 H.264 的 Transformed Coefficients 位元數少於 H.263 的結論。

2.4 H.263 與 H.264 像素熵編碼比較

本節中，我們比較 CAVLC 與 H.263 Run – Level Coding，並且定性敘述兩者的優劣。

兩者的比較：

- **CAVLC 採用彈性語法（Context - Adaptive）**

所謂彈性語法（Context - Adaptive），指的是標記所對應的二進位代號會隨著前後標記的不同而改變。相對於彈性語法，H.263 與 MPEG4 採用固定語法（Fixed Context）。也就是 H.263 的二進位代號與前後標記內容互相獨立。

CAVLC 採用彈性語法。以 CAVLC 區段 A 的 total_coeff 爲例，total_coeff 的二進位代號是 nC 的函數（見附錄 Table A-2）；nC 是鄰近方格的 total_coeff 平均。畫面內的方格經由 Intra 預測與轉換，每個 4x4 方格內的 Coefficient 數量有很高的相關性。藉由 nC，CAVLC 得以與鄰近方格進行”**Context – based Adaptive**”的編碼；並且在設計 CAVLC 編碼表時可以將相鄰方格的相關性考量入內，減少位元總數。

- **CAVLC 的 Distribution Coding 與 Level Coding 分開進行**

H.263 的 Run – Level Coding 中，二進位代號同時紀錄了 Coefficient 與 Coefficient 之間零的數量與 Coefficient 的大小。Coefficient 之間零的數量反映了 Coefficient 的分布情形，因此 Run – Level Coding 同時進行了 Distribution 與 Level Coding。Distribution 與 Level Coding 同時進行減少了計算機編碼過程的複雜度，但是有其尚待改善之處。我們以下列兩個數列爲例，解釋 Distribution 與 Level Coding 同時進行的缺點：

數列 A（0,3,0,0,0,0,3,0,0,0,……..,0），數列 B（0,0,0,0,3,0,3,0,0,0,……..,0）。

數列 A 與數列 B 都有兩個 Non Zero Coefficient，儘管位置不同，但都有相同的（Run，Level）：（1，3）與（4，3）。Run – L evel Coding 不論（1，3）與（4，3）在數列上哪一個位置，都對應相同的二進位代號，因此數列 A 與數列 B 經由 H.263 的 Run – Level Coding 產生相同的位元總數。很明顯的，數列 A 出現的機率遠比數列 B 低，H.263 的編碼卻產生相同的位元數，這違背了熵編碼的基本原則，造成了 Run – L evel Coding 的缺點。

CAVLC 的 Distribution Coding（區段 C） 與 Level Coding（區段 B）分開進行。如此的做法克服了 Run – L evel Coding 的問題，因爲 CAVLC 的 Level Coding 提供不同的頻率不同的二進位代號。

- **CAVLC 的動態範圍往低頻遞增**

CAVLC 的 Level Coding 隨著 Coefficient 所在的頻率而不同，動態範圍越往低頻越大。H.263 Run – Level Coding 的動態範圍與所在的頻率無關，只與 Coefficient 之間零的數目（Zero Run）有關。根據 Run – Level 編碼表（附錄 Table A-1）所定義的動態範圍，我們表列動態範圍與 Zero Run 的關係：

表 2-13 動態範圍與 Zero – Run 的關係

Zero Run	Dynamic Range
0	-12 ~ +12
1	-6 ~ +6
2	-4 ~ +4
3	-3 ~ +3
4	-3 ~ +3
5	-3 ~ +3
6	-3 ~ +3
7	-2 ~ +2
8	-2 ~ +2
9	-2 ~ +2
10	-2 ~ +2
11	-1 ~ +1
12	-1 ~ +1
13	-1 ~ +1

使用 Run – Level Coding 的 H.263，當 Coefficient 之間的 Zero Run 固定，根據 Run – Level 編碼表所定義的動態範圍也固定。

CAVLC 的動態範圍隨著頻率而改變，**往低頻移動一格，動態範圍位階晉級，Entropy Coder 所能描述的動態範圍就放大一倍。**

兩者的優劣：

從模擬的結果看出 CAVLC 的壓縮表現較佳。CAVLC 的編碼器卻因爲需要儲存大量的編碼表（附錄 A-2 到 A-6），以及彈性語法需要儲存前後標記的資料以查表，CAVLC 的編碼器需要龐大的緩衝器。