國 立 交 通 大 學

電 控 工 程 研 究 所

碩 士 論 文

研究生：邱致瀚

指導教授：董蘭榮　博士

用於低位元率輕量級視訊壓縮的熵編碼模式切換技術

Study on Dual-mode Entropy Coding Selection for Low Bit Rate Lightweight Video Compression

中 華 民 國 九 十 八 年 九 月

用於低位元率輕量級視訊壓縮的

熵編碼模式切換技術

Study on Dual-mode Entropy Coding Selection for Low Bit Rate Lightweight Video Compression

研究生：邱致瀚　　Student：Zhi-Han Qiu

指導教授：董蘭榮 博士　　Advisor：Lan-Rong Dung

國 立 交 通 大 學

電 控 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Department of Electrical and Control Engineering

College of Electrical Engineering and Computer Science

National Chaio-Tung University

in Partial Fulfillment of the Requirements

for the Degree of Master

in

Electrical and Control Engineering

September 2009

Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 八 年 九 月

用於低位元率輕量級視訊壓縮的

熵編碼模式切換技術

研究生：邱致瀚　　指導教授：董蘭榮　博士

國立交通大學

電控工程研究所

摘要

這篇論文的主要目標是為低位元率輕量級視訊壓縮設計一種熵編碼模式切換演算法。在低位元率輕量級視訊壓縮中，熵編碼為系統中最花費時間的步驟。而熵編碼採用 CABAC 平均約需要 CAVLC 1.5 倍的時間。所以我們發展熵編碼模式切換演算法以達到有效切換模式的目的。我們利用熵編碼模式切換演算法計算在使用位元率控制下，單張畫面使用兩種熵編碼模式會造成的畫質差。當 CABAC 能比 CAVLC 能有效提升畫面達 0.5dB 以上，才使用 CABAC 模式，反之則使用 CAVLC 以節省時間。在演算法中，我們同時也為 CABAC 發展碼率預估演算法而能用更快的速度取得 CABAC 的壓縮量。從模擬結果顯示能比只用 CAVLC 模式平均每張畫面能達到 0.7dB 的畫質提昇，而相對 CABAC 比 CAVLC 需要的時間，使用切換演算法平均能節省 40%的時間。

Study on Dual-mode Entropy Coding Selection for Low Bit Rate Lightweight Video Compression

Graduate Student: Zhi-Han Qiu Advisor: Dr. Lan-Rong Dung

Department of Electrical and Control Engineering

National Chiao Tung University

Abstract

The objective of this thesis is to develop a dual-mode entropy coding selection algorithm for the low bit-rate lightweight video compression system. In applications of low bit-rate lightweight video compression system, entropy coding stage costs the most time and the time of CABAC is 1.5 times the time of CAVLC on the average. As a result, we develop a dual-mode entropy coding selection algorithm to choose the better mode for each frame encoded. Dual-mode entropy coding selection algorithm can be used to compute the PSNR difference of one frame which encoded by two kinds of entropy coding mode under rate control. When CABAC increases better PSNR value than CAVLC by 0.5 dB, the algorithm chooses CABAC mode to encode. When the value is below the threshold, the algorithm selects another mode. In our algorithm, we also present a CABAC rate estimation method to save computation complexity. Simulations shows that the proposed algorithm can produce better PSNR value by 0.7 dB than encoding by CAVLC only and save up to 40% additional simulation time than encoding by CABAC only.

誌　謝

這篇論文能夠完成，要謝謝許多照顧我以及幫助我的人。

首先謝謝我的指導教授，董蘭榮老師。在研究所的兩年中，董教授不厭其煩地為我指點迷津，而也總能幫我考慮到許多原本沒想到的地方，讓我獲益良多。

另外，謝謝我的朋友們，能把我挫折時的沮喪換為歡笑，並且包容我的一切幼稚以及任性。這幾年能夠跟你們相處在一起是我的幸運，除了謝謝之外，也希望我們能夠一起成長。

同時，也感謝實驗室的學長—盟淳、穎毅、信丞、貫康、詠麟、嘉宏、博仁，在求學和研究過程中給於指點及幫助，以及同學們—智聖、嘉鴻、昶翔，在課業與生活上的互相扶持。另外要謝謝振揚同學於研究中的幫忙。

同時，謝謝我最親愛的家人，從我外出念書開始就不斷地關心和照顧我，給予我沒有後顧之憂的生活，我能夠順利完成學業都要謝謝你們。最後也要謝謝我的女朋友妙如，一路陪伴以來不斷的照顧和鼓勵，感謝妳。

這幾年謝謝你們了。

2009.9.22

目　錄

圖 目 錄

表目錄

第一章　緒論

1.1 研究動機與目的

在[1]中提到，視訊標準經過多年來的發展，一個基本的視訊壓縮系統會包含動態估測、動態補償、訊號轉換、和熵編碼四個主要的部份。而這四個部分是由最原始的differential pulse code modulation (DPCM) 編碼加上離散餘弦轉換(DCT)發展而來。DPCM [2]、[3]是利用預測資料和原始資料的相關性去減少壓縮資料量，而相減完的殘餘資料在經過 DCT [4]轉換到頻域以減少資料量，最後的熵編碼則是透過移除統計的相關性來壓縮資料。而這種由預測和訊號轉換所組成的系統也被稱為混合視訊壓縮(hybrid video coding)[5]、[6].

在 2003 年最新制定新一代的視訊標準 H.264/MPEG-4 AVC [7]、[8]，和之前的MPEG-4、H.263、MPEG-2 比較，H.264 可以達到 39%、49%、64%的碼率降低 [9]，所以 H.264 可以提供更高的編碼效能。

低位元率輕量化視訊壓縮為系統具有的傳輸頻寬很低，位元率可能只有 1~2M (bit/s)，而系統本身為了滿足可攜性的功能，所以採用電池供電。為了讓電池可以儘可能的長時間使用，所以整個壓縮系統不採用耗電的動態估測技術或模式選擇(mode decision)等技術。低位元率輕量化視訊壓縮最典型的例子為無線膠囊內視鏡系統[10]、[11]、[12]、[13]、[14]、[15]，後兩者為之前我們所開發的系統。系統壓縮影像尺寸為512×512，而 frame rate 為 2 (frames/sec)，當位元率為 2M (bits/s)時。因此視訊壓縮演算法必須儘可能的減少功率消耗，但又必須儘可能提高影像畫質。

在 H.264 中定義了兩種的熵編碼模式，分別為內容適應性變動長度編碼(CAVLC)和

內容適應性二元算術編碼(CABAC)。根據 [16]、[17]的研究結果，在相同的畫質之下，CABAC 平均可以比 CAVLC 節省 9%~14%的壓縮資料量，但是因為兩種熵編碼性質上的不同，即 CAVLC 為直接對整個符號編碼，而 CABAC 卻是以符號位元為單位進行編碼，所以對於同樣的內容，CABAC 需要比較多的時間去編碼，根據我們的模擬，在壓縮單張 d1 的畫面時，CABAC 平均要花 1.5 倍 CAVLC 的時間。

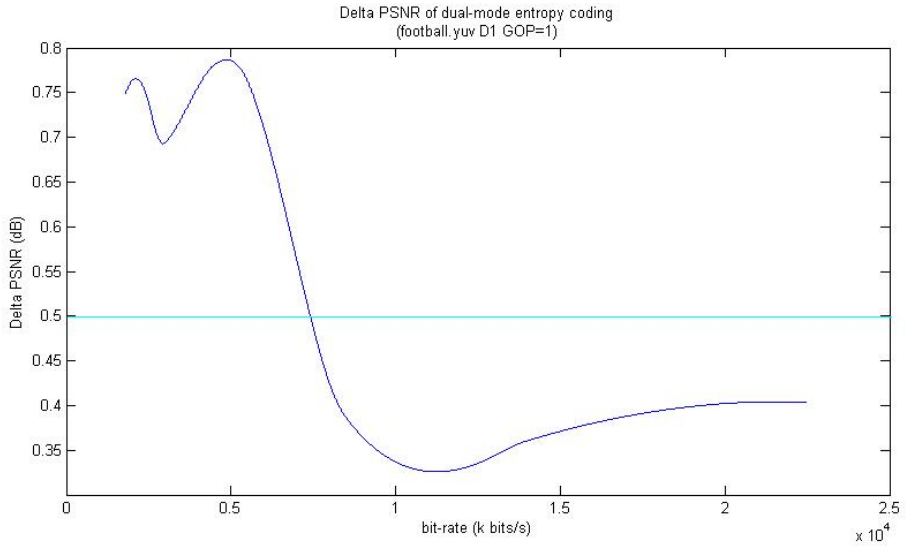


圖 1.1 delta-PSNR 圖

圖 1.1 為使用 football.yuv (D1,4:2:0)以 JM11.0[18] 實際壓縮的結果，橫軸為每秒的位元率(bits/s)，縱軸為熵編碼的 PSNR 差，為 CABAC 減去 CAVLC 的 PSNR 值差距。從圖 1.1 來看，在固定 x 軸的值之下，對應到 y 軸的兩點，即為兩種模式在固定位元率時的 PSNR 差距。但是在不同的位元率之下，兩者間的 PSNR 差會有所改變。在固定位元率且兩者壓縮的 PSNR 差距小於 0.5dB 時，表示兩種方法此時並無差異，那當這種情況發生時，我們應該要使用 CAVLC 來壓縮以節省壓縮時間。這代表我們可以設計出一種機制去估計出當前壓縮畫面兩者的 PSNR 差，而根據這個 PSNR 差來切換熵編碼模式。

Part	(%)
Entropy coding	90.56
Q/IQ	3.44
DCT/IDCT	1.29
Intra prediction	4.57
Others	0.14
total	100

表 1.1 內視鏡系統模擬時間比例表

在我們之前設計的內視鏡系統中[15]，熵編碼只使用 CAVLC 模式。從表 1.1 模擬時間表可以看出熵編碼階段在整個系統中要消耗最多的執行時間，而 CABAC 消耗的時間又比 CAVLC 更為增加。所以為了要在輕量化視訊壓縮系統中有效地利用 CABAC 來增加畫質，所以要發展熵編碼模式切換演算法，為當前畫面選擇最有利的熵編碼模式。當畫質可以明顯改善時，選擇 CABAC 模式，反之則使用 CAVLC 以減少熵編碼階段的時間消耗。我們將先計算出固定頻寬下每張畫面的編碼預算，再利用熵編碼模式切換演算法去選擇最有利的熵編碼模式。

在本文中，我們針對低位元率輕量化視訊壓縮設計了一種熵編碼模式切換演算法。在系統前端的畫面內估測，我們保留之前改良的設計，只使用一種模式來估測。接下來，利用演算法從系統位元率計算出每張畫面的編碼預算，從編碼預算利用 R-QP 模型和 PSNR-QP 模型去計算兩種熵編碼要符合編碼預算的量化值。從量化值導致的畫質差是否有明顯改善在兩種熵編碼間選擇當下最有利的模式和對應的量畫值。為了避免硬體實現時除法器的使用，所以在兩個模型的使用上，改用以查表的方法實現。為了讓在每張畫面的壓縮時間在演算法執行時不會超過使用 CABAC 壓縮的時間，我們發展了

CABAC 碼率估測演算法去估計 CABAC 的壓縮量。CABAC 碼率估測演算法主要是利用減少原始 CABAC 的執行步驟以加快速的猜測出壓縮量。在碼率估測演算法的發展中，我們根據模擬結果發現碼率估測演算法的準確度會對模式造成影響，所以我們以誤差值去選擇最後使用的方法。而最後我們的 CABAC 碼率估測演算法平均可以比 CABAC 減少約三分之二的執行時間又能準確提供切換訊息且以硬體實現時只需要些微消耗。從模擬結果顯示，能比只用 CAVLC 模式壓縮平均每張畫面能達到 0.7dB 的畫質提昇，而跟 CABAC 相比，使用切換演算法平均能節省 40%的多餘時間，且能同時達成位元率的控制。

1.2 論文架構

第一章 緒論

包括整個論文的介紹和說明研究動機和方法

第二章 背景介紹與文獻回顧

對 H.264 的熵編碼演算法和位元率控制演算法有初步的介紹，最後再探討碼率預測相關的論文

第三章 熵編碼模式切換演算法的設計

針對研究目的提出本論文的演算法，並逐節介紹演算法各步驟的設計過程。針對兩種熵編碼的碼率預估進行各種方式的改善並模擬使用結果。

第四章 模擬結果與比較

分析熵編碼模式切換演算法的整體模擬結果和模擬以硬體實現的結果。

第五章 結論與未來展望

分析說明整體結果，並由訂出未來可進行的方向。

第二章 背景介紹與文獻回顧

在本章的內容之中，將以無線膠囊內視鏡中演算法的技術作為輕量化視訊壓縮的流程基礎來進行說明，圖 2.1 為整個輕量化視訊壓縮演算法的架構圖。從 2.1 節開始，將從影像格式 bayer pattern 開始，逐步介紹這個系統中各個部份用到的技術。在第 2.3 節裡面，將詳細介紹兩種熵編碼演算法。在 2.4 節中將介紹碼率預估演算法的文獻回顧。第 2.5 節將簡述系統中利用的位元率控制演算法。

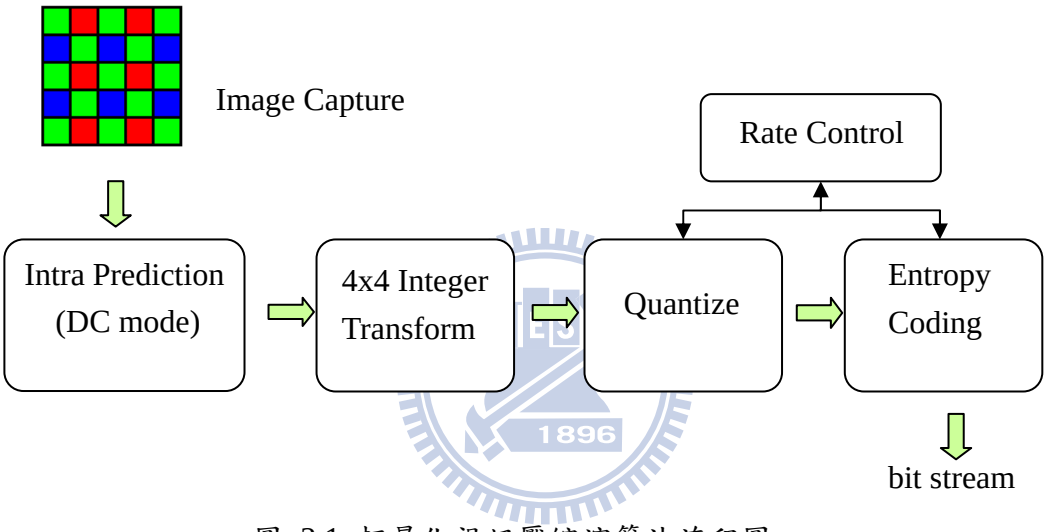


圖 2.1 輕量化視訊壓縮演算法流程圖

2.1 膠囊內視鏡取像格式

膠囊內視鏡的影像捕捉方式是透過單片色彩濾鏡的旋轉在 CMOS 感應器上的每個像素點紀錄不同顏色的值。紀錄的排列方式如圖 2.1 中，左上角的彩圖所示。即取像畫面上，G 平面的取樣像素資料會呈現對角線的排列，而在不同的列中，在分別對 R 平面和 B 平面取樣。因為人類視覺系統對 G plane 的變化較為敏銳，且 G plane 的能量在自然界中所佔的比例也比其他的兩個顏色較高，所以 RGB 三者取樣的資料量比為 1:2:1。

在我們的演算法中，對於影像感應器(image sensor)捕捉的 raw image 不先經過解馬賽克(Demosaicking)的運算做像素三原色補色，也不將補色後的彩色像素做色彩空間轉換(Color-Space Transform)。即不把彩色像素 R、G、B 三原色轉換成 Y、Cb、Cr，而是直接進行壓縮，這樣可以減少執行色彩空間轉換的功率消耗。

2.2 H.264/MPEG-4 AVC 畫面內估測的壓縮演算法

接下來的幾個部份所用到的基本方法都是參考於 H.264/MPEG-4 AVC[8]中使用的技技術。在標準中，基本的畫面估測方式有兩種:分別為畫面內估測(intra prediction)，和畫面間估測(inter prediction)，為了延長電池使用時間，所以整個壓縮流程不採用會消耗功率很大的畫面間估測而只使用畫面內估測。

2.2.1 畫面內估測 (Intra prediction)

畫面內估測的原理主要是利用當前編碼區塊周圍的資訊去建立一個預測區塊，然後只壓縮兩者的差值。當預測區塊的值越接近編碼區塊時，兩者相減所得的殘餘資料量會越低，而壓縮率會越高。H.264 的基本編碼單位區塊為巨方塊(macroblock : MB)，在 4:2:0 取樣格式下的話，一個巨方塊由 1 個 16×16 的亮度區塊和 2 個 8×8 的彩度區塊所組成。而亮度區塊和彩度區塊可以切割成的最小尺寸為 4×4。在膠囊內視鏡系統中，因為色彩空間不是使用 YUV，而是使用 bayer pattern，所以我們的最小編碼區塊的尺寸就直接設為 4×4 以方便壓縮，而這樣也省去要壓縮巨方塊檔頭的資料量。在建立預測區塊的內容時，主要是利用已經編碼過，然後再重建完的相鄰編碼區塊的內容去猜測。這裡的相鄰區塊是該編碼區塊上方、右上方和左方的編碼區塊，因為編碼順序的關係，所以這三個單位的像素內容在要壓縮當前區塊時已經是重建過後的資訊，即解碼端對應的資料。猜測的方式是利用上方、右上方和左方編碼內容去重建一個預測區塊。圖 2.2 為[1]中所提

到相鄰像素值的關係圖，圖中的 a 到 p 為要猜測的像素值位置，而 A 到 L 則是相鄰的編碼區塊已知像素值位置。

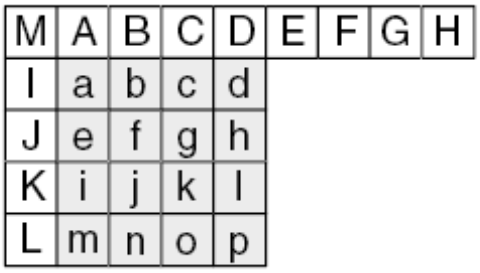


圖 2.2 相鄰像素值的關係圖 節錄自[1]

在標準中，針對不同的編碼單位尺寸有不同的猜測方式可供選擇。對一個 16×16 巨方塊有 4 種模式可供選擇，而 4×4 編碼單位有 9 種方式可供選擇，這 9 種方式列於表 2.1 中，而圖 2.3 為[1]中各模式對應的猜測方向。

NUM	Intra 4x4 prediction Mode
0	Vertical
1	Horizontal
2	DC
3	Diagonal-down-left
4	Diagonal-down-right
5	Vertical-Right
6	Horizontal-down
7	Vertical-left
	Horizontal-up

表 2.1 畫面內估測的模式 節錄自[1]

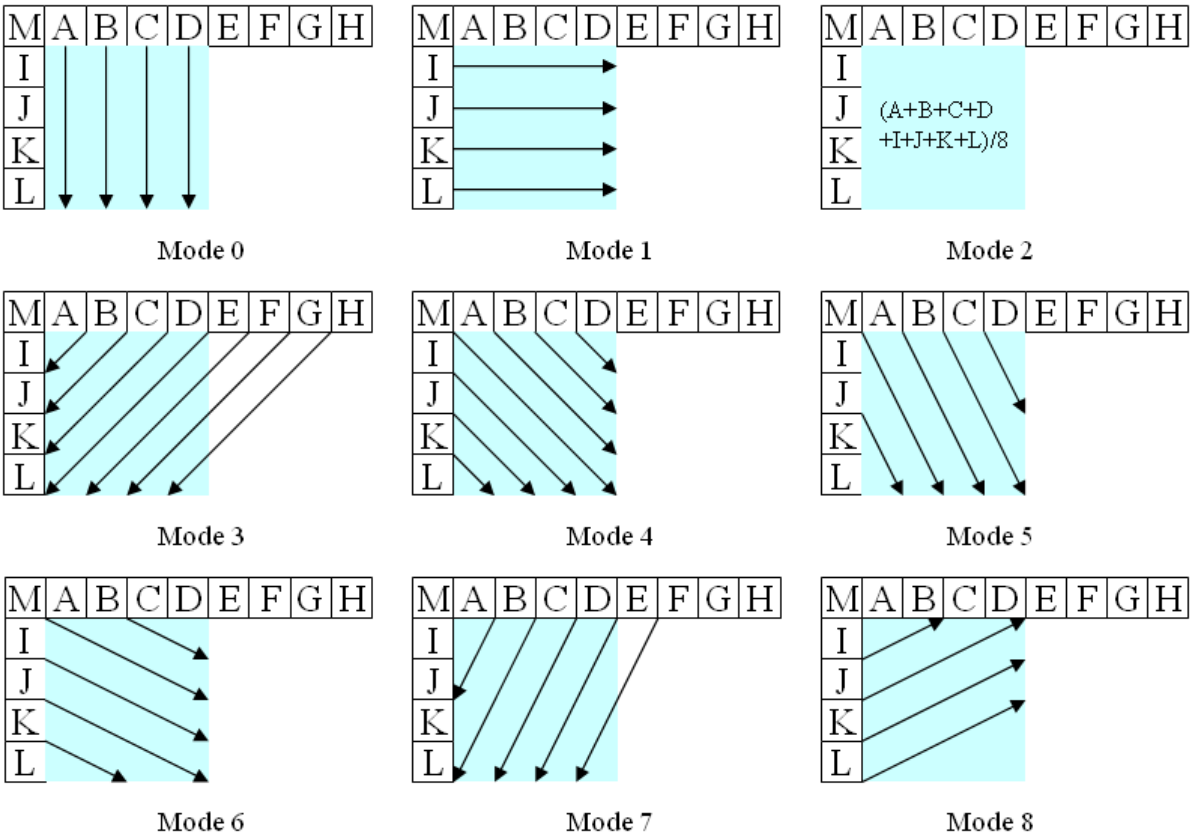


圖 2.3 各種模式的猜測方向 節錄自[1]

2.2.2 整數轉換 (4×4 IntegerTransform)

因為傳統壓縮標準中的離散餘弦轉換(discrete cosine transform，簡稱為 DCT)會帶來硬體上實現的麻煩，所以在 H.264 中，為了簡化複雜度，提出了整數轉換。整數轉換是近似的 4×4 離散餘弦轉換，最大的優點是可以把傳統的浮點運算改成只需要加法和移位的運算。下面的推導過程參考於[19]。

傳統 DCT 公式以矩陣形式表示:

$$\mathbf{Y}=\mathbf{AXA}^{\mathbf{T}}=\begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix}\begin{bmatrix} \mathbf{X} \end{bmatrix}\begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix} \tag{2.1}$$

這裡的 a,b,c 分別為:

$$a=\frac{1}{2},\quad b=\sqrt{\frac{1}{2}}\cos(\frac{\pi}{8}),\quad c=\sqrt{\frac{1}{2}}\cos(\frac{3\pi}{8})$$

為了簡化運算，將矩陣乘法中重複的項提到外面。

所以可以將(2.1 式)改寫成以(2.2 式)表示:

$$\mathbf{Y}=(\mathbf{CXC}^{\mathbf{T}})\otimes\mathbf{E}=\left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & - \\ d & -1 & 1 & -d \end{bmatrix}\begin{bmatrix} \mathbf{X} \end{bmatrix}\begin{bmatrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{bmatrix}\right)\otimes\begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \tag{2.2}$$

這裡的 a,b,d 分別為，c 值和前頁相同:

$$a=\frac{1}{2},\quad b=\sqrt{\frac{1}{2}}\cos(\frac{\pi}{8}),\quad d=\frac{c}{b}$$

近似離散餘弦轉換的整數轉換則寫成為:

$$\mathbf{Y}=\left(\begin{bmatrix}1&1&1&1\\2&1&-1&-2\\1&-1&-1&1\\1&-2&2&-1\end{bmatrix}\begin{bmatrix}\mathbf{X}\end{bmatrix}\begin{bmatrix}1&2&1&1\\1&1&-1&-2\\1&-1&-1&2\\1&-2&1&-1\end{bmatrix}\right)\otimes\begin{bmatrix}a^2&ab/2&a^2&ab/2\\ab/2&b^2/4&ab/2&b^2/4\\a^2&ab/2&a^2&ab/2\\ab/2&b^2/4&ab/2&b^2/4\end{bmatrix} \tag{2.3}$$

反整數轉換則寫成為:

$$\mathbf{Y}=\begin{bmatrix}1&1&1&0.5\\1&0.5&-1&-1\\1&-0.5&-1&1\\1&-1&1&-0.5\end{bmatrix}\left(\begin{bmatrix}\mathbf{X}\end{bmatrix}\otimes\begin{bmatrix}a^2&ab&a^2&ab\\ab&b^2&ab&b^2\\a^2&ab&a^2&ab\\ab&b^2&ab&b^2\end{bmatrix}\right)\begin{bmatrix}1&1&1&1\\1&0.5&-0.5&-1\\1&-1&-1&1\\0.5&-1&1&-0.5\end{bmatrix} \tag{2.4}$$

這裡的 a,b,d 則分別為:

$$a=\frac{1}{2},\quad b=\sqrt{\frac{2}{5}},\quad d=\frac{1}{2}$$

2.2.3 量化 (Quantization)

在整個壓縮編碼的流程之中，經過畫面內估測後的殘餘值資料經過整數轉換之後，會在經過量化這一個步驟，才把資料傳給熵編碼端。量化步驟在整個壓縮中所扮演的角色是利用量化值去調整壓縮的品質。當使用較大的量化值，會讓殘餘值的資料變的較少，可以讓壓縮率上升，但因為資料表示位數減少的關係而有較大的失真，這會導致影格解壓縮回來之後的品質下降，反之亦然。

在 H.264 的標準中，定義了 52 個量化值，如表 2.2 中所示，每個量化值 QP 都可以對應到一個真實的量化參數 Q_{step}。當 QP 相差 6，則 Q_{step} 值為兩倍的增減。

QP	0	1	2	3	4	5	6	7	8	9	10	11	12
Q_{step}	0.625	0.6875	0.8125	0.875	1	1.125	1.25	1.375	1.625	1.75	2	2.25	2.5
QP	13	14	15	16	17	18	19	20	21	22	23	24	25
Q_{step}	2.75	3.25	3.5	4	4.5	5	5.5	6.5	7	8	9	10	11
QP	26	27	28	29	30	31	32	33	34	35	36	37	38
Q_{step}	13	14	16	18	20	22	26	28	32	36	40	44	52
QP	39	40	41	42	43	44	45	46	47	48	49	50	51
Q_{step}	56	64	72	80	88	104	112	128	144	160	176	208	224

表 2.2 *QP* 和 Q_{step} 對應表

整個量化的步驟可以如(2.5 式)所示:

$$Y_{ij} = round(X_{ij} / Q_{step}) \tag{2.5}$$

其中，X_{ij} 為經過整數轉換後在矩陣中的係數，Q_{step} 為前頁所提到的量化參數，而 Y_{ij} 為量化後經四捨五入的係數。

2.3 熵編碼

熵編碼的功用主要是去除編碼符號間的重複性，按照符號出現的機率，用預先設計好的碼字去置換符號，而碼字的長度會和符號的出現機率成反比，以達成壓縮的功能。在本節中，將詳述兩種熵編碼的演算法，而在 2.3.3 小節中將介紹關於兩種熵編碼的碼率預估演算法。

2.3.1 內容適應性變動長度編碼法(CAVLC)

CAVLC 的設計原理和視訊壓縮標準 MPEG-2、MPEG-4 所用的熵編碼是類似的[20]。在轉換步驟中，不論採用的是離散餘弦轉換或是整數轉換，經過量化後的 4×4 的殘餘值區塊之中，其係數值會出現一種特別的分布，即非零整數值會分布在區塊的左上角，而左上角在物理意義上代表低頻的部份，相對的，區塊的高頻部份，即右下角的部份，係數值的絕對值通常是比較小的，或是值都為零。

CAVLC 的壓縮原理是將一個 4×4 的殘餘值區塊以反向的之字型(zigzag)順序掃出，將 16 個元素排成一維向量(1×16)。而這個向量可以用一些預先設計好的統計量來描述，因為在解碼端可以利用這些統計量的值還原成原本的殘餘值區塊。這些統計量即為 CAVLC 定義的編碼符號。對於編碼符號，CAVLC 會根據相鄰區塊的統計值分布，適應性地去選擇編碼表格來壓縮。

在一個 4×4 的殘餘值區塊之中，CAVLC 需要的統計量和對應的編碼符號為:

編碼符號	統計量
coeff_token	非零係數的個數、連續 1 的個數
trailing_ones_sign_flag	連續 1 的正負號
level	非零係數的絕對值和正負號
total_zeros	非零係數間所夾的零的個數 (第一個非零係數和 最後一個非零係數之間)
run_before	兩個非零係數間所夾的零的個數

表 2.3　CAVLC 所用的編碼符號和統計量

而它們的編碼順序則如下面圖 2.4 所示:

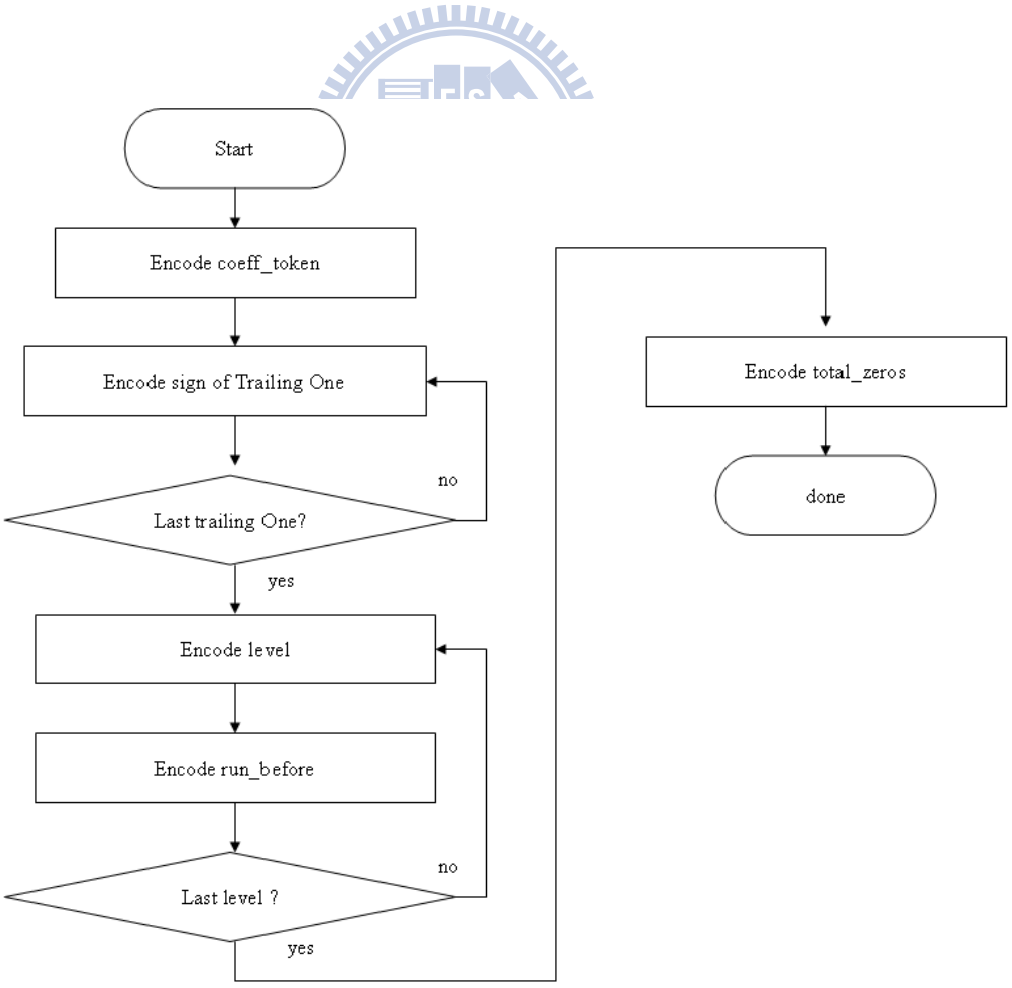


圖 2.4　CAVLC 編碼 4x4 殘餘係數區塊的流程圖

每一個編碼符號都有自己的編碼表格。在編碼每個符號時，除了要具備前面所列表 2.3 的統計量之外，在適應性地調整編碼表格時所用的參考統計量亦不同，如下面表 2.4 所示。也就是在編碼時，除了該編碼符號的值之外，其它同一個係數區塊內的編碼符號值也會參考到，而也有可能要參考不同係數區塊內的編碼符號。

編碼符號	參考統計量
coeff_token	MB 類型，上方和左方係數區塊的非零係數的個數
level	同一矩陣編碼過的最大係數絕對值
total_zeros	非零係數的個數
run_before	剩下的係數間零的個數

表 2.4 編碼符號需要的參考統計量

CAVLC 編碼範例[1]:

圖 2.5 上面方格中為一個 4×4 殘餘係數區塊原始內容，經過反向之字形掃描的向量列在下面，而 Bitstream 為壓縮結果。

0	3	-1	0
0	-1	1	0
1	0	0	0
0	0	0	0

Backward scan : 0 0 0 0 0 0 0 0 1 0 -1 -1 1 0 3 0

Bitstream : 0 0 0 0 1 0 0 0 1 1 1 0 0 1 0 1 1 1 1 0 1 1 0 1

Symbol	Value	Codword
Coeff_token	TC = 5, T1 = 3	0000100
T1 sign	+	0
T1 sign	-	1
T1 sign	-	1
Level	+1 (initial : Level_VLC0)	1
Level	+3 (3>0 : Level_VLC1)	0010
Total Zero	3	111
Run	ZeroLeft = 3; run = 1	10
Run	ZeroLeft = 2; run = 0	1
Run	ZeroLeft = 2; run = 0	1
Run	ZeroLeft = 2; run = 1	01
Run	ZeroLeft = 1; run = 1	x

圖 2.5 CAVLC範例 節錄自[1]

2.3.2 內容適應性二元算術編碼(CABAC)

下圖 2.6 是[17]中所提到的 CABAC 編碼流程圖，CABAC 在編碼的步驟上可以分為三個步驟。這三個步驟分別為二元轉換、機率模型選擇、和最後由虛線所框起來的二元算術編碼。CABAC 和 CAVLC 一個不同的地方是 CABAC 除了會去編碼量化過的 4×4 殘餘係數區塊之外，也會對一些檔頭資料進行編碼，像是 MB 類型，移動向量，參考影格等等。在本節中將詳細介紹 CABAC 的三個步驟。在我們的系統中，因為檔頭資料統一使用 Exp-Golomb Coding 編碼，所以在這裡會著重在殘餘係數區塊的編碼。

CABAC 把基本的編碼符號稱為符號元素(syntax element,在圖 2.6 中簡稱為 s.e)。符號元素在資料結構上可能是一個位元或是一個整數。經過二元化之後，符號元素將被轉換成一個符號位元序列(bin string)，該序列由符號位元所組成，而位元值為 0 或 1。CABAC 編碼的方式則是以符號位元(bin)為單位進行編碼，每一個符號位元會有一個預設的機率模型可以使用，符號位元在經過編碼後，會根據實際編碼的值去更新機率模型的值。而機率模型預測的結果越準確，CABAC 就可以得到更高的壓縮效率。

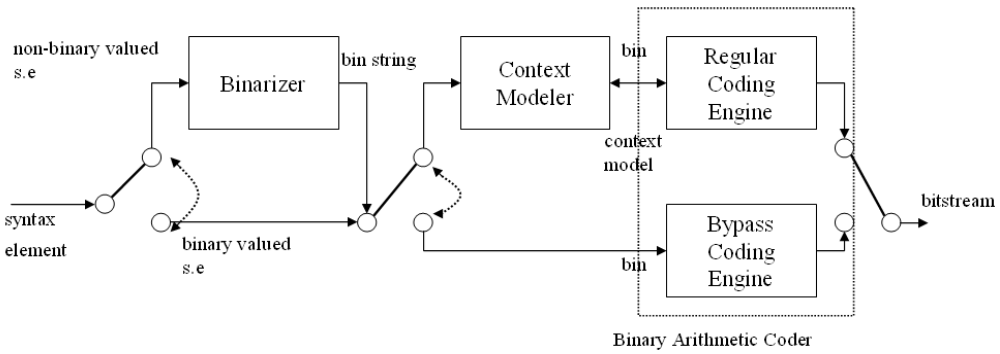


圖 2.6 CABAC編碼符號位元的流程圖 節錄自[17]

2.3.2.1 二元轉換

因為符號元素在資料結構上可能是一個位元或是一個整數。所以並不是所有的符號元素都要經過二元轉換。要做二元轉換的原因是因為 CABAC 對壓縮連續出現的相同 0 或 1 可以較佔優勢，所以當符號元素以原本的二進位表示時，其二元出現的機率較難以預測時，即會進行二元轉換以轉換成適當的符號位元序列以方便機率模型預測，而會進行二元轉換的符號元素在標準中是規定好的。

二元轉換的方式是將符號元素的值以另外一種二元的編碼來表示。編碼的內容皆為 0 和 1 的序列。而編碼的方式有下列六種。

Unary code

當符號元素的值為 n 時，所對應的 unary code 為由 n 個‘1’和 1 個‘0’所組成。

Truncated unary code

truncated unary code 轉換的規則和 unary code 很像。但是額外加了一個門檻值 x 當符號元素的值 n 小於 x，所對應的 truncated unary code 為由 n 個‘1’和 1 個‘0’所組成。而當 n 大於或等於 x 時，則所對應的 truncated unary code 為由 x 個‘1’所組成。

Fixed-length code

fixed-length code 就是直接使用符號元素原本的二進位表示。

The kth order Exp-Golomb code

the kth order Exp-Golomb code 編碼方式見下面圖 2.7 [17]。

```
while(1)
{
 if(x>=(1<<k))
 { put(1)
   x = x – (1<<k)
   k++
 }
 else
 { put(0)
   while(k--)
   put((x>>k)&0x01)
   break
 }
}
```

圖 2.7 合併Unary code 和the kth order Exp-Golomb code 的虛擬碼 [17]

Table mapping code

採用直接查表的方式轉換。

合併使用

一個符號元素中，分段使用上面兩種轉換方式。

2.3.2.2 基於內容的機率模型選擇

符號位元(bin)在編碼的時候，是由該位元採用的二元算術編碼方式來決定是否會使用機率模型。在圖 2.6 中，當二元算術編碼的方式是採用 regular coding engine 時，才會去為這個位元選擇機率模型。而採用 bypass coding engine 時，則不選擇機率模型。因為 bypass coding engine 是用來編碼當符號位元序列(bin string)中，1 或 0 出現的機率接近的情況，所以就直接以出現機率 0.5 猜測，而不使用機率模型。

預測機率模型表裡面存放的值為機率狀態(pstate)和符號位元的較高機率出現值(valMPS，以下簡稱 MPS)。pstate 代表這個符號位元會出現 MPS 的機率，而 MPS 的內容為 0 或 1，代表兩者中哪者出現的機率高。不同的符號位元可能會對應到相同的機率模型位置，而該位置在標準中以機率狀態索引(ctxidx)表示。

在使用 CABAC 時，機率模型表的使用週期為 slice 定義的大小。在我們的設計中，則是以一張畫面為使用週期。在每個使用週期一開始，必須以畫面的 QP 值和畫面型態去重設機率模型表中存放的值。

在基於內容的機率模型選擇步驟中，對應一個進來的符號位元，必須根據它所屬的符號元素去選擇 ctxidx 的計算公式。

例如:要編碼 coded_block_flag 的公式為(2.6 式) [8]:

(定義:一個 4×4 殘餘係數區塊是否所有的元素皆為零)

$$ctxidx = ctxidxOffset + ctxCatOffset + ctxIdxInc \tag{2.6}$$

在這裡，ctxidxOffset 為 coded_block_flag 這個符號元素在機率模型表中起始的位置，而 ctxIdxInc 為這個符號位元在符號位元序列中的位置對應值或是要參考係數區塊的值去做調整，ctxCatOffset 則代表係數區塊類型的權重，如亮度區塊和彩度區塊所對應的值即不同。

在下一個步驟的時候，會因為符號位元實際值和 MPS 的相同與否，去決定 pstate 的增減與否，而改變後 pstate 和 MPS 必須再記錄回機率模型表之中，以達到隨內容更新的作用。

2.3.2.3 適應性二元算術編碼

二元算術編碼可以算是一種簡化的算術編碼。算術編碼在運作上必須先知道每個符號的出現機率，而符號的出現機率可以對應到一個範圍。在編碼時，會一直遞迴地去更新範圍，而這個範圍即是代表符號組合出現的機率。二元算術編碼是因為要編碼的符號元素只有 0 和 1，所以整個更新的步驟可以大為簡化，因為更新的過程就是在原來的過程中產生兩個新的範圍。在二元算術編碼中，也因為只會產生兩個新範圍，所以編碼的方式就可以大為簡化，而不用像原始的算術編碼在記錄代表機率的範圍時，需要使用很多位元數。

在標準中所規定的二元算術編碼，基本原理和上述的相同。下面的圖 2.8 可以代表二元算術編碼每次在編碼時如何更新範圍。圖 2.8 中的 Range 和 Low 代表要壓縮一個符號位元所需要的數值。Range 代表這個符號位元所在的區域，而 Low 所代表的則是整個範圍的下界，所以 Range 和 Low 的總和即為整個區域的上界。CABAC 使用的編碼範圍大小為[0~1023]。在前文所提到該符號位元對應的 pstate 和 MPS 即可以計算出這個符號位元代表 MPS 和 LPS 的範圍，分別為 R_{MPS} 和 R_{LPS}。LPS 指的是出現機率較小的 0 或 1，相對於 LPS 而言。

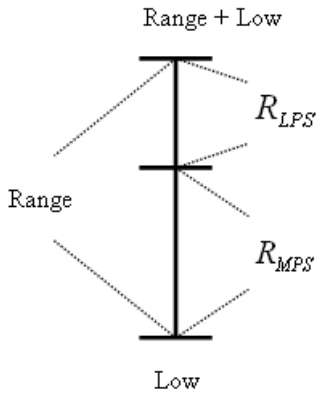


圖 2.8　Range 和 Low 示意圖

這裡的 R_{LPS} 可以由公式(2.7)所得到，P_{LPS} 紀錄的值即為 LPS 會發生的機率。R_{MPS} 亦然。

$$R_{LPS} = R \times P_{LPS} \tag{2.7}$$

但是為了實現不使用乘法的演算法，所以 pstate 中紀錄的是預先定義好的機率狀態值。在 CABAC 中，利用 pstate 和 Range 就可以查表得到現在的 R_{LPS}。而 R_{MPS} 則是用 Range 減去 R_{LPS} 就能計算出。二元算術編碼有三種運作方式，當符號元素中 MPS 和 LPS 兩者的出現機率相差較大時，則採用 encodeDecision 的方式，而當出現機率接近的時候則使用 encodebBypass 的方式。下圖(2.8)和圖(2.9)是定義在[8]中，二元算術編碼使用 encodeDecision 方式的流程圖。另兩種列於後頁。

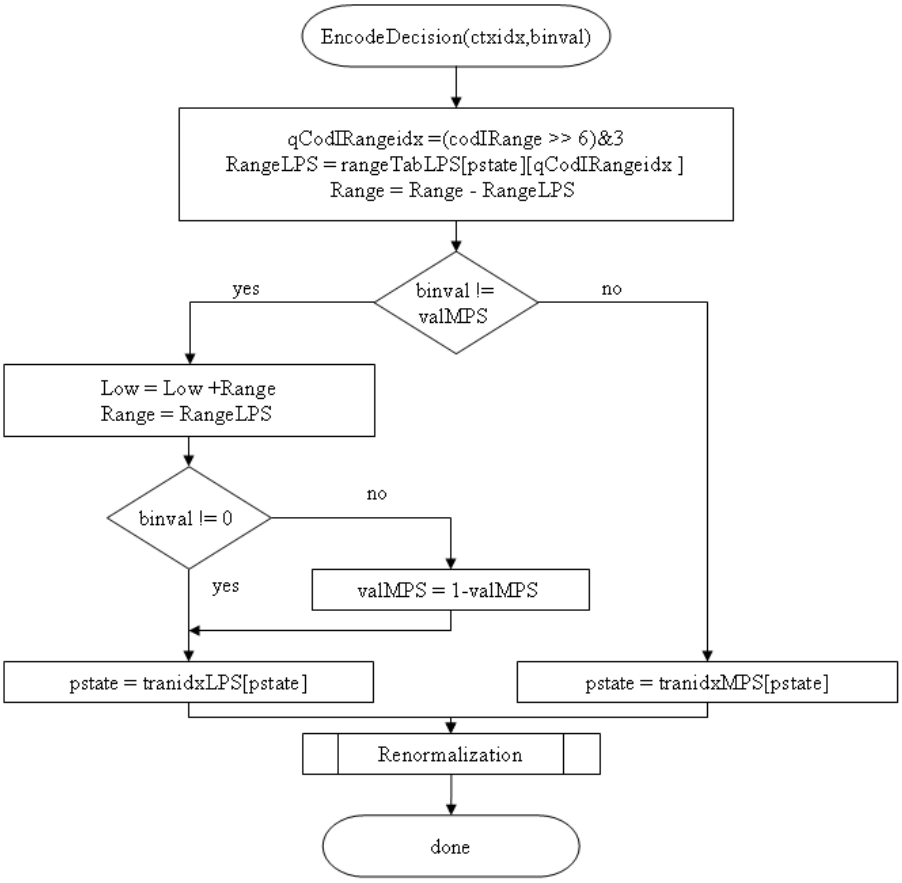


圖 2.9 適應性二元算術編碼 (encodeDecision) 流程圖 節錄自[8]

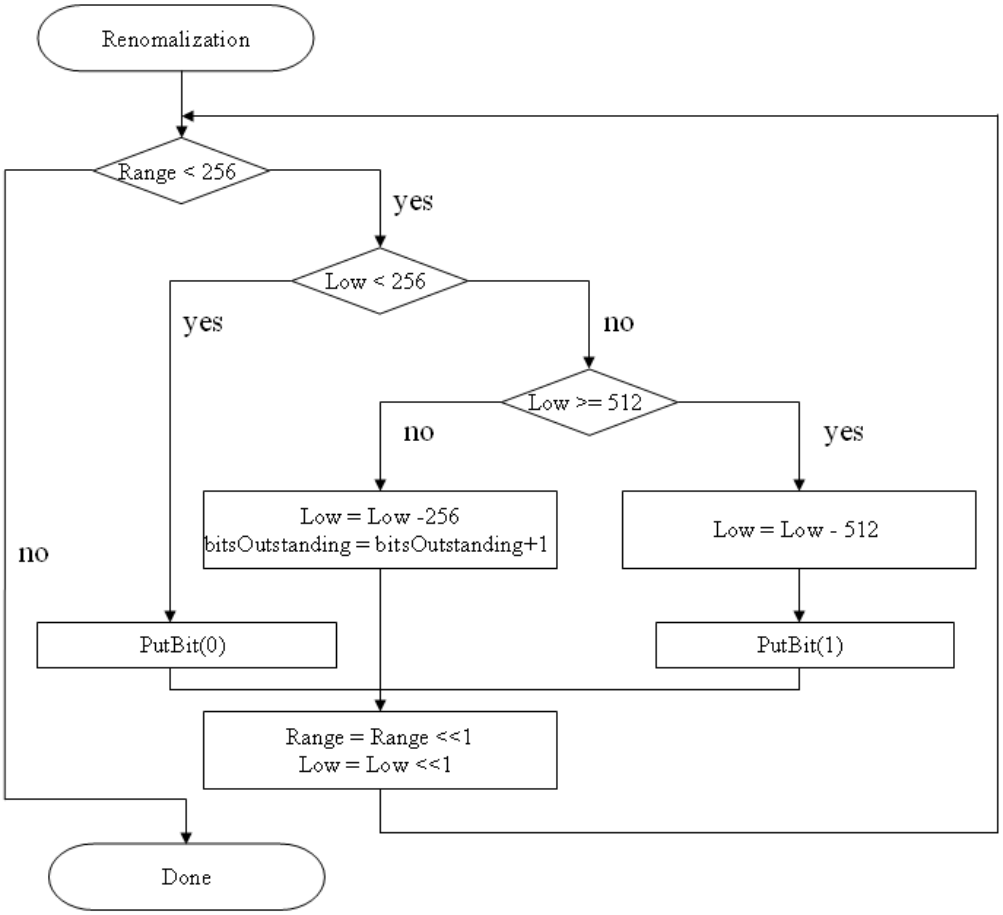


圖 2.10　Renormalization流程圖 節錄自[8]

在圖 2.9 中，第一個方塊即代表 R_{LPS} 的查表過程。在計算出 R_{LPS} 和 R_{MPS} 後，就可以根據現在要壓縮的符號位元和機率模型中紀錄的 MPS 是否相同來進行編碼。當兩者相同的情況發生時，也就是猜對的情況，則新的範圍會被 R_{MPS} 取代，反之猜錯則由 R_{LPS} 取代。而新的 Low 也必須一起更新。而圖 2.10 則是代表在圖 2.9 中 Renormalize 區塊的流程圖。

為了讓在壓縮每一個符號位元時，所用的 Range 和 Low 都能在保持在定義好的區間中，所以在圖 2.9 中，上半段的步驟已經按照更新了 Range 和 Low，且必須讓新的 Range 和 Low 回到合理的範圍。Range 合理的範圍為[256~512]，而 Low 則是[0~1023]。在圖 2.10 中，會先判斷 Range 是否大於 256，小於 256 時，就會進行 Renormalize 的步驟，而這也是 CABAC 輸出位元的機制。當整個編碼範圍確保完全落於上半區間時，即

Range 小於 256，而 low 大於 512 時，即會輸出 1，反之則輸出 0。當上下區間都有接觸到時，則讓未來輸出位元長度(bitsoutstanding)加 1。

接下來介紹剩下的兩種運作方式，當符號元素中 MPS 和 LPS 兩者的出現機率相差較小時，即兩者的機率都接近 0.5 時，則採用 encodeBypass 的方式。當符號位元採用 encodeBypass 的方式編碼，則不需要選擇機率表格，而當定義的 CABAC 編碼範圍要結束時，則採用 encodeTermination 的方式。下面的圖 2.11 和圖 2.12 為 encodeBypass 的流程圖和 encodeTermination 的流程圖。兩者進行的流程和圖 2.9 類似。

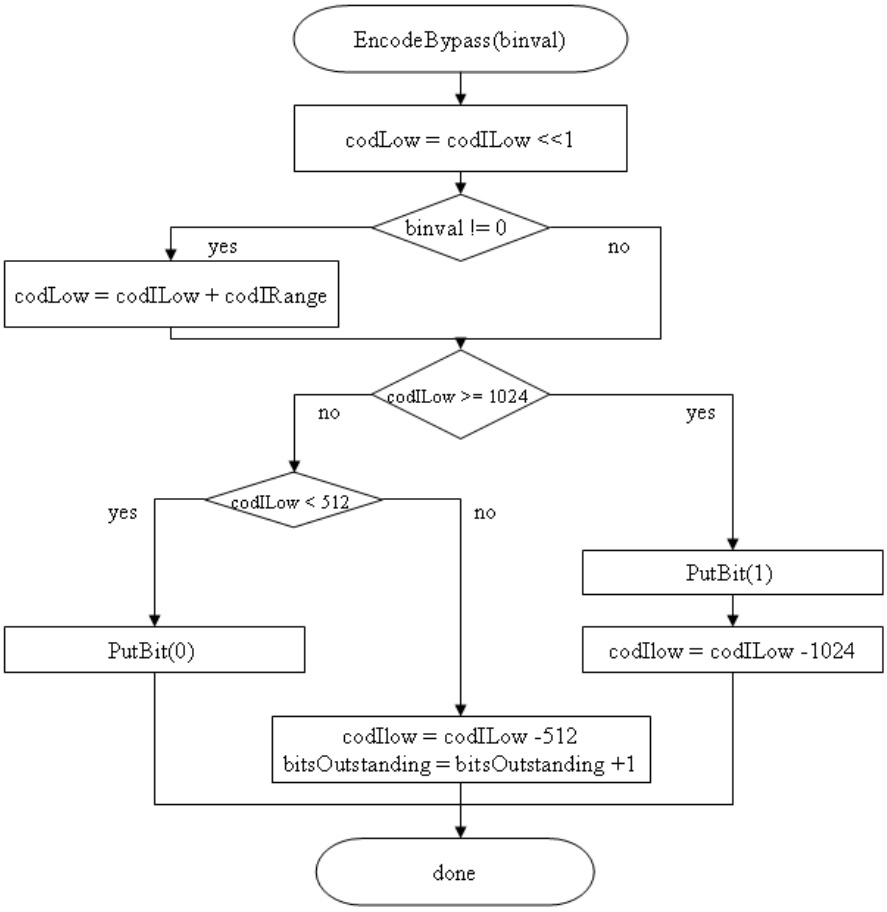


圖 2.11 適應性二元算術編碼 (encodeBypass) 流程圖 節錄自[8]

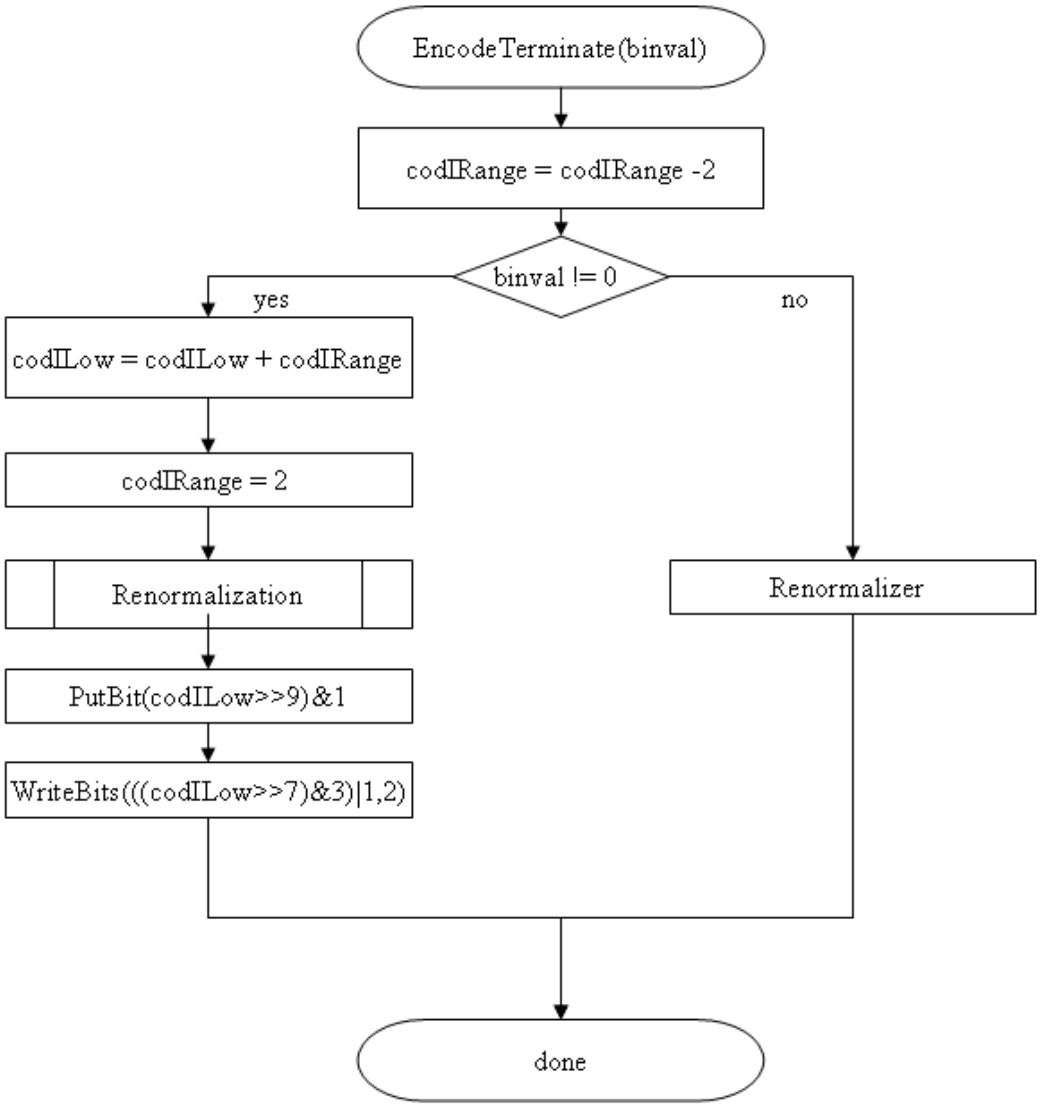


圖 2.12 適應性二元算術編碼二元算術編碼 (encodeTerminate) 流程圖 節錄自[8]

2.4 碼率預估 (rate estimation) 的文獻回顧

在我們設計的演算法中，因為要同時在壓縮的步驟中使用 CABAC 和 CAVLC，但是又只需要知道壓縮完的資料量就可以更新模型，所以我們可以選擇在壓縮一張畫面時只真正執行其中一個，而另外一個熵編碼就採用碼率預估的方式取得壓縮完的資料量以減少計算時間上的消耗。另外，因為猜測的準確度會影響到切換模式，所以在我們的設計上，也必須同時兼顧猜測的準確性。

在[21]、[22]、[23]這三篇論文中，都有提到要怎麼去猜測一個殘餘係數區塊經過熵編碼後的資料量。[21]是針對 CAVLC 的資料量猜測，而[22]和[23]則是針對 CABAC 的資料量猜測。這三篇提出壓縮資料量猜測的原因都是為了去減少執行 rate distortion optimization (RDO)的時間。RDO 執行的方式如下:在 H.264 中，因為一個 MB 定義了 16 種的變動尺寸大小區塊切割模式，而要選擇最佳的模式，根據 Lagrange optimization theory ，必須利用每一種切割模式會產生的誤差和實際編碼量去計算代價函數(cost function)的值，而以模式能有最小的代價函數為所選定的模式。但是為了求出實際編碼量，殘餘係數區塊必須經過所有的編碼步驟，這樣會太消耗時間，所以在熵編碼的步驟上，就有一些研究提出了關於壓縮資料量的猜測方法。

在[21]中，預估殘餘係數區塊經過 CAVLC 壓縮後資料量的方法，也是要先求得在 CAVLC 執行時需要的編碼符號。在利用[21]所提出的(2.8 式)去計算殘餘係數區塊壓縮後的資料量。在(2.8 式)裡，R 為預估的資料量，T_c 為區塊中的非零係數個數，T_z 為最後一個非零係數前的零的個數， SAT_l 為所有非零係數值的絕對值總和，而 f_k 為非零係數出現的順序，從低頻往高頻的方向排，從 1 開始。

$$R = T_c + T_z + SAT_l + 0.3\sum_{k=1}^{Tc} f_k \tag{2.8}$$

這個方法等於是將編碼表格中的編碼資料長度和統計量的關係簡化成函數。雖然可以表達出兩個不同矩陣壓縮完的資料量變化，但是這個方法在預估不是用查表編碼的編碼符號上，如 Level，會有比較大的預估誤差，而 Run_zero 的部份用固定權重係數 0.3 在某些情況下也會有預估誤差。

在[22]中所提出的猜測方法是利用原始的熵定義。上一節中，我們列出了用 CABAC 編碼一個 4×4 係數矩陣所需要的編碼符號。而[22]所提出的方法是:在編碼單位都經過二元化之後，會變成符號位元序列。對同一個編碼符號所產生的序列而言，去統計出每個序列中出現的 0 和 1 分別的個數，就可以計算 0 和 1 在這個序列中發生的機率，再利用(2.9 式) 和(2.10 式)算出兩者等效的編碼位元數，最後再利用(2.11 式)就能計算這個位元序列的編碼量。每個編碼符號都按照這個方式計算就可以預估整個殘餘係數區塊資料量。在(2.9~2.11 式中)，R 為預估的資料量，B_{zero} 為位元序列編碼「1」的編碼代價，而 B_{one} 為編碼「0」的代價。

$$B_{zero} = -\log_2 P_{zero} = \ln P_{zero} / \ln(0.5) \tag{2.9}$$

$$B_{one} = \ln(1 - P_{zero}) / \ln(0.5) \tag{2.10}$$

$$R = B_{zero} \times N_{zero} + B_{one} \times N_{one} \tag{2.11}$$

在[23]中所提出的方法，同樣需要對應的編碼符號。和[22]不同的是，[23]的方法對應不同的編碼符號必須使用不同的預估公式。對 significant_coeff_flag 和 last_significant_coeff_flag 這兩個編碼符號來說，使用(2.12 式)，而對

coeff_abs_level_minus1 來說，則使用(2.13 式)。編碼符號的定義請參閱[8]。(2.13 式)的使用會根據預先定義好的門檻值，而門檻值由(2.14 式)和(2.15 式)決定。在[22]和[23]中，對於 coeff_block_flag 都是直接用一個數字猜測，因為它的資料長度只有 1 位。而 coeff_sign_flag 則不需要猜測，因為壓縮完的資料等於原本的資料量。

$$R_{s\_map} = C_z \times N_z + C_{sm} \times N_s \quad C_z = 1, C_{sm} = 1.5 \tag{2.12}$$

$$R = \sum_{i=1}^{total\_coeff} Est(L_i) \tag{2.13}$$

$$Est(L) = \begin{cases} L & if\ L \le T \\ avg(L) & otherwise \end{cases} \tag{2.14}$$

$$T = \begin{cases} 5 & QP > 15 \\ 8 & otherwise \end{cases} \tag{2.15}$$

2.5 位元率控制 (rate control)

在一個壓縮的系統中，因為進入熵編碼端的資料流量是固定的，但是壓縮所產生的輸出資料量卻會隨著畫面的內容而變。為了不讓輸出的資料量長度造成緩衝器資料超過範圍或是沒有資料被讀取，必須使用位元率控制來讓編碼完的位元率達到我們所要求的值。下面將介紹所使用的位元率控制演算法[24]。

在[24]中提出了配合 H.264 技術的位元率控制演算法，位元率控制的最小單元稱為 basic unit。basic unit 由任意個 MB 所組成。在我們的設計中，basic unit 將設定為整個畫面。位元率控制演算法的完整步驟如下:

1. 使用 fluid traffic model 和 linear tracking theory [25]去計算當前面的編碼預算。先利用系統每秒輸出的位元率和畫面數去計算整個畫面群組(GOP)擁有的預算。然後在壓縮每一張畫面後，計算出目標的緩衝器佔有率。利用這兩項值去計算當前畫面的編碼預算。
2. 如果設定 basic unit 小於整個畫面的話，把剩餘的預算平均分配給目前畫面中剩下的 basic unit。
3. 利用前一個畫面中，相同位置 basic unit 的平均絕對值誤差總和(MAD)，去猜測目前 basic unit 的 MAD
4. 使用 quadratic R-D model [26]、[27]去計算出壓縮這個 basic unit 需要的 QP
5. 利用計算的 QP 去壓縮目前 basic unit 內的 MB

第三章 熵編碼模式切換演算法的設計

在本章中，我們將介紹熵編碼模式切換演算法。為了使用一個完整的輕量級視訊壓縮系統進行演算法的模擬，前端的畫面內估測，轉換和量化等步驟將使用[15]的系統設計。在 3.1 節將簡述之前研究在畫面內估測的改良。3.2 節將介紹合併位元率控制之後的熵編碼模式切換演算法。3.3 節為碼率預估演算法的深入分析。

3.1 畫面內估測模式的簡化

在[15]的研究之中，我們已經提出了一個輕量級視訊壓縮系統編碼器。為了節省功率的消耗，所以我們選擇只採用畫面內估測的方式，而完全不使用畫面間偵測。在我們的系統中，最小的編碼單位為 4×4 的殘餘係數區塊。但是因為 bayer pattern 格式包含四個間隔的資料要分別做壓縮，所以每次從記憶體中讀取的大小為 8×8 的區塊，而讀取出來的資料要再重新整理成四個不同顏色的平面，如圖 3.1 所示。而 G1 平面和 G2 平面因為資料的相關性很高，所以在分析的過程中，兩者的表現也很接近。

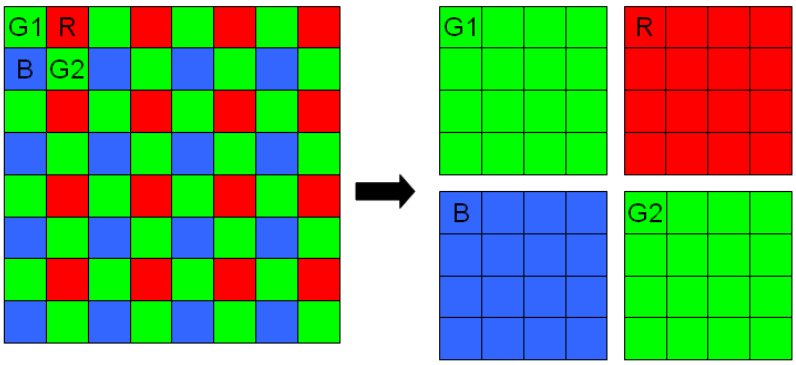


圖 3.1　bayer pattern 影像的資料重排

3.1.1 DC 模式

如 2.2.1 節所提到的，對於 4×4 的殘餘係數區塊，畫面內估測有 9 種模式可以使用。在[15]的研究中，我們先證明了在打開 RDO 之後，使用所有尺寸的畫面內估測和只使用 4×4 的 9 種模式，在我們使用的輸出位元率下，兩者的表現是很接近的。而只使用 8×8 的模式反而相差的比較遠，所以先拿掉 8×8 的模式，以節省功率消耗。接下來，再使用 4×4 的 9 種模式和只用 4×4 的 DC 模式做比較。DC 模式是利用相鄰區塊的值取平均後所得的值去建立預測區塊，如圖 2.3 的 mode 2 所示。而這個區塊的特性是每一個元素都是計算出的平均值。在經過分析後發現到在三個顏色平面的模擬中，兩者的 PSNR 差距在使用的位元率下都小於 0.3 dB，因此我們只用 DC 模式來執行畫面內估測。

3.1.2 重建 G1 模式

同樣在[15]的研究之中，我們也提出重建 G1 模式。這個模式建立的原理是利用 G1 平面和 G2 平面兩者資料的相關性很高。在圖 3.1 中，右上方的 G1 區塊經過整個編碼流程之後，會計算出一個新的重建區塊給下一個要編碼的 G1 區塊去當作預測參考。而這個重建區塊和左下方的 G2 區塊資料關聯性很高，所以用這個重建區塊去當作 G2 區塊的預測矩陣應該可以猜的更準，也就是讓殘餘資料量變的更低，同時 G2 區塊可以節省下重建的步驟。圖 3.2 為在[15]中分析 G2 區塊使用 DC 模式和使用重建 G1 模式的 R-D 曲線比較圖。在 PSNR 值為 40dB 時，兩者的差距大約 0.6dB。

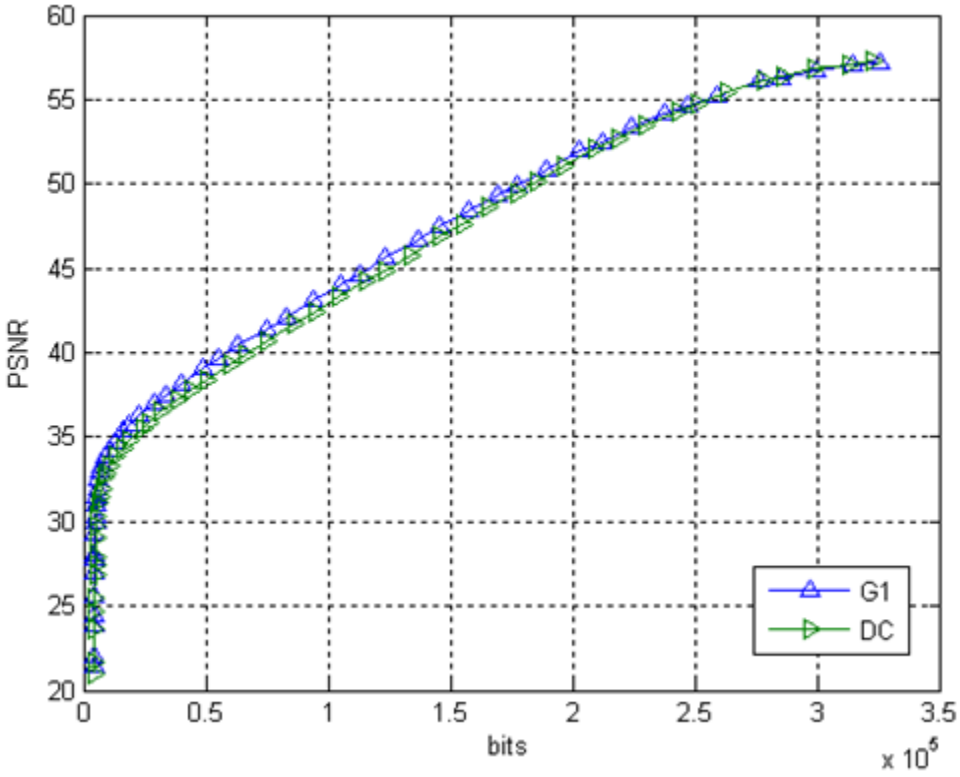


圖 3.2 兩種模式的R-D 曲線比較圖 節錄自[15]

3.2 熵編碼模式切換演算法

在本節中，我們會提出一個切換熵編碼模式的演算法。這個方法將利用 2.5 節中所提到的位元率控制演算法計算需要的資訊。在壓縮每一張畫面之前，先計算出當前畫面的編碼預算，再利用 rate-QP model(R-Q model) 去計算出要符合畫面預算需要的 QP 值，而對兩種不同的熵編碼模式則會各算出自己對應的 QP 值，在將兩者 QP 值的差異對應到 PSNR-Q model 上，計算出用兩種模式壓縮對畫質的影響。在根據畫質差是否有超過預設值而在兩者間切換。我們選定去判斷切換的預設值為 0.5dB。下面圖 3.3 代表我們在影像中執行熵編碼模式切換演算法的步驟

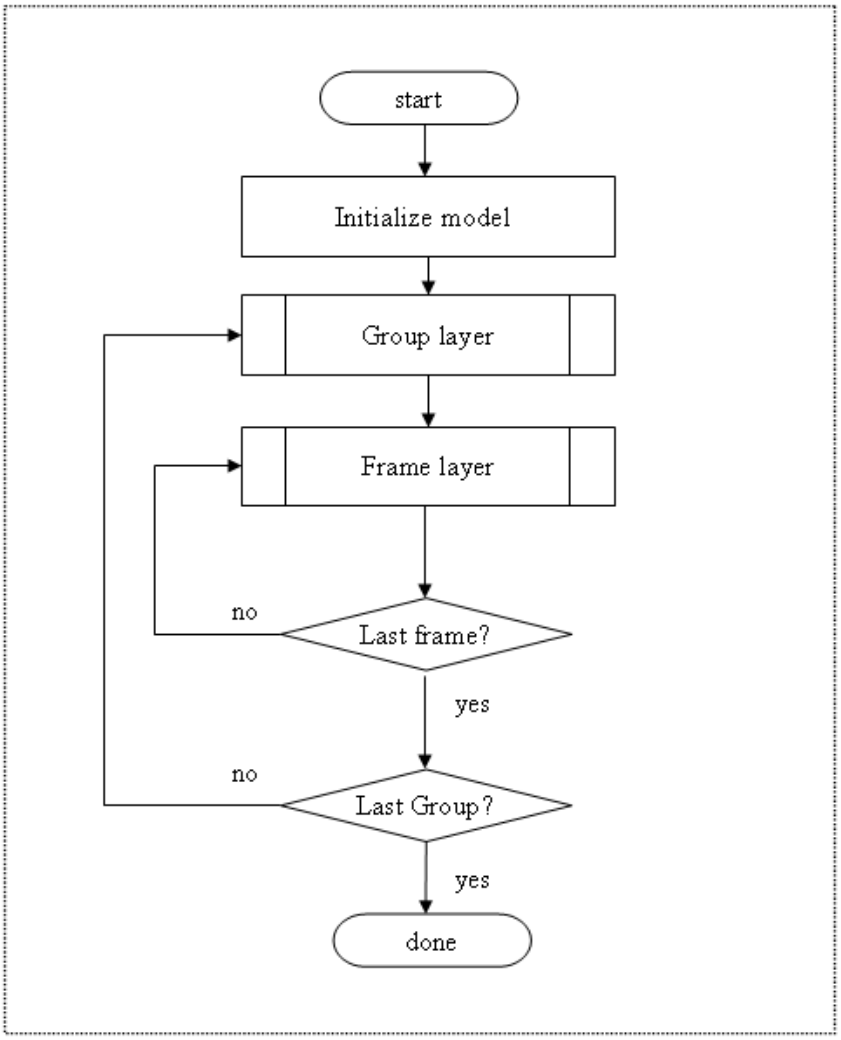


圖 3.3 熵編碼切換演算法的流程圖

在[24]的位元率控制演算法中，如果選擇一張畫面為基本單位(basic unit)，那整個控制流程會分成兩個層面進行控制。這兩個層面分別為 GOP 層控制和 frame 層控制，我們為了保留使用 GOP 層控制的優點，和減少碼率預估的時間消耗，所以將 N 張畫面組成一個 group。我們預設的 N 值為 4，以方便於硬體實現時執行除法運算。

整個演算法最外圍以 group 為單位執行。而在每個 group 中，才針對每張畫面進行模式切換，所以同一個 group 中的畫面不一定會使用同一個熵編碼模式。在圖 3.3 中，一開始執行的 initialize model 步驟是為了計算 R-Q model 和 PSNR-Q model 的係數初始值並設定。而因為 initialize model 步驟在整個編碼流程中只會執行一次且為了在編碼第一個 group 時就可以切換模式，所以實際上這個步驟將預先執行好，而不會放在系統中。圖 3.4 和 圖 3.5 兩個流程圖則分別代表在圖 3.3 中的 group layer 和 frame layer 步驟。在接下來的小節中，將依序對流程圖中的每個步驟進行介紹。

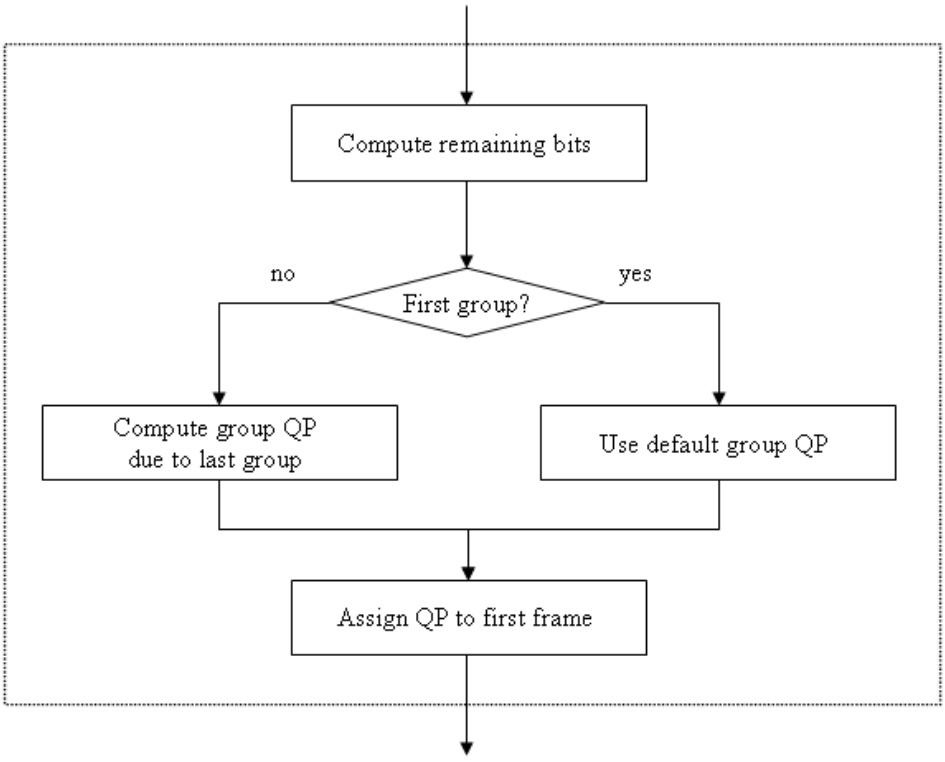


圖 3.4　group layer 流程圖

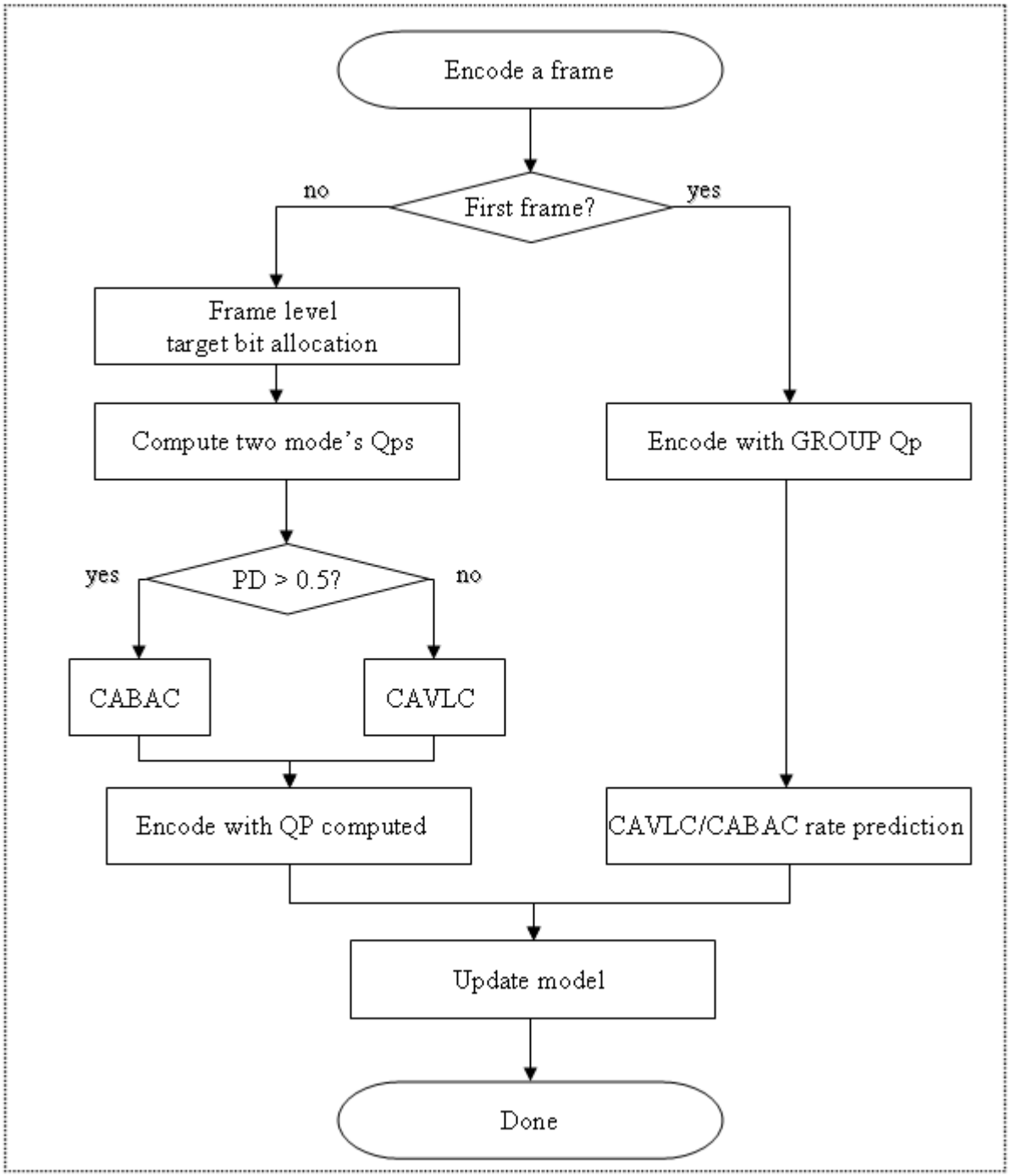


圖 3.5 frame layer 流程圖

3.2.1 畫面編碼預算的計算

在[24]所用的 GOP 層控制中，在每個 GOP 前兩張壓縮的畫面，不會為畫面計算編碼預算，而是第三張畫面後才開始計算。GOP 層控制主要的目的為: (1) 根據一個 GOP 中包含的畫面張數，利用每張畫面可以用的編碼量來為整個 GOP 設定壓縮量預算。(2) 當編碼畫面屬於另一個 GOP 時，前一個 GOP 如果有多餘的預算編碼量或是多用了預算編碼量，可以把這個量傳遞給下一個 GOP，以符合位元率控制。(3)為了讓輸出位元率在控制時，值不會太過劇烈變動，前一個 GOP 每張 P 畫面所用的 QP 值會被記錄下來，最後算出的平均單張畫面 QP 值會成為下個 GOP 前兩張畫面的初始 QP 值。

在我們的演算法中，因為沒有使用畫面間估測，因此所有的畫面都屬於 I frame。所以在我們的設計中，就沒有 GOP 的區別。但是為了保持上面所列出的三項優點，所以我們利用 GOP 的觀念將 N 張畫面組成一個 group。除了第一個 group 使用預設的 QP 當作初始值，之後其他的 group 則同樣採用上述的方法傳遞資料，如圖 3.4 所示。這樣設計可以讓後面的 group 可以利用前面留下來的多餘預算或是在這個 group 必須降低預算。上述的方法簡化後的公式如(3.1 式)和(3.2 式)所示。

$$R_{group(i+1)} = \frac{u(n)}{F_r} \times N_{group} + R_{group(i)} \tag{3.1}$$

$$QP_{group(i+1)} = \frac{sum(QP(I_i))}{N_{group}} \tag{3.2}$$

在(3.1 式)和(3.2 式)中，下標的 group(i+1) 表示影片中預編碼的第 i+1 個 group，所以 $R_{group(i+1)}$ 代表第(i+1)個 group 在這個 group 開始編碼時的預算編碼量，$R_{group(i)}$ 為上一個 group 經過編碼後剩下的預算編碼量。$u(n)$ 為輸出頻寬，單位為 bits/s。F_r 為每秒壓

縮的畫面張數，單位為 frames/s。而 N_{group} 為定義一個 group 內的畫面數。$QP_{group(i+1)}$ 為第(i+1)個 group 的初始 QP，這個 group 的第一個畫面將使用 QP 進行量化。$sum(QP(I_i))$ 為第 i 個 group 內每張畫面所用的 QP 值總和。

在圖 3.5 中，frame layer 第一個步驟就是要去判斷現在要編碼的畫面是否為該 group 中的第一張畫面，如果是的話，就直接用 $QP_{group(i+1)}$ 去壓縮，而此時不進行熵編碼模式切換的選擇。在第一張畫面中要同時執行 CAVLC 的編碼和 CABAC 的碼率猜測。這是為了避免前一個 group 如果都選擇了相同的熵編碼模式，則另一個模式的 R-Q model 將失去更新的機會，而這也會讓模型和實際值的誤差隨著沒被挑選中而越大。所以為了在每一個 group 中，兩種熵編碼模式都至少能更新一次模型，所以要在第一張畫面得出兩種熵編碼的碼率以對兩種模式的 R-Q model 進行更新。但是實際上只需要執行一種熵編碼，所以我們選擇執行 CAVLC 的編碼和 CABAC 的碼率猜測，因為這種組合的模擬時間會小於 CABAC 的編碼和 CAVLC 的碼率猜測。當畫面不為 group 中的第一張時，則要進行熵編碼模式切換的選擇，而此時必須計算出當前畫面的編碼預算，group 的設計如圖 3.6 所示，在 group 中的第一張畫面要執行兩種熵編碼模式，而後面的畫面則只需要執行選擇的熵編碼模式。

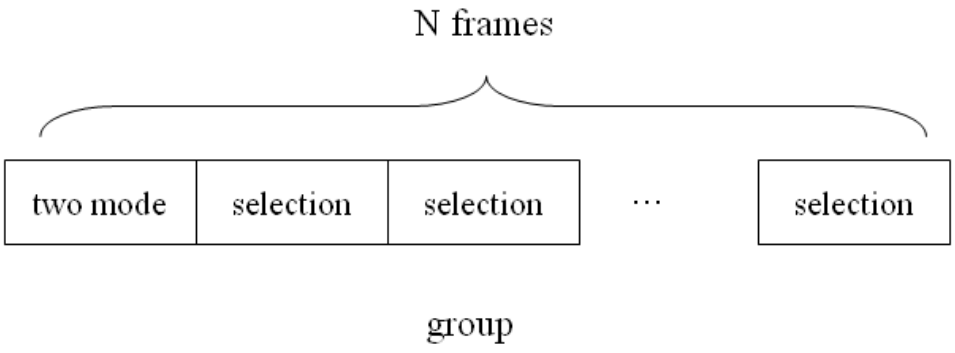


圖 3.6 group layer 的設計

在計算畫面的編碼預算時，我們利用[24]中演算法的第一個步驟。這個步驟在執行上要先使用預先設定好的虛擬緩衝器。接下來，再定義兩個描述緩衝器內資料狀態的變數: target buffer level(*Tbl*)和 actual buffer occupancy(B_c)，兩個值的單位皆為 bits。*Tbl* 代表的是位元率控制預期達到的緩衝器佔有率，而 B_c 則是實際上緩衝器的佔有率。在編碼完 GOP 中第一個 P 畫面的時候，*Tbl* 會被重設為 B_c ，這時 B_c 的值代表前兩張畫面在緩衝器內的佔有量。接下來在編碼每一張 P 畫面之後，*Tbl* 會每張逐次的減少，以回到 0 的值，這代表希望在編碼完這個 GOP 之後，緩衝器可以將資料清空。在我們的設計中，同樣也以這兩個變數來描述編碼資料在虛擬緩衝器中的狀態。

因為我們使用 group 的設計，所以在編碼完第一張畫面後，就可以重設預期的緩衝器佔有率*Tbl* 。再利用修改後的(3.3 式)、(3.4 式)、(3.5 式)就可以計算出當前畫面的編碼預算。(3.3 式)是利用緩衝器佔有率去估算編碼預算 $\tilde{f}(n_i)$ ，而(3.4 式)是利用整個 group 所具有的編碼預算平均分配給剩下的畫面去估算單張畫面的編碼預算 $\hat{f}(n_i)$ 。3.5 式則是以前兩式的加權平均作為計算出的編碼預算。

$$\tilde{f}(n_i) = \frac{u(n)}{F_r} - 0.5(B_c(n_{i-1}) - Tbl(n_{i-1})) \tag{3.3}$$

$$\hat{f}(n_i) = \frac{R_{group}(n_i)}{N} \tag{3.4}$$

$$f(n) = 0.5\tilde{f}(n) + 0.5\hat{f}(n) \tag{3.5}$$

上面三個公式中，n 代表目前要編碼的畫面。$Tbl(n_{i-1})$ 和 $B_c(n_{i-1})$ 為上一張畫面編碼過的在虛擬緩衝器中的佔有率。$R_{group}(n_{i-1})$ 為上一張畫面編碼過剩下的 group 編碼預算。N 為 group 中剩下的未編碼畫面數，包括當前要編碼這一張。

3.2.2 QP 的計算

在利用位元率控制演算法計算出當前畫面的編碼預算後，接下來，必須去計算這個編碼預算在兩種熵編碼模式的等效 QP，如圖 3.5 所示。而要計算出兩種熵編碼模式的 QP，必須使用 R-Q model。R-Q model 是描述壓縮演算法在選擇不同的 QP 去壓縮時所對應的編碼量的關係。

在[24]中所使用的 quadratic R-D model 在 R-Q domain 上可以表示為(3.6 式)。在(3.6 式)中，R 代表編碼後的資料量，而 a 和 b 為二次模型的兩個係數。而以(3.7 式)則可以從單張畫面的編碼預算 R 得出 QP。

$$R = \frac{a}{QP} + \frac{b}{QP^2} \tag{3.6}$$

$$QP = \frac{a \pm \sqrt{a^2 + 4bR}}{2R} \tag{3.7}$$

但是考慮(3.6 式)和(3.7 式)計算時和硬體實現上的複雜度，且二次曲線模型在以線性迴歸的方式更新模型係數時，需要比較多的計算量，而且針對兩種熵編碼模式都必須分別計算模型係數，所以在此功率的消耗必須倍增。而且在[28]、[29]中，都有提出使用線性模型去取代二次曲線模型的想法。而在[28]中，更提出在線性模型中，使用 Q_{step} domain 會比 QP domian 更為準確，所以我們打算使用 $R-Q_{step}$ 線性模型去計算。

$R-Q_{step}$ 線性模型如(3.8 式)所示。

$$R = \frac{a}{Q_{step}} + b \tag{3.8}$$

所以從單張畫面的編碼預算得出 Q_{step} 的方式如(3.9 式)所示。而兩種熵編碼模式將各自由自己的 $R-Q_{step}$ model 計算 Q_{step}。

$$Q_{step}=\frac{a}{R-b} \tag{3.9}$$

在 2.2.3 節中，已經介紹過 QP 和 Q_{step} 的關係。從(3.9 式)得出的 Q_{step} 可以直接以表 2.2 換算等效的 QP 值。QP 代表量化表的順序，從 0 開始到 51 結束，總共 52 個值，每個值對應一個 Q_{step}，也就是實際上的量化數，兩者間可以互換。從表 2.2 中可以看到，當 QP 小於 4 時，Q_{step} 將小於 1，而 Q_{step} 等於 1 時的壓縮量，已經近乎於無損壓縮的資料量，為了讓壓縮後的資料量不在上昇而增加功率消耗，而且此時與 PSNR 值的關係變成難以用線性模型描述，所以我們在使用上，不會用到 QP 小於 4 的四個值，所以有效的 QP 個數為 48。實際上在計算時，都是用 Q_{step} 去計算，不過因為 Q_{step} 的值為非線性變化，而 QP 值為線性變化，所以在描述內容和顯示結果時，都是用 QP 值表示所用的 Q_{step} 值。

我們定義在線性模型上以 QP 值 52 對應的資料量為 $R_{\max}$，而 QP 值 4 對應的資料量為 $R_{\min}$。$R_{\max}$ 和 $R_{\min}$ 是用來處理當編碼預算不在這兩個值之間時，即當編碼預算大於 $R_{\max}$ 或小於 $R_{\min}$，則所得出的 QP 值分別為 52 或 4。

3.2.3 PNSR 差值的計算

在我們的切換演算法中，定義兩種熵編碼模式壓縮完的 PNSR 要相差 0.5dB 以上才會進行模式的切換。在圖 3.5 中，PD 代表兩者壓縮後的 PNSR 差值。在前兩小節中，我們利用位元率控制演算法根據輸出頻寬計算出單張畫面編碼預算，然後再用 $R-Q_{step}$ 線性模型算出需要的 QP 值，而兩種熵編碼會根據自己的線性模型計算出對應的 QP 值。而這個 QP 值剛好可以拿來計算兩者壓縮後的 PNSR 差值。

在壓縮演算法中，QP 值會設計成和壓縮完的 PNSR 值成線性關係[30]，所以我們同樣只要用一個簡單的線性模型來描述，如(3.10 式)所示。而對同一張畫面來說，當使用的 QP 值固定，不管熵編碼使用的模式為何，都不會影響到壓縮後的 PNSR 值。所以在切換演算法中，只需要使用一個 PNSR-QP 模型。

$$PSNR = c \times QP + d \tag{3.10}$$

當兩種熵編碼算出的 QP 值分別為 QP_{bac} 和 QP_{vlc}，則兩者的差值可以(3.11 式)表示。從(3.11 式)可以得知當兩者 QP 值相差大於 1 時，只要當 a 值大於 0.5 時，即代表 PNSR 差值大於 0.5 dB。

$$PD = c \times (QP_{bac} - QP_{vlc}) \tag{3.11}$$

因為 CABAC 的壓縮表現比 CAVLC 優異，所以 CABAC 的 R-D 曲線會在 CAVLC 的上方，即使用固定 QP 值壓縮，畫面熵編碼模式採用 CABAC，壓縮後的 PNSR 值會大於等於 CAVLC 的 PNSR 值。所以(3.11 式)是用 $QP_{bac} - QP_{vlc}$ 表示兩者的 QP 差值。在

我們演算法的設計中，會在 initialize model 步驟中預設一個初始的熵編碼模式，因為 group 中的第一張畫面不做編碼模式切換，所以在第一個 group 的第一張畫面，即使用預設的熵編碼模式編碼。而之後除了每個 group 的第一張畫面，則都會進行編碼模式切換。當 PNSR 差值大於 0.5dB，表示 CABAC 在當前頻寬下，壓縮的畫質可以顯著提昇；而當 PNSR 差值小於 0.5 dB，為了節省時間，則將編碼模式保持在 CAVLC。

3.2.4 模型係數更新

圖 3.5 的流程中，在壓縮完每一張畫面後，要對三個線性模型進行係數更新。分別為兩種熵編碼分別的 $R-Q_{step}$ 模型和共同使用的 $PSNR-QP$ 模型。更新的方式是將過程中記錄的固定筆的資料用線性迴歸去計算模型係數[31]，以 $PSNR-QP$ 模型(3.10 式)為例，係數 a，b 更新的公式如(3.12 式)、(3.13 式)所示。P_i 為每張畫面壓縮時的 PSNR 值，而 QP_i 為量化使用的 QP 值，n 則為記錄的資料個數。

$$c=\frac{\sum P_iQP_i-\frac{\sum P_i\sum QP_i}{n}}{\sum QP_i^2-\frac{(\sum QP_i)^2}{n}} \tag{3.12}$$

$$d=\frac{1}{n}(\sum P_i-a\sum QP_i)=\overline{P}-aQ\overline{P} \tag{3.13}$$

記錄資料更新的方式則像是 FIFO 緩衝器的運作，如圖 3.7 所示。

圖 3.7 資料更新方式

以 $R-Q_{step}$ 模型為例，假設我們的系統存 n 筆資料，對熵編碼模式的線性模型來說，每一筆資料代表畫面序列實際壓縮後的(R, Q_{step})資料。在 initialize model 步驟中，即會把 n 筆資料放滿，而同時也計算出初始模型係數值。在壓縮的過程中，當壓縮完一張畫面後，新的(R, Q_{step})資料會存放在第一個位置，而第 n 個位置的資料會被第 n-1 個位置的資料覆蓋，所以第 n 筆的資料等於被拋棄。在資料更新完畢之後，即利用被記錄的這 n 筆資料去計算模型係數。我們預設的 n 為 8，為 2 倍 group 內包含的畫面張數。

3.2.5 線性模型的硬體實現分析

在前三小節中，我們在利用(3.9 式)計算 QP 和(3.12 式)更新係數時，都會需要浮點數的除法器，為了減低演算法以硬體實現的成本，所以我們使用查表的方式來取代除法的運算。在 $R-Q_{step}$ 模型的計算上，我們使用 R-QP model 記憶體來取代，如圖 3.8 所示。

兩種熵編碼都需要自己的 R-QP model 記憶體。以一種熵編碼來說明，我們的記憶體的位址為 52，代表 52 種 QP 值的變化，不過實際上只使用 48 種 QP 值。而每個位址會紀錄 N 筆的 R 值，N 為同一個 QP 值之下要紀錄的資料數，而 R 為畫面以該 QP 值實際壓縮的資料量。

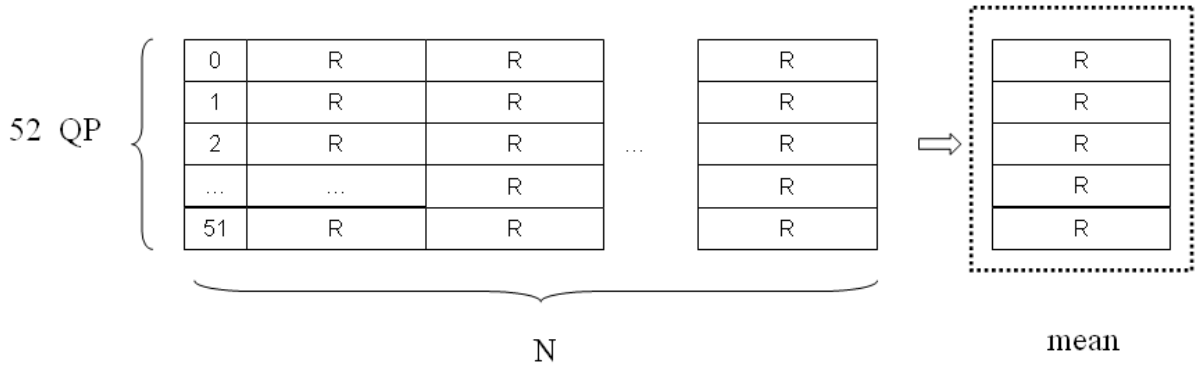


圖 3.8 R-QP model 記憶體

所以這個記憶體的使用方式為，在演算法實際執行前，我們先收集畫面在 48 種 QP 值壓縮下的 R 值以當作記憶體的預設值。而每次壓縮完，則將壓縮量 R 儲存到對應的 QP 位址，儲存的方式和圖 3.7 的方式相同，所以只會儲存最近的 N 筆資料。每次儲存完之後，會針對每個 QP 值內儲存的資料取平均。平均完的資料則用來為編碼預算選擇要使用的 QP 值。和 3.2.2 節相同，我們同樣要先計算畫面的編碼預算，在將編碼預算和平均後的 52 個資料做比較，以相差最少者對應的位址為選定的 QP 值，而兩種熵編碼同樣都以這個方法得出對應的 QP 值。

1	2	3
13.34%	10.58%	18.88%

表 3.1 記憶體所造成的切換誤差率

表 3.1 為使用 R-QP model 記憶體進行切換和實際以(3.9 式)計算兩者的模式切換誤差，可以發現在 3 個測試影像中所造成的平均誤差約 14%，從第四章的模擬結果來看，這個模式切換誤差是可以接受的，因為對整體的結果沒有造成很大的影響，而且可以省去使用浮點數除法器。當記憶體的 N 取 4，而畫面壓縮最大資料量為 1M (bits/frame)時，需要的記憶體大小為 2×48×4×20，約為 0.94 KB。但是因為在位元率控制中，畫面編碼預算的變動量約為±50%之間，所以以頻寬 250k bits/frame 為例，需要的記憶體大小為

2×10×4×20，約 0.2 KB。

而在 PSNR-QP 模型的計算上，我們使用 PSNR-QP table 來取代，如圖 3.9 所示。當要判斷 PD 有沒有大於 0.5 時，其實不需要把 c 的值很準確的算出來，而是只要知道 c 的值乘上多大的整數可以大於 0.5 即可，我們將這個整數定為 M，利用(3.14 式)可以計算 M 值，所以模式切換的判斷變成如(3.15 式)所示。

$$M = ceil(0.5/c) \tag{3.14}$$

$$\begin{aligned} &if\ M \le (QP_{bac}\text{-}QP_{vlc}) \\ &\quad \text{mode} = CABAC \\ &else \\ &\quad \text{mode} = CAVLC \end{aligned} \tag{3.15}$$

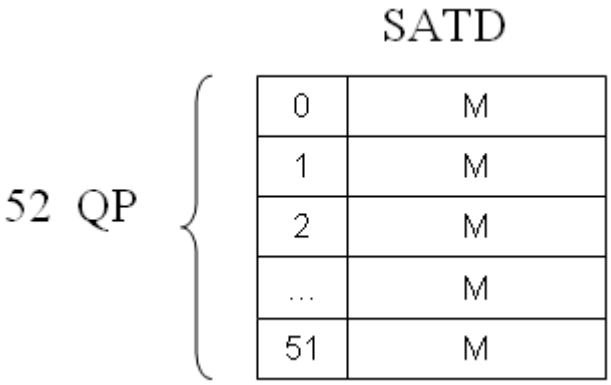


圖 3.9 PSNR-QP table

PSNR-QP table 的建立，是將圖片以預測後的 SATD 值去分類圖片。將 SATD 值分成數個區間，而同一個區間內的圖片則使用相同的 PSNR-QP table。PSNR-QP table 在每個區間中要為 52 個 QP 值建立預先計算好的 M 值。在這裡，M 的紀錄大小為 0~4。0 代表此處的等效 PSNR 已經大於 50 dB 以上，不進行模式的切換，而 M 值出現 4 以上的機率很低，所以 4 以上的情況都化簡以 4 表示。PSNR-QP table 的使用方式為利用畫

面實際壓縮時的 SATD 值去選擇對應的表格，在用兩種熵編碼對應的值去選擇 M 值，當選擇出的 M 不同時，以較小值為主，這樣做的理由是 M 值出現較小值的機率會比較大值高。

	4	8	16	32	64	128
SATD	0.1301	0.1271	0.1169	0.1141	0.103	0.104

表 3.2　SATD 分類區間數所造成的 M 值 MSE

表 3.2 為測試影像在不同 SATD 分類區間數下所造成的 M 值 MSE 變化。MSE 值的計算為同一 SATD 分類下單一畫面的 M 值和整個區間的平均值計算的結果。我們最後選擇的分類數為 16，因為分類數 8 到 16 是 MSE 減少最大的地方，而雖然分類數越大可以造成 MSE 值的下降，但是分類數越大將直接導致表格變大，且 MSE 值的下降趨勢漸趨於飽和。同樣在位元率控制中，QP 值的變動最大約為±10，所以實際的表格大小約為 16×20×3，約為 1Kb。

3.3 熵編碼的碼率預估演算法

在這節中，將介紹我們設計的兩種熵編碼位元率預估演算法。在每一小節的最後面將列出各種方法的預估 QP 誤差值和各種方法的速度。速度方面希望能盡量減少 CAVLC 的編碼和 CABAC 的碼率猜測兩者執行時間總合和 CABAC 執行編碼時間的比值，以真實達到節省時間的效果。我們在兩種熵編碼所用的碼率預估演算法基本上也是要先去統計出必要的編碼符號，再利用編碼符號實際上的值去換算出預估的壓縮位元率。

3.3.1 CAVLC 碼率預估演算法

在第 2.3.1 節中，已經介紹過 CAVLC 編碼需要的統計量。這些統計量中，除了編碼符號 level 之外，所有其他的編碼符號在壓縮時的編碼內容都可以透過查表而得。因此，除了 level 之外，所有其他的編碼符號在我們的猜測方法中，也都是透過查表的方式去得到編碼長度。而所有的表格，則依然參考[8]裡面所定義的表格。這樣做的好處是，在之前的設計中[15]，CAVLC 查表的部份原本就有紀錄編碼長度，所以如果這個碼率預估演算法要以硬體實現，不會有任何額外的硬體消耗。

而 level 的部份，我們必須從實際壓縮殘餘係數區塊得到 level 的值，去計算出實際的編碼長度為何。下面的表 3.1 是利用測試平台 jm11.0[18]實際編碼 level 的結果。使用的 level 編碼參數 suffix_length 值為 0 到 3，而 suffix_length 值的整個範圍為 0 到 6。

Level	suffix_length : 0
1	1
-1	01
2	001
-2	0001
3	00001
-3	000001
…	…
-7	00000000000001
±8 ~±15	000000000000001xxxx
≧ ±16	0000000000000001xxxxxxxxxxxx

Level	suffix_length : 1
1	10
-1	11
2	010
-2	011
3	0010
-3	0011
…	…
14	000000000000010
-14	000000000000011
15	0000000000000010
-15	0000000000000011
> ±15	0000000000000001xxxxxxxxxxxx

Level	suffix_length : 2
1	100
-1	101
2	110
-2	111
3	0100
-3	0101
…	…
29	00000000000000100
-29	00000000000000101
30	00000000000000110
-30	00000000000000111
> ±30	0000000000000001xxxxxxxxxxxx

Level	suffix_length : 3
1	1000
-1	1001
2	1010
-2	1011
3	1100
-3	1101
…	…
59	000000000000001100
-59	000000000000001101
60	000000000000001110
-60	000000000000001111
> ±60	0000000000000001xxxxxxxxxxxx

表 3.3 suffix_length 值為 0 到 4 對應的 level 編碼結果

從四個表格中可看到，除了 suffix_length 值等於 0 的情況之外，其他三個表格皆會在當 level 值大於某個門檻值之後，編碼長度即固定為 28，即表格中呈現灰色的部份。而在小於門檻值時，則編碼長度會和 level 絕對值呈正比。而 suffix_length 值等於 0 時，則具有兩個門檻值。表 3.4 列出不同 suffix_length 值之下，影響編碼值長度變動或固定的 level 門檻值，意即當 level 小於門檻值時，編碼長度屬於變動值。

suffix_length	level 門檻值
0	8
1	15
2	30
3	60
4	120
5	240
6	480

表 3.4　影響編碼長度的 level 門檻值

在[32]中提到，level 編碼的結果在 level 值小於門檻值時(即表格中白色部分的編碼)可以分成 prefix 和 suffix 兩個部份。prefix 部份可以相當於表 3.3 中每個碼字前面連續零的部分，而 suffix 部分為第一個 1 之後所包含的部份。這兩個部份的編碼長度都會受到 suffix_length 這個參數所影響。而當編碼的過程中，區塊中 level 的絕對值超過規定的門檻值，則 suffix_length 的值會增加。門檻值和 suffix_length 的關係[1]表示在表 3.5 中。

suffix_length	增加 suffix_length 的 level 門檻值
0	0
1	3
2	6
3	12
4	24
5	48
6	不存在

表 3.5 對應suffix_length變動的level門檻值 節錄自[1]

在[32]中也提出了 prefix 和 suffix 兩部分和 suffix_length 的關係。關係如(3.16 式)、(3.17 式)、(3.18 式)所示。(3.16 式)是把 level 值轉成以 level_index 表示，而 level_index 可以等效成在 suffix_length 表格中的位置。(3.17 式)中的 *prefix* 代表編碼結果連續零的個數，也就是 prefix 部分的長度。而(3.16 式)中的 *suffix* 為 suffix 部分扣掉第一個 1 之後的二進位值，其編碼長度恰好為 suffix_length。

$$level\_index = \begin{cases} 2\times(level-1) & level > 0 \\ -(2\times level)-1 & level < 0 \end{cases} \tag{3.16}$$

$$prefix = level\_index/(2^{suffix\_length}) \tag{3.17}$$

$$suffix = level\_index \,\%\, (2^{suffix\_length}) \tag{3.18}$$

但是我們在這裡只需要知道 level 編出來的編碼長度，綜合前面三式的結果，level 編碼長度可以用(3.19 式)、(3.20 式)表示。code_length 為計算出來的 level 編碼長度，*prefix* 和 *suffix* 為 3.17 式和 3.18 式所計算之值，*threshold* 為表 3.4 之門檻值。

當 suffix_length 值為 0:

$$code\_length = \begin{cases} prefix + suffix + 1 & abs(level) < 8 \\ 19 & 15 \geq abs(level) \geq 8 \\ 28 & 15 \leq abs(level) \end{cases} \tag{3.19}$$

為其他值時:

$$code\_length = \begin{cases} prefix + suffix + 1 & abs(level) \leq threshold \\ 28 & abs(level) > threshold \end{cases} \tag{3.20}$$

我們提出的方法為除了編碼符號 level 用(3.19 式)和(3.20 式)去計算編碼長度，而其它編碼符號的編碼長度皆使用直接查表的方法。下表 3.6 為提出的 CAVLC 碼率預估演算法和[21]的模擬比較。測試影像來源為北醫和中科院的內視鏡影像，給定的模擬單張壓縮預算範圍為 50k bits 到 1M bits，每次增加 50k bits，共 20 個值。

方法	CAVLC	Proposed	[21]
QP SAD	0	0	64.9
Error rate(%)	0	0	86
Time/frame(s)	25.38	11.24	2.56

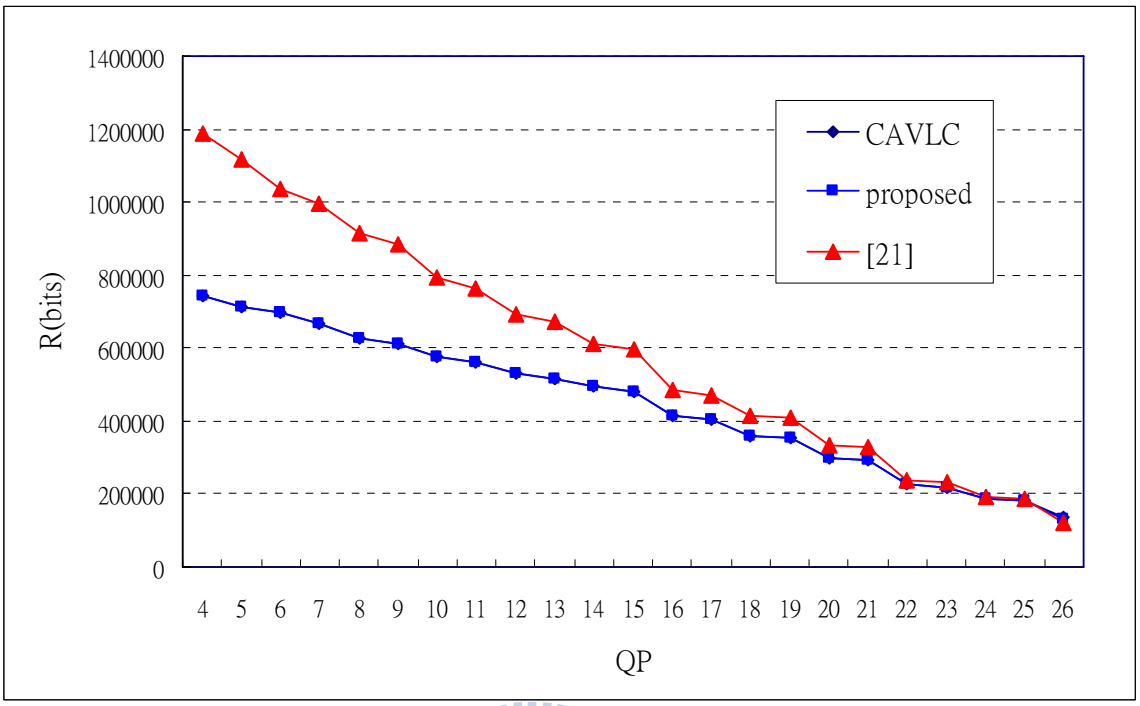
表 3.6　CAVLC 碼率預估演算法模擬比較

模擬的方式為將內視鏡影像以原始的 CAVLC 壓縮先求得在 48 個 QP 值的壓縮量，在以模擬的單張畫面壓縮預算去計算對應的 QP 值為何，而提出的預估演算法和[21]同樣使用相同的方法去反推 QP 值。而這個 QP 值即等效使用這個預估演算法的線性模型在單張畫面壓縮預算給定下會計算出的 QP 值。所以利用兩者和原始 CAVLC 壓縮的 QP 值差異就可以推論預估演算法準確度對線性模型預估準確性的影響。

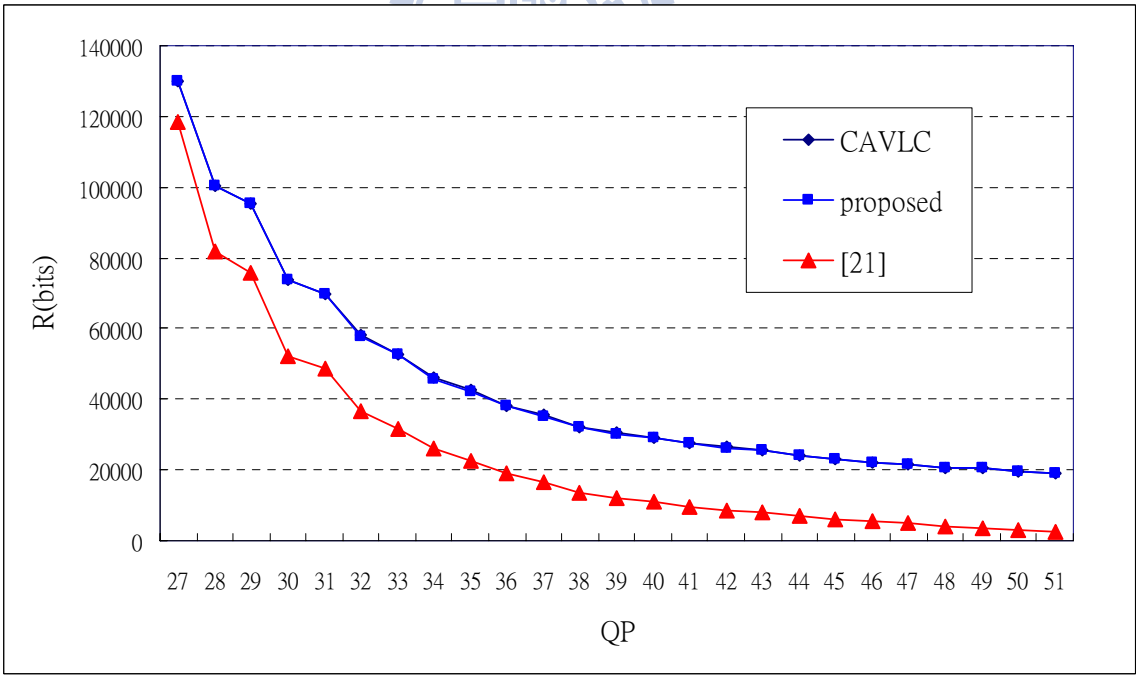
表 3.6 第二行的資料代表直接用 CAVLC 壓縮的結果，同時我們也利用這個結果當做標準，去計算提出的預估演算法和[21]兩者的 QP SAD 值和錯誤率，分別列於表中第二和第三列。QP SAD 值為兩種預估演算法和 CAVLC 壓縮所預估出的 QP 差值的絕對值總和而錯誤率為 20 個模擬值發生錯誤的機率。表中第四列的數據代表使用三種方式在取得單張圖片壓縮量的平均模擬時間。模擬平台是使用 MATLAB 7.6.0。從模擬結果的數據可以看到我們的預估演算法雖然不能像[21]比原始的壓縮大幅減低時間，但也平均減少了一半的時間以上，甚至約達1/3，但是能保證猜測出的 QP 值和直接用 CAVLC 編碼猜測的相同。而[21]的方式則產生很大的誤差。

圖 3.10 顯示其中一張測試影像以三種方法壓縮的位元率。圖 3.10 上半部顯示 QP 值 4 到 27，而下半部則顯示 QP 值 28 以後的結果。其中 CAVLC 為直接以固定 QP 值壓縮的結果，而提出的預估演算法和[21]則是直接以公式計算。因為預估演算法和直接以 CAVLC 壓縮的兩條線幾乎重疊，所以猜測的 QP 值才會完全相同。從圖中可以發現，[21]的猜測方法只有在 QP 值在 21 到 26 時才會和原始 CAVLC 的壓縮結果接近，其它的部份都具有相當的差距，所以其計算出的猜測 QP 值也都具有相當的誤差，而這就能解釋表 3.6 的模擬結果。雖然為了節省執行時間，在整體的熵編碼模式切換演算法設計中我們不使用 CAVLC 的碼率預估，但是從這個模擬結果，碼率預估演算法預估的編碼量必須儘可能接近原始的壓縮結果，才不會造成過大的猜測 QP 值誤差。

圖 3.10　CAVLC 編碼率預測的 R-QP 關係圖



(a)　QP from 4 to 26



(b)　QP from 27 to 51

3.3.2 CABAC 碼率預估演算法

在第二章中已經提過，CABAC 的完整步驟包含二元轉換，機率模型選擇，內容適應性二元算術編碼。我們所提出的 CABAC 碼率預估演算法，同樣也是要執行 CABAC 的前兩個步驟，而是改變了適應性二元算術編碼的流程，以達到減少執行時間並取得編碼量的目的。而針對適應性二元算術編碼採用的三種方式:encodeDecision、encodeBypass、encodeTerminate，則會修改成對應的預估演算法。發展演算法的目的主要是去預估一個 4×4 殘餘係數區塊經過 CABAC 壓縮後的資料量。同樣是要先計算需要的統計量，二元轉換後再進行每個符號位元的機率選擇，再以 CABAC 碼率預估演算法預估每個符號位元的編碼長度。在這一節中，前面將介紹我們自己發展的 CABAC 位元率預估演算法。最後一節則是各種方法的分析結果比較。

3.3.2.1 初步的 CABAC 碼率預估演算法

在預估 CABAC 位元率和預估 CAVLC 位元率在基本面上有一個很大的不同，也就是在預估一個 4×4 殘餘係數區塊的編碼量時，如果相鄰區塊的參考值已知，那 CAVLC 的壓縮結果只會和區塊內的係數排列有關。但是在預估一個 4×4 殘餘係數區塊使用 CABAC 壓縮的編碼量時，不只和相鄰區塊的參考值有關，還和 CABAC 內適應性二元算術編碼的傳遞量有關。這些傳遞量包括:Range、Low、bitsoutstanding。下面將使用一個例子說明在使用 CABAC 壓縮 4×4 殘餘係數區塊時，傳遞值會扮演的角色。

假設我們使用 CABAC 壓縮的殘餘係數區塊為:

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

則換算出的編碼符號依序為 coeff_block_flag : 1、significant_coeff_flag : {1,1} last_significant_coeff_flag : {0,1}，coeff_abs_level_minus1:{{0}， {0}}，而兩個非零係數的 coeff_sign_flag 皆為 0。後面的中括號的值即為預編碼的值。在此，我們只觀察 significant_coeff_flag 和 last_significant_coeff_flag 這兩個編碼符號。因為這兩個編碼符號不需要經過二元轉換，所以中括號內的值即為兩者的符號位元。我們可以將 significant_coeff_flag 和 last_significant_coeff_flag 這兩個編碼符號的符號位元進入二元算術編碼的過程以圖 3.11 表示。因為兩個編碼符號的符號位元編碼的順序如圖 2.13 所示將依序輪流，所以圖 3.11 中的符號位元(bin)依序為{1 0 1 1}。

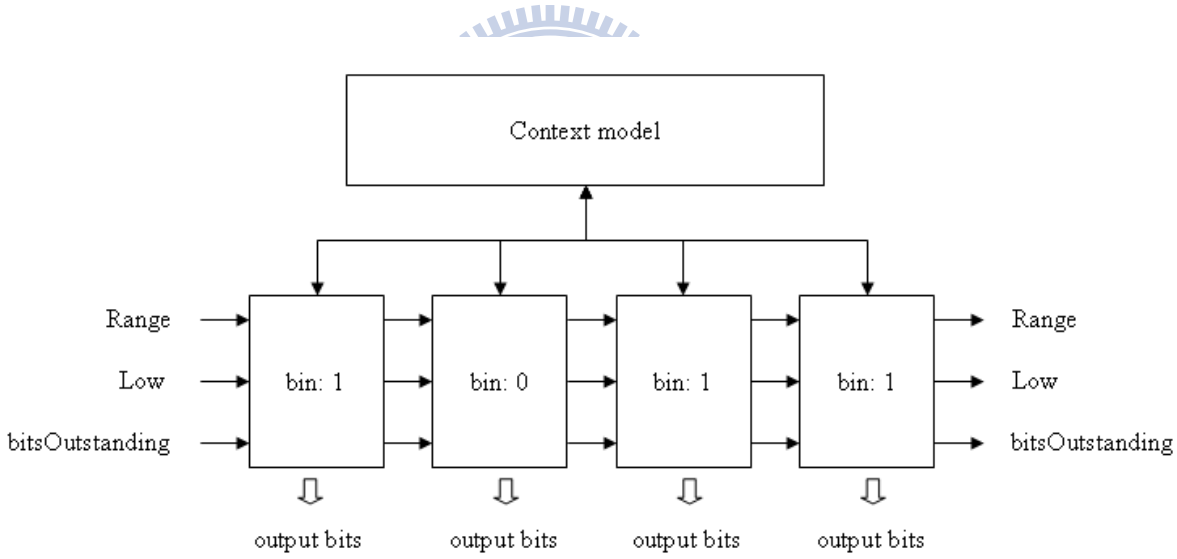


圖 3.11 符號位元(bin)編碼時的資料傳遞

圖 3.11 中每個包含 bin:x，$x \in \{0,1\}$ 的直立長方格代表進行一次適應性二元算術編碼，而 x 為當前的符號位元值，和上面橫向方格的箭頭則代表每個符號位元都要指定一個機率模型，並在編碼後進行模型的更新。在圖 3.11 中可以看到，每次進行二元算術編碼所需的傳遞量都相同，而這些傳遞量在編碼完每一個符號位元後都會更新，以傳給下一個符號位元使用，所以每一個符號位元產生的編碼結果是前後相關的。簡化圖 3.11

的過程，則單獨一個符號位元的適應性二元算術編碼可以用圖 3.12 表示，而進入符號位元和送出符號位元的傳遞量也用不同的名稱表示，以方便接下來的說明。在機率模型(context model)資訊已知之下，如果我們找到單獨符號位元接收的傳遞值和輸出編碼長度的關係，我們就可以去預估一個符號位元在固定傳遞值下的編碼長度。擴充到整個區塊的符號序列即可以去預估 4×4 殘餘係數區塊整個的編碼長度。當把圖 3.12 當成一個黑盒子，由傳遞值直接去預估當前符號位元的輸出位元長度，則輸出編碼長度的值可以用(3.21 式)表示。

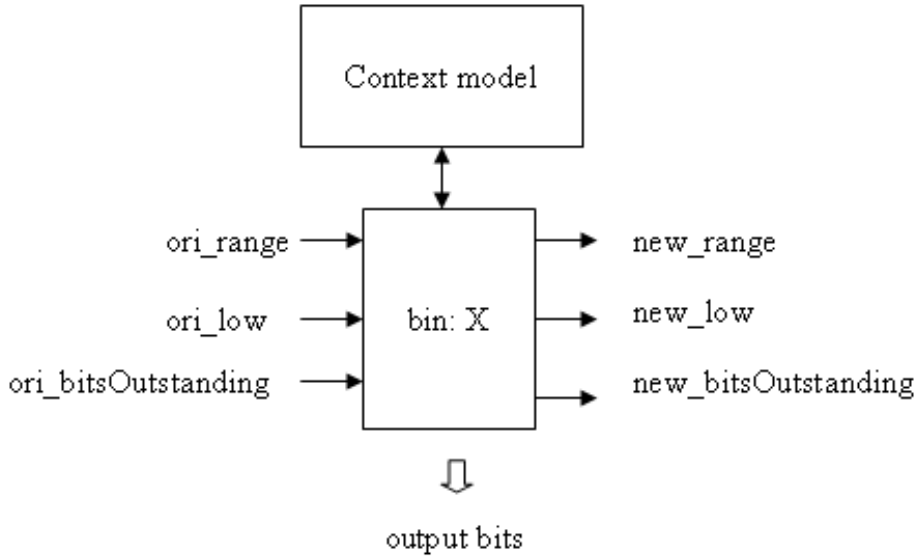


圖 3.12 單符號位元(bin) 於適應性二元算術編碼時的資料傳遞

$$bits\_length = (new\_bitsOutstanding - ori\_bitsOutstanding) + length(output\ bits) \qquad (3.21)$$

在(3.21 式)中，bits_length 表示這個符號位元等效的編碼長度貢獻，為真實輸出位元長度 *length(output bits)* 和 bitsOutstanding 的增加量之和。bitsOutstanding 的增加量為送出的 bitsOutstanding 值減去進入的 bitsOutstanding 值。bitsOutstanding 值雖然不一定為當前符號位元貢獻編碼長度，但是一定會為接下來後面的符號位元貢獻編碼長度。所以

該值的增加也代表這個符號位元等效的編碼長度貢獻。將傳遞量在二元算術編碼內的改變也考慮進來，則將圖 3.12 改成以圖 3.13 表示。圖 3.13 只展現二元算術編碼採用 encodeDecision 方式的符號位元。其它兩個方式將於後面介紹。

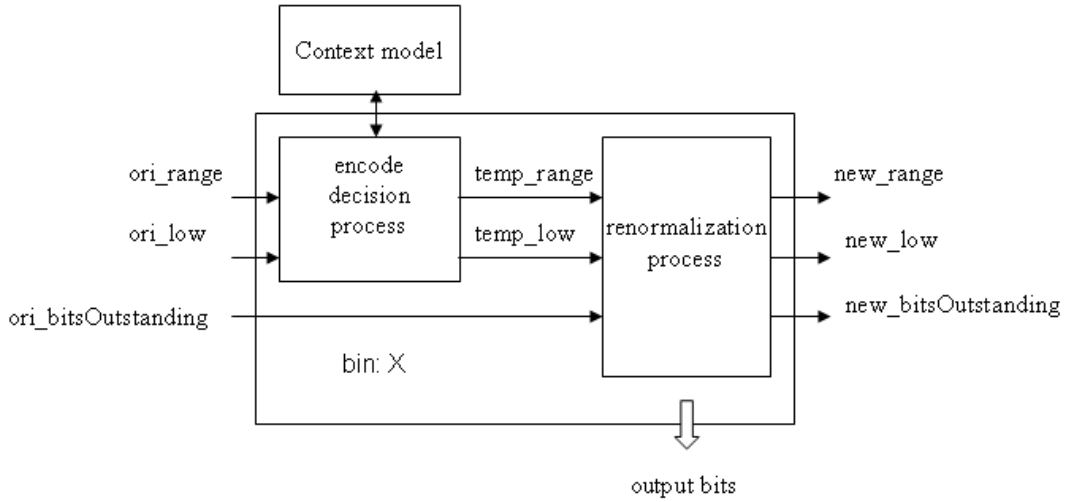


圖 3.13 單符號位元(bin) 於適應性二元算術編碼的內外部資料傳遞

在圖 3.13 中的 encode decision process 區塊為圖 2.8 流程圖裡面到 renormalization 前的部份，而後面的 renormalization process 區塊則為圖 2.9 流程圖的部分。圖 2.8 encodeDecision 流程圖中，針對符號位元的值和記錄在機率模型中的 MPS 相同與否去決定猜對或猜錯，不論猜對和猜錯，都會對 Range 和 Low 進行更新，更新值在圖 3.13 中以 temp_range 和 temp_low 表示。而在這個階段，還沒有任何編碼輸出。更新後的 temp_range 和 temp_low 值在進入 renormalization 步驟後，會根據此時 temp_range 的值去判斷要進行 renormalization 步驟幾次，直到讓 temp_range 大於等於 256 為止。將這個遞迴次數定義為 T，則上述關係可以如(3.22 式)表示。

$$T = ceil(\log_2 \frac{256}{temp\_range}) \qquad temp\_range < 256 \tag{3.22}$$

當 temp_range 小於 256 時，每次會去判斷 temp_low 的值落於不同的範圍之內，而對應輸出，每次的輸出不是真實輸出位元，就是讓 bitsOutstanding 加 1。所以次數 T 即為這個 temp_range 值對應的 bits_length，也就是這個符號位元在 temp_range 值時的編碼長度，如(3.23 式)所示。

$$bits\_length = ceil(\log_2 \frac{256}{temp\_range}) \qquad temp\_range < 256 \tag{3.23}$$

而要計算 temp_range 要乘以 2 幾次才能超過 256 的另一個方法，則將 temp_range 以二進位表示，再計算從 MSB 開始的第二個位元到第一個 1 發生經過幾個連續的零，連續零的個數即為 bits_length。

利用(3.23 式)，我們在二元算術編碼採用 encodeDecision 方式的估測演算法將變成如圖 3.14 所示。圖 3.14 為單符號位元的碼率預估，此時不需要再傳遞 bitsOutstanding 的值，而只要利用比對 MPS 之後，更新後的 temp_range 利用(3.23 式)去計算 bits_length 即可，跟原本直接以 CABAC 壓縮來比，這可以省去執行 renormalization 步驟的時間。而最後傳遞出的 new_low 值則按照 bits_length 值去計算。計算的過程在圖中以 prediction process 表示。將單符號位元(bin)的碼率預估延伸到整個區塊的符號序列，則我們就可以用這個方式來預估整個 4×4 殘餘係數區塊的編碼量。

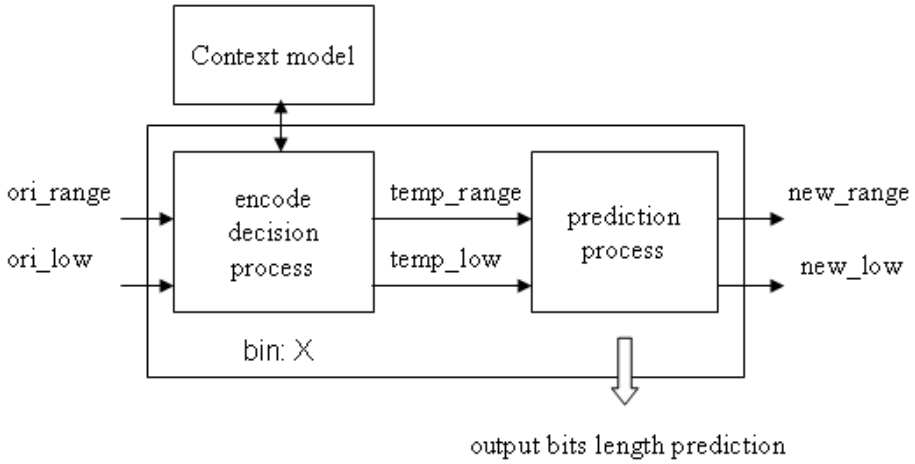


圖 3.14 單符號位元(bin) 的碼率估測

3.3.2.2 改良的 CABAC 碼率預估演算法

在前一小節中，我們發展出了基本的猜測方法。但是計算 Low 值的變化會額外花費時間。綜合前一節的結論和圖 2.10 的流程可以說 Range 值會影響到符號位元在二元算術編碼內的輸出編碼長度，而 Low 值會影響到輸出編碼內容。

將圖 3.14 中 Range 值和 low 值在 encode decision process 於猜對和猜錯兩種情況下的變化以下面的(3.24 式)到(3.27 式)表示。

猜對:

$$temp\_range = ori\_range - codIRangeLPS \tag{3.24}$$

$$temp\_low = ori\_low \tag{3.25}$$

猜錯:

$$temp\_range = codIRangeLPS \tag{3.26}$$

$$temp\_low = ori\_low + ori\_range - codIRangeLPS \tag{3.27}$$

從(3.24 式)到(3.27 式)可以歸納出，不論猜對和猜錯的情況下，Range 值在 encode

decision process 內的變化和 Low 無關，所以為了要加快猜測速度，我們將省去 Low 值的計算，而只計算 Range 值的變化。所以在我們的猜測方法中，將不會用到(3.25 式)和(3.27 式)。另外，不論猜對和猜錯的情況下，temp_range 值都和 *codIRangeLPS* 有關，而 *codIRangeLPS* 則由傳遞進來的 ori_range 值和機率模型內的 pstate 查表而得，如圖 2.9 第一個步驟。當猜錯時，temp_range 的值則等於 *codIRangeLPS* ，而 *codIRangeLPS* 為查表值，所以可以將 *codIRangeLPS* 的值先代入(3.23 式)，則查表同時也可得出猜錯時的 bits_length，而省去計算的時間，如表 3.7 所示。表 3.7 的數據為由[8]中的表 9-35 *codIRangeLPS* 表格計算而成。粗線所圍出的區域由上而下依序為 1 到 7。這個值即為在猜錯的情況下各種 *codIRangeLPS* 值對應的等效 bits_length 值。而猜對的情況，則只能在算出 temp_range 後，再使用(3.23 式)計算。

pstate	qCodIRangeIdx				pstate	qCodIRangeIdx			
	0	1	2	3		0	1	2	3
0	1	1	1	1	32	4	3	3	3
1	1	1	1	1	33	4	4	3	3
2	1	1	1	1	34	4	4	3	3
3	2	1	1	1	35	4	4	3	3
4	2	1	1	1	36	4	4	3	3
5	2	1	1	1	37	4	4	4	3
6	2	1	1	1	38	4	4	4	3
7	2	2	1	1	39	4	4	4	4
8	2	2	1	1	40	4	4	4	4
9	2	2	1	1	41	4	4	4	4
10	2	2	2	1	42	4	4	4	4
11	2	2	2	1	43	5	4	4	4
12	2	2	2	1	44	5	4	4	4
13	2	2	2	2	45	5	4	4	4
14	2	2	2	2	46	5	4	4	4
15	3	2	2	2	47	5	5	4	4
16	3	2	2	2	48	5	5	4	4

17	3	2	2	2	49	5	5	4	4
18	3	2	2	2	50	5	5	5	4
19	3	2	2	2	51	5	5	5	4
20	3	3	2	2	52	5	5	5	4
21	3	3	2	2	53	5	5	5	5
22	3	3	2	2	54	5	5	5	5
23	3	3	3	2	55	5	5	5	5
24	3	3	3	2	56	5	5	5	5
25	3	3	3	2	57	6	5	5	5
26	3	3	3	3	58	6	5	5	5
27	3	3	3	3	59	6	5	5	5
28	3	3	3	3	60	6	5	5	5
29	3	3	3	3	61	6	6	5	5
30	4	3	3	3	62	6	6	5	5
31	4	3	3	3	63	7	7	7	7

表 3.7 *codIRangeLPS* 值對應的 bits_length 表格

表 3.7 上面所列的 pstate 和 qCodIRangeIdx 兩個值是用來查詢 *codIRangeLPS* 值。我們將整理後的單符號位元的碼率估測以圖 3.15 表示。圖 3.15 的方法只要計算 Range 值的變化即可。

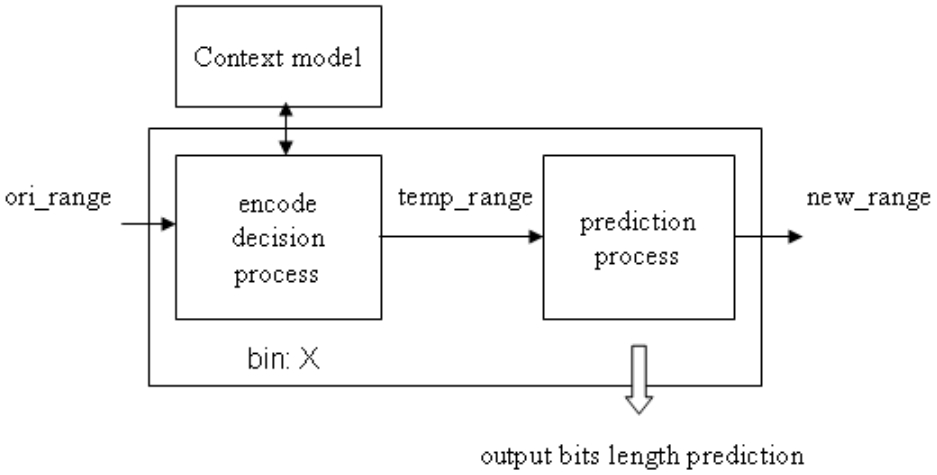
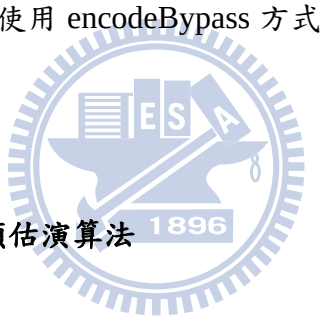


圖 3.15 改良的單符號位元(bin) 碼率估測

圖 3.15 中，ori_range 和 temp_range 值的關係如(3.26 式)和(3.26 式)所示。而 temp_range 和 new_range 的關係如(3.28 式)所示。式中 bit_length 由(3.23 式)計算而得。

$$new\_range = temp\_range \times 2^{bit\_length} \tag{3.28}$$

在 2.3.2.3 小節中，除了 encodeDecision 之外，適應性二元算術編碼還包括 encodeBypass 和 encodeTerminate 兩種方式，其中後者在我們的設計中，每張畫面只會發生一次，即在畫面中所有的 4×4 殘餘係數區塊都被編碼完之後，結束此張畫面的 CABAC 編碼。而 encodeTerminate 產生的編碼量很少，所以在整張畫面的碼率估測中不進行計算。而前者 encodeBypass 方式的估測必須參考圖 2.11 的流程圖。從圖 2.11 中可以發現，不論進入的 Range 值為何，Range 值最後都不會改變，而一個符號位元一定會對應出輸出編碼長度 1。所以使用 encodeBypass 方式編碼的符號序列，其輸出編碼長度即等於符號序列長度。



3.3.2.3 位元平行化碼率預估演算法

在前面的兩個小節中，我們利用了簡化過的適應性二元算術編碼步驟，實現了 CABAC 碼率預估演算法。但是這個方法的缺點為編碼符號的符號位元必須依序的進行預估，如圖 3.11 的順序，因為前一個符號位元更新後的 Range 要傳遞給下一個符號位元當作參考值。

如果要進行平行化地進行預估，即對同一個符號序列中所包含的符號位元都可以一起預估，必須打破位元間 Range 的傳遞方式。我們想到的方式是:每個符號位元收到的 Range 值不是從前一個符號位元計算而得，而是用一個預先定義好的值，這樣就可以同是對複數個符號位元進行碼率預估。這樣做的結果會讓預測的速度變快，但是會犧牲預估編碼率的準確度。

Range 值在符號位元間傳遞的合理範圍為 256 到 511，而在 H.264 [8]中，查詢 *codIRangeLPS* 表格使用的 Range 間隔為 64，所以我們以下面 5 個數值當做預先 Range 值的候選，分別為 256、320、384、448、510，彼此間的間隔為 64。因為最大值無法到 512，所以最大值設定為 510。接下來我們用這 5 個值，當作 Range 的預設值進行模擬。模擬的目的是為了去觀察使用固定的 Range 會對預估的編碼量造成什麼影響。同樣以內視鏡影像進行測試，影像大小為 512x480。平均單張圖片的壓縮量結果記錄在表 3.8 中。第一列顯示使用的 Range 值，CABAC 代表實際以 CABAC 壓縮，而後面的五個值即代表使用的 Range 預設值。第二列的數值是代表各種預設值下預估的平均單張畫面壓縮量，第一個值是實際壓縮編碼量，而後面的五個值為預估編碼量。而第三列的誤差率則是分別和第一行的數值計算。公式為: ((固定 Range 值預估編碼量-實際壓縮編碼量)/ 實際壓縮編碼量)×100%。

	CABAC	256	320	384	448	510
R (bits/frame)	258934	431507	300079	235168	189864	160919
誤差率(%)	0	66.65	15.89	-9.18	-26.67	-37.85

表 3.8 各種預設 Range 值的壓縮量模擬結果

從數據結果可以看到以 384 當作預設 Range 值時，和使用 CABAC 壓縮的編碼量最接近，所以使用 384 當作我們設計上的 Range 預設值。而 384 剛好也是 Range 傳遞範圍最大值和最小值的平均值。採用預設的 Range 值會改變每個符號位元碼率預估的流程。新流程以圖 3.16 表示。原本圖 3.15 中的 ori_range 將變成 384，而 temp_range 和 new_range 即不需要計算，因為不需要傳給下一級。並且不論猜對或猜錯，都可以直接從表 3.9 利用 pstate 查出等效的 bit_length 值。這個方法我們稱為位元平行化碼率預估演

算法。

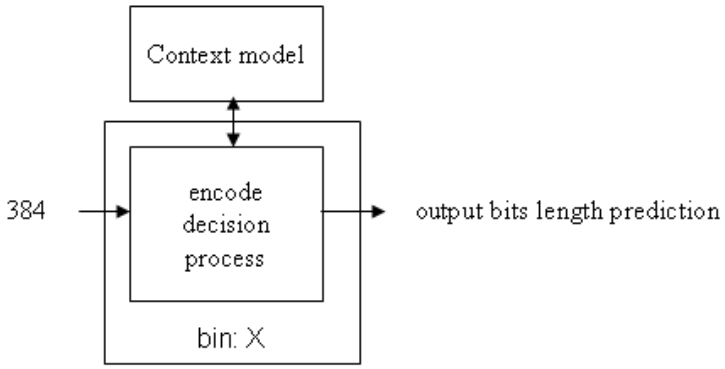


圖 3.16 單符號位元(bin) 的平行化碼率估測

表 3.9 猜錯部份的值是直接從表 3.7 中第三行複製過來的，因為 Range 值 384 對應的查表係數 qCodIRangeIdx 為 2，兩者換算的公式參考圖 2.9 中的第一個步驟。

猜錯的情況:

pStateIdx	0	1	2	3	4	5	6	7	8	9	10	11	12
bit_length	1	1	1	1	1	1	1	1	1	1	2	2	2
pStateIdx	13	14	15	16	17	18	19	20	21	22	23	24	25
bit_length	2	2	2	2	2	2	2	2	2	2	3	3	3
pStateIdx	26	27	28	29	30	31	32	33	34	35	36	37	38
bit_length	3	3	3	3	3	3	3	3	3	3	3	4	4
pStateIdx	39	40	41	42	43	44	45	46	47	48	49	50	51
bit_length	4	4	4	4	4	4	4	4	4	4	4	5	5
pStateIdx	52	53	54	55	56	57	58	59	60	61	62	63	
bit_length	5	5	5	5	5	5	5	5	5	5	5	7	

猜對的情況:

pStateIdx	0	1	2	3	4	5	6	7	8	9	10	…	63
bit_length	1	1	1	1	1	1	1	1	1	1	1	0	0

表 3.9 採用預設的 Range 值下的 bit_length 表

而表 3.9 中猜對部份的 bit_length 值計算如下。當符號位元的值和 MPS 不同，代表猜錯。根據(3.24 式)，此時的 temp_range = 384 – codIRangeLPS。 而 codIRangeLPS 的值是由 pstate 和 qCodIRangeIdx 查表可得，而 qCodIRangeIdx 的值此時為 2，所以將導致 temp_range 變成 pstate 的單變數函數(temp_range = f(pStateIdx))。舉例，當 pstate 等於 0 時，計算出的 temp_range 值為 176，再套用(3.24 式)可得 bit_length 值為 1。而只有在 pstate 小於 10 的值，temp_range 才會小於 256，其它的值，算出來的 temp_range 都大於 256，換算的 bit_length 即為零。

位元平行化碼率預估演算法的好處是同一個編碼符號下的所有的符號位元可以一起編碼，但是在更新機率模型時仍必須每個符號位元分開執行。

3.3.2.4 level 查表碼率預估演算法

在[23]中，對預估 CABAC 在 level 部分的編碼量，使用了更為簡單的方法。而這也給我們新的靈感。他們猜測 level 編碼量的方法如(2.13 式)、(2.14 式)、(2.15 式)所示。主要的原理是針對使用不同 QP 值壓縮的畫面給定 level 門檻值 T，當 level 值小於門檻值 T 時，則以 level 絕對值當作預估的編碼長度，當超過門檻值時，則使用前一張畫面該 level 值的平均編碼長度。要使用這個方法必須在前一張畫面在實際壓縮時，要去計算每個 level 值的編碼總和，以算出平均的編碼量。如果兩張畫面間的殘餘係數區塊中

非零係數值的分布差異性很小，這個方法可以快速地預估出編碼量。而實際上使用的缺點則是對一張畫面來說，在不同的 QP 值壓縮情形之下，並不知道 level 會出現的範圍有多大，意即無法去推估區塊中非零係數的最大值，而當出現的 level 值分佈範圍過大時，儲存編碼總和將消耗過多的記憶體。

考慮實際可行性，我們將這個方法的使用限制在 level 絕對值小於 14 的範圍。因為這個部份是 level 部分編碼量主要的貢獻來源。而這樣每張畫面只需要記憶 14 筆的資料。下圖 3.17 為一張內視鏡影像經不同 QP 值壓縮下 level 絕對值小於 14 的平均編碼量。橫軸為 level 絕對值，縱軸為每個 level 絕對值對應的平均編碼量 R，單位為 bits，下圖包含 4 個 QP 值所畫成的線和一條比對的線(m=1，表示斜率等於 1)。黑線的值可以看成直接用 level 絕對值當成編碼量的預估方式。可以發現到，即使在[23]所定義的 level 門檻值 5 之下，在不同 QP 下的平均編碼量仍然和絕對值(黑線)有一定差距。

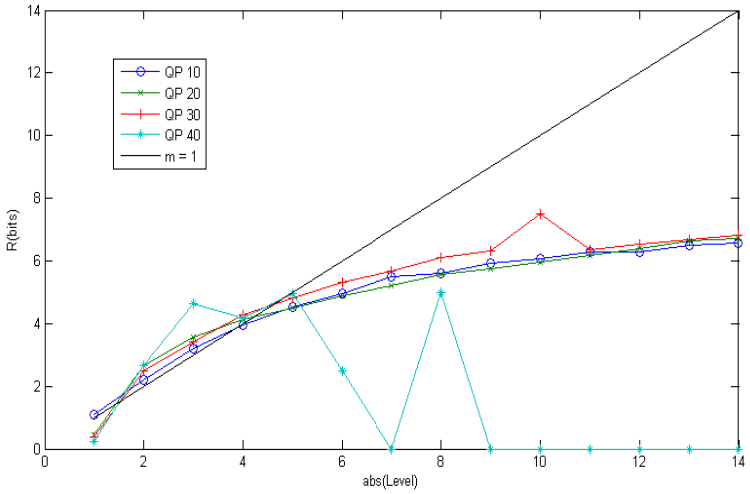


圖 3.17 不同 QP 值壓縮下 level 的平均編碼量

所以我們下面將模擬比較部份使用平均編碼量和全用平均編碼量兩種 level 預估方式和實際壓縮的誤差。這裡的平均編碼量表將採用前一張畫面產生。表 3.10 則紀錄兩種預估方式和實際壓縮在整張畫面下 level 部分的預估量和誤差率。這裡的預估量只有 level 絕對值小於 14 的部份，因為大於 14 的部份採用原始的編碼方式。從數據結果可

以看出當 level 絕對值小於 14 時，全用平均編碼量去預估會比部分使用編碼量和實際壓縮的結果比較接近。誤差率的計算公式為: ((預估編碼量-實際壓縮編碼量)/ 實際壓縮編碼量)×100%。

	CABAC	部分使用	全部使用
R (bits)	97023	107664	93703
Error rate(%)	0	10.97	-3.42

表 3.10　level 的平均預估編碼量

3.3.2.5 各種方法的優劣比較

在這一小節中，將比較 CABAC 各種碼率預測演算法的預測時間和預測 QP 值誤差。和 3.3.1 小節模擬的方式類似，我們將以實際 CABAC 的壓縮結果和計算的 QP 值當做計算標準。我們將 3.3.2.1 和 3.3.2.2 提出的基本碼率預估方法稱為 CABAC_pre，而將位元平行化碼率預估演算法稱為 CABAC_pp。下表 3.11 為提出的兩種 CABAC 位元率估測法和[22]、[23]的模擬比較。測試圖片和條件和 CAVLC 相同。

	CABAC	CABAC_pre	CABAC_pp	[22]	[23]
R (bits/frame)	363818	363816	334554	320781	411472
QP SAD	0	0	20.8	21.2	38.5
QP Error rate(%)	0	0	63	62.5	81
Time/frame(s)	37.04	11.5	6.95	4.29	6.66(2.06)

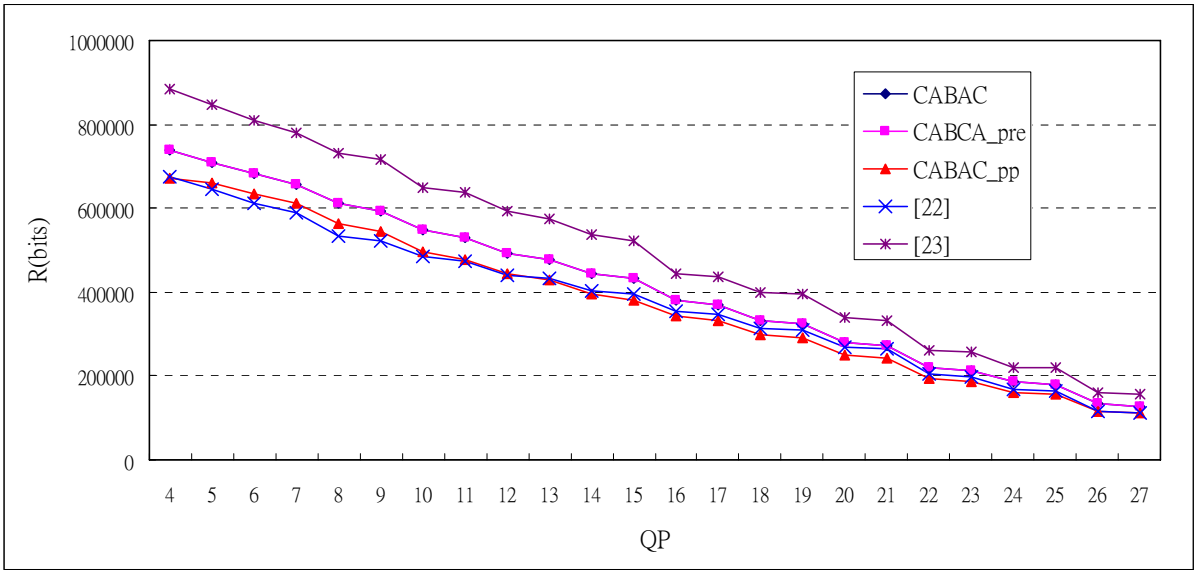
表 3.11　CABAC 碼率預估演算法模擬比較

[23]的測試時間有兩個值，因為演算法所要用的 level 平均壓縮量需要在前一張畫面先算好，所以在模擬時，前一張畫面是採用 CABAC_pre 的方式去猜測，而這一張畫面才能直接用查表的方式去預估。所以括號值為 level 表格已經存在之下，猜測單張畫面編碼量所需的時間。另一個值則是括號值和前一張預估時間的平均，代表實際猜測這張畫面的時間。

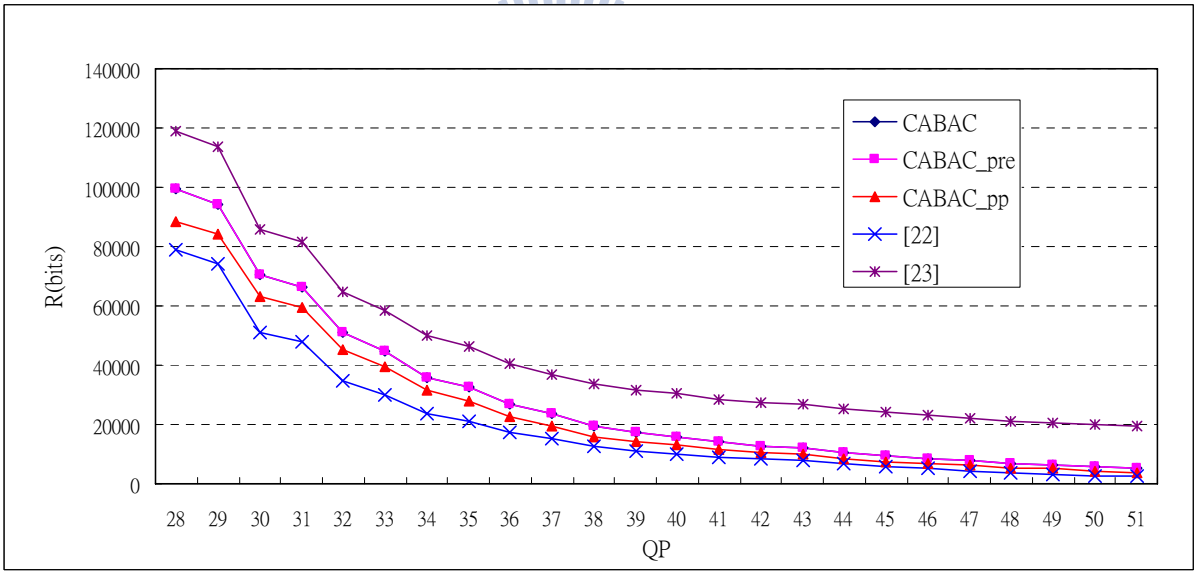
從測試結果來看，可以發現到雖然平均單張預估編碼量(R)在各種方法下的變動量不大，但是卻會對 QP 的 SAD 值和錯誤率造成很大的影響。所以[22]和[23]的方法雖然估計的時間很快，但是造成的誤差卻很大。而我們所發展的位元平行化碼率預估演算法雖然可以再減少猜測時間約達基本碼率預估演算法時間的一半，但是在 20 個單張壓縮預算值模擬下的平均 QP 估計誤差卻高達 63%，和[22]的誤差值很接近。從數據結果來看，後三種方法的皆會造成預估 QP 的誤差太高，誤差率都超過 50%，這代表在一半的壓縮預算值下，都會造成誤差，而基本碼率預估演算法猜測的結果卻能和直接壓縮極為接近，且同時節省約三分之二的時間。所以基本碼率預估演算法是最好的選擇。

圖 3.18 同樣為一張畫面以各種 CABAC 預估演算法模擬的 R-QP 關係圖，上半部顯示 QP 值 4 到 27，而下半部則顯示 QP 值 28 以後的結果。從圖中可以看出不論在任何 QP 值，[23]的方法和其他方法都有相當的誤差，所以造成高達 81%的 QP 值猜測誤差。而 CABAC 和 CABAC_pre 兩條線近乎重合。而 CABAC_pp 在 QP 值 12 到 25 的範圍，預估的編碼量的差距比[22]還要大，其他的部份，尤其在高 QP 值，差距才會比較小。

圖 3.18　CABAC 編碼率預測的 R-QP 關係圖



(a)　QP from 4 to 27



(b)　QP from 28 to 51

接下來模擬 CABAC_pre 和 CABAC_pp 兩個演算法搭費使用 level table 的預估效果。即兩者在猜測 level 部分的編碼量時，直接使用前一張統計的 level table 去猜測。表 3.12 為模擬結果。從結果來看 CABAC_pre 搭配 level table 後會造成微量的編碼預估誤差，但是時間上的改變也很微量，比沒使用時平均快了約 1 秒而已，這代表預估主要的

時間消耗並不在 level 部分，而是在編碼符號 significant_coeff_flag 和 last_significant_coeff_flag 上面。

而 CABAC_pp 的部份則是編碼預估誤差減少了約 10K bits，但是平均預估時間卻反而增加約 1 秒，而誤差率依然大於 50%，表示除了 level 外的統計量仍然貢獻很大的誤差。CABAC_pre 搭配 level table 後產生了約 5k bits，但是 QP 預估錯誤率則升高至 13.5%。但是實現上，除了前一張畫面只能使用 CABAC_pre 進行預估，另外要付出的代價為儲存空間和至少 14 次的除法的運算。

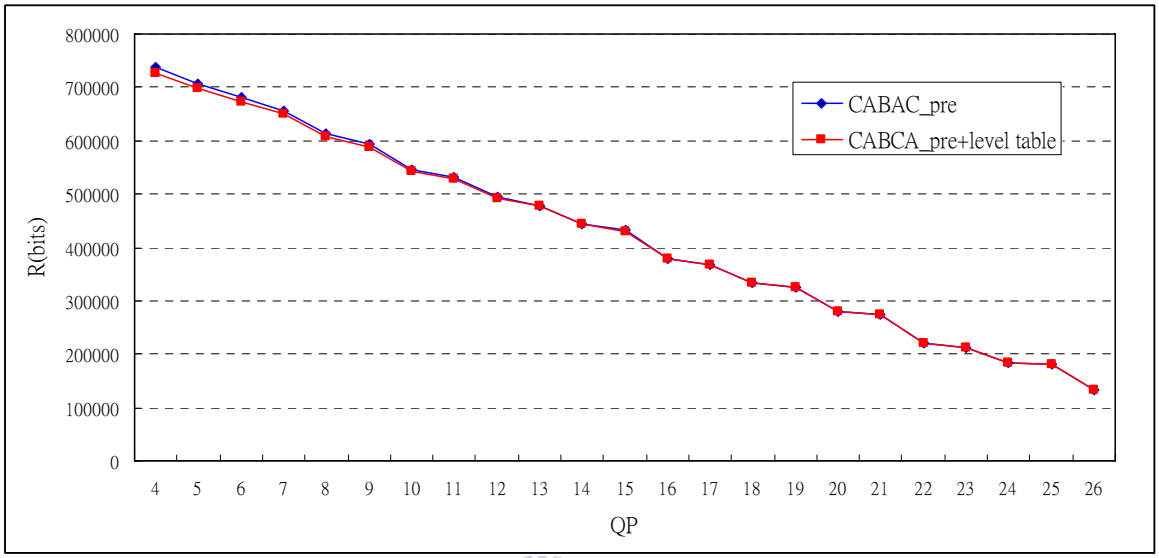
	CABAC	CABAC_pre	CABAC_pre (level table)	CABAC_pp (level table)
R (bits/frame)	363818	363816	358682	342404
QP SAD	0	0	2.1	15.3
QP error rate(%)	0	0	13.5	52.5
Time/frame (s)	37.04	11.5	10.07(8.42)	8.5(5.04)

表 3.12　使用 level table 的模擬結果

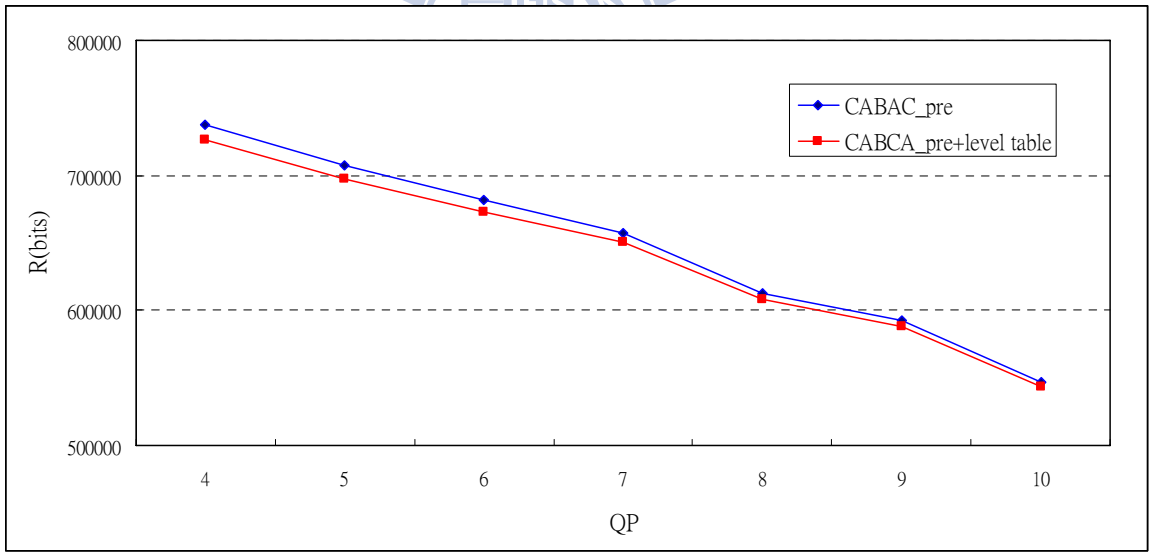
圖 3.19 為 CABAC_pre 使用 level table 所造成的誤差模擬，下面的圖是把上面的圖中 low QP 的部份放大來看，綜合兩者的趨勢，可以發現 level table 在 low QP 的誤差會比較大，而誤差則隨 QP 上升而遞減，因為畫面以 low QP 壓縮會導致區塊中的非零係數具有比較高的絕對值，這也會讓畫面間的係數絕對值分布發生較大的改變，所以當 QP 上升時，前一張畫面產生的 level table 才會較為準確。在 low QP 所產生的預估編碼量誤差則是導致表 3.11 模擬結果的原因。所以當使用的畫面頻寬較小時，才比較適合

搭配 level table 使用，因為較小的畫面頻寬代表使用較大的 QP 值量化。

圖 3.19 CABAC_pre 使用 level table 所造成的誤差



(a) QP from 4 to 26



(b) QP from 4 to 10

第四章 熵編碼模式切換演算法模擬結果

4.1 演算法模擬結果

在這一章中，我們將使用中科院和北醫的膠囊內視鏡影像來進行熵編碼模式切換演算法的模擬。測試的影像序列規格如表 4.1 所示。而在表 4.2 中，我們則分成三個畫面頻寬來進行模擬，單張畫面壓縮量分別為 500k、250k、125k (bits)，每秒壓縮畫面為 2(frames/s) ，group 的畫面張數為 4，而更新 model 用的紀錄資料筆數為 8。

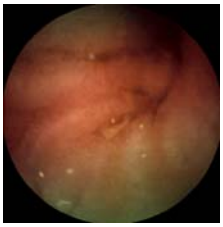
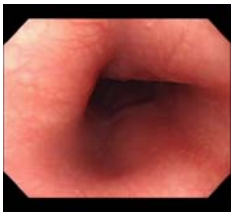
Pattern		
1	2	3
		
512×512 (160 frames)	512×480 (400 frames)	256×256 (400 frames)

表 4.1 測試影像

		PSNR (dB/frame)			Time (s/frame)			Probability of CABAC(%)
		CABAC	CAVLC	dual	CABAC	CAVLC	dual	
500k	1	46.4	45.08	46.26	47	31.69	45.31	78.75
	2	48.18	45.82	47.74	50.25	30.82	48.79	84.25
	3	44.69	44.16	44.64	13.09	6.8	10.8	65.75
250k	1	39.67	39.18	39.59	43.9	24.46	36.11	68.75
	2	41.46	40.79	41.38	46.58	23.03	38.81	51.56
	3	38.43	37.8	39.39	12.23	6.56	10.3	78.5
125k	1	36.48	36.35	36.38	12.87	8.93	9.33	18.75
	2	38.23	37.86	38.07	13.46	9	10	34.25
	3	32.99	32.06	32.94	3.1	2.57	2.94	89.25

表 4.2 熵編碼模式切換演算法的模擬

表 4.2 中，紀錄三個測試影使用熵編碼模式切換演算法和原始兩種熵編碼壓縮的模擬結果。右邊的三個數值分別是模擬的頻寬，和對應的三個測試影像。上面列的三項分別為平均單張畫面 PSNR 值，平均單張畫面執行時間，和選擇 CABAC 模式編碼的機率。前兩項分別記錄只使用 CABAC，只使用 CAVLC 和模式切換的模擬結果。選擇 CABAC 模式編碼的機率為扣掉每個 group 的第一張畫面後，剩下畫面根據模式切換演算法切換到 CABAC 模式的機率。

因為測試影像 3 的圖片尺寸為前兩者的四分之一，為了模擬出實際頻寬的影響，所以將實際模擬位元率降低成約四分之一，所以測試影像 3 模擬時用的單張畫面壓縮量分別為 125k、60k、30k (bits)。從表 4.2 的模擬結果來看，在三個模擬畫面頻寬之下，平均每次以模式切換演算法壓縮在單張畫面下比單用 CAVLC 壓縮有接近 0.7 dB 的畫質改善，這代表演算法能提供顯著的畫質改善。而除了切換率低的情況之下，模式切換演算法皆相當接近 CABAC 的表現。而平均切換率 63%也代表切換演算法確實有用，而切換

率沒有出現接近 0%或是 100%的情況也代表只用一種熵編碼在某些畫面確實會有缺陷。CABAC 模式壓縮而除了測試影像 3 之外，測試影像 1 和 2 隨著畫面頻寬的降低，切換到 CABAC 的機率同樣隨之下降；而測試影像 3 恰好相反，說明這個切換趨勢可能和內容有關。在模擬頻寬 125k (bits) 之下，測試影像 1 和 2 的切換機率分別為 18.75%和 34.25%，遠低於平均值，這是因為這兩個測試影像在使用原始熵編碼的畫質差已經很接近，分別為 0.13dB 和 0.37dB，都小於 0.5dB，所以這也証明我們的模式切換演算法都能正確地選擇 CAVLC 為壓縮的熵編碼模式。

從第二大項的模擬時間也可以看出我們的演算法在執行時間上的節省，因為紀錄的數值為平均當張畫面的執行時間，所以乘以整段影像的畫面數將可以有顯著的時間節省。和 CABAC 多於 CAVLC 的時間相比較，使用我們的模式切換演算法平均每張畫面可以節省約 40%的時間，所以執行時間約為單用 CABAC 和 CAVLC 的一半。

4.2 硬體成本模擬結果

在 3.3.2 節中，我們發展出 CABAC 基本的碼率預估演算法。而這個預估演算法除了預估準確度的優點之外，在以硬體實現時可以完全使用 CABAC 的硬體。表 4.3 為 CABAC_pre 和 CABAC 以 90nm 實現的邏輯閘數比較，合成的週期為 40ns。CABAC_pre 和 CABAC 只需要實現編碼殘餘係數區塊的功能，所以機率模型使用的記憶體很少，所以合成出來的邏輯閘數很小。在第二列中，為單獨實現 CABAC 功能的邏輯閘數，而在第三列中則是將 CABAC_pre 整合在 CABAC 中需要的邏輯閘數，從兩者的邏輯閘個數差可以知道要以硬體實現我們所發展的 CABAC 基本的碼率預估演算法幾乎可以完全使用 CABAC 原有的功能，而不會額外消耗太多的硬體成本。

function	Gate count
CABAC	9378
CABAC_pre+CABAC	9655

表 4.3 邏輯閘數統計

第五章　結論

在這篇論文中，我們為輕量化的視訊壓縮提出了熵編碼模式切換的演算法，在兩種熵編碼模式都可以使用的前提之下，利用我們的演算法可以有效地在兩種熵編碼模式間進行切換。當畫質可以明顯提升時，才使用 CABAC 進行熵編碼，在我們的模擬中，平均可以比單使用 CAVLC 進行壓縮，單張畫面的畫質可以提升 0.7 dB。而當 CABAC 無法使畫質明顯提升時，即使用 CAVLC 進行熵編碼的方式則讓我們的方法平均在單張畫面下可以比 CABAC 節省約 40%的時間。而平均切換率 63%也代表使用熵編碼模式切換演算法能比單用一種熵編碼壓縮能更適應性針對畫面去調整以得到畫質提昇，且使用熵編碼模式切換演算法的同時還可以進行壓縮系統的位元率控制。

在使用演算法切換的過程中，每個 group 的第一張畫面，因為必須同時執行 CAVLC 編碼和 CABAC 碼率預估演算法，從 3.3.1 和 3.3.2 的平均模擬時間可以知道兩者的執行時間總和會略小於以 CABAC 編碼，所以在影像的整體使用上，單張畫面需要最多執行時間為使用 CABAC 編碼的畫面，而和全使用 CABAC 編碼比較起來，切換到 CAVLC 則能有效節省時間。而 CABAC 碼率預估演算法近乎能完全使用 CABAC 的硬體，所以使用 CABAC 碼率預估演算法在硬體實現上不會額外消耗太多的硬體成本。

參考文獻

[1] E.G. Richardson, “H.264 and MPEG-4 Video Compression,” Chichester: John Wiley & Sons, 2003.

[2] V. Bhaskaran, K. Konstantinides, “Image and Video Compression Standards: Algorithms and Architectures,” Boston: Kluwer, 1997.

[3] R. C. Gonzales, R. E. Woods, “Digital Image Processing,” Upper Saddle River, New Jersey: Prentice Hall, 2002.

[4] N. Ahmed, T. Natarajan, and K. R. Rao, “Discrete cosine transform,” IEEE Transactions on Computers, vol. 23, pp. 90-93, Jan. 1974.

[5] Y. Wang, J. Ostermann, Y. Q. Zhang, “Video processing and communications,” Upper Saddle River, New Jersey: Prentice Hall, 2002.

[6] T. Wedi and H. G. Musmann, “Motion- and aliasing-compensated prediction for hybrid video coding, ” IEEE Transactions on Circuit and System for Video Technology, vol.13, no.7, pp.577-586, 2003.

[7] T. Wiegand, G. J. Sullivan, G. Bjontegaard and A. Luthra“Overview of the H.264/AVC video coding standard,” IEEE Transactions on Circuits and Systems for Video Technology, vol.13, no.7, pp: 560-576, Jul. 2003.

[8] Draft ITU-T Recommendation and Final Draft Intenational Standard of Joint Video Specification (ITU-T Rec.H.264|ISO/IEC 14496-10 AVC), Jul. 2003.

[9] Y. W. Huang, B. Y. Hsieh, T. C. Chen, and L. G. Chen, “Analysis, fast algorithm, and VLSI architecture design for H.264/AVC intra frame coder,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 15, no. 3, pp.378–401, Mar. 2005.

[10] F. Gong. P. Swain, and T. Mills, “Wireless endoscopy,” Gastrointestinal Endoscopy,vol.51, no. 6, pp. 725-729, Jun. 2000.

[11] H. J.Park, H.W. Nam, B.S. Song, J.L. Choi, H.C. Choi, J.C. Park, M.N. Kim, J.T. Lee,and J.H. Cho, “Design of bi-directional and multi-channel miniaturized telemetry module for wireless endoscopy,” in the 2nd Annual Int’l IEEE-EMBS Special Topic Conference on Microtechnologies in Medicine and Biology, May 2-4, 2002, Madison, USA, pp. 273-276.

[12] http://www.givenimaging.com/Cultures/en-US/given/english

[13] http://www.rfsystemlab.com/

[14] Meng Chun Lin, Lan Rong Dung and Ping Kuo Weng, “A Ultra-low-power Image Compressor for Capsule Endoscope,” BioMedical Engineering OnLine, vol.5, pp.14.1-14.8, Feb. 2006.

[15] H.C. Lai, “A Modified H.264 Intra-frame Encoder for Gastrointestinal Image,” M.S. thesis, Dept. ECE, NCTU Univ., Hsinchu, Taiwan, 2007.

[16] A.Puri, X. Chen, and A. Luthra, “Video coding using the H.264/MPEG-4 AVC compression standard,” Signal Processing: Image Communication, vol. 19, pp. 793–849, 2004.

[17] D.Marpe, H. Schwarz, and T. Wiegand, “Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard,” IEEE Transactions on Circuits and Systems for Video Technology, vol.13, no.7, pp: 620-636, Jul. 2003.

[18] JVT H.264/AVC Reference Software version JM 11.0,http://iphome.hhi.de/suehring/tml/download/old_jm/ jm11.0.zip

[19] A.Hallapuro, M. Karczewicz, and H. Malvar, “Low Complexity Transform and Quantization – Part I: basic implementation,” at the 2nd JVT Meeting, JVT- B038, Geneva, CH, Feb. 2002.

[20] G. Bjontegaard and K. Lillevold, “Context-Adaptive VLC Coding of Coefficients”, at the 3rd JVT Meeting, JVT- C028, Fairfax, Virginia, May 2002.

[21] M.G. Sarwer, L.M. Po “Fast Bit Rate Estimation for Mode Decision of H.264/AVC,” IEEE Transactions on Circuits and Systems for Video Technology, vol.17, No.10, pp.1402-1407, Oct. 2007.

[22] S.M. Chen, S.W. Sun “Efficient Bit-Rate Estimation Technique for CABAC,” IEEE Asia Pacific Conference on Circuits and Systems, pp.514-517, Nov. 30 2008-Dec. 3 2008

[23] L. Liu, X.H. Zhuang, “CABAC Based Bit Estimation for Fast H.264 RD Optimization Decision” IEEE Conference on Consumer Communications and Networking, pp.1-5, Jan. 2009.

[24] Z. G. Li, F. Pan, K. P. Lim, G. N. Feng, X. Lin, and S. Rahardja, “Adaptive basic unit layer rate control for JVT,” JVT-G012-r1, in the 7th Meeting, Pattaya II, Bangkok,Thailand, Mar. 2003.

[25] C.T. Chen, “Linear System Theory and Design,” New York: Holt, Rinehart, and Winston, 1984.

[26] H.J.Lee, T.H.Chiang, Y.Q.Zhang, “Scalable Rate Control for MPEG-4 Video,”IEEE Transactions on Circuits and Systems for Video Technology, Vol. 10, No.10, pp.878 – 894, Sep. 2000.

[27] A.Vetro, H.Sun, Y.Wang, “MPEG-4 rate control for multiple video objects,” IEEE Transactions on Circuits and Systems for Video Technology, Vol. 9, No.1,pp.186 – 199, Feb. 1999.

[28] S. Ma, W. Gao, and Y. Lu, “Rate-Distortion Analysis for H.264/AVC Video Coding and its Application to Rate Control,” IEEE Transactions on Circuits and Systems for Video Technology, Vol. 15, No.12, pp.1533 – 1544, Dec. 2005.

[29] D.K. Kwon, M.Y. Shen, and C.C.J. Kuo, “Rate Control for H.264 Video with Enhanced Rate and Distortion models,” IEEE Transactions on Circuits and Systems for Video Technology, Vol. 17, No. 5, pp.517–529, May 2007.

[30] L. Zhuo, X. J.Gao, Z. Y. Wang, D.D. Feng, and L. S. Shen, “A Novel Rate-Quality Model based H.264/AVC Frame Layer Rate Control Method,” IEEE International Conference on Information, Communications and Signal Processing , pp.1-5, Dec. 2007.

[31] T.H.Chiang, Y.Q.Zhang, “A New Rate Control Scheme Using Quadratic Rate Distortion Model,” IEEE Transactions on Circuits and Systems for Video Technology, Vol. 7, No.1,pp.246 – 250, Feb. 1997.

[32] CAVLC in H.264, http://140.113.13.90/material/multimedia%20com/CAVLC.pdf