第一章 影像壓縮概論

預測編碼（Prediction Coding）、轉換編碼（Transform Coding）、以及熵編碼（Entropy Coding）是進行動態影像壓縮的三個主要步驟，見圖 1-1：

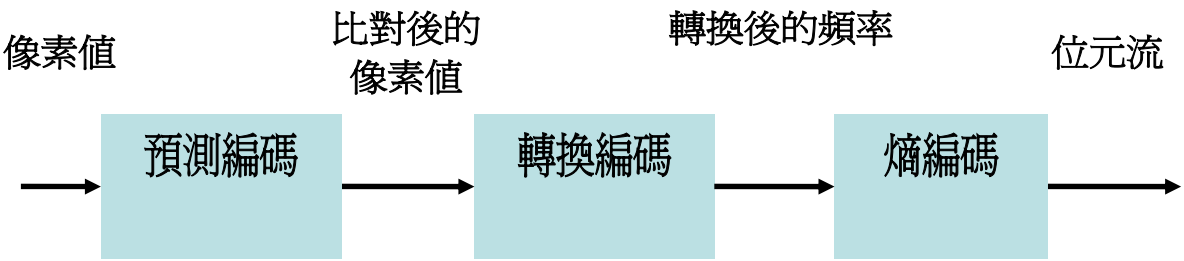

圖 1-1 影像壓縮中的預測編碼，轉換編碼與熵編碼

不論是預測編碼、轉換編碼、或熵編碼，其理論或作法都是降低影像資料之間的相關性。預測編碼尋找並消除影像區塊（Macroblock）與區塊之間的相似處，所以預測編碼就是在減少區塊之間的相關性。轉換編碼將影像區塊裡的像素大小轉換成頻率，利用轉換基底的特性將能量集中在特定頻率。如此一來編譯器可以專注處理特定的頻率，至於不重要的頻率就可以忽略。所以我們可以說，轉換編碼就是在減少像素間的相關性。熵編碼則是使用數學理論，找出影像的統計特性，用短的位元長度來描述經常出現的標記（Symbol），以達到減少整體位元總數的目的。

本章介紹影像的組成以及階層式編碼（Layer Coding）的觀念，並且以導論的方式概述動態影像編碼的三個主要步驟。在 1.2 節我們介紹 8x8 離散餘弦轉換以及 H.264 的核心轉換（Core Transform），並且介紹 H.264 的量化方式。

1.1 視訊影像的組成與階層式編碼

1.1.1 視訊影像的組成

視訊影像的組成，包含一系列連續的單張畫面。每張畫面由一定數量區塊（Macroblock）構成，而不同的畫面形式（Format）包含不同數量的方格。（表 1-1）〔1〕

表 1-1 不同畫面形式的解析度與區塊大小

	解析度 (水平 x 垂直)	區塊大小 (水平 x 垂直)	畫面中區塊總數
CIF	352 x 288	22 x 18	396
4CIF	704 x 576	44 x 36	1584
16CIF	1408 x 1152	88 x 72	6336
QCIF	176 x 144	11 x 9	99
Sub-QCIF	128 x 96	8 x 6	48

區塊內的像素（Pixel）是畫面最基本的元素。每個像素由紅，綠，藍（RGB）三個原色所組成。由於肉眼對三個顏色的敏銳度不同，因此影像處理又定義了明度像素（Luminance Pixel，代號 Y）與兩個色度像素（Chrominance Pixel，代號 Cb 與 Cr）。明度像素與色度像素是三個原色的線性轉換，Y，Cb，Cr，與 RGB 六者間的關係如**(1.1)**〔2〕:

$$Y = 0.299R + 0.587G + 0.114B ,$$
$$Cb = -0.169R – 0.331G + 0.500B ,$$
$$Cr = 0.500R – 0.419G – 0.081B . \qquad \textbf{(1.1)}$$

明度像素構成明度方格（Luminance Block），而兩個色度像素構成色度方格（Chrominance Block）。在顏色比例 4：2：0 的區塊格式下，四個明度方格與兩個色度方格共同組成一個區塊。而區塊構成畫面，連續的畫面再形成一個視訊序列。我們把可以把以上的組成繪成一個以時間爲橫軸的三維圖：

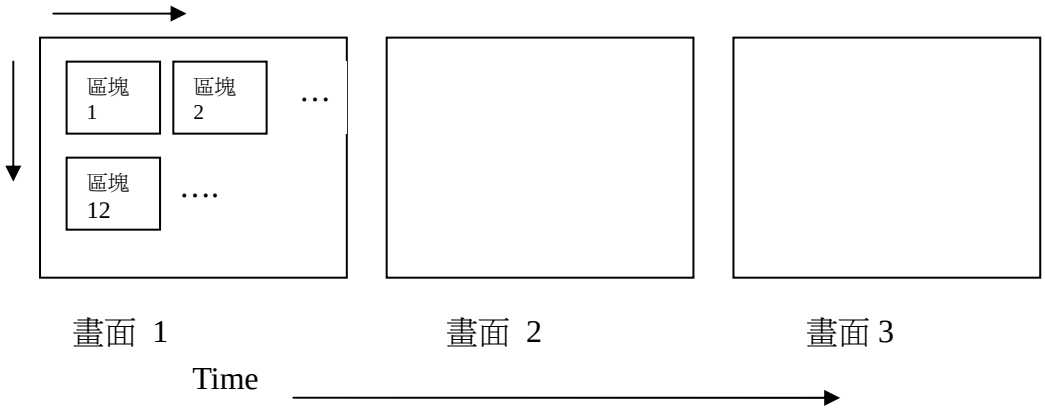

圖 1-2 畫面序列三維概念圖

1.1.2 階層式編碼

圖 1-2 顯示了視訊序列在壓縮編碼前的三維架構。而編碼的結果是一維的二進位資料，因此編碼的過程需要將三維的架構”重組”成一維的架構。爲了方便描述與分析，視訊壓縮標準會以階層式的概念表達壓縮編碼後的一維視訊資料。以階層方式進行的壓縮編碼，叫做階層式編碼（Layer Coding）。

透過階層式編碼壓縮過後的視訊資料，分爲三個主要階層：序列層（Sequence layer），畫面層（Frame Layer），與區塊層（MacroBlock Layer）。H.263，H.264 與 MPEG4 都使用階層式編碼，也都是按照這三個主要階層設計其編碼方式。

一長串的視訊資料的編碼結果叫做序列層資料。按照編碼的畫面數量以及次序，序列層資料包含許多畫面層資料。而每一個畫面層資料又包含數十個區塊層資料。

每一個層級的個別內容，都以標頭（Header）開始。標頭內容描述了該層級的屬性與特徵。例如 H.263 的畫面標頭，包含了量化位階等資訊，並且透過畫面標頭顯示該畫面是 I 畫面或 P 畫面（有關 I 畫面與 P 畫面的定義請見 1.3 節）。

1.2 轉換編碼

明度或色度方格內的像素大小呈現平均分布（Uniform Distribution），如此的分布情況使得編碼後的位元長度過長。轉換編碼（Transform Coding）將方格內的像素大小由像素值域（Pixel Domain）轉換成頻率值域（Frequency Domain），由於轉換基底（Transform Basis）的特性，轉換後能量會集中在特定頻率，通常是低頻〔3〕。轉換後低頻能量大於高頻能量的分布特性使得壓縮編碼器能夠專注於低頻的編碼，至於高頻可以忽略。轉換編碼後使得能量集中有利於編碼器進行壓縮編碼就是轉換編碼的目的。

以下的模擬我們觀察轉換編碼對方格內像素的分布特性的影響。我們取測試序列 Coastguard 第一張畫面的第十三個區塊的第一個明度方格（圖 1-3），以 H.263 壓縮方式來觀察轉換編碼對方格內像素大小的影響。

圖 1-3 Coastguard 樣本畫面

圖 1-3 箭頭所指的是 Coastguard 第一張畫面中第十三個區塊的第一個明度方格。我們拿這個方格進行模擬。表 1-2 顯示這個大小 8x8 的方格裡，未經轉換的 64 個像素在像素值域的像素大小。從表中可看出像素值域裡的 64 個像素大小差別不大，平均分佈在大約 50 到 80 的範圍。我們可以概略的說，像素值域中未經轉換的像素值大小呈現平均分布。

表 1-2 未經轉換的方格內像素大小（像素值域）

55	56	71	80	79	81	85	87
55	46	58	72	73	69	75	72
42	49	66	65	70	73	70	68
60	68	71	59	62	64	63	70
55	57	60	66	65	65	77	81
46	55	63	72	74	75	76	73
56	58	58	61	67	62	49	49
52	61	67	60	58	59	54	53

表 1-3 轉換後方格內的像素值大小（頻率值域）

514.75	-41.22	-22.06	-8.81	-0.5	1.11	2.91	2.42
28	-26	0.11	-2.27	5.47	7.98	1.11	2.34
-1.4	5.04	-10.51	3.79	2.68	3.04	1.56	-0.91
23.29	-18.49	-0.17	1.04	7.02	4.09	-5.1	-2.5
10.5	6.49	14.53	-9.11	-3.75	-1.54	0.47	-0.26
-0.02	13.39	-2.72	1.64	-4.25	-8.14	4.34	-0.39
10.32	-14.87	-7.93	-5.88	-7.87	0.42	1.51	-0.24
-4.4	-5.1	-1.53	0.92	1.79	0.83	-0.1	-0.04

表 1-3 顯示表 1-2 內的 64 個像素經由 8x8 離散餘弦轉換轉換，由像素值域轉換到頻率值域的結果。表 1-3 的左上角是低頻，右下角是高頻。表中明顯地顯示頻率值域從左上角遞減到右下角，左上角的頻率位置（DC）最大，高達 500，而右下半的值大約都只有個位數字的大小。從這個例子我們看到，轉換編碼能夠將方格內的像素從平均分布轉變成能量集中在低頻。

轉換編碼將 m x n 個像素與 m x n 個基底進行乘法運算。以幾何概念來說，這樣的計算可以看成是像素資料”投影”在個別基底上；而從數學觀點來說，計算後的結果則是像素資料與基底之間的內積（Inner Product）。轉換基底是轉換編碼的靈魂，基底的選擇與設計決定了轉換結果的分布情形。

爲了使轉換結果有利於壓縮編碼，基底的選擇與設計通常包含兩項特性：

一、基底之間正交（Orthogonal）

基底之間正交就是相同基底之間的內積爲一（Orthonormal），不同基底之間內積爲零。從投影的觀點來看，正交的特性使得像素資料投影在各基底上的結果保有”獨特性”，各轉換結果之間沒有贅述（Redanduncy）。

二、基底近似弦波（Sinosoidal Wave）

轉換後的頻率是像素資料與基底之間的內積，如果基底的統計特性與方格內的像素近似，轉換後的能量會集中在特定頻率。例如離散餘弦轉換（Discrete Cosine Transform，DCT）的基底是弦波，與自然界影像的特徵近似〔3〕。

1.2.1 離散餘弦轉換

在影像處理時，離散餘弦轉換是最常用的。MPEG4〔4〕與 H.263〔5〕都使用 8x8 離散餘弦轉換。8x8 離散餘弦轉換將區塊內 8x8 方格裡的像素投影在離散餘弦轉換的轉換基底上。轉換的公式見**(1.2)**〔6〕。

$$y(u,v)=\alpha(u)\alpha(v)\sum_{a=0}^{7}\sum_{b=0}^{7}x(a,b)\cos\left[\frac{(2a+1)u\pi}{16}\right]\cos\left[\frac{(2b+1)v\pi}{16}\right]$$

$$\alpha(u)=\begin{cases}\sqrt{\frac{1}{8}}, u=0\\ \sqrt{\frac{1}{4}}, u=1,2,.....7\end{cases}$$

$$u,v=0,1,2,.....7$$

(1.2)

(1.2)中 $x(a,b)$ 是像素值域裡的像素大小， $y(u,v)$ 是轉換後頻域值域裡的像素大小。 a,b 代表了轉換前像素在像素值域裡的位置， u,v 轉換後頻率值域裡的頻率位置。

1.2.2 H.264 的核心編碼

H.264〔7〕的轉換編碼稱爲核心轉換（Core Transform）〔8〕。核心轉換以 4x4 大小爲單位進行運算，一個區塊有一個 16x16 明度方格以及兩個 8x8 色度方格，因此一個區塊要進行 16 次明度像素的核心轉換以及 8 次色度像素的核心轉換。核心轉換演變自 4x4 離散餘弦轉換，他保留了離散餘弦轉換基底之間相互正交的特質，但是稍微改變了離散餘弦轉換的基底。核心轉換相對於離散餘弦轉換的最大差別是運算過程的簡化。以下我們分別介紹核心轉換的基底以及運算的簡化方式。

核心編碼演變自 4x4 離散餘弦轉換，4x4 離散餘弦轉換的公式如**(1.3)**〔6〕。

$$y(u,v)=\alpha(u)\alpha(v)\sum_{a=0}^{3}\sum_{b=0}^{3}x(a,b)\cos\left[\frac{(2a+1)u\pi}{8}\right]\cos\left[\frac{(2b+1)v\pi}{8}\right]$$

$$\alpha(u)=\begin{cases}\sqrt{\frac{1}{4}}, u=0\\ \sqrt{\frac{1}{2}}, u=1,2,3\end{cases}$$

$$u,v=0,1,2,3$$

(1.3)

(1.3)中 $x(a,b)$ 是像素值域裡的像素大小， $y(u,v)$ 是轉換後頻域值域裡的像素大小。 a,b 代表了轉換前像素在像素值域裡的位置， u,v 轉換後頻率值域裡的頻率位置。我們再將**(1.3)**改以矩陣方式表示：

$$Y = A_{DCT} X A_{DCT}^{T}$$

$$= \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.6533 & 0.2706 & -0.2706 & -0.6533 \\ 0.5 & -0.5 & -0.5 & 0.5 \\ 0.2706 & -0.6533 & 0.6533 & -0.2706 \end{bmatrix} \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} 0.5 & 0.6533 & 0.5 & 0.2706 \\ 0.5 & 0.2706 & -0.5 & -0.6533 \\ 0.5 & -0.2706 & -0.5 & 0.6533 \\ 0.5 & -0.6533 & 0.5 & -0.2706 \end{bmatrix}$$

(1.4)

(1.4)中，等式右端的 x_{00} 、 $x_{01} \ldots\ldots\ldots x_{33}$ 表示**(1.3)**中的 $x(0,0)$ 、 $x(0,1) \ldots\ldots\ldots x(3,3)$ 。這 16 個像素值域像素大小，依據所在的位置構成了矩陣 X 。矩陣 X 裡的 x_{00} 、 $x_{01} \ldots\ldots\ldots x_{33}$ 經由 4x4 離散餘弦轉換，產生了頻率值域矩陣 Y 。 A_{DCT} 則是 4x4 離散餘弦轉換的轉換矩陣。

(1.4)顯示了 4x4 離散餘弦轉換如何透過 A_{DCT} 將矩陣 X 轉換成爲矩陣 Y 。H.264 的核心轉換改變**(1.4)**中的 A_{DCT} ，成爲 A ，核心轉換中矩陣 X 與矩陣 Y 的關係如**(1.5)**：

$$Y = AXA^{T}$$

$$= \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.6325 & 0.3162 & -0.3162 & -0.6325 \\ 0.5 & -0.5 & -0.5 & 0.5 \\ 0.3162 & -0.6325 & 0.6325 & -0.3162 \end{bmatrix} \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} 0.5 & 0.6325 & 0.5 & 0.3162 \\ 0.5 & 0.3162 & -0.5 & -0.6325 \\ 0.5 & -0.3162 & -0.5 & 0.6325 \\ 0.5 & 0.6325 & 0.5 & -0.3162 \end{bmatrix}$$

(1.5)

(1.5)就是核心轉換的數學式，而矩陣 A 是核心轉換的轉換矩陣。從**(1.5)**我們可看到，像素值域的矩陣 X 透過轉換矩陣 A 轉換至頻率值域的矩陣 Y 。從**(1.4)**與**(1.5)**我們也可看到，核心轉換只有稍微改變 4x4 離散餘弦轉換，兩者的轉換矩陣近似。核心轉換將矩陣 A_{DCT} 中第二列的 0.6533 改爲 0.6525，0.2706 改爲 0.3162。接下來我們介紹 H.264 如何簡化**(1.5)**的運算。

H.264 的簡化方式，是將**(1.5)**的運算分解成兩個階段的運算：第一個階段的矩陣運算與第二個階段的浮點數乘法運算。圖 1-4 是簡化後的核心轉換方塊圖。

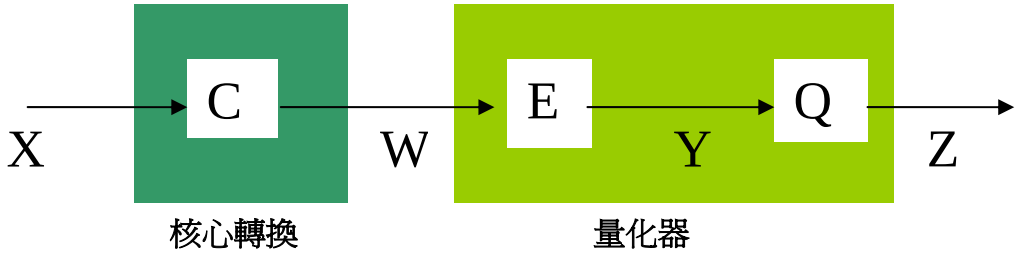

圖 1-4 簡化後的核心轉換方塊示意圖

第一個階段中，矩陣 X 透過矩陣 C 的矩陣運算產生矩陣 W

$$W = CXC^T \qquad \textbf{(1.6)}$$

第二個階段中，矩陣 W 進行浮點數乘法運算產生矩陣 Y ：

$$Y = W \otimes E \qquad \textbf{(1.7)}$$

(1.7)裡的運算符號⊗表示兩個大小相同的矩陣之間，相同矩陣位置的矩陣元素之間進行乘法運算。**(1.6)**與**(1.7)**的推導方式如下：將**(1.5)**中的矩陣 A 單位化（Normalize），也就是提出第一列與第三列的 0.5，第二列與第四列的 0.3162，我們得到**(1.8)**：

$$Y = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} 0.5 & 0.6325 & 0.5 & 0.3162 \\ 0.5 & 0.3162 & -0.5 & -0.6325 \\ 0.5 & -0.3162 & -0.5 & 0.6325 \\ 0.5 & -0.6325 & 0.5 & -0.3162 \end{bmatrix} \right) \otimes \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.3162 & 0.3162 & 0.3162 & 0.3162 \\ 0.5 & 0.5 & 0.5 & 0.5 \\ 0.3162 & 0.3162 & 0.3162 & 0.3162 \end{bmatrix}$$

$$= \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} 0.25 & 0.1581 & 0.25 & 0.1581 \\ 0.1581 & 0.1 & 0.1581 & 0.1 \\ 0.25 & 0.1581 & 0.25 & 0.1581 \\ 0.1581 & 0.1 & 0.1581 & 0.1 \end{bmatrix}$$

$$= \left(CXC^T \right) \otimes E$$

$$Y = \left(CXC^T \right) \otimes E \qquad \textbf{(1.8)}$$

(1.8)的前半段矩陣運算就是**(1.6)**；後半段就是**(1.7)**。經由對**(1.5)**的單位化，我們得到了矩陣 C ；矩陣 X 與矩陣 C 進行矩陣運算，產生矩陣 W **(1.6)**。由於矩陣 C 內的元素大小只有 1 與 2，第一個階段的矩陣運算中只需使用少數的移位器與加法器，硬體實踐（Hardware Implementation）相對簡單許多。第一個階段的矩陣運算結果產生矩陣 W ；第二個階段的浮點數乘法運算，矩陣 W 並不直接與矩陣 E 裡的浮點數進行乘法運算；矩陣 E 會與量化器的量化位階結合（參考圖 1-4），矩陣 W 與此量化位階運算後得到矩陣 Z ，完成核心轉換與量化的工作。整體而言，H.264 的核心轉換只需少數的移位器與加法器。

1.2.3 H.264 的量化

H.264 的量化器，將W_{ij}乘以**(1.8)**中矩陣E裡的乘法元素PF_{ij}，再除以量化位階$QStep$，產生Z_{ij} **(1.9)**〔9〕。

$$Z_{ij} = W_{ij} \times \frac{PF_{ij}}{QStep} \qquad \textbf{(1.9)}$$
$$i, j = 0,1,2,3$$

(1.9)中W_{ij}是核心轉換的結果（參考**(1.6)**），而$QStep$是量化參數（簡稱QP）的函數。使用者在壓縮影像時，從 0 到 51 輸入一個特定的QP值，作爲壓縮影像時的量化依據。在 H.264 的量化位階設計中，QP每增加 6，$QStep$增大一倍，QP與$QStep$的關係如表 1-4。〔10〕

表 1-4 量化參數QP與量化位階$QStep$的關係

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

H.264 量化過程的簡化方式

(1.9)的運算方式需要用到浮點數乘法以及除法，爲了簡化量化的運算過程，H.264 利用QP每增加 6，$QStep$增大一倍的特性減少**(1.9)**的運算複雜度。簡化的方式是

將等式**(1.9)**的浮點數乘法以及除法改成整數乘法以及位移，如**(1.10)**。

$$Z_{ij} = (W_{ij} \times MF_{ij}) >> qbits$$
$$i, j = 0,1,2,3 \qquad \textbf{(1.10)}$$

根據**(1.10)**，Z_{ij} 等於W_{ij} 乘以 MF_{ij} 再右移 $qbits$ 次。其中 $qbits$ 以及 MF_{ij} 定義如**(1.11)** **(1.12)**：

$$qbits = 15 + QP/6 \qquad \textbf{(1.11)}$$

$$MF_{ij} = \frac{PF_{ij} \times 2^{qbits}}{QStep} \qquad \textbf{(1.12)}$$

MF_{ij} 以及 $qbits$ 的大小影響到了等式**(1.10)**的計算複雜度。下面我們分別計算分析 MF_{ij} 以及 $qbits$ 。

MF_{ij} 的計算

我們使用**(1.12)**計算 MF_{ij} 。根據**(1.11)**， QP 每增加 6， $qbits$ 增加 1；而從表 1-4 我們知道 QP 每增加 6， $QStep$ 增大一倍。由這兩個特點， QP 每增加 6，**(1.12)** 中右端的分子與分母同時增大一倍； QP 增加 6 不會改變到 MF_{ij} 的大小。由此可知 MF_{ij} 大小與 $QStep$ 、 QP 無關； MF_{ij} 是 $QP \bmod 6$（ QP 除以 6 得到的餘數）以及 PF_{ij} 的函數。從**(1.8)**我們知道 $PF_{ij} = 0.25, 0.1, 0.1581$ 。接著分別將 $QP \bmod 6 = 0,1,2,3,4,5$ 以及 $PF_{ij} = 0.25, 0.1, 0.1581$ 帶入**(1.12)**，計算出 MF_{ij} 如表 1-5：〔11〕

表 1-5 MF_{ij} 與 QP 以及 PF_{ij} 的關係

	$QP \bmod 6$					
	0	1	2	3	4	5
PF_{ij} =0.25	13107	11916	10082	9362	8192	7282
PF_{ij} =0.1	5243	4660	4194	3647	3355	2893
PF_{ij} =0.1581	8066	7490	6554	5825	5243	4559

qbits 的計算

根據等式**(1.11)**，*qbits* 是 *QP* 的函數，其大小從 15 到 23，如表 1-6。

表 1-6 *qbits* 與 *QP* 的關係

QP range	*qbits*
0 – 5	15
6 – 11	16
12 – 17	17
18 – 23	18
24 – 29	19
30 – 35	20
36 – 41	21
42 - 47	22

量化器複雜度的整體分析

計算分析完了 MF_{ij} 與 *qbits* ，我們整體分析量化器的複雜度。從等式**(1.10)**，我們知道簡化後的量化運算使用到 MF_{ij} 與 *qbits* 這兩個變數。量化器根據 $QP \bmod 6$ 以及 W_{ij} 的 (i, j) 值，從表 1-5 選擇一個 MF_{ij}。量化器根據 *QP* ，從表 1-6 選擇一個 *qbits* 。 W_{ij} 乘以 MF_{ij} 再右移 *qbits*，完成量化運算。

MF_{ij} 根據 *QP* 以及 W_{ij} 的 (i, j) 值，只有 18 個選擇，並且都是整數。 *qbits* 大小從 15 到 23。H.264 量化器複雜度，只需進行整數乘法運算以及 15 到 23 的位移。

1.3 預測編碼

動態影像區塊（方格）之間的像素大小極爲近似，這是視訊的特質。單張畫面中，背景的鄰近區塊之間呈現極爲雷同的顏色，因此鄰近方格（方格）的像素大小近似。動態影像由連續的畫面組成，景物的移動現象也增加了連續畫面之間的近似性。利用動態影像區塊（方格）像素大小的近似特質進行的編碼方式稱爲預測編碼（Prediction Coding）。找尋影像區塊之間雷同的地方，是預測編碼的主要工作。依照找尋的方式，預測編碼分成兩大種類：找尋單張畫面鄰近影像區塊（方格）之間的近似處，是 Intra 編碼（Intra Prediction Coding），畫面若以 Intra 進行編碼稱爲 Intra Frame，簡稱 **I**；找尋連續畫面區塊（方格）之間的近似處，是 Inter 編碼（Inter Prediction Coding），畫面若以 Inter 進行編碼則稱爲 Inter Frame，簡稱 **P**。動態壓縮編碼過程中，通常兩者同時進行，例如連續的畫面，每隔十張便進行 Intra 編碼，中間的畫面則使用 Inter 編碼。如以下的畫面序列所示：

I P P P P P P P P P I P P P P P P P P P I P P P P P P.....

1.3.1 Intra 預測編碼

Intra 預測編碼器首先逐一從鄰近區塊（方格）中，找尋一個與目前區塊（方格）最近似的區塊（方格）。找到的結果會與目前區塊（方格）相比對（像素大小相減），因此兩個方格越是近似，相減後的像素大小越小，編碼效果越好。由於每一個鄰近方格都代表一個特定的 Intra 模式，因此找尋的過程又叫做模式選擇（Mode Decision）。找到了合適的模式，再進行比對，消除兩者近似之處。

模式選擇決定了比對的鄰近方格，也決定了編碼的效果。精確的模式選擇方式可以找到與目前方格最”匹配”的鄰近區塊（方格），但是耗費計算機效能。不同視訊標準定義不同的模式選擇方式，在第三章我們深入分析 H.263、H.264 與 MPEG4 的模式選擇方式與編碼效果的關係。

1.3.2 Inter 預測編碼

與 Intra 編碼不同，Inter 編碼是從標準畫面（通常是前一張畫面）中，找尋一個與目前區塊（方格）最近似，最雷同的區域。也就是說，Inter 編碼的找尋過程是跨畫面的，這也是爲何使用”Inter”這個字的原因。

Inter 編碼找尋近似區域的過程，叫做動態預測（Motion Estimation）。這個名稱婉轉的表達了 Inter 編碼的精神：標準畫面中，與目前區塊（方格）最近似的區域就是景物移動的位置，動態預測反映了畫面的移動情形。

我們以移動向量（Motion Vector）來表示這個區域的位置。移動向量是二維的位置描述，包含水平與垂直兩個分量。移動向量所標示的位置是相對的，相對於目前區塊的位置。

與 Intra 編碼相同，不同的移動向量找尋方式耗費不同的計算機效能，也產生不同的編碼效果。以固定方格大小或以變動方格大小進行動態預測也產生不同的效果。第四章我們分析比較 H.263、H.264、MPEG4 在這方面的異同。

1.3.3 動態位元數，像素位元數與計算機運算次數的權衡

不論 Inter 或 Intra 預測，都減少了區塊內的像素大小，並以 Intra 模式或移動向量描述區塊的移動現象。因此預測編碼減少了像素的位元數，卻因爲區塊標頭的 Intra 模式或移動向量佔用額外的位元而增加了額外的動態位元數。

預測編碼是近似區塊的找尋過程，需要耗費計算機運算次數。精確的找尋過程（Intra 的模式選擇，Inter 的動態預測）耗費計算機運算次數，但是可以找到最近似的區塊。

由上所述，我們看到預測編碼的三大關鍵要素：動態位元數、像素位元數、與計算機運算次數。如何在最少的計算機運算次數下，產生最少的動態位元數與像素位元數是預測編碼設計時的一個重要課題。（參考圖 1-5）

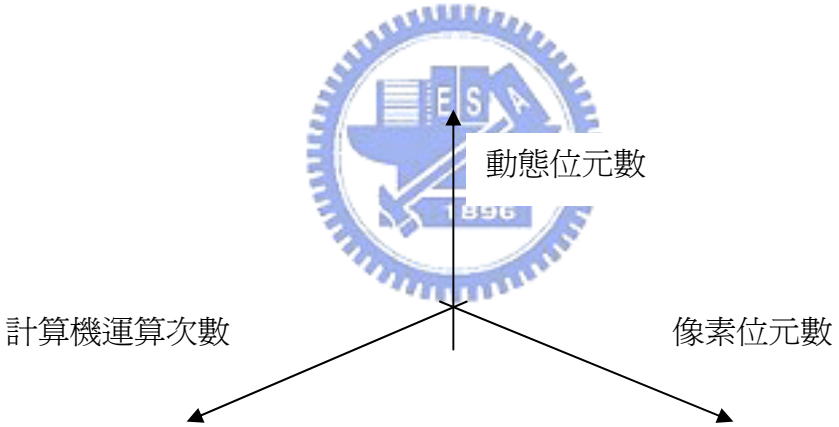

圖 1-5 動態位元數，像素位元數以及計算機運算次數的權衡

1.4 熵編碼

編碼（Encode）就是將標記（Symbol）轉換成特定的二進位代號（Codeword），而熵編碼（Entropy Coding）就是要以最少的代號長度（Codeword Length）來描述此標記。利用不同標記出現的機率不同的特性，熵編碼採取非等長碼（Variable Length Code），也就是偶而出現的標記對應到長碼（Long Codeword），常出現的標記對應到短碼（Short Codeword），使得編碼後整體的位元總數最少。設計熵編碼器時，會替個別標記找尋合適長度的二進位代號，標記與代號兩者之間呈現一對一的對應關係。根據各標記與其所對應的二進位代號所繪製的表格，就是編碼表（Codeword Table）。由於熵編碼是編碼必經也是最重要的環節，編碼表會在後續章節一再出現。由此可知，熵編碼的設計重點在於熵編碼的設計與各標記出現機率的預估。

視訊壓縮中的熵編碼設計，重點在各層級標頭（序列層標頭，畫面層標頭，區塊層標頭）以及區塊裡的移動向量，像素轉換後的編碼。