國 立 中 央 大 學

電 機 工 程 研 究 所

碩 士 論 文

快速多階連續消除移動預估演算法
應用於 H.26L 視訊編碼標準

指導教授：張 寶 基 博士

研 究 生：蔡 國 基

中華民國九十一年六月

國立中央大學圖書館
碩博士論文授權書

(91 年 5 月最新修正版)

本授權書所授權之論文全文與電子檔，為本人於國立中央大學，撰寫之碩/博士學位論文。(以下請擇一勾選)

(✓)**同意** (立即開放)

(　　)**同意** (一年後開放)，原因是：＿＿＿＿＿＿＿＿

(　　)**同意** (二年後開放)，原因是：＿＿＿＿＿＿＿＿

(　　)**不同意**，原因是：＿＿＿＿＿＿＿＿＿＿

以非專屬、無償授權國立中央大學圖書館與國家圖書館，基於推動讀者間「資源共享、互惠合作」之理念，於回饋社會與學術研究之目的，得不限地域、時間與次數，以紙本、光碟、網路或其它各種方法收錄、重製、與發行，或再授權他人以各種方法重製與利用。以提供讀者基於個人非營利性質之線上檢索、閱覽、下載或列印。

研究生簽名: 蔡　國　基

論文名稱: 快速多階連續消除移動預估演算法應用於 H.26L 視訊編碼標準

指導教授姓名： 張　寶　基　教授

系所　：　電 機 工程研究　所　☐博士　☑碩士班

學號：　89521042

日期：民國　91　年　6　月　24　日

備註：

1. 本授權書請填寫列印**兩份紙本**，並親筆簽名後（全文電子檔內之授權書可用電腦打字代替），一份請裝訂於紙本論文封面後之次頁，一份於辦理離校時，交圖書館服務台（以統一轉交國家圖書館），未附本授權書，圖書館將不予驗收。
2. 讀者基於個人非營利性質之線上檢索、閱覽、下載或列印上列論文，應依著作權法相關規定辦理。

快速多階連續消除移動預估演算法應用於 H.26L 視訊編碼標準

摘要

關鍵字 –H.26L視訊編碼標準、快速移動預估演算法、多階連續消除演算法(MSEA)、精確位移向量、中斷決策。

移動預估在視訊壓縮編碼上一直扮演著重要的角色,其最主要的目的,是利用畫面間的相關性,來移除多餘的資料量,以達到壓縮目的。因此移動預估演算法的好壞,對整個視訊壓縮品質有極大的影響。

H.26L是目前已發展完成並且最有效率的視訊編碼標準,它使用了多重模式,即不同區塊大小的移動預估,來改進及增進移動向量的精確度。然而,在H.26L視訊編碼標準上,使用全域搜尋演算法(Full Search)會造成龐大的計算量。為了要降低全域搜尋演算法的複雜度,本論文提出快速多階連續消除演算法(FMSEA)於H.26L視訊編碼標準,來解決H.26L在多重模式移動預估上的搜尋。本論文所提出的方法,最主要是修改多階連續消除演算法,並且結合精確位移向量(motion refinement)及利用中斷決策來判斷是否繼續搜尋4x8、8x4及4x4這三個模式。實驗的結果顯示,本論文所提出的快速多階連續消除演算法,有效地降低複雜度並且畫面品質極接近全域搜尋法(Full Search),適合運用在視訊壓縮標準H.26L上。

Fast Multi-level Successive Elimination Algorithm for Motion Estimation in H.26L

Abstract

Keyword- H.26L Video Coding、Fast Motion Estimation、Multi-level Successive Elimination Algorithm (MSEA)、Refinement、Half Stop Decision.

Motion estimation plays an extremely important role in the video coding. The objective of the motion estimation is to remove the temporal redundancy between video frames so that the motion compensated frames can be coded efficiently.

H.26L video coding is the most efficient coding standard currently available. It uses multi-mode with variable block-size motion estimation to improve the accuracy. However, the conventional full search algorithm will be a heavy computational load in this situation. To reduce the complexity, we propose a fast multi-level successive elimination algorithm (FMSEA) for H.26L multi-mode motion estimation search. The proposed method is mainly based on the combination of a modified multi-level successive elimination algorithm (MSEA) with a motion refinement approach and a half-stop decision that skips the 8x4, 4x8, and 4x4 sub-block motion searches. Experimental results show that FMSEA is very efficient in terms of the computational speedup and video reconstruction quality for H.26L.

目　錄

第一章 緒論.....1
1.1 簡介.....1
1.2 動機與目的.....2
1.3 論文架構.....2
第二章 視訊壓縮標準簡介.....3
2.1 資料壓縮簡介.....3
2.2 H. 263 視訊壓縮標準簡介.....4
2.2.1 影像大小格式.....5
2.2.2 區塊組成.....6
2.2.3 DCT 與 ZigZag.....8
2.2.4 量化(Quantization).....9
2.2.5 半像素(Half Pixel).....10
2.2.6 移動向量與參考向量.....11
2.2.7 四種選擇性編碼.....12
2.2.8 H.263 壓縮流程.....16
2.3 H. 26L 視訊壓縮標準簡介.....20
2.3.1 Intra/Chroma/Inter 預測模式.....23
2.3.2 Transform Coding.....31
2.3.3 UVLC.....34
2.3.4 High/Low Complexity.....35
2.4 H .26L 複雜度分析.....39
第三章 快速位移估計演算法簡介.....42
3.1 無失真快速位移估計演算法(Lossless).....43

3.1.1 連續消除演算法(SEA) 43
3.1.2 多階層連續消除演算法(MSEA).....46
3.1.3 一維投影演算法(1-D Projection) 47
3.1.4 部份消除演算法(PDE) 48
3.2 失真快速位移估計演算法 49
3.2.1 三步搜尋演算法(3SS) 49
3.2.2 新三步搜尋演算法(NTSS) 50
3.2.3 SES 搜尋演算法.....51
3.2.4 四步搜尋演算法(4SS) 53
3.2.5 鑽石搜尋演算法(DS).....54
3.2.6 區塊梯度搜尋演算法(BBGD).....56
3.2.7 基因搜尋演算法(Genetic) 57
3.2.8 使用空間域演算法(CAS) 59
第四章 快速多階連續消除演算法於 H.26L 視訊編碼標準.....61
4.1 運用連續消除演算法及多階連續消除法在 H.26L 的分析.....61
4.2 H. 26L 模式(Inter Mode)分析 70
4.3 運用既有連續消除所算出的 16x16SAD 來降低複雜度 76
4.4 精細小區塊的位移向量(Refinement) 79
4.5 中斷(Half Stop Mode) 83
4.6 快速多階連續消除演算法流程圖 84

第五章 實驗結果分析與討論 86
5.1 環境設定及所使用的視訊樣本 86
5.2 運用快速多階連續消除移動預估演算法加速的效果 89
5.3 Performance 評估 94
第六章 結論與未來展望 101

參考文獻..102

List of Figures

Figure 2.1 H.263 區塊組成架構圖 7
Figure 2.2 Y Cb Cr 與 6 塊 blocks 的關係圖 7
Figure 2.3 ZigZag 排列 8
Figure 2.4 量化的範例 9
Figure 2.5 H.263 內差法公式 10
Figure 2.6 經過內差後的圖 11
Figure 2.7 移動向量示意圖 12
Figure 2.8 參考向量示意圖 13
Figure 2.9 PB-Frames 之說明 16
Figure 2.10 I 畫面的壓縮流程圖 17
Figure 2.11 P 畫面的壓縮流程圖 18
Figure 2.12 全部壓縮流程圖 19
Figure 2.13 視訊壓縮發展示意圖 21
Figure 2.14 視訊標準 H.26L 結構圖 23
Figure 2.15 Intra coding 4x4 區塊示意圖 23
Figure 2.16 Intra coding 4x4 區塊方向示意圖 24
Figure 2.17 Intra coding 4x4 區塊 Mode1 示意圖 24
Figure 2.18 Intra coding 4x4 區塊 Mode2 示意圖 24
Figure 2.19 Intra coding 4x4 區塊 Mode3 示意圖 25
Figure 2.20 Intra coding 4x4 區塊 Mode4 示意圖 25
Figure 2.21 Intra coding 4x4 區塊 Mode5 示意圖 25
Figure 2.22 Intra coding 16x16 區塊方向示意圖 26
Figure 2.23 Intra coding Chrominance 區塊預測示意圖 27

Figure 2.24 Inter coding 7 個區塊模式示意圖 28
Figure 2.25 Inter coding 參考向量示意圖 29
Figure 2.26 Inter coding 參考向量方向示意圖 30
Figure 2.27 1-D Integer Transform and Inverse Transform 31
Figure 2.28 Luminance and Chrominance 編碼示意圖 32
Figure 2.29 Chrominace DC Transform 33
Figure 2.30 ZigZag Simple Scan 33
Figure 2.31 ZigZag Double Scan.....33
Figure 2.32 量化對照表 34
Figure 2.33 UVLC 示意圖 34
Figure 2.34 Intra Coding 模式機率表.....36
Figure 2.35 Interpolation 示意圖 38
Figure 2.36 H.26L 整體複雜度評估圖.....41
Figure 2.37 H.26L 位移估計複雜度評估圖.....41
Figure 3.1 螺旋式搜尋 45
Figure 3.2 柵欄式搜尋 45
Figure 3.3 連續消除演算法流程圖 45
Figure 3.4 一維區塊投影比對演算法示意圖 47
Figure 3.5 三步搜尋演算法示意圖 49
Figure 3.6 新三步搜尋演算法示意圖 50
Figure 3.7 SES 判斷象限示意圖 51
Figure 3.8 SES 判斷象限後增加點數示意圖 52
Figure 3.9 SES 搜尋演算法整體示意圖 52
Figure 3.10 四步搜尋演算法分解步驟示意圖 53
Figure 3.11 四步搜尋演算法整體示意圖 54
Figure 3.12 鑽石搜尋演算法分解步驟示意圖 54

Figure 3.13 鑽石搜尋演算法整體示意圖 55
Figure 3.14 區塊梯度搜尋演算法整體示意圖 56
Figure 3.15 基因演算法交配步驟示意圖 58
Figure 3.16 參考位移向量圖 59
Figure 4.1 將 SEA 演算法放入 H.26L 流程圖 62
Figure 4.2 使用一個 16x16 模式的狀況下，TML8.0、16x16SEA、8x8MSEA 及 4x4MSEA 壓一張所需要花的時間 63
Figure 4.3 使用一個 16x16 模式的狀況下，TML8.0、16x16SEA、8x8MSEA 及 4x4MSEA 畫面品質的比較 63
Figure 4.4 使用一個 16x16 模式的狀況下，TML8.0、16x16SEA、8x8MSEA 及 4x4MSEA 位元率的比較 64
Figure 4.5 使用一個 16x16 模式的狀況下，16x16SEA、8x8MSEA 及 4x4MSEA 所需要檢查的點數 64
Figure 4.6 使用一個 16x16 模式的狀況下，16x16SEA、8x8MSEA 及 4x4MSEA 算 Sum Norm 的時間 65
Figure 4.7 使用四個模式 16x16-8x8 的狀況下，TML8.0、8x8MSEA 及 4x4MSEA 壓一張所需要花的時間 65
Figure 4.8 使用四個模式 16x16-8x8 的狀況下，TML8.0、8x8MSEA 及 4x4MSEA 畫面品質的比較 66
Figure 4.9 使用四個模式 16x16-8x8 的狀況下，TML8.0、8x8MSEA 及 4x4MSEA 位元率的比較 66
Figure 4.10 使用四個模式 16x16-8x8 的狀況下，8x8MSEA 及 4x4MSEA 所需要檢查的點數 67
Figure 4.11 使用七個模式 16x16-4x4 的狀況下，TML8.0、及 4x4MSEA 壓一張所需要花的時間 67

Figure 4.12 使用七個模式 16x16-4x4 的狀況下，TML8.0、8x8MSEA 及 4x4MSEA 畫面品質的比較 67
Figure 4.13 使用七個模式 16x16-4x4 的狀況下，TML8.0、8x8MSEA 及 4x4MSEA 位元率的比較 68
Figure 4.14 模式分佈 - 中位元率 (No Frameskip) 70
Figure 4.15 模式分佈 - 中位元率 (Frameskip 1) 71
Figure 4.16 模式分佈 - 中位元率 (Frameskip 2) 71
Figure 4.17 模式分佈 - 低位元率 (No Frame skip) 72
Figure 4.18 模式分佈 - 低位元率 (Frameskip 1) 72
Figure 4.19 模式分佈 - 低位元率 (Frameskip 2) 73
Figure 4.20 模式分佈 - 高位元率 (No Frameskip) 74
Figure 4.21 模式分佈 - 高位元率 (Frameskip 1) 74
Figure 4.22 模式分佈 - 高位元率 (Frameskip 2) 74
Figure 4.23 以 Foreman Sequence 使用 1 Mode、4 Mode 及 7 Mode 的 Rate Distortion 分析 76
Figure 4.24 以 Foreman Sequence 在各種位元率下使用 1 Mode、4 Mode 及 7 Mode 所得到降低位元率比例圖 77
Figure 4.25 使用 8x8 MSEA 搜尋 16x16 區塊模式，並使用所得的 SAD 值繼續搜尋 16x8 – 4x4 區塊模式，並和 TML 8.0 區塊模式的準確度 78
Figure 4.26 使用 8x8 MSEA 搜尋 16x16 區塊模式，並使用所得的 SAD 值繼續搜尋 16x8 – 4x4 區塊模式，並和 TML 8.0 位移向量的準確度 79
Figure 4.27 以 8x8MSEA 來搜尋 16x16 區塊模式，再利用其殘餘的 SAD 值，往下搜尋小區塊的位移向量，與 TML8.0 所產生

模分佈的差異圖.....80
Figure 4.28 以 16x16 所搜尋過的點，利用其 SAD 值，往下搜尋小區塊的位移向量，得到其中最小的一個，圖中的距離就是 TML8.0 和上述方法的距離統計圖.....81
Figure 4.29 3x3 搜尋視窗.....81
Figure 4.30 5x5 搜尋視窗.....81
Figure 4.31 二步搜尋法.....82
Figure 4.32 利用周圍 3x3 、5x5 search window 及 2SS(two step search)的方式來進行精確小區塊中位移向量的相似度比較.82
Figure 4.33 Half stop decision 流程圖.....83
Figure 4.34 利用 Half stop decision 判斷成功的機率.....84
Figure 4.34 快速多階連續消除演算法流程圖.....85
Figure 5.1 各種快速演算法用在 Akiyo Sequence 編碼時間的比較.....90
Figure 5.2 各種快速演算法用在 News Sequence 編碼時間的比較.....90
Figure 5.3 各種快速演算法用在 Carphone Sequence 編碼時間的比較.....91
Figure 5.4 各種快速演算法用在 Foreman Sequence 編碼時間的比較.....91
Figure 5.5 各種快速演算法用在 Stefan Sequence 編碼時間的比較.....92
Figure 5.6 各種快速演算法用在 Bream Sequence 編碼時間的比較.....92
Figure 5.7 各種快速演算法用在 Akiyo Sequence Rate-Distortion 的比較.....95

Figure 5.8 各種快速演算法用在 News Sequence Rate-Distortion 的比較.....95
Figure 5.9 在高位元率，各種快速演算法用在 Carphone Sequence Rate-Distortion 的比較(1).....96
Figure 5.10 在中位元率，各種快速演算法用在 Carphone Sequence Rate-Distortion 的比較(2).....96
Figure 5.11 在低位元率，各種快速演算法用在 Carphone Sequence Rate-Distortion 的比較(3).....97
Figure 5.12 在中高位元率，各種快速演算法用在 Foreman Sequence Rate-Distortion 的比較(1).....97
Figure 5.13 在中低位元率，各種快速演算法用在 Foreman Sequence Rate-Distortion 的比較(2).....98
Figure 5.14 各種快速演算法用在 Stefan Sequence Rate-Distortion 的比較.....98
Figure 5.15 在中高位元率，各種快速演算法用在 Bream Sequence Rate-Distortion 的比較(1).....99
Figure 5.15 在中低位元率，各種快速演算法用在 Bream Sequence Rate-Distortion 的比較(2).....99

List of Tables

Table2.1 H.263 支援影像大小格式.....5
Table2.2 Y Cb Cr 解碼還原表.....6
Table5.1 視訊樣本的分類表.....86
Table5.2 各種快速演算法編碼張數比較表.....89
Table6.1 利用 FSEA 在 H.26L 上所得到各 Function 的時間...101

第一章 緒論

1.1 簡介

隨著科技的進步，人們常享受著科技的便利，如 MPEG-1 音視訊壓縮標準在 VCD 及 MP3 的應用，漸漸地取代舊時的錄影帶及錄音帶；緊接著而來的是，利用 MPEG-2 音視訊壓縮標準，比 VCD 擁有更高的畫質及音質的 DVD，使我們在家中就如同置身在高水準、高品質的電影院中。由於訊號壓縮技術的發展，加上寬頻網路建設不斷的擴展，促使多媒體音視訊在網路上的應用大放異彩，例如用在公眾網路或網際網路(Internet)上來達成視訊會議(Video Conferencing)的 H.263，以及有高抗錯性及物件為基礎特性的 MPEG-4，不但適合用在儲存音視資料上，而且其高抗錯能力及低位元輸出率，極適合在無線的環境下傳輸。

然而，新一代的視訊標準 H.26L 也即將問世，其性能不但勝過 H.263 低位元率及 MPEG-4 高抗錯性的特性，同時也考慮到網路傳輸的功能。雖然 H.26L 還在制定中，確已成為學術及工業界研究的熱門話題。

1.2 動機與目的

由於人眼的視覺神經有視覺暫留的特性，視訊動畫每秒播 30 張的畫面，便不會察覺有不連續的效果，如此高的播放速度，鄰近畫面間的相關性必定很高，所以一般的視訊壓縮標準，會利用位移估計(Motion Estimation)及位移補償(Motion Compensation)等技術來消除此時間上的相關性。

所以要評估一套視訊編碼標準的效能，就非位移估計的演算法莫

屬了，而本篇論文將針對位移估計演算法做些研究及探討，並提出一套針對在 H.26L 的快速演算法。這套新一代的視訊標準，雖然有很多數據証明其效能，比從前 H.263 及 MPEG-4 好[23]，但從一些文獻[22]及實驗結果証實，其編碼的複雜度太高，特別是在位移估計這個部份，使其難以運用在即時的壓縮上，所以本論文將針對其演算法複雜度，做一定的加速，並且不會降低太多畫面的品質。

1.3 論文架構

在本篇論文的第一章中，描述論文動機；第二章介紹為什麼要資料壓縮、視訊標準 H.263 以及新一代的視訊標準 H.26L 並就其複雜度作深入的分析；第三章敘述各種快速的位移估計演算法；第四章會提出本論文所發展出來的快速多階連續消除移動預估演算法；第五章為實驗結果與討論；最後，第六章則為結論及未來展望。

第二章 視訊壓縮標準簡介

第二章的主要內容，首先在第一節中會介紹基本的視訊壓縮標準，包含為什麼需要資料壓縮。第二節會介紹用在低位元率的視訊標準 H.263[2][3]。在第三節會介紹新一代的視訊壓縮標準 H.26L[5]，也是本論文所採用之視訊標準。第四節會用軟體來分析 H.26L 的複雜度。

2.1 資料壓縮簡介

為什麼資料需要壓縮呢? 原來就是因為我們將聲音、影像和動態的畫面數位化後，產生的大量資料，使得在儲存或傳輸有種種的限制及不便。舉一個例子，以一個 352*240 影像大小的及播放的速度為每秒 30 張全彩畫面(RGB)視訊，要存放在一片 650Mbyte 的 CD 中，大約只能儲存 90 秒中，計算方式如下：

352 * 240 pixels/frame * 3 bytes/pixel * 30 frames/sec=7603200 bytes/sec

650*1024*1024 bytes/7603200 bytes/sec= 89.64 sec

從以上的例子，可知資料若不採用資料壓縮的方式來降低資料量，那所要儲存及傳輸的資料必定是個天文的數字。資料壓縮的好處一，就是將資料量降低，可使得傳輸的費用降低，或者是在一定的傳輸頻寬下，得到更好的視訊品質；好處二，在儲存上，可以將一部兩

小時的電影，直接放入一片小小的 CD 中，由此可見其重要性。

而在資料壓縮中，有分為兩種：

- 無損耗性編碼(Lossless compression)
- 損耗性編碼(Lossy compression)

無損耗性編碼代表解碼得到的資料和原始資料完全相同，通常是用在不容有誤差存在的文件資料，以及應用在電腦系統內部無限次傳輸、複製及再生等處理方面，通常其壓縮比率並不大。

損耗性編碼，顧名思意就是解碼之後與原來的不同，特別可用在影像、聲音及視訊的訊號上，主要是因為人的眼睛及耳朵對影像及聲音訊號有部份，並不是那麼的敏感，因此這部份的資訊可以不用那麼重視，甚至可以忽略。

以下，就以簡單的方式講解幾個重要的視訊壓縮標準。

2.2 H.263 視訊壓縮標準

H.263[2][3]是特別用在低位率(Low Bit Rate)傳輸的視訊標準，目的是用在低寬帶(Low Bandwidth)上傳輸視訊，擁有如下的特點：

1.提供五種視訊格式，分別是 sub-qcif、qcif、cif、4cif、16cif。

2.列狀結構的區塊組層。

3.半像素(Sub-pixel)的移動向量補償。

4.四種選擇性編碼模式。

•無限制的移動向量模式(Unrestricted Motion Vector mode)

•結構式算數編碼模式（Syntax-based Arithmetic Coding mode）

•進階預測模式(Advanced Prediction mode)

•PB 畫面模式

2.2.1 影像大小格式

H.263 支援的影像大小格式有五種，分別為 sub-qcif、qcif、cif、4cif、16cif，其五種格式大小如 Table 2.1:

Table 2.1 H.263 支援影像大小格式

Picture Format	Number of pixels for luminance (dx)	Number of lines for luminance (dy)	Number of pixels for chrominance (dx/2)	number of lines for chrominance (dy/2)
Sub-QCIF	128	96	64	48
QCIF	176	144	88	72
CIF	352	288	176	144
4CIF	704	576	352	288
16CIF	1408	1152	704	576

一般來說，影像都是由 R(紅色)、G(綠色)、B(藍色)三原色來表達色彩，在經過矩陣轉換後可將 RGB 訊號轉換成一個亮度(luminance)訊號 Y 及二個色差訊號(color difference)Cb、Cr，用 Y、Cb、Cr 三個訊號即可組成一個完整的顏色，而且人眼在處理顏色上，最敏感是亮度訊號，其次才是色差訊號，所以我們利用了這點特性來作壓縮，所以我們從上表可知 Y、Cb、Cr 的比值是 4:1:1，從這樣資訊中要如何還原圖案呢？

假設有個圖案(4x4)資料流是這樣的

亮度訊號：$Y_1\ Y_2\ Y_3\ Y_4\ Y_5\ Y_6\ Y_7\ Y_8\ Y_9\ Y_{10}\ Y_{11}\ Y_{12}\ Y_{13}\ Y_{14}\ Y_{15}\ Y_{16}$

色差訊號：$Cb_1\ Cb_2\ Cb_3\ Cb_4$

色差訊號：$Cr_1\ Cr_2\ Cr_3\ Cr_4$

Table 2.2 表示一個 4x4 的還原圖案表，各點都有三個 Y Cb Cr 值，所以依照這樣的方法就可以還原 qcif 影像。

Table 2.2 Y Cb Cr 解碼還原表

$Y_1\ Cb_1\ Cr_1$	$Y_2\ Cb_1\ Cr_1$	$Y_3\ Cb_2\ Cr_2$	$Y_4\ Cb_2\ Cr_2$
$Y_5\ Cb_1\ Cr_1$	$Y_6\ Cb_1\ Cr_1$	$Y_7\ Cb_2\ Cr_2$	$Y_8\ Cb_2\ Cr_2$
$Y_9\ Cb_3\ Cr_3$	$Y_{10}\ Cb_3\ Cr_3$	$Y_{11}\ Cb_4\ Cr_4$	$Y_{12}\ Cb_4\ Cr_4$
$Y_{13}\ Cb_3\ Cr_3$	$Y_{14}\ Cb_3\ Cr_3$	$Y_{15}\ Cb_4\ Cr_4$	$Y_{16}\ Cb_4\ Cr_4$

從上表可知只要使用 4:1:1 的資料量，就可以表示 4:4:4 的的原始資料，而且人的肉眼對這樣的改變是不敏感的，以這樣的方式可以節省 1/2 的資料量，這是在使用 DCT 及量化壓縮法以前，針對原始資料所作的壓縮。

2.2.2 區塊組成

H.263 是以一種階層式的結構組成，由上而下分成四層，依序為 Picture layer、Group of blocks layer、Macro block layer、block。其架構如 Figure 2.1 所示：

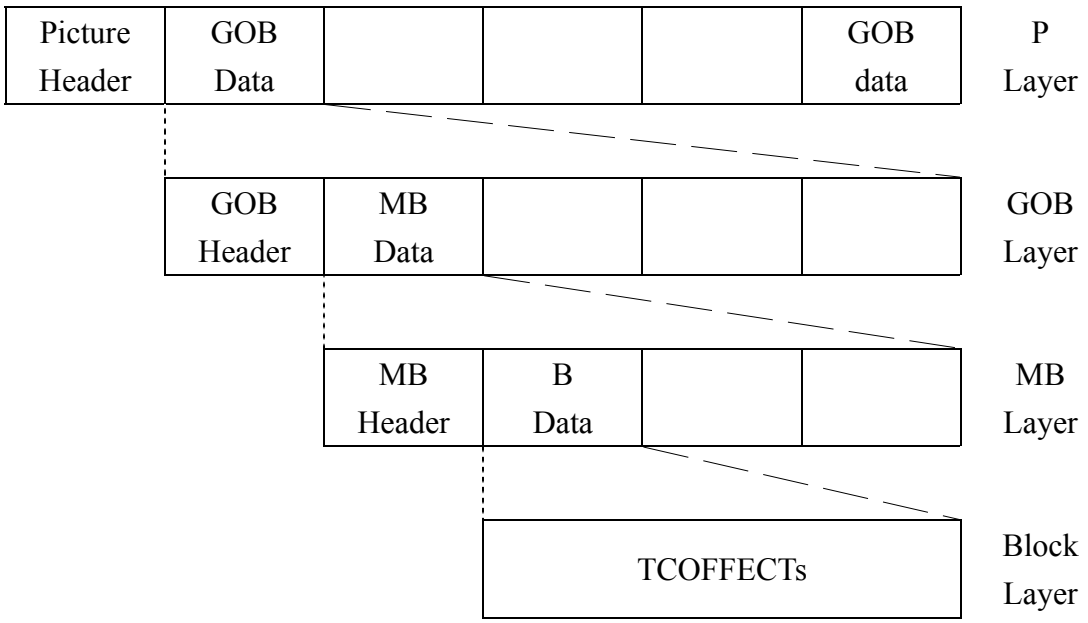


Figure 2.1 H.263 區塊組成架構

當然在不同的規格大小中，其各 GOB 區塊的組成大小也不盡相同。假設一張圖的大小為 176x144，其中每個 MB 的大小為 16x16，則每一張圖有 9 條 GOB，每個 GOB 有 11 個 MB，而每個 MB 則有 6 塊 8x8 的 block，所以每一張圖則有 594(9*11*6)個 block，Figure2.2 就是 block 在 MB 中的排序。

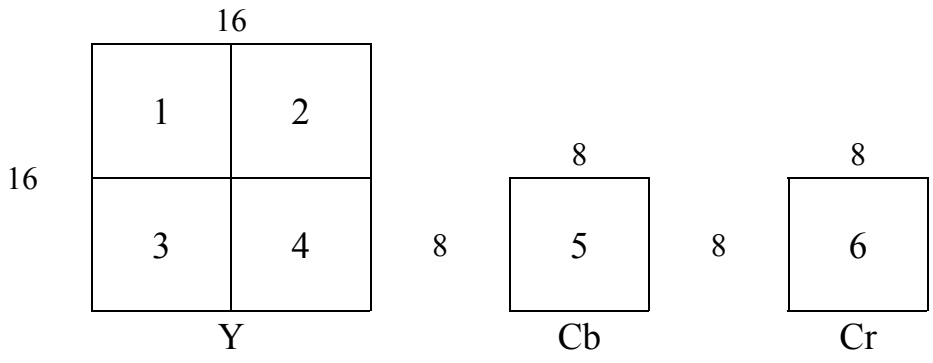
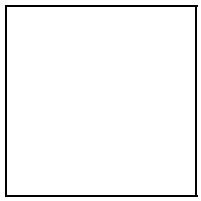


Figure 2.2 Y Cb Cr 與 6 塊 blocks 的關係

2.2.3 DCT 與 ZigZag

使用 DCT 的結果，會使得 block 圖像的 DC 值偏向於 block 的左上角，再用 ZigZag 的排列方式，排成一串頭重腳輕的資料流，這樣一來，之後我們使用 vlc 編碼時，由於資料流後面的值太小而成為一連串的 Zero，而可以編成少量的碼，以下就是當一個 block(8x8)當成一個矩陣的作法。



with , v, x, y=0, 1, 2,..., 7

where x,y = spatial coordinates in the pixel domain,

u,v = coordinates in the transform domain,

C(u) = 1/ $\sqrt{2}$ for u = 0, otherwise 1,

C(v) = 1/ $\sqrt{2}$ for v = 0, otherwise 1.

1	2	6	7	15	16	28	29
3	5	8	14	17	27	30	43
4	9	13	18	26	31	42	44
10	12	19	25	32	41	45	54
11	20	24	33	40	46	53	55
21	23	34	39	47	52	56	61
22	35	38	48	51	57	60	62
36	37	49	50	58	59	63	64

Figure 2.3 ZigZag 排列

Figure 2.3 為 ZigZag 排列表，排完後的資料流，前面是左上角的數值，後面則是右下角的數值。

2.2.4 量化(Quantization)

一個 block(8x8)經過 dct 和 zigzag 的轉換後，形成一個數值都集中前面的資料流，如果再經過量化處理就會使得數值變小，資料流後頭的數值都因太小而被量化作 zero,而使得後來的編碼更為減少。還原時一定要把量化值放在 H.263 檔案的檔頭，以便作反還原之用，還原結果當然會與原始碼有所差距，但如果量化值取得適當，其畫面不僅可讓人接受，其壓縮出來的碼也會相當的少，在每個 MB 中還可以在不同的情況下有自己的有自己的量化值。量化值的範圍為 1 至 31，值愈小畫面愈好，壓出來的碼也愈多。Figure 2.4 是個量化的例子:

QUANT=8

827	-12	11	9	0	0	0	-10
-9	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

經過 DCT、ZigZag 的 block

103	-1	1	1	0	0	0	-1
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

再經過 quantization

Figure 2.4 量化的範例

2.2.5 半像素(half-pixel)

每個影像都是由一個一個像素所組合而成的，但由於在處理 P frames 找尋 motion vector 時，我們可利用半像素使得 motion vector 以"0.5"為基本單位，來作更精細的調整，可是在原本影像中，顏色資訊是以數位化的方式儲存，不可會有"半點"，產生在點與點中間，此

時我們要利用 H.263 的標準公式來幫我們作圖像的擴張，這樣一來就會使得原本 176x144 的影像變成 352x288 大小的影像，這樣一來二點中間就會產一個像素，我們稱之為"半像素”。

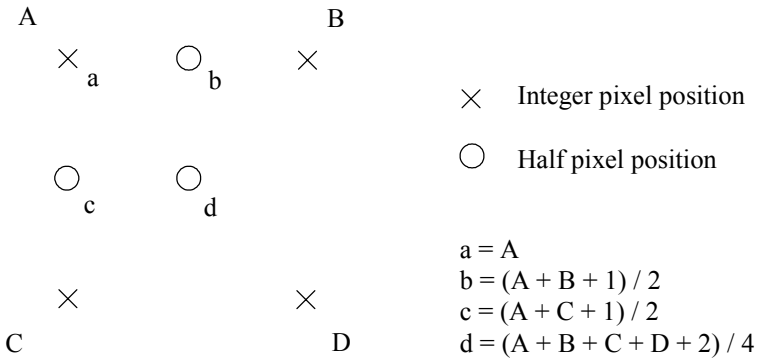


Figure 2.5　H.263 標準的內差法

因半像素是用來使 motion vector 能精準，而在 H.263 只在亮度(luminance)上尋找 motion vector，所以下圖片以灰階來表示。

Figure 2.6 經過內差後的圖

Figure 2.6 左圖是經過重建後的影像，右圖是經過公式計算後的影像。

2.2.6 移動向量與參考向量

在 inter-frame 編碼中，是利用了鄰近的二張畫面中，會有相當程度上的類似，所以將第二張畫面中，以大區塊(16x16)為單位與前一張的重建後的畫面作預估，並且判斷其大區塊的編碼方式，若有在第一張重建後的畫面中找到接近的大區塊，則可以針對移動向量與參考向量的差異量和二個大區塊的差異量作編碼，以達到影像壓縮目的。以下分別介紹"移動向量"、"參考向量"和"大區塊間的差異量"。

2.2.6.1 移動向量

如果在前一張重建後的影像中找到一個相似的大區塊(16x16)，則將(X1,Y1)－(X2,Y2)作為移動向量。

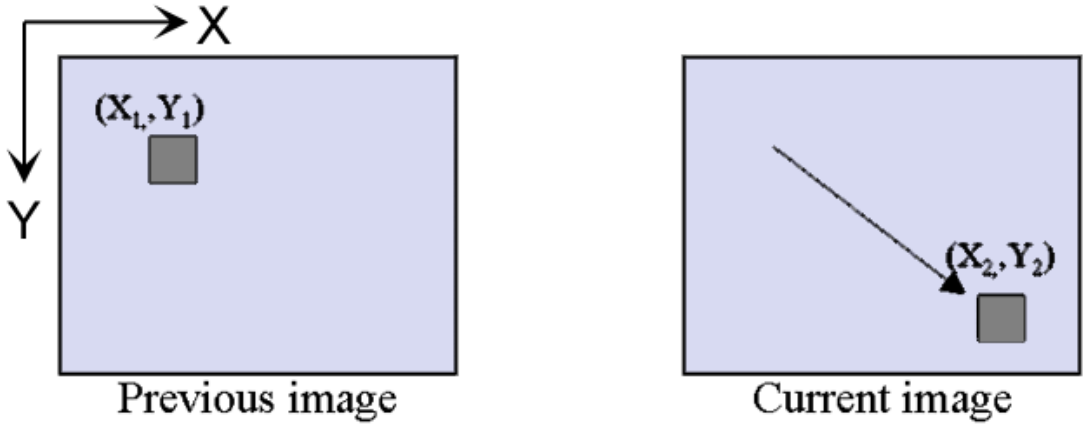


Figure 2.7 移動向量示意圖

2.2.6.2 參考向量

在一般的影像中，全大部份的畫面是連續的，利用這樣的特點，在編位移向量前，利用大區塊附近的向量，做為參考向量，再將移動向量減去參考向量，若區塊移動的方向及大小和附近的區塊相似，則

其要傳的位移向量幾乎為零，這樣的作法可以增加不少壓縮率。參考向量的作法是，分別對 MV1、MV2、MV3 的 X、Y 值，取出中間值作為參考向量；如果大區塊是在圖案的邊緣，則利用 Figure 2.8 的規則即可。

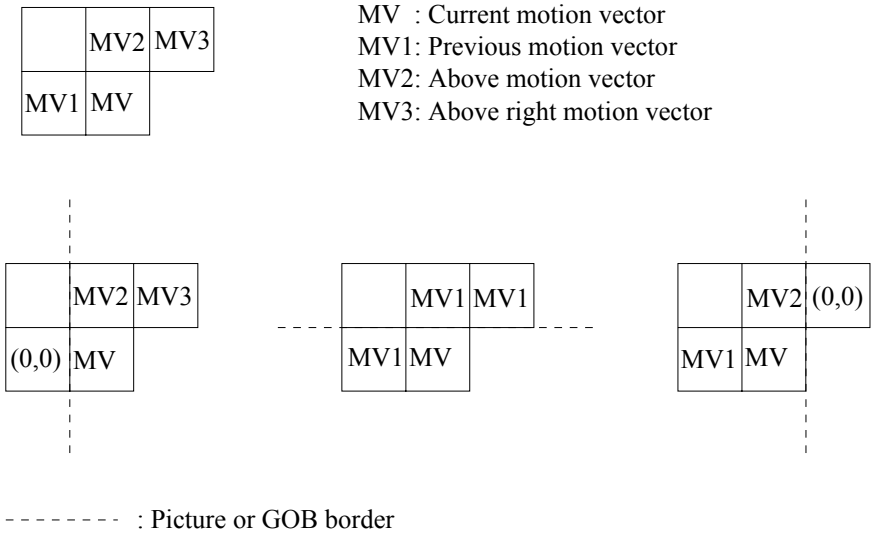


Figure 2.8 參考向量示意圖

2.2.6.3 大區塊間的差異量

雖然找到了 motion vector，但這只是代表著這二個大區塊之間的亮度差異度比較小，為了彌補其差異，待 motion vector 傳送出去，再將現在的大區塊減去前一張的大區塊，所得出來的值用 VLC 編碼傳送。解碼時，只要將其差異加回前一張的大區塊，就可以相當近似原資料，用此方法可使畫質不會太差。

2.2.7 四種選擇性編碼模式

2.2.7.1 無限制的移動向量模式(UMV mode）

在 H.263 的基本預測模式中，移動向量被限制不能指到畫面外，如此的限制造成在畫面邊緣的大區塊，可能無法找到最佳的預測區

塊。而在這個選項模式中，則允許移動向量指到畫面外面，而畫面外不存在的像素，就用邊界像素來取代。如此一來，即使在畫面邊緣有物體移動產生，也可得到不錯的預測效果。

擴展的像素值如何定義呢？我們以 QCIF 的灰階訊號（Y）說明，首先定義如下之式子：

Rumv(x, y) = R(x', y')

其中

x, y, x', y' = 在像素域（Pixel domain）的空間座標

Rumv（x, y）=參考畫面（Reference picture）在（x, y）位置的像素值（Pixel value）

R（x', y'）=參考畫面在（x', y'）位置的像素值

其中

$$x = \begin{cases} 0 & \text{if } x < 0 \\ 175 & \text{if } x > 175 \\ x & \text{otherwise} \end{cases}$$

$$y = \begin{cases} 0 & \text{if } y < 0 \\ 143 & \text{if } y > 143 \\ y & \text{otherwise} \end{cases}$$

此外，使用這個選項讓移動向量的預測範圍增大，水平和垂直移

動向量值的範圍由（-16, 15.5）增加變為（-31.5, 31.5），所以這個選項更適合於攝影機產生移動 (camera moment) 或較大的畫面格式 (如 4 CIF 或 16 CIF) 的情形下。

2.2.7.2 結構式算數編碼模式(SAC mode）

在這個選項中是在無失真（lossless）的編碼方式中，以算術編碼模式（Arithmetic Coding）來代替 H.263 原來的可變長度編碼（Variable Length Coding），使用這個選項可以提高編碼效率，其缺點是運算相當複雜。

2.2.7.3 進階預測模式（AP mode）

在這個選項中採用了重疊區塊移動補償（Overlapped Block Motion Compensation, OBMC），重疊區塊移動補償只針對亮度（luminance）區塊做處理，使用重疊區塊移動補償可以減少區塊效應（block artifact），提高畫面品質。此外，並且可以採用 4 個 8x8 區塊的移動向量來代替原來只有一個 16x16 大區塊的移動向量，雖然 4 個移動向量使用較多的位元，但是卻能預測更好的重建畫面。若同時使用 Unrestricted Motion Vector mode，這 4 個 8x8 區塊的移動向量也允許超出圖形的邊界及擴展移動向量。這個選項如果和 PB-frames mode 一起使用，則重疊區塊移動補償僅使用在 P-picture，而不使用在 B-picture 中。

2.2.7.4 PB 畫面模式（PB-Frames mode）

一組 PB-Frames 是由一張 P（Prediction）畫面和一張 B（Bi-directional）畫面所構成的，這張 P 畫面是由前一張解碼後的 P 畫面所預測出來的，至於 B 畫面，則是由前一張解碼後的 P 畫面和目前這一組中被解碼過的 P 畫面所預測出來的，之所以取名為 B 畫面是因為它能雙向地由前後兩張 P 畫面來預測自己的畫面，如圖 2.6 所示。

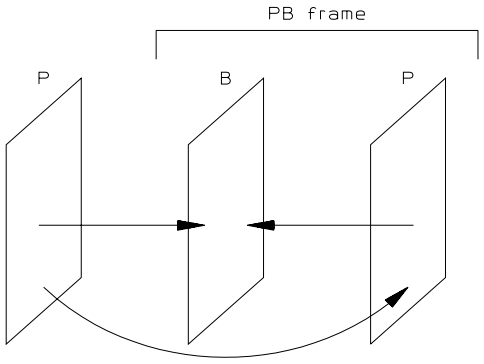


Figure 2.9：PB-Frames 之說明

在一組 PB-Frames 中 P 畫面產生一個大區塊，B 畫面也緊跟著產生一個大區塊，所以若使用 PB-frames mode，在一個大區塊的階層中，包含 12 個區塊（前面 6 個區塊屬於 P 畫面，後面 6 個區塊屬於 B 畫面），這一點，不同於 MPEG 產生一張 P 畫面之後，再產生 B 畫面。所以在 H.263 中把 PB-Frames 當做一組來處理，可以減少一些標頭（Header）所用的位元率，有效地降低位元率。

2.2.8 H.263 壓縮流程

2.2.8.1 I 畫面的壓縮流程

接下來介紹的流程，只用於基本的壓縮，也就是第一張是 I 畫面，接下來的就是 P 畫面，所以我們將分別說明。

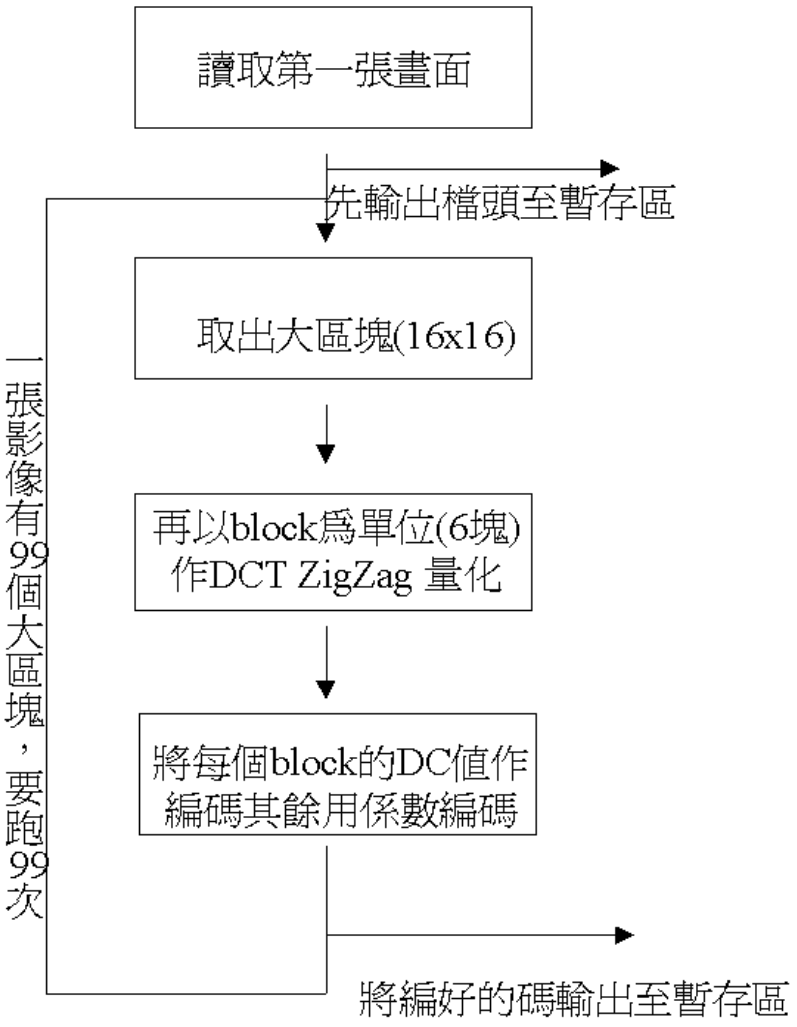


Figure 2.10 I 畫面的壓縮流程圖

2.2.8.2 P 畫面的壓縮流程

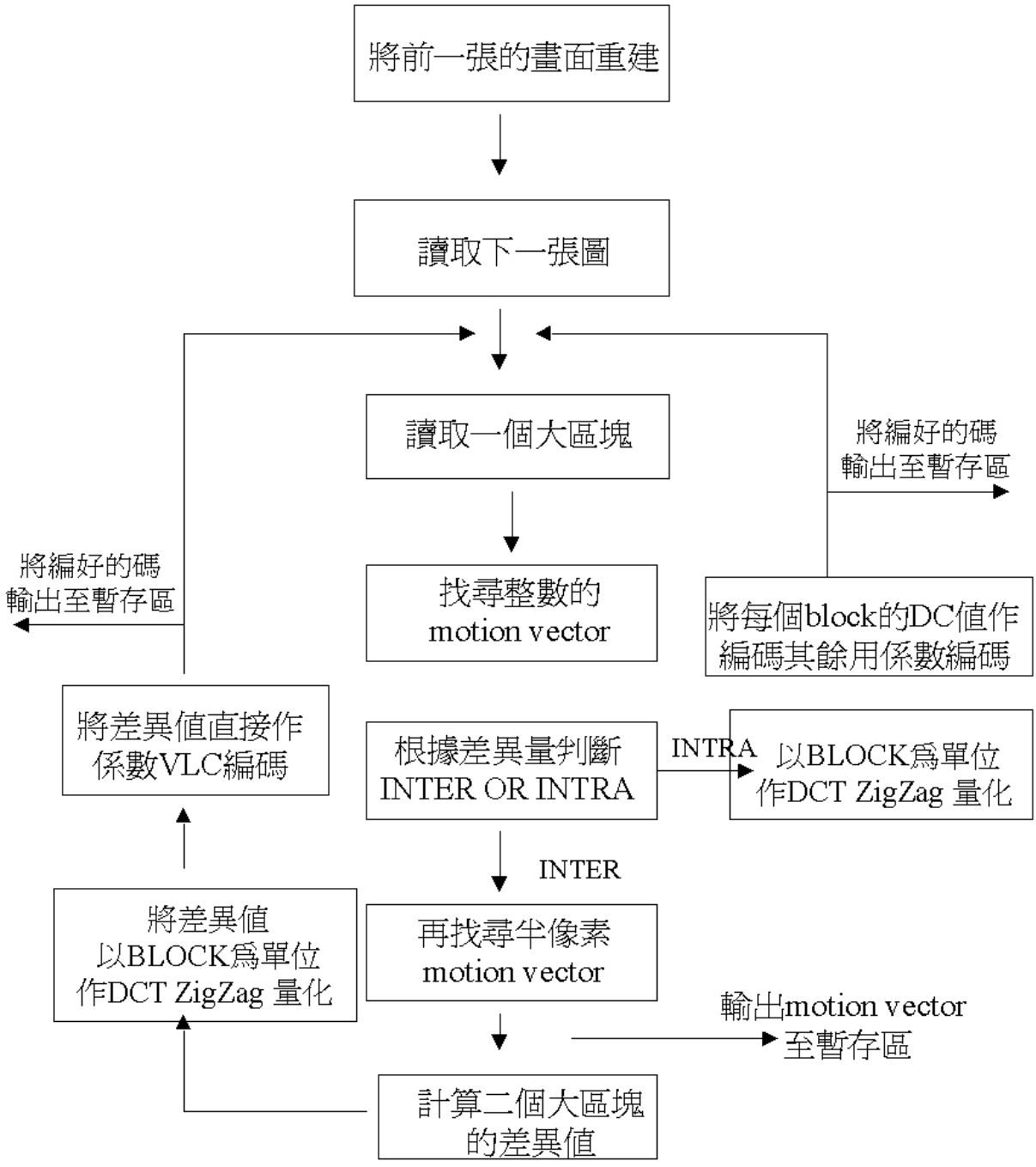


Figure 2.11 P 畫面的壓縮流程圖

2.2.8.3 全部壓縮流程

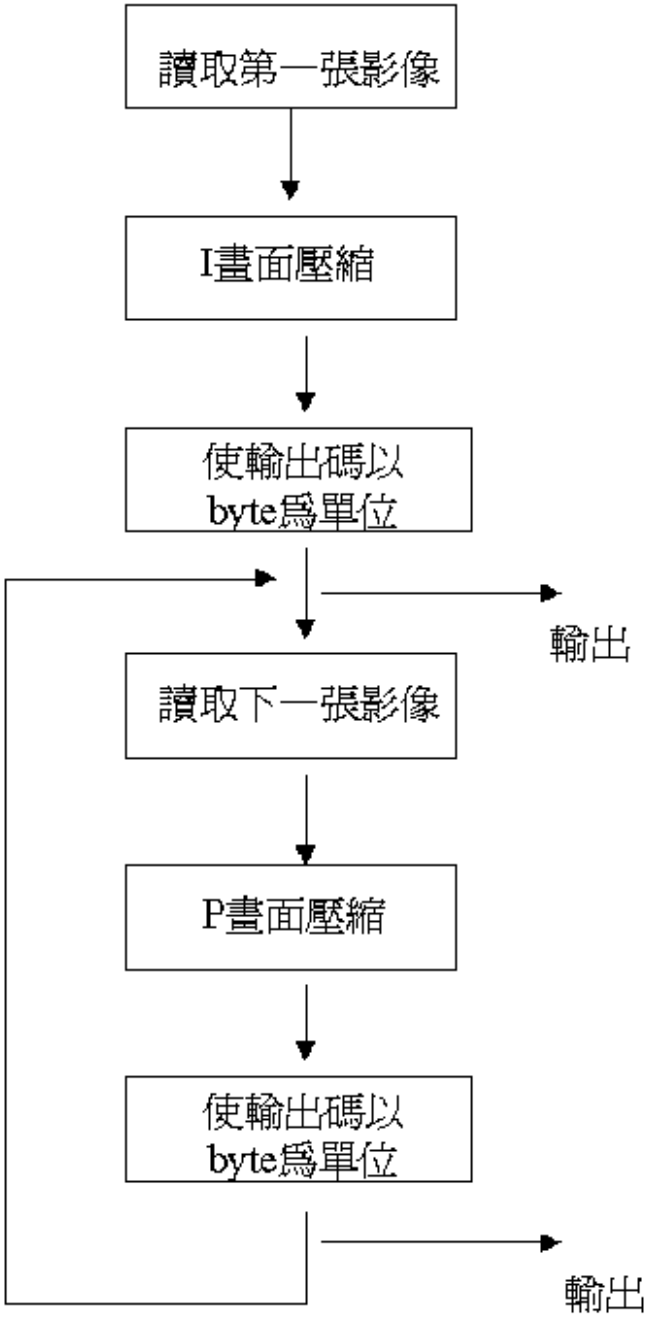


Figure 2.12 全部壓縮流程圖

2.3 H.26L 視訊壓縮標準簡介

緊接 H.263 之後，在西元 1998 年，國際電信組織 (ITU-T) 已經開始著手制定下一代的視訊標準，H.26L(Long Term)[5]，其目標有以下幾點：

- 具高壓縮率的效能

  和 H.263 比較，以相同的品質，但只需要其不到一半的位元率。

- 高畫質的應用[6]

  不但能用在低位元率的壓縮，而且在高位元率所壓縮出的位元率，皆比之前的視訊壓縮標準低，並且畫面品質與先前的差不多。

- 具計算複雜度的可調適性(Complexity scalability in encoder and decoder)

  可調適於畫面的品質及編碼器的複雜度，因所用的計算機的能力而有所區分。

- 具有高抗錯的能力(Error Resilience)

- 友善的網路傳輸設計

- 詳細定義解碼器規格

  不會與編碼器有不協調的狀況發生(no mis-match)。

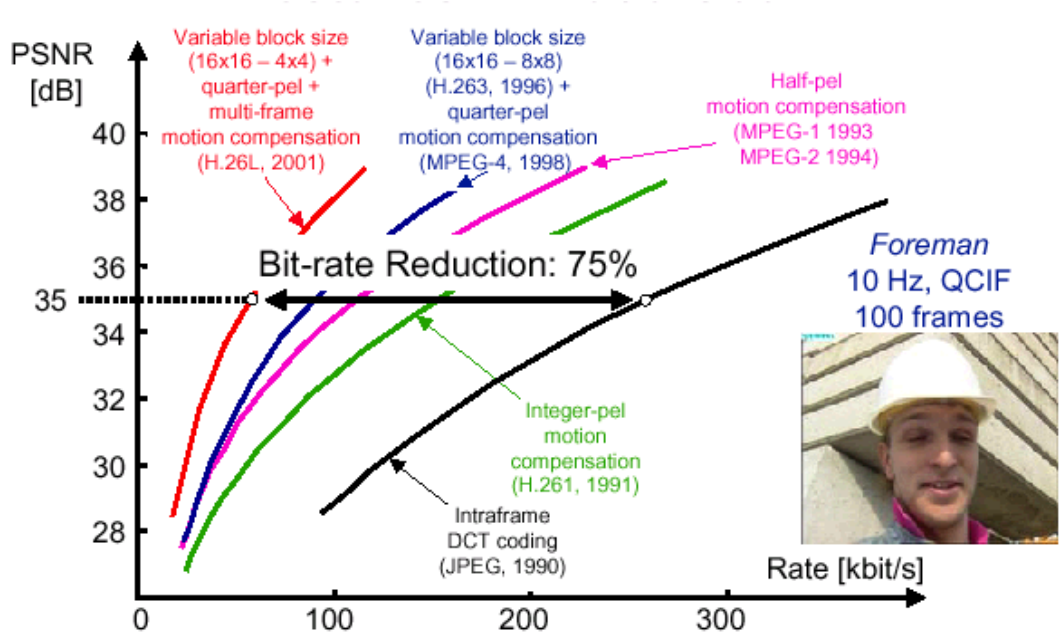


Figure 2.13 視訊壓縮發展示意圖

From Thomas Wiegand, Heinrich-Hertz-Institute Berlin

高壓縮率

由 Figure 2.13 可一覽視訊壓縮標準的發展史，依序是

- Intraframe DCT coding (JPEG, 1990)
- Integer-Pel Motion Compensation (H.261, 1991)
- Half-Pel Motion Compensation (MPEG-1, 1993; MPEG-2, 1994)
- Quarter-Pel Motion Compensation(MPEG-4, 1998) + Variable Block Size <16x16 和 8x8> (H. 263, 1996)
- Quarter-Pel + Multi-Frame Motion Compensation + Variable Block Size <16x16 - 4x4> (H. 26L, 2001)

在 1990 年，最早的視訊壓縮，是用 JPEG 一張張的圖來做壓縮，後來有 H. 261[1]用 Block based matching Integer-pel motion compensation，由 Figure 2.13 可觀察到，增進移動預估精確度確實可以在一定畫面品質之下，壓縮的位元率比先前的降低了有

30%-40%，而到了 1993、1994 年的 MPEG-1(VCD)、MPEG-2(DVD)， 再用更精細的移動預估，精細到 1/2 像素為單位去尋找位移向量，到了 98 年的 MPEG-4[4]，再精細到 1/4 像素，壓縮的位元率比 JPEG 壓縮降低了有 50%-60%，由以上的視訊壓縮發展過程，可讓我們了解一件事，**位移預估演算法若做的恰當，會使整個視訊壓縮的效能達到最好**。另一方面，若將搜尋的區塊縮小(16x16 變成 8x8 或 更小的 4x4)，或許可以找到比原先 16x 16 更像的區塊，可以增進編碼的效率； 但也未必是小的區塊就最省，因為這也要看影像是那一類的，是平滑的或是複雜的，所以如果大小區塊都找，在配合一些 Rate-Distortion[8][9]的機制和之前的 1/4 Pixel 的 motion compensation，就是最基本的 H. 26L 的樣子。若要再增進壓縮的效率，就要利用到編碼之前的好幾張的畫面，利用這些畫面，可以找到一個和現在這個區塊最像的，特別是對動作較大的影片，效果最好，所以 H.26L 位元率與最初利用 JPEG 壓縮的來比，大概降低有 75%的位元率。

Figure 2.14 是 H.26L 整體的架構圖，以下就開始介紹 H. 26L 與之前的標準不同之處，介紹的順序，會依據編碼的順序，內容會包含 Intra/Chroma/Inter prediction mode、Transform coding：4x4 Integer Transform、VLC：Universal VLC、Zig-zag scan

/Quantization、Motion Estimation： High/Low Complexity。

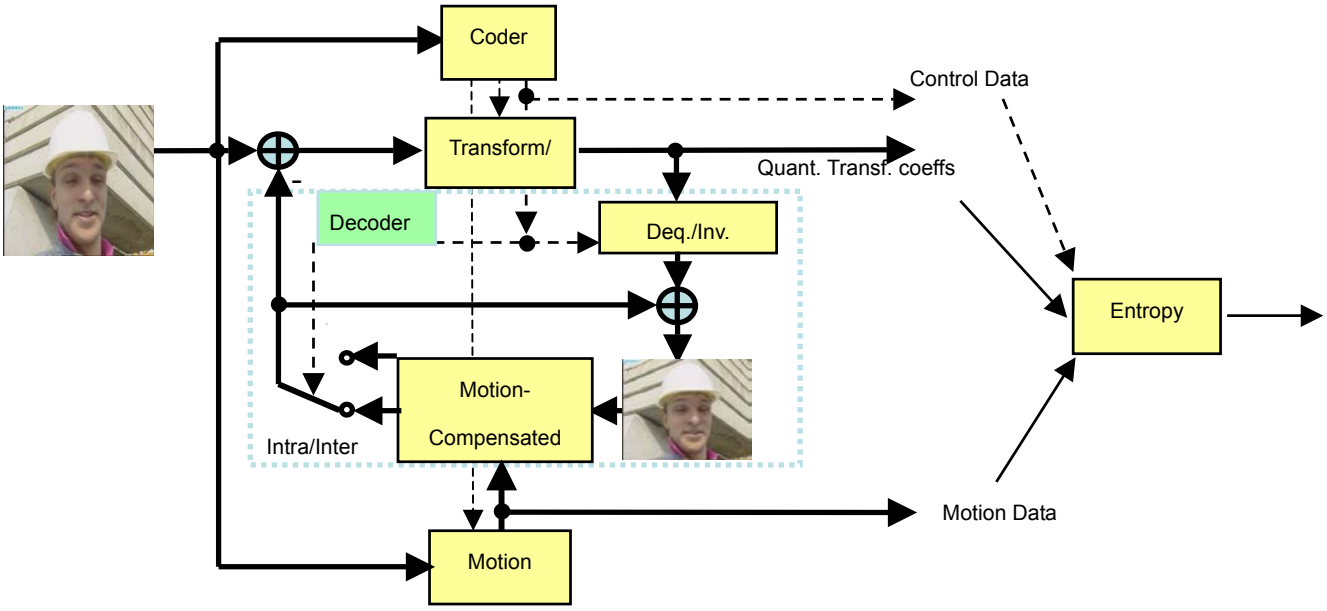


Figure 2.14視訊標準H.26L結構圖
From Thomas Wiegand, Heinrich-Hertz-Institute Berlin

2.3.1 Intra/Chroma/Inter 預測模式:

2.3.1.1 Intra prediction mode:

分別為 4x4 及 16x16 兩種區塊，4x4 又有六種不同的模式

(Mode)，分別是：

0. DC
1. Diagonal Vertical 22.5deg
2. Vertical
3. Diagonal 45deg
4. Horizontal
5. Diagonal Horizontal -22.5deg

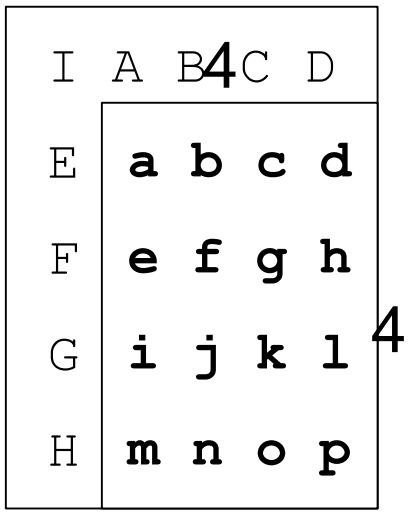


Figure 2.15

Intra Prediction mode 是利用其周圍已解壓縮的像素，即 Figure 2.15 中大寫 A-I 的像素，然後根據 Figure 2.16 的五個方向，也就是這五個模式，做預測，至於是那一個像素預測那一個，依據模式的不同，而有變化。

Mode 0: DC prediction

這種模式是利用，(A+B+C+D+E+F+G+H)/8，這八個點的平均值，然後利用此平均值，減去小寫 a-p 的像素值，再利用剩下的差異值，去編碼。假若區塊上方或左方的四個像素都沒有的話，就單用左方或上方的四個像素值做平均再相減，若左方和上方都沒有的話就用 128 去相減。

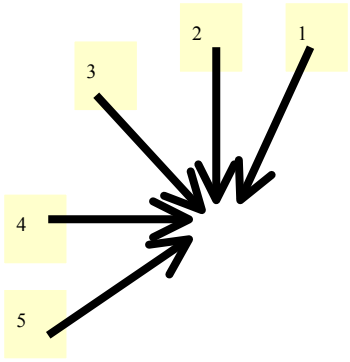


Figure 2.16

Mode 1: Diagonal vertical 22.5deg

a 是由 (A+B)/2來預測的。

e 是由 B來預測的

b,i 是 (B+C)/2來預測的。

f,m 是由 C來預測的。

c,j 是由 (C+D)/2來預測的。

d,g,h,k,l,n,o,p 是由 D來預測的

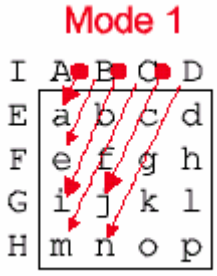
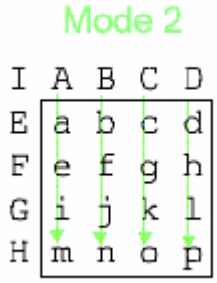


Figure 2.17



由Figure 2.17的示意圖可表達出以上所陳述的。

Mode 2: Vertical

Vertical mode 一定要有大寫的 A-D 周圍四個點。由 Figure 2.18 示意圖可看出，利用 A-a、A-e、A-i、A-m 得到差值，進行編碼，其它的行(B、C、D)依此類推。

Mode 3: Diagonal 45deg

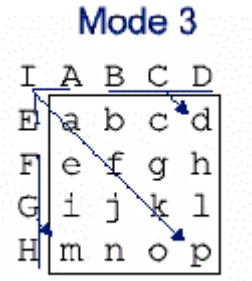


Figure 2.19

m 是由 (H+2G+F)/4來預測的。

i,n 是由 (G+2F+E)/4來預測的。

e,j,o 是由 (F+2E+I)/4來預測的。

a,f,k,p 是由 (E+2I+A)/4來預測的。

b,g,l 是由 (I+2A+B)/4來預測的。

c,h 是由 (A+2B+C)/4來預測的。

d 是由 (B+2C+D)/4來預測的。

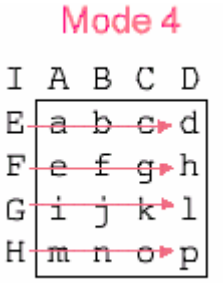


Figure 2.20

由Figure 2.19的示意圖可表達出以上所陳述的。

Mode 4: Horizontal

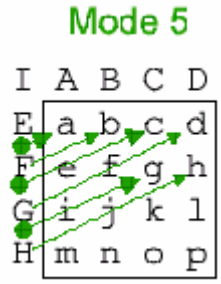


Figure 2.21

Horizontal mode 一定要有大寫的 E-H 周圍四個點。由 Figure 2.20 示意圖可看出，利用 E-a、E-b、E-c、E-d 得到差值，進行編碼，其它的行(B、C、D)依此類推。

Mode 5: Diagonal horizontal -22.5deg

a 是由 (E+F)/2來預測的。

b 是由 F來預測的。

c,e 是由 (F+G)/2來預測的。

f,d 是由 G來預測的。

i,g 是由 (G+H)/2來預測的。

h,j,k,l,m,n,o,p 是由 H來預測的。

由Figure 2.22 的示意圖可表達出以上所陳述的。

另外 16x16 區塊有 4 個模式

1. Vertical
2. Horizontal
3. DC prediction
4. Plane prediction

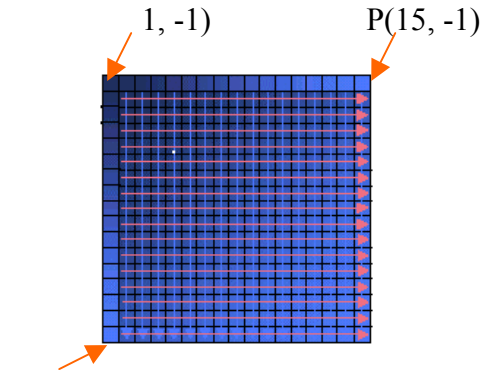


Figure 2.22

利用的技術和 4x4 區塊的方法一樣，只是在第四個模式，是用類似漸層的方式，以下是它的數學式子。

$$Pred(i,j) = [a + b(i-7) + c(j-7) + 16]/32$$

Where:

- $a = 16 \times [P(-1,15) + P(15,-1)]$
- $b = 5 \times (H/4)/16$
- $c = 5 \times (V/4)/16$
- $H = \sum_{i=1}^{8} i \times (P(7+i,-1) + P(7-i,-1))$
- $V = \sum_{j=1}^{8} j \times [P(-1,7+j) + P(-1,7-j)]$

Mode 4 的每一個預測的值都不同，所以這個式子的 i 和 j 就代表 16x16 區塊中的 X、Y 軸，a、b、c、H 和 V 是這個 mode 的參數。

2.3.1.2 Chroma prediction mode:

在 Figure 2.23 中，S0、S1、S2 和 S3 各是 A 的上方左方、B 的上方、C 的左方最鄰近的四個點，加起來並平均。假若 S0、S1、S2 和 S3 都存在的話，

A = (S0 + S2 + 4)/8

B = (S1 + 2)/4

C = (S3 + 2)/4

D = (S1 + S3 + 4)/8

若只有 S2 和 S3 存在的話，

A = (S0 + 2)/4

B = (S1 + 2)/4

C = (S0 + 2)/4

D = (S1 + 2)/4

若只有 S0 和 S1 存在的話，

A = (S2 + 2)/4

B = (S2 + 2)/4

C = (S3 + 2)/4

D = (S3 + 2)/4

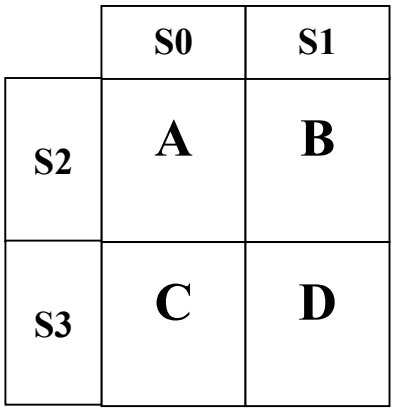


Figure 2.23 Chrominance 區塊預測示意圖

最後，如果剛好是這四個都不存在，A=B=C=D=128，這四個 A、

B、C、和 D，就是預測值，預測值減去原先的像素值，就是要拿來編碼的值。

2.3.1.3 Inter Prediction mode:

在 Inter mode 中，有七種不同的區塊大小可以用來做位移補償(Motion Compensation)，其中在一個 Macroblock 中，依不同的區塊包含有 1、2、4、8 或 16 個位移向量。一個 Macroblock 分成小的子區塊的方式，如 Figure 2.24 所示。

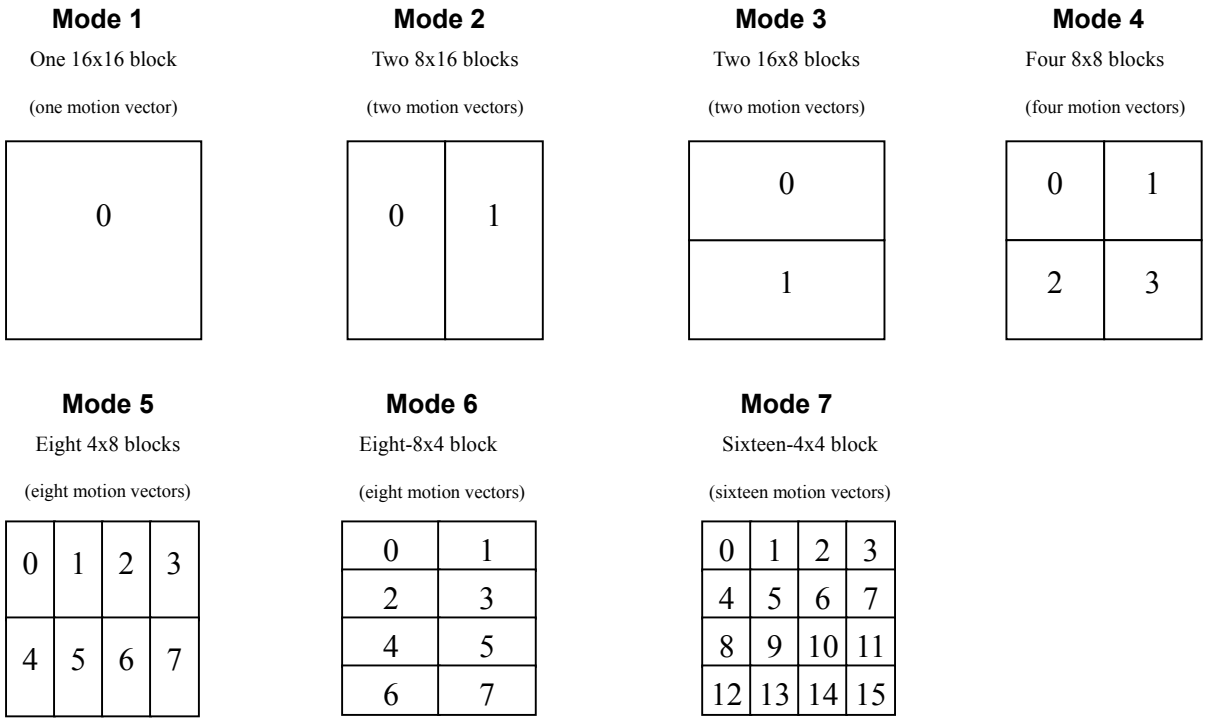
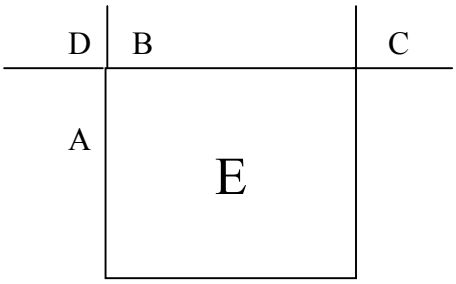


Figure 2.24　Inter coding 7 個區塊模式示意圖

我們可以清楚的看到，雖然小的區塊，具有比較大的彈性可以找到一個與它最像的區塊，但它必須利用 16 個位移向量才可以完成一個 Macroblock，所以在決定要用那一種模式之前，系統必須將所要花

費編位移向量的位元併入考量，這樣才能達到低位元率的效果。



- A 是在 E 區塊的左方靠上區塊的位移向量
- B 是在 E 區塊的上方靠左區塊的位移向量
- C 區塊的對角線右上區塊的位移向量
- D 區塊的反對角線左上區塊的位移向量

Figure 2.25 Inter coding 參考向量示意圖

為了讓位移向量的編碼效率增加，在每個子區塊所找到的位移向量，會做適當的預測。如果區塊是正方形的(4x4、8x8 及 16x16)，會利用 A、B、C、D 取中間值的方法，來與 E 取差值，進行編碼。倘若 A、B、C 及 D 位移向量剛好在圖片的邊緣，則可能會有不存在的，所以系統有以下的替代方案：

1. 如果 A 及 D 位移向量都不存在，就將這兩個位移向量的值設為 0。
2. 如果 D、B 及 C 位移向量都不存在，預測值就用 A 位移向量。
3. 如果 C 位移向量不存在，就用 D 取代 C。

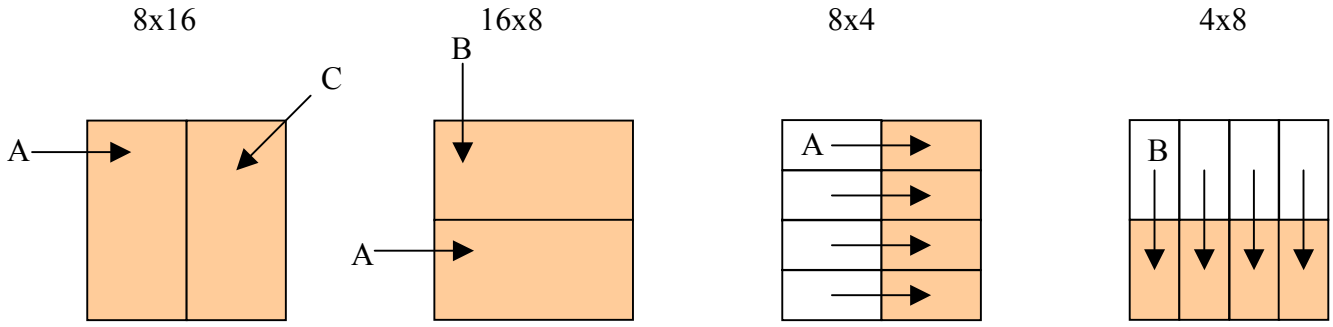


Figure 2.26 Inter coding 參考向量方向示意圖

若區塊是長條形的(8x16、16x8、4x8 及 8x4)，系統會採取方向性的預測，現在就逐一以 Figure 2.26 來說明。

8x16 區塊：

左半邊的區塊，用其左上方的 A 位移向量來預測，右半邊的區塊，用其對角線右上方的 C 位移向量來預測，如果 A 或 C 位移向量不是與 E 位移向量同一個畫面找到的，那就會採用 A、B、C、D 取中間值的方法。

16x8 區塊：

上半邊的區塊，用其上方左邊的 B 位移向量來預測，下半邊的區塊，用其左上方的 A 位移向量，如果 A 或 B 位移向量不是與 E 位移向量同一個畫面找到的，那就會採用 A、B、C、D 位移向量取中間值的方法。

8x4 區塊：

左半邊四個白色的區塊，是用其附近 A、B、C、D 位移向量取中

間值的方法，有顏色的部份是利用其左邊的 A 位移向量。

4x8 區塊

上半部四個白色的區塊，是用其附近 A、B、C、D 位移向量取中間值的方法，有顏色的部份是利用其上方的 B 位移向量。

假如，方向性預測區塊的預測值有不存在的，就同樣利用正方形區所採用的方法。

2.3.2 Transform Coding

2.3.2.1 4x4 Integer DCT Transform

不論是 Intra 或 Inter mode，殘餘的訊號利用 Integer DCT (見 Figure 2.27) 進行訊號能量的集中。它的功能如同一般的 DCT，不同的是它在反轉換的時候，在解碼端得到的結果和編碼端相同，這樣就不會產生 Mis-match 的結果。過往 DCT 轉換都是做在 8x8 區塊上，如今將轉換做在 4x4 區塊上，可降低在轉換上的計算的複雜度，並且在畫面的邊緣少一些編碼的誤差。

1-D Integer Transform	1-D Inverse Integer Transform
A = 13a + 13b + 13c + 13d	a' = 13A + 17B + 13C + 7D
B = 17a + 7b - 7c - 17d	b' = 13A + 7B - 13C – 17D
C = 13a - 13b – 13c + 13d	c' = 13A – 7B – 13C + 17
D = 7a - 17b + 17c - 7d	d' = 13A – 17B + 13C - 7D

Figure 2.27 1-D Integer Transform and Inverse Transform

利用一維的 Integer 轉換，可在水平的像素上做一次，再在垂直的像素上做一次，即可得到轉換過後二維的係數。而反轉換也是利用同樣的原理，只是反轉換出來的值是先前的 676 倍(ex. a' = 676a)。

2.3.2.2 2x2 transform of chroma DC coefficients

之前 4x4 Integer 轉換是用在亮度(Luma)上，H. 26L 特別為了提昇彩度(Chroma)部份品質，彩度部份在編碼時，其在 U 及 V 的部份各有 4 個 DC 值，將各自 4 個 DC 值，再做一次轉換，而有 2x2 transform for chroma (見 Figure 2.28 與 Figure 2.29)。

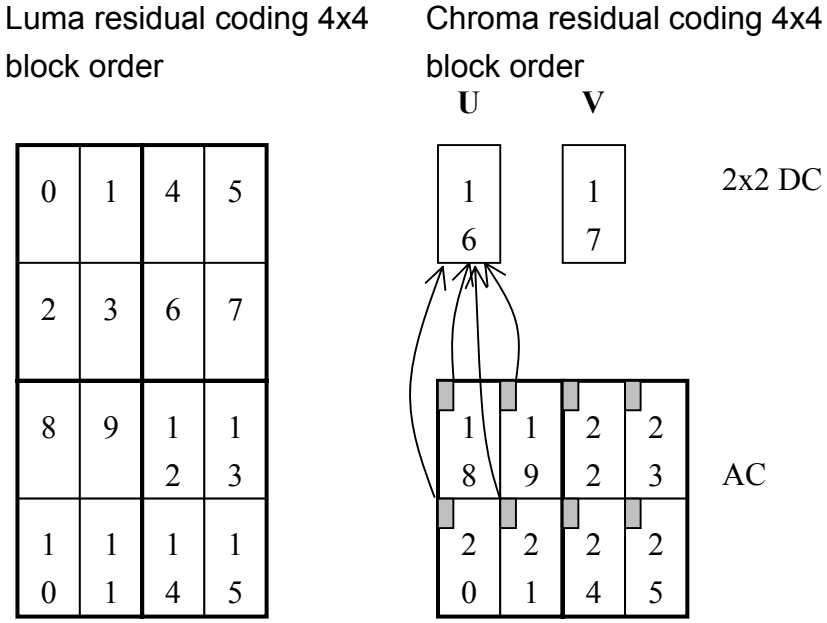


Figure 2.28 Luminance and Chrominance 編碼示意圖

1-D 2x2 transform for DC Chroma

DCC(0,0) = (DC0+DC1+DC2+DC3)/2

DCC(1,0) = (DC0-DC1+DC2-DC3)/2

DCC(0,1) = (DC0+DC1-DC2-DC3)/2

DCC(1,1) = (DC0-DC1-DC2+DC3)/2

1-D 2x2 Inverse transform for DC Chroma

DC0 = [DCC(0,0)+ DCC(1,0)+ DCC(0,1)+ DCC(1,1)]/2

DC1 = [DCC(0,0)- DCC(1,0)+ DCC(0,1)- DCC(1,1)]/2

DC2 = [DCC(0,0)+ DCC(1,0)- DCC(0,1)- DCC(1,1)]/2

DC3 = [DCC(0,0)- DCC(1,0)- DCC(0,1)+ DCC(1,1)]/2

Figure 2.29 Chrominace DC Transform

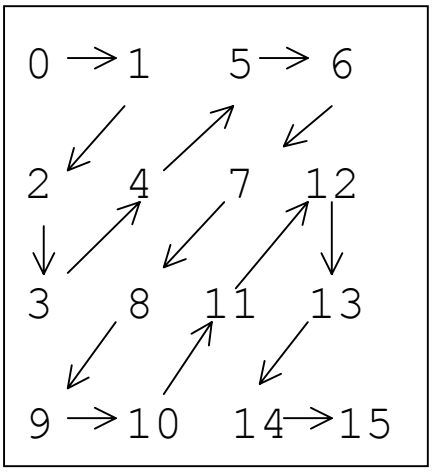


Figure 2.30 Simple Scan

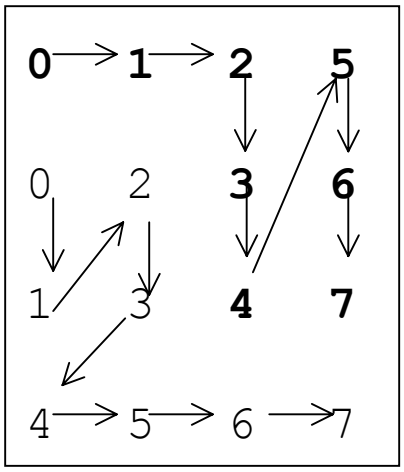


Figure 2.31 Double Scan

2.3.2.3 Zig-zag scan /Quantization

Simple Scan 使用的時機，除了 Intra coding 亮度(luma)QP<24 的部份之外，系統都用此形式做 Run Length Coding，如 Figure 2.30。

當系統使用 Simple Scan 在一些 Inter blocks 和 Intra blocks (使用在較大 QP 值) 去編 VLC 時，從機率的概念來說，用一個位元編 EOB(End of Block)是最適合不過的。但在 4x4 區塊 Intra coding 中，平均會有一個以上非零的 Coefficients，所以用一個位元去編 EOB，反而就不是最好的設計。為了要解決此一問題，改成一個 4x4 的區塊中使用 Double Scan，如 Figure 2.31。

QP_{luma}	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
QP_{chroma}	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	17	18	19	20	20	21	22	22	23	23	24	24	25	25

Figure 2.32 量化對照表

在量化及反量化過程中，在亮度中所用 QP_{Luma} 值的間隔不是用等距的，而是以每個 Step Size 以 12% 增加，換言之，每增加 QP 值在六個 Step size 之後，QP 值間距比原先增一倍。量化的過程並不採用的 Dead zone 的方式量化。

另外在彩度方面，因為考量人的眼睛對彩度並不是很敏感，所以在 QP_{chroma} 等於 17 之後，採非等間距的方式 (詳見 Figure 2.32)。

1
0 x_0 1
0 x_1 0 x_0 1
0 x_2 0 x_1 0 x_0 1
0 x_3 0 x_2 0 x_1 0 x_0 1

Figure 2.33 UVLC 示意圖

2.3.3 VLC：Universal VLC

UVLC[7]有以下幾個優點：

1. 編解碼有一定的規則，容易辨認其碼字
2. 具有抵抗錯誤的能力
3. 使編碼更有效率

由 Figure 2.33 可看到類似金字塔的架構，並且在每一層中有固

定的長度， $x_0\ x_1\ x_2\ ....$可以是 0 或是 1，而將 $x_0\ x_1\ x_2\ x_3$ 組合起來的二元碼字，就是在這一層的編號，若是要完全解碼，就要使用這個式子 Code_number = $2^{L/2}$ + INFO -1 (L/2 會判斷在第幾層，INFO 會指在這層中的編號是幾號，除了在 L = 1 時，INFO 必須要等於 0)。

換言之，我們可以利用此特性，就可辨別碼字，並且也可利用此隔一個位元就有 0，若有兩個 1 連續出現，那麼代表有錯，來加強其抗錯誤的能力。

2.3.4 Motion Estimation： High /Low Complexity

因 H.26L 視訊標準的壓縮率高，但也大量的增加計算複雜度，所以在編碼端有二種不同程度的複雜度的分別，低複雜度及高複雜度。

2.3.4.1 Low Complexity

低複雜度，以消耗較少資源計算量，達成壓縮的目的，以下敘述其詳細過程，說明過程分為 Intra 及 Inter 兩個部份：

Intra Prediction mode

Intra 4x4 mode decision：[Dc,Hor,Ver,Diag_RL,Diag_LR,Diag_45]

- 加上 qp 值*24 到 intra_sad 的計算中，主要是為了和 Inter prediction mode 比較，若是與 Intra16x16 比較，不必加此項。
- 將五種 Mode 用在 16 個 4x4 區塊中。

i. 用周圍區塊所用的模式(上及左)，預測現在區塊會使用哪個機率比較大，式子為
QP0(QP) × Order_of_prediction_mode
order of prediction_mode 要查 Figure 2.34，並且這樣的預測都是以二個相鄰的區塊方式進行。

ii. 加上每一個 block 中預測值和實際值相減所得的 SAD 值。

iii. 將以上 16 個區塊所算出來的 SAD 值加起來。

iv. 再做完 5 個 4x4 區塊模式後，再互相比較(current_intra_sad)。

- 決定了一個模式之後，再和 16x16Intra mode 比較。
- 利用剛才決定的模式，開始編亮度的部份。

B\A	outside	0	1	2	3	4	5
outside	0-----	021---	102---	201---	012---	012---	012---
0	045---	041352	104325	230415	304215	043152	043512
1	045---	014325	102435	203145	032145	041325	014352
2	045---	012345	102345	210345	302145	042135	013245
3	045---	304152	310425	231054	304215	403512	305412
4	405---	403512	401532	240351	430512	403512	405312
5	504---	540312	015432	201453	530412	450312	504132

Figure 2.34 Intra Coding 模式機率表

Intra(16x16) mode decision： [Dc, Hor, Ver, Plane]

- 先算四個 16x16 區塊模式，在做完預測之後的 SAD 值，會用 Hadamard transform。(會包含 16 個區塊 Hadamard transform 和 DC Hadamard transform)
- 取 MiniSAD Mode 當做 16x16 的模式。

- 若 16x16 區塊所得的 tot_intra_sad2 小於 4x4 的 tot_intra_sad 就開始編 16x16 亮度的部份。否則還是會沿用剛才 4x4 所編的碼，不做 16x16 區塊的編碼。

Inter mode decision：[copy,16x16,16x8,8x16,8x8,8x4,4x8,4x4]

- 加上 qp 值*min(ref,1)*2 到 Tot_inter_sad 主要是為了預測多張畫面而設計的 Weighting。
- Inter mode (7 個 mode, Integer search、half pixel、1/4 pixel)
    - i. Integer search(只是單純的用 SAD 來算並沒有用 Haramard Transform)
        1. Prediction biased (not center biased)
        2. Motion Vector cost (SAD+qp*MV_usebits， 如果是遇到 16x16 並且 candidate=(0,0)，減去 qp*16)
    - ii. Half、1/4 pixel search (若在 option 中有選 Hadamard，這裏全部都要用)
        1. 只搜尋周圍的八個點，見 Figure 2.35。
        2. Motion Vector cost(SAD+qp*MV_usebits 如果是遇到 16x16 且 candidate=(0,0)，減去 qp*16)

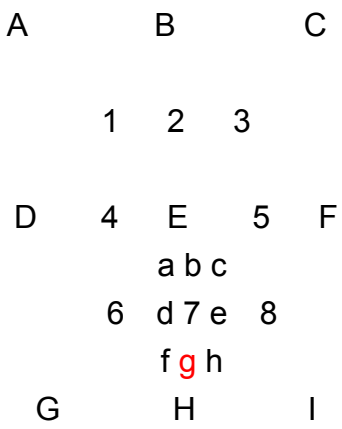


Figure 2.35 Interpolation 示意圖

2.3.4.2 High Complexity

H.26L 為了以更低的位元率達到更高的品質，採用高複雜度的編碼，以下敘述其詳細過程：

其使用的技術最重要的是用在亮度和彩度的 SSD(the Sun of Squared Differences)，及 Rate Distortion Lagrange model。

Initiate the parameter (Lagrange parameters)

1. Intra mode decision

$$\lambda_{MODE,P} = 5 \cdot e^{QP/10} \cdot \left(\frac{QP+5}{34-QP}\right)$$

2. Motion vector (Inter)

$$\lambda_{MOTION,P} = \sqrt{5} \cdot e^{QP/20} \cdot \sqrt{\frac{QP+5}{34-QP}}$$

Intra mode (4x4)

- 用 SSD 算 4x4 五種區塊模式中的失真(Distortion)值，並由最小的失真值，代表 4x4 區塊模式。
- 實際去編所得到的最佳模式(Y、U 及 V)，並記錄所花的位元。

- Lagrange_Intra_mode decision 乘上編完各區塊模式所得到的位元，再加上用 SSD 所算出的失真值，此值是為了和其它的 Intra16x16 及 Inter mode 比較。

Intra (16x16)

- 決定那個模式的方法是和 Low complexity 一樣
- 然後決定編的那個模式的 Y(luma)、U 及 V。
- 將實際編出來的 mode* Lagrange_Intra_mode+Y、U、及 V 的 Rate*Lagrange_Intra_mode(目的是要和其它的 Intra 4x4 及 Inter mode 比較)

Copy(原先的未經編過)

- 算其 YUV SSD(多張 Frame)以決定用哪一張。
- 然後去編碼(若是超過最小的 min_rdcost，就跳到其它的 mode 去)
- 將實際編出來的 mode* Lagrange_Intra_mode+Y、U、及 V 的 Rate*Lagrange_Intra_mode(目的是要和 Inter mode 比)

- Inter mode (7 modes)
  - tot_inter_sad = (int)floor(rdopt->lambda_motion * (1+2*floor(log(k+1)/log(2)+1e-10)));
  - 和 Low complexity search 的過程一樣
  - 然後去編(Y、U、V 若是超過最小的 min_rdcost，就跳到其它的 mode 去)

- 將實際編出來的 mode* Lagrange_Intra_mode+Y、U、及 V 的 Rate*Lagrange_Intra_mode

2.4 H.26L 複雜度分析

這個分析主要是針對低複雜度來做測試，基本上以下面的參數使用 Intel VTune Performance 測試軟體來測驗。

測試的環境

使用 Pentium III 733 桌上型電腦

RAM 128M

H.26L 參數 (Reference Software: TML8.0)

使用 Sequence Foreman

Image format: QCIF

Frame Skip 2 (總共 100 張 Frame)

Hadamard transform: Not used

Reference frames used in P prediction: 1

Sequence type: IPPP (QP: I=17, P=17)

Entropy coding method: UVLC

Search range restrictions: none

RD-optimized mode decision: Low Complexity

MV resolution: 1/4-pel

Blocktype: 7 modes

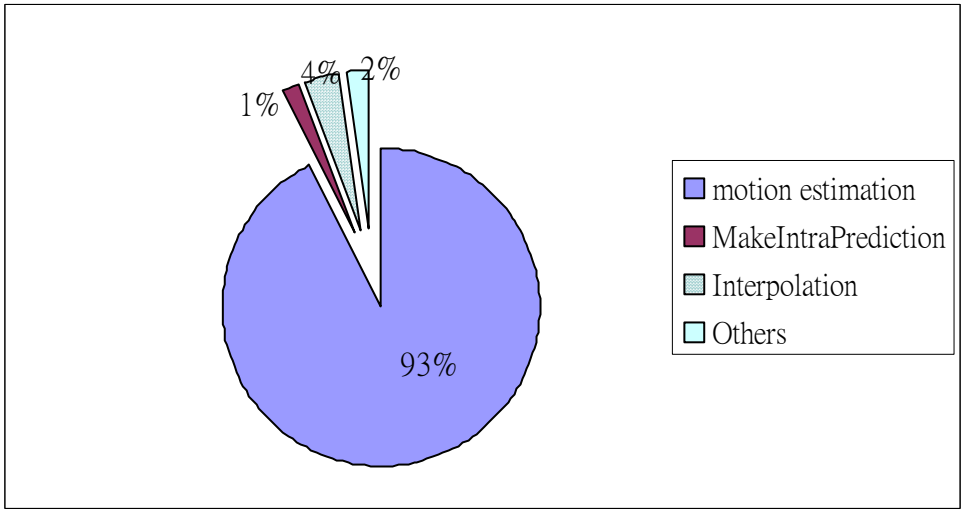


Figure 2.36　H.26L 整體複雜度評估圖

由 Figure 2.36 的百分比圖可看出，Motion Estimation 佔了百分之九十三的壓縮時間，其它的部份分別是 Interpolation 百分之四，Intra Coding(MakeIntraPrediction)百分之一，其它包含 Integer

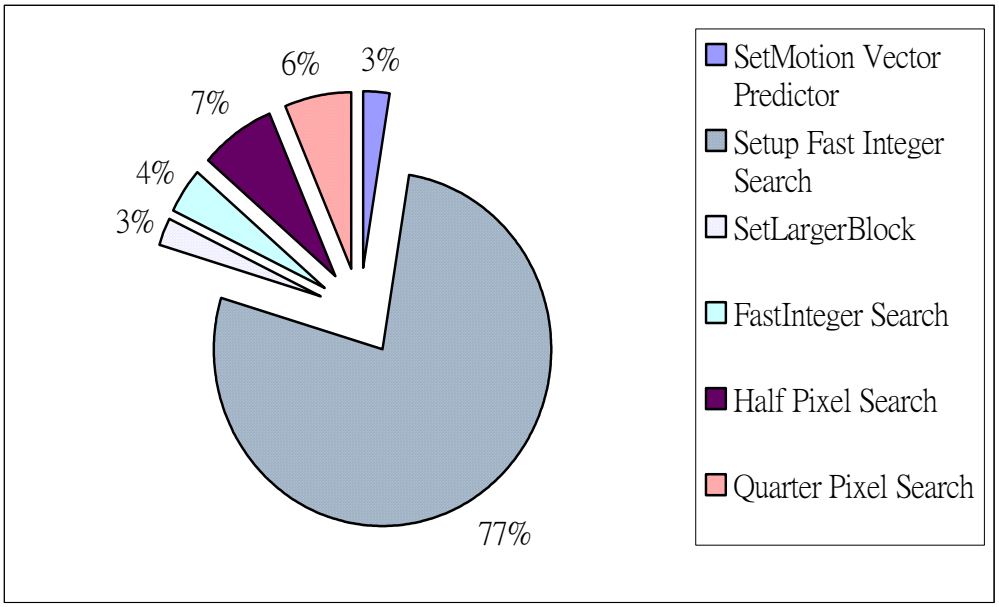


Figure 2.37 H.26L 位移估計複雜度評估圖

DCT、UVLC 編碼等百分之二。

接著來看 Motion Estimation 的子項，由於程式中的 Integer Pixel Searching 是包含兩個部份，一是事先將在搜尋範圍中的 4x4 區塊中的 SAD 值都把它先計算出來(SetupFastIntegeSearch)，再把 4x4 區塊疊合起來(SetupLargeBlock)，並且再做一個比較(FastIntegerSearch)，總共佔了百分之八十四；Half Pixel 和 Quarter Pixel 分別佔了百分之七到八。

所以 H.26L 雖有高的壓縮率，但其複雜度太高了[21][22]，若要真正能夠在即時壓縮上應用，可看出尤其是在 Motion Estimation 上需要快速演算法，這也是本論文所要提出的一套方法來解決此一問題。

第三章 快速位移估計演算法簡介

位移估計(motion estimation) 在以位移補償為基礎的視訊編碼標準上扮演了十分重要的角色，通常是以區塊比對(block-matching)的方式來處理，所謂的區塊比對是指在處理一張圖片的時候(即一個frame)，先將 frame 分割成一個個 16x16 的區塊，然後再以此區塊為單位來預估位移向量。 位移向量是以 current frame 為基準，在previous frame 有限的區域(±16)內，找尋到最接近並誤差最小的區塊，然後將兩個 frame 的座標相減，即是位移向量。在找尋最佳位移向量的過程，我們稱為搜尋演算法(Search Algorithm)。之前所講到的Inter 模式即是這樣的方式由 previous frame 來預測 current frame，以達到壓縮資料量的目的。

除了全域搜尋區塊比對演算法(Full Search)外，基本上，應用在視訊標準上的快速位移演算法，大概分為二大類：1)無失真(Lossless)快速位移估計演算法，例如連續消除演算法(SEA)[10]，2)失真(Lossy)快速位移估計演算法，例如三步搜尋演算法[13]。

無失真位移估計演算法，主要是利用其它方式，來取代並降低算SAD(Sum of Absolute Difference)計算量，其所得的位元率及 PSNR 值和全域搜尋區塊比對演算法完全相同，並且編碼的速度可快速增加，但在某些特別的情況，會使加速的效果，沒有這麼好，還可能會徒增編碼的時間。失真位移演算法，主要是利用 Uni-model Error Surface Assumption(UESA)，將搜尋範圍中尋找最小值的問題，假定只有一個區域的最小值(Local Minimum)，也就是全域的最小值(Global Minimum)，但事實上並非如此，所以用此類演算法，所找到的最小

值，通常是區域的最小值，所以其速度雖然比無失真快速位移演算法快，但是因為找到的是區域最小值，所以位元率上升並且畫面品質變差。

3.1 無失真快速位移估計演算法

以下我們將焦點放在無失真快速位移估計演算法。誠如之前所言，每一個區塊都必須以搜尋演算法來找尋找最像的區塊，所以我們可知其重要性。全域搜尋區塊比對演算法(Full Search)是最原始、最簡單的一種;但相對的,由於全域搜尋區塊比對演算法將 current frame block 的每一點與 previous frame search area 的每一點做運算,因此 Full search 得到是最精確的搜尋結果，但計算量十分龐大，所耗的搜尋時間亦不低。

所以是否可以不要用每一點去相減，來得到最相似的位移區塊，這就是無失真快速位移估計演算法最重要的精神。

3.1.1 連續消除演算法(SEA)

連續消除演算法(Successive Elimination Algorithm)[10]在 1995 年被提出，它可以減少全域搜尋區塊比對演算法的高運算量，同時可以得到和全搜尋比對演算法相同的結果,使它比其它許多必須要犧牲峰值信號雜訊比(PSNR)的快速演算法，例如三步搜尋(Three Step Search)、鑽石搜尋(Diamond Search)等等，更加吸引人採用。其主要精神可以以下式表示：

$$\begin{aligned} \mathrm{SAD(m,n)} &= \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |\, c(i,j) - s(i+m, j+n)\,| \\ &\geq \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |\, c(i,j)\,| - \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |\, s(i+m, j+n)\,| = K - SB(m,n) \equiv Sum\ Norm(m,n) \end{aligned} \tag{1}$$

(1)式中，K 代表目前區塊中所有像素之強度和，SB(m, n)代表在搜尋位置(m, n)的候選區塊之所有像素強度的和。對每個搜尋位置(m,

n)而言，計算 Sum Norm(m, n)值比計算 SAD(m, n)值要容易許多，因為所有搜尋位置(m, n)只需要計算 K 值一次，而 SB(m, n)可由左方搜尋位置之 SB(m-1, n)求出，如(2)所示：

$$SB(m,n) = SB(m-1,n) + \sum_{a=0}^{N-1} s(m+N-1, n+a) - \sum_{a=0}^{N-1} s(m-1, n+b) \quad (2)$$

如果 Sum Norm(m, n)比已計算出來的 SAD 中之最小值 SAD_{min} 還大的話，由(1)式可保證 SAD(m, n)一定比 Sum Norm(m, n)還大，因此搜尋位置(m, n)的 SAD 計算就可以省略，否則，仍必須計算 SAD(m, n)。很顯然地，如果一開始就能對移動向量有很好的初始猜測，使一開始就有一較小的 SAD_{min} 值，則可以省略搜尋位置的機會，有效的降低運算量，因此，搜尋位置的掃瞄順序變得十分重要。有研究者提出以移動向量的預測向量為第一個的搜尋位置，預測向量即為左方、上方及右上方的移動向量的中位數；另有研究者提出以螺旋狀掃瞄(Spiral Scan)，如 Figure 3.1 所示，來取代傳統的光柵掃瞄(Raster Scan)，如 Figure 3.2 所示； 然而，若真正的移動向量超出搜尋範圍，連續消除演算法對搜尋位置的省略比例甚至有可能會低到使該移動向量之運算時間比全域搜尋區塊比對演算法還要久，Figure 3.3 是連續消除演算法的流程圖。

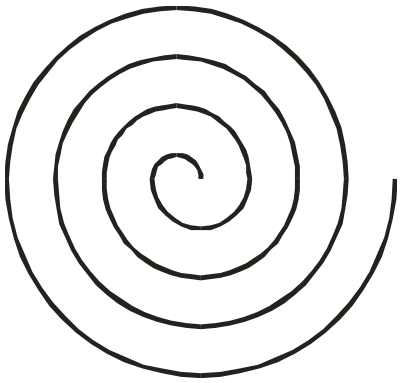


Figure 3.1 螺旋式搜尋

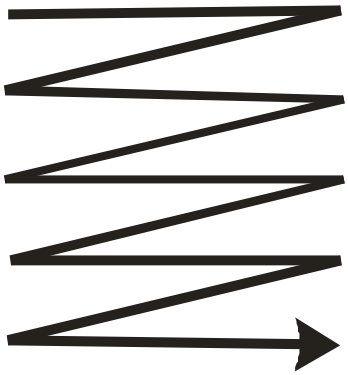


Figure 3.2 柵欄式搜尋

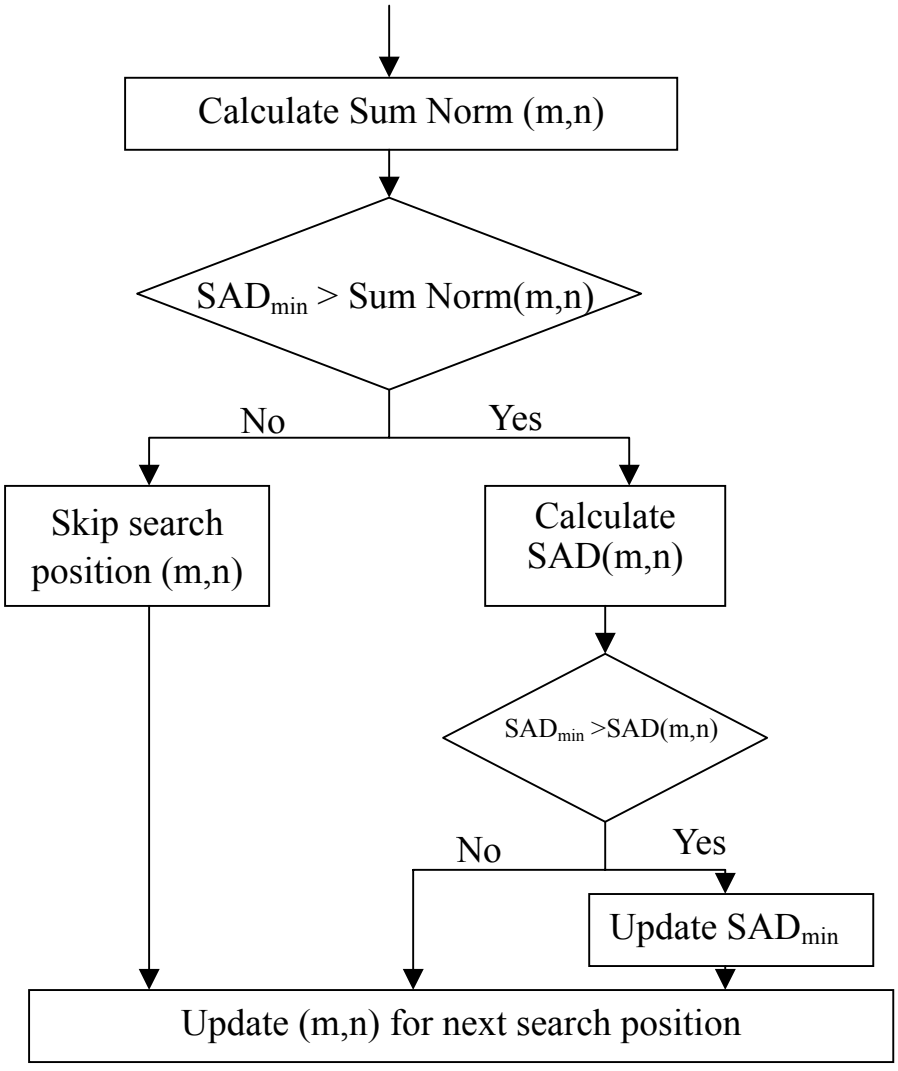


Figure 3.3 連續消除演算法流程圖

3.1.2 多階層連續消除演算法

有研究者於西元 2000 年提出多階層消除演算法(Multi- Level Successive Elimination Algorithm)[11]，其原理為將(1)式修改為下列不等式：

$$\begin{aligned}
&\mathrm{SAD(m,n)} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |\, c(i,j) - s(i+m, j+n)\,| \\
&\geq \sum_{q=0}^{L-1} |\, Kq - SBq(m,n)\,| \equiv \mathrm{M\,Sum\ Norm}(m,n) \qquad (3) \\
&\geq \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |\, c(i,j)\,| - \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |\, s(i+m, j+n)\,| = K - SB(m,n) \equiv \mathrm{Sum\ Norm}(m,n)
\end{aligned}$$

其中，一個 NxN 區塊被分成 L 個子區塊，K_q 代表在目前區塊中第 q 個子區塊內所有像素之強度和，SB_q(m,n)代表在搜尋位置(m,n)計算出 M(Multi-level)Sum Norm(m,n)，如果 MSum Norm(m,n)比計算出來的 SAD 中之最小值 SAD 還大的話，由(3)式可保証 SAD(m,n)一定比 MSum Norm(m,n)還大，自然也比 SAD_{min} 還大，因此搜尋位置(m,n)的 SAD(m,n)計算就可以被省略，否則，仍必須計算 SAD(m,n)。此外，由(3)式我們可以發現 MSum Norm(m,n)大於或等於 Sum Norm(m,n)，因此，在相同的掃瞄順序下，多階層連續消除法可更進一步降低連續消除演算法的運算量，若配合由預測向量開始搜尋或螺旋狀掃瞄，多階層連續消除演算法搜尋位置省略比率在 50%~90%，隨著不同的的視訊樣本會有不同的值，且不同的畫面間的省略比例也不同，這個特點和連續消除演算法是一樣的。每個移動向量估計所需的值不相同且無法預測。

多階層連續消除演算法的流程圖和連續消除演算法幾乎完全相同，只要將 Figure 3.3 中的 Sum Norm(m,n)改成 MSum Norm(m,n)即可。因此，連續消除演算法會遭遇到的問題，多階層連續消除演算法也會有相同的情況。

3.1.3 一維投影區塊比對演算法(1D-Projection)

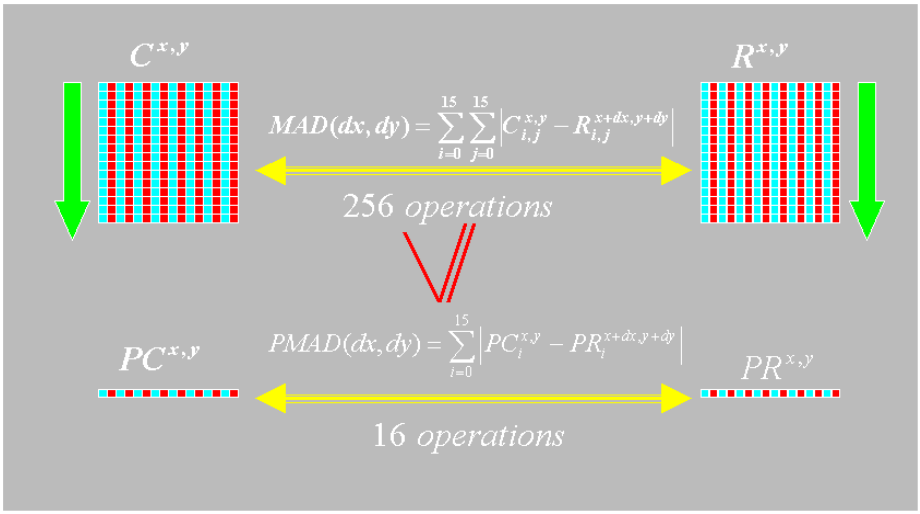


Figure 3.4 一維區塊投影比對演算法

M. Brunig 在西元 2000 年提出一維投影區塊比對演算法 [12] ，其原理由下式來說明：

$$MAD(dx,dy)=\sum_{i=0}^{B_h-1}\sum_{j=0}^{B_v-1}\left|C_{i,j}^{x,y}-R_{i,j}^{x+dx,y+dy}\right|$$

$$=\sum_{i=0}^{B_h-1}\left|\sum_{j=0}^{B_v-1}\left|C_{i,j}^{x,y}-R_{i,j}^{x+dx,y+dy}\right|\right|\geq\sum_{i=0}^{B_h-1}\left|PC_{i}^{x,y}-PR_{i}^{x+dx,y+dy}\right|=PMAD(dx,dy)\quad(4)$$

其所使用的方法，主要是將一個 NxN 的 block 中的一整行 Column 做相加，類似投影的方法，相加所留下來的數值分別為 $PC^{x,y}$ 及 $PR^{x+dx,y+dy}$，這樣就可使二維的區塊比對變成一維區塊，降低其計算量，如 Figure 3.4 所示，再由(4)中可看到類似在 SEA 演算法中出現的不等式，只是現在是以一個 Column 做相加而非一個 Block。同樣用此方法也會遇到和 SEA 演算法的困擾，所以初始的猜測變得很重要。

3.1.4 部份區塊比對演算法(PDE)

主要是利用當二個 Current 區塊和 Previous 區塊在算 SAD 時，在每算完一個 Row 的 SAD 值後，做一個檢查的動作，看是不是比 SAD_{min} 還要大，如果是真的話就不再計算，而再檢查在搜尋範圍的下一個點，這種方法類似 Half Stop 的意味[24]。

3.2 失真快速位移估計演算法

3.2.1 三步搜尋演算法(3SS)[13]

1981 年由 J. Jain 和 A. Jain 所提出的方法，其作法如 Figure 3.5 所示：

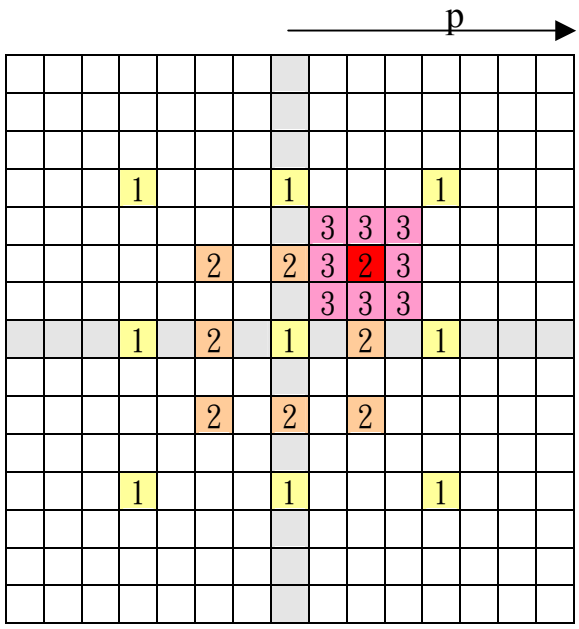


Figure 3.5 三步搜尋演算法

Step 1：當 Previous 和 Current Macroblock 開始做搜尋的時候，三步搜尋演算法會先對整個範圍相距 p/2(p:Search Range) 像素點及原點共九個點做比較，也就是圖中標示為 1 的區塊，並判定出最接近的區塊。

Step 2：接下來將目標點轉移到前一步驟所判定出的最接近點，並以此點原地縮小範圍，找尋相距 p/4 的八個像素點，再次比較其差異度，決定出最接近點。(圖中標示為 2)

Step 3：如前所示，我們將目標點再次轉移到最接近點，並以此點縮小範圍，找尋相鄰的八個像素點，比較其差異，找出最接近點，即為所求。我們可以看到圖中被紅色

所填滿的方塊就是找到的位移向量的位置。

因此，三步搜尋法，每一個區塊所需要搜尋的點數 25 點。

3.2.2 新三步搜尋演算法(N3SS)[14]

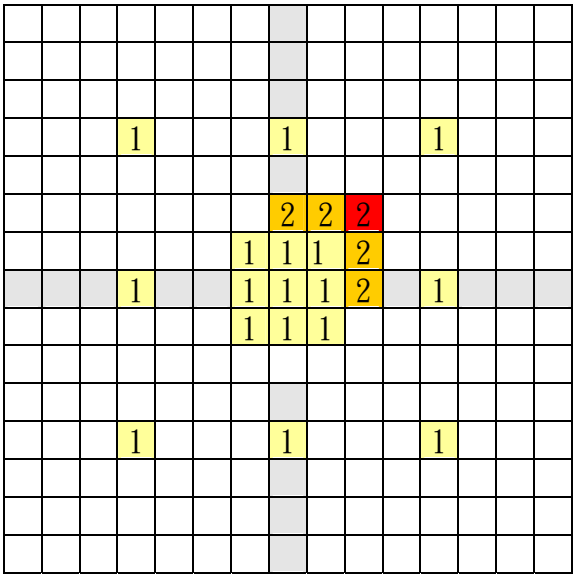


Figure 3.6 新三步搜尋演算法

1994 年由 Renxiang Li 所提出的方法;三步搜尋演算法中,在 Step 1 使用平均配置(Uniform)檢查點，但在一般的動態影像中，尤其是在低位元率中，位移向量通常是 Smooth 及變化緩慢，所以採用所謂 Center biased 的方式來取代平均配置。

Step 1：新三步搜尋演算法會先對整個範圍相距 p/2 (p: Search Range)的 8 個像素點、原點及原點周圍 8 個像素點做比較，也就是圖中標示為 1 的區塊，並判定出最接近的區塊。根據最小 SAD 值的那點，判斷是否要要繼續檢查，若在 17 個點中，中心點有最小值，則最佳的點就是中心點,若是在周圍 8 點中的其中一點則跳到 Step 2，若是在最外的那 8 個點，則跳到 Step3。

Step 2：再以這點檢查周圍的 8 個點,其中有部份的點重複了，

所以實際上要檢查的點，只有 3 到 5 點，其中最佳的點就是有最小 SAD 值，此時即結束整個搜尋過程。

Step 3：而在最外面的 8 個點，則以原本 Three Step Search 的方式進行。

因此，新三步搜尋法，每個區塊需要 17 到 33 個點。

3.2.3 SES 搜尋演算法[15]

1997 年由 Jianhua Lu 所提出的方法;SES 主要是改進三步搜尋演算法的平均配置檢查點,提出所謂在 UESA 的情況下相反方向的不需檢查，也就是是利用方向來找到最小 SAD 值。

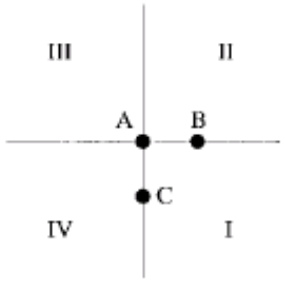


Figure 3.7 SES 判斷象限示意圖

Step 1: 和 Three Step Search 一樣，相距原點 p/2 之九個點，但是此次先檢查三個點，就如 Figure 3.7 所示，A,B,C，再來利用不等式，如(5)式，來判定尋找的方向，

If MAD (A) >=MAD(B) and MAD(A)>=MAD(C), I is selected
If MAD (A) >=MAD(B) and MAD(A) < MAD(C), II is selected
If MAD (A) < MAD(B) and MAD(A) < MAD(C), III is selected (5)
If MAD (A) < MAD(B) and MAD(A) >=MAD(C), IV is selected

再由 Figure 3.8 所示，先決定好方向，再以其方向加上必要的點數，並尋找下一個 Step 的最適合的 Candidate。

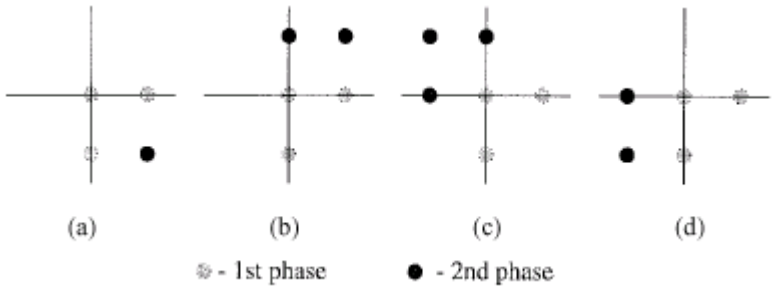


Figure 3.8 判斷象限後增加點數示意圖

Step 2：依照同樣的方法並以上個所找到的最佳點，將其相距 p/4 的八個點，和 Step 1 同樣的方法，先判定方向，再增加點。

Step 3：和 Step 1 和 Step 2 類似，但是此次限定相鄰的八個點。

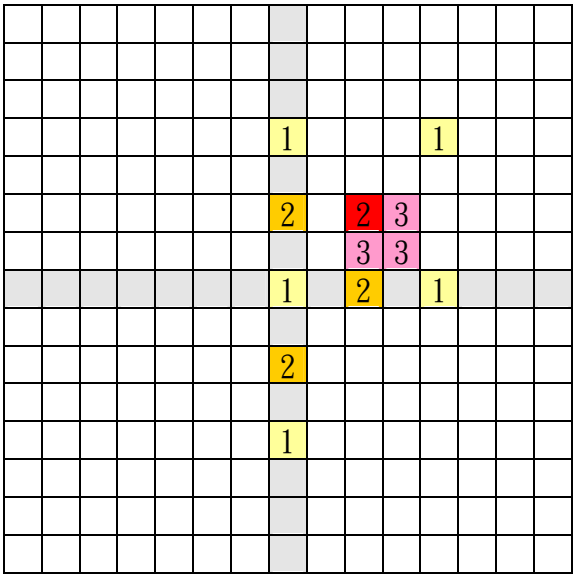


Figure 3.9 SES 搜尋演算法整體示意圖

Figure 3.9 是 SES 搜尋演算法的例子，每個區塊需要 10 到 17 個點。

3.2.4 四步搜尋演算法(4SS)[16]

1996 年由 Lai-Mao Po 所提出的方法，如 Figure 3.10 所示：

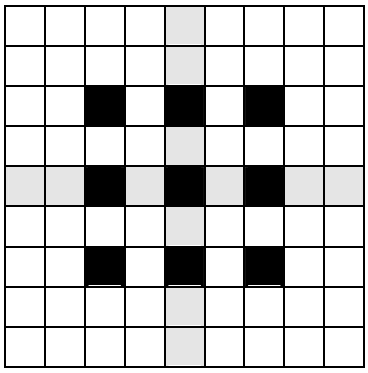


Figure 3.10(a)

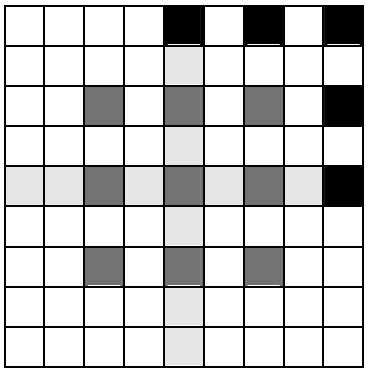


Figure 3.10(b)

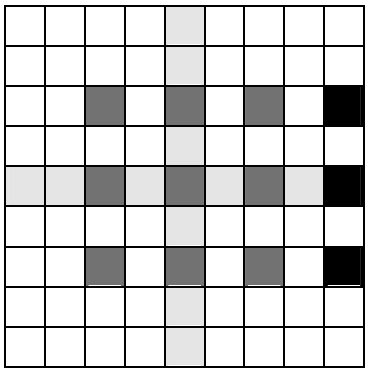


Figure 3.10(c)

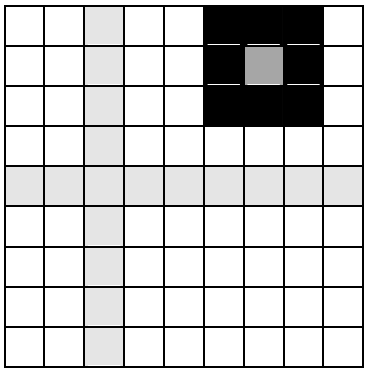


Figure 3.10(d)

Step 1：由搜尋區域的中心點開始，如 Figure 3.10(a)所示，以中心點周圍 5x5 的範圍八個點，找尋誤差最小的點。若最小值在中心點則跳到 Step 4，否則跳到 Step 2。

Step 2：以 Step1 所找到的最小值為中心，周圍 5x5 範圍八個點，省略 Step 1 找過的點，如 Figure 3.10(b)、3.10(c)，若最小值在 Step 2 時的中心點則跳到 Step 4，否則跳到 Step 3。

Step 3：步驟和 Step 2 一樣，只是做完直接到 Step 4。

Step 4： 以上一個 Step 最小值為中心，如 Figure3.10(d)，找尋周圍八個的點，找到即是此演算法最佳的位移向量的位置。

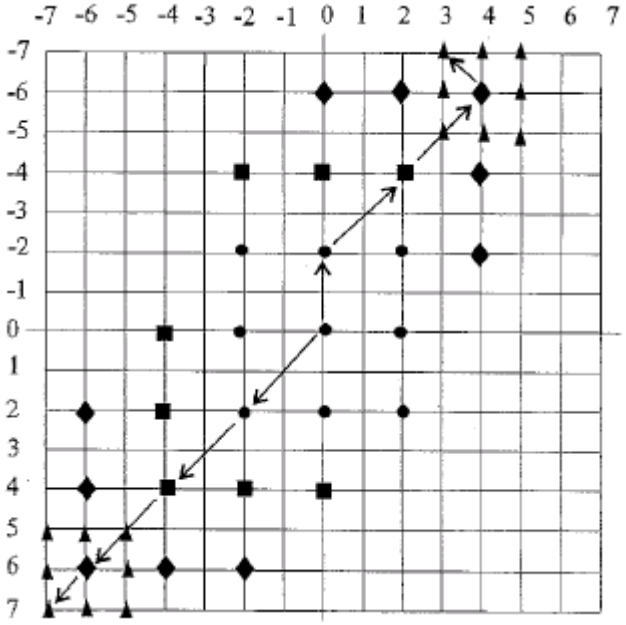


Figure 3.11 四步搜尋演算法整體示意圖

Figure 3.9 是 SES 搜尋演算法的例子，每個區塊需要 17 到 27 個點。

3.2.5 鑽石搜尋演算法(DS)[17]

1997 年由 Shan Zhu、Kai-Kuang Ma 所提出的方法，如 Fig 3.12 所示：

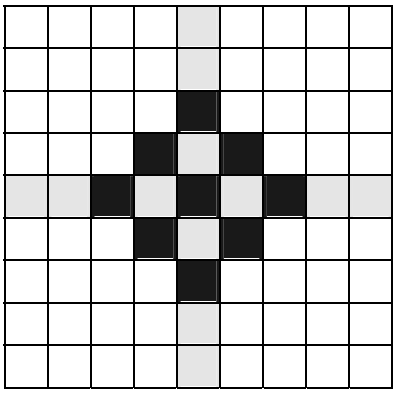


Figure 3.12(a)

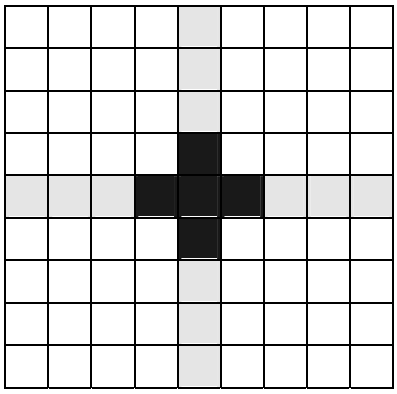


Figure 3.12(b)

Step 1：由搜尋區域的中心點開始搜尋，由 Figure 3.12(a)的範圍開始搜尋，找尋最接近的像素點。

Step 2-1：如果找到的點為中心點的話，則原地縮小範圍，以 Figure

3.12(b)的範圍繼續搜尋，下一步跳至 step 3-1。

Step 2-2：若非中心點，則以最接近點為中心點，重複 step 1，直到最接近點為中心點為止。

Step 3-1：如果以 Figure 3.12(b)的範圍，找到的點為中心點的話，則搜尋結束，並以此點為所求。

Step 3-2：若非中心點，則重複 step 2-1，直到找到的點為中心點為止。Figure 3.13 中紅色方塊即為所求。

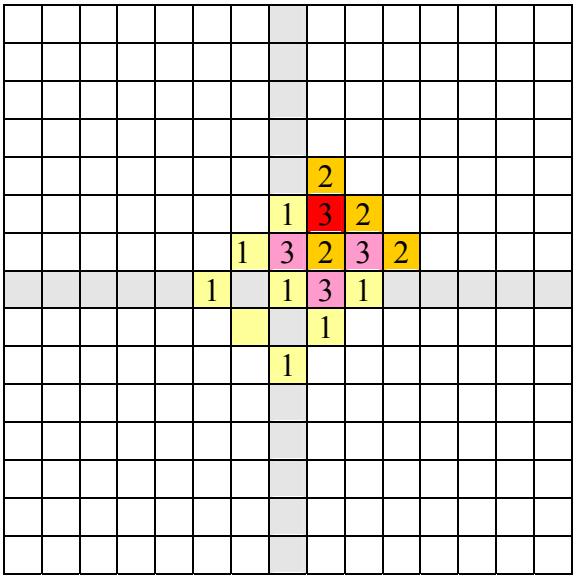


Figure 3.13 鑽石搜尋演算法整體示意圖

3.2.6 區塊梯度搜尋演算法(BBGD)[18]

1996 年由 Lurng-Kuo Liu 和 Ephraim Feig 所提出，如 Figure 3.14 所示：

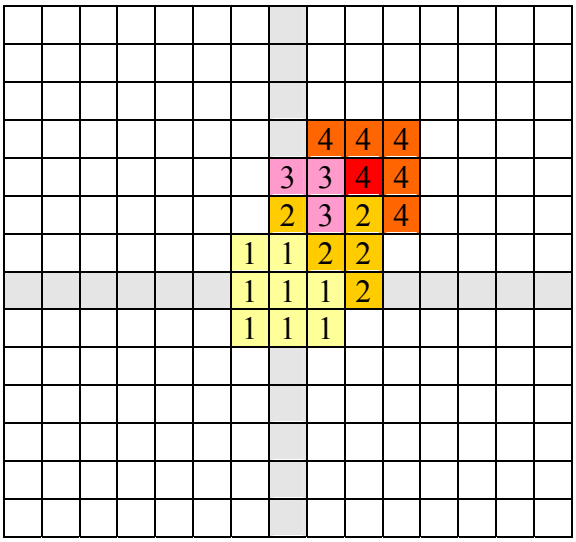


Figure 3.14 區塊梯度搜尋演算法整體示意圖

Step 1：以 Macroblock 的中心點為基準，檢測環繞此點八個像素的差異值，找出最接近點。

Step 2-1：如果最接近點為中心點的話，結束搜尋。此點即為所求。

Step 2-2：如果最接近點不是在中心點，則以此點為新的中心，並再次檢測環繞此點的八個 pixel。

Step 3：重複 step 2，直到結束搜尋。(圖中紅色的方塊，即為所求)

3.2.7 基因搜尋演算法(Genetic Algorithm)[19]

1993 年由 Hung-Kei Chow 所提出，可以說是基因演算法用於移動向量預估之先河，十分地具有指標性的意義。以下我將就此演算法做介紹，由這篇論文我們可以發現基因演算法用於移動向量之預估是可行的。

1. Initialization：一開始選取親代的 search point 時，我們以隨機的方式在 search window 中挑選，挑選的原則是找中心附近的點。即便 Genetic Algorithm 告訴我們，中心點並不一定是 maximum optimum，但是在大多數的情況下，maximum optimum 是在中心附近的。
2. Evaluation & Selection：上一步已經選取若干 search point，並以其為親代的 chromosomes。接下來將進行物競天擇的繁殖。我們以某種衡量標準來判斷哪些 chromosomes 是比較優秀，並去除比較不符合衡量標準的 chromosomes。理所當然的衡量標準，當然還是 SAD 或 MSE 這兩種廣為人們所使用的方式。於是乎，達成較小 MSE 或 SAD 的 chromosomes 就可以被保留下來，剩下的將被去除，如此將可以保證下一代會有比較優秀的子代。
3. Crossover：親代挑選剩下的菁英，彼此再進行交配的動作，在生物體進行染色體 cross over 時，會發生基因互換的現象。我們可以藉由另外一篇論文來闡述這樣的觀念，Figure 3.15 是出自於 J. A. Handcock 在 1999 年所發表的論文，我們可以很清楚的看到親代跟子代 cross over 後交換彼此資訊的互動關係。而[19]則提供一個實質的方式：

$$\oplus \{(X_{p1},X_{p2})\} = (X_{c1},X_{c2}) \text{ ， } X_{c1}=(X_{p1} \cap M) \cup (X_{p2} \cap \overline{M}) \text{ 、 } X_{c2}=(X_{p1} \cap \overline{M}) \cup (X_{p2} \cap M)$$

在這裡各參數的意義如下：

⊕：cross over。

X_{p1},X_{p2}：親代的 search point 位置。

X_{c1},X_{c2}：子代的 search point 位置。

M：一個隨機產生的陣列，而其長度和 chromosomes 一樣長。

~~M~~：是 M 的 inverse。

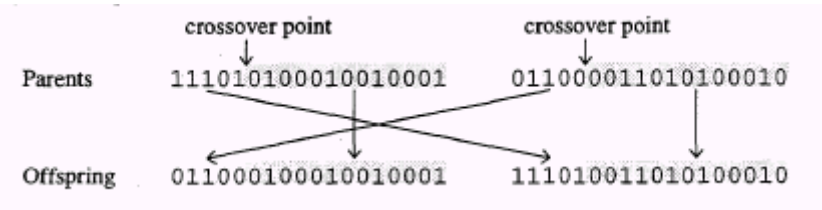


Figure 3.15 基因演算法交配步驟示意圖

4. Mutation：在此是加入一些變異性，以隨機產生的碼，改變 search point 的值。這樣做法的原因乃是為了跳離 local optimum，到另外一個區域。如果另外的地方真的比較好，那麼其子代一定可以通過天擇的檢驗，否則，仍然會回歸到目前的趨近。Mutation 是這類 Genetic Algorithm 的精華之所在，也只有這樣，才和一些 fast search algorithm 有所差距，更容易找到 global optimum。

5. 重複 1-4 的步驟，每重複一次，意味著一個 generation。在一般的 Genetic Algorithm 的使用上，會執行上百個子代，藉由電腦高速的運算能力來簡化問題。比如在茫茫的資料庫中找尋一粟，如果要研發出最好的演算法，勢必不易。就算擁有一種不錯的演算法，其效能卻又不一定會好。但是 Genetic Algorithm 卻保有簡單

的執行,分析的方式簡單、判斷的方式亦簡單。所以一般的 Genetic Algorithm 才會執行上百個世代,但是在視訊系統中,這樣的事情可不能成立。如我們所知視訊中的 motion estimation,找尋到的 optimum 往往都不會偏離中心太多,因此太多的子代實在不必,而且太多的子代將會影響到整體的效能,阻礙 real time 的實現。

3.2.8 使用空間域演算法(CAS)[20]

2001 年由 Hyun Mun Kim 所提出,主要是利用周圍已編出來的位移向量,來縮窄搜尋的範圍。

	MV2	MV3
MV1	MV	

Figure 3.16 參考位移向量圖

而會用到幾個數值,如(6)式中的 Motion Predictor,及(7)式中的周圍三個 MV1, MV2, MV3 的 x,y 最大值和最小值的差值。

$$\begin{aligned} Px &= Median(MV1x, MV2x, MV3x) \\ Py &= Median(MV1y, MV2y, MV3y) \end{aligned} \quad (6)$$

$$\begin{aligned} Rx &= MAX(MV1x, MV2x, MV3x) - MIN(MV1x, MV2x, MV3x) \\ Rx &= MAX(MV1x, MV2x, MV3x) - MIN(MV1x, MV2x, MV3x) \end{aligned} \quad (7)$$

而這個演算法為什麼要用到上面(6)及(7)式,最主要是觀察其周圍是否有變動量,若有則擴大區域搜尋。以下是演算法的判斷式。

```
If(Px=MV1x) then
    i=1
Else if(Px=MV2x)then
    i=2
Else
    i=3
Endif
```

```
If(Py=MV1x) then
    j=1
Else if(Py=MV2x)then
    j=2
Else
    j=3
Endif
```

```
If(i=j)then
  Apply motion estimation using 3x3 search window centered y (Px,Py)
Else
  Apply motion estimation using axb search window centered by (Px,Py);where default value of  a=b=5;
If(Rx<=3)set a=3;
If(Ry<=3)set b=3;
```

第四章 快速多階連續消除移動預估演算法

由於 H.26L 壓縮編碼時間，主要花在搜尋七個不同大小區塊模式(16x16 - 4x4)的移動估計，所花費的時間是以往以 16x16 為搜尋單位的數倍，為了要快速的尋找到最佳的位移向量，並且不損失畫面品質及增加壓縮的位元率，本論文會以連續消除演算法或(多階)連續消除演算法為出發點，並加以改良使搜尋的速度增快。

以下是本章的簡介，在第一節中，會將(多階)連續消除演算法實際實現在 H.26L TML8.0 中，並且分析其效能。第二節中，會分析七個區塊模式在各種情況下所佔的分佈比率。第三節中，會以模式及位移向量的相似度來分析，利用 8x8 MSEA 在只算出 16x16 區塊模式下的 SAD 值，暫存並利用其來做為小區塊的搜尋。第四節會以第三節為基礎，並加以精細位移向量。第五節會以第二節所分析的模式的分佈，去除可省略搜尋的區塊模式。最後在第六節中，是本論文所提出快速多階連續消除演算法的流程圖。

4.1 運用連續消除演算法及多階連續消除演算法在 H.26L 的分析

本節會以分析連續消除演算法及多階連續消除演算法，用在 16x16 單一區塊模式上的效率，接著是四個模式及七個模式的分析。連續消除演算法及多階連續消除演算法，主要有兩個部份為最重要，

1)計算區塊 Sum Norm 值，2) 計算不等式條件下區塊 SAD 值。以下就是本論文的分析。

A. 只啟動一個 16x16 區塊模式編碼

以 16x16 SEA、8x8 MSEA 及 4x4 MSEA 演算法，由以下 Figure 4.1 的流程圖來進行比較及計算，和過往的視訊標準不同的是，在位移估計部份需要加上位移向量位元數(motion vector bit-use)，來使所找出的位移向量可符合 Rate Distortion 的原則。

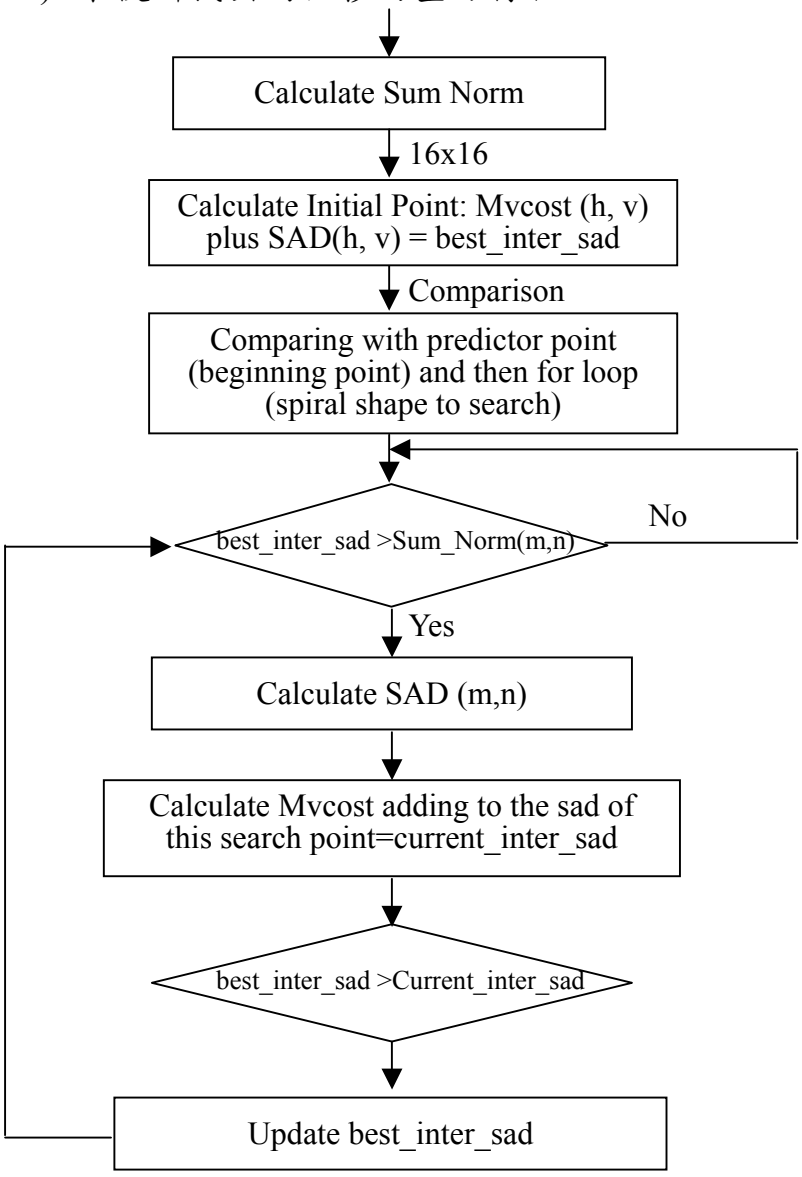


Figure 4.1 將 SEA 演算法放入 H.26L 的流程圖

我們來比較其所花費的時間，由 Figure 4.2 可觀察到結果。

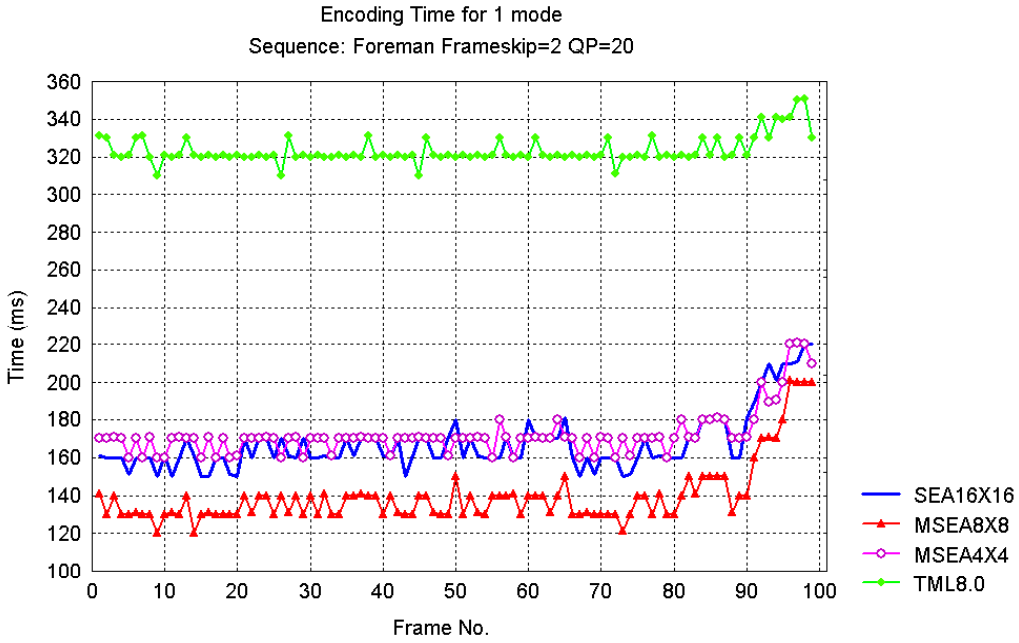
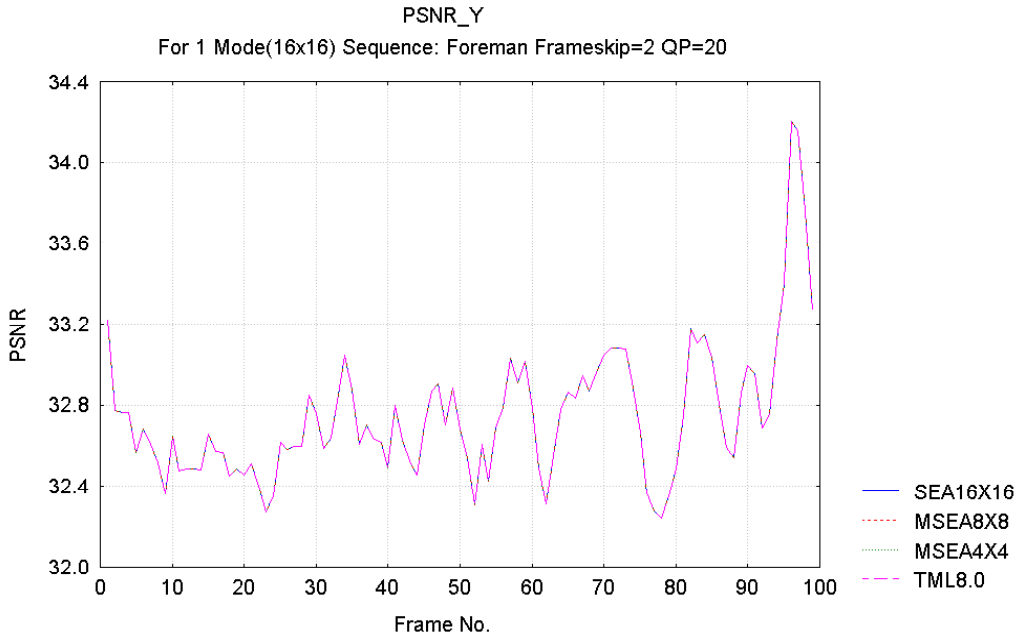


Figure 4.2 使用一個16x16區塊模式的狀況下，TML8.0、16x16SEA、8x8MSEA及4x4MSEA壓一張所需要花的時間

8x8 MSEA 所花的時間最短，其次是 16x16 SEA，最後是 4x4 MSEA，都遠比原始 TML8.0 的 Full Search 好，節省了百分之五十以上的時間，並且由 Figure 4.3 及 Figure 4.4 所示，並沒有增加位元率或降低畫面品質。



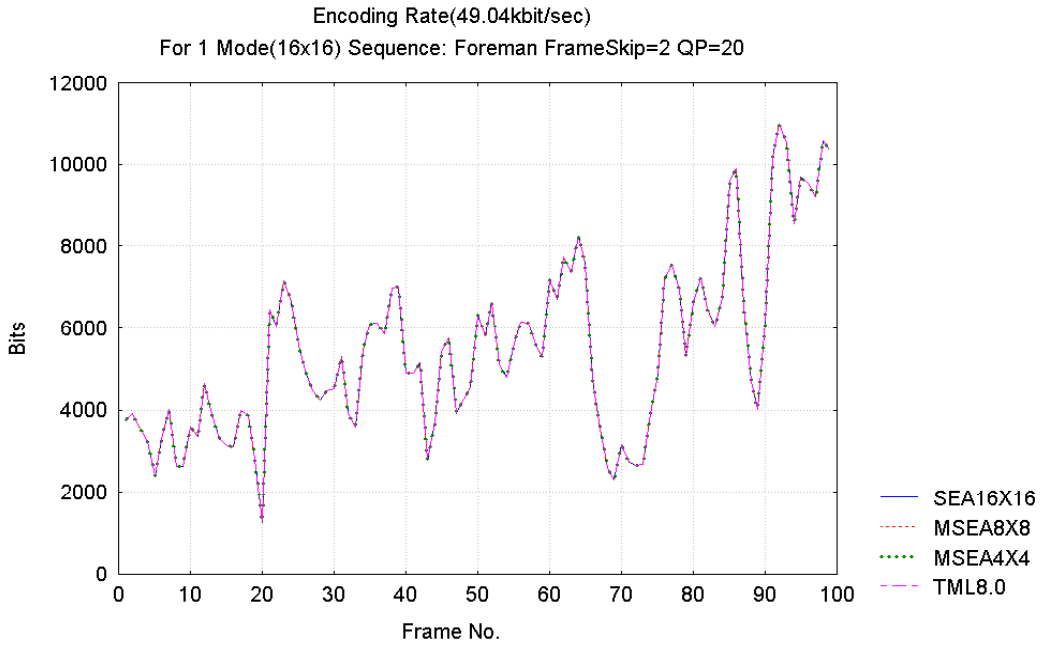


Figure 4.4 使用一個16x16模式的狀況下，TML8.0、16x16SEA、8x8MSEA及4x4MSEA位元率的比較

而從 Figure 4.2 所示，為什麼 8x8 MSEA 會比其它的 16x16 SEA、及 4x4 MSEA 快呢？主要是因為 16x16 SEA 花費在算不等式條件下區塊的 SAD 值太多，可由 Figure 4.5 所示。而 4x4 MSEA 則是花太多的時間在算 4x4 Sum Norm，可由 Figure 4.6 所示。

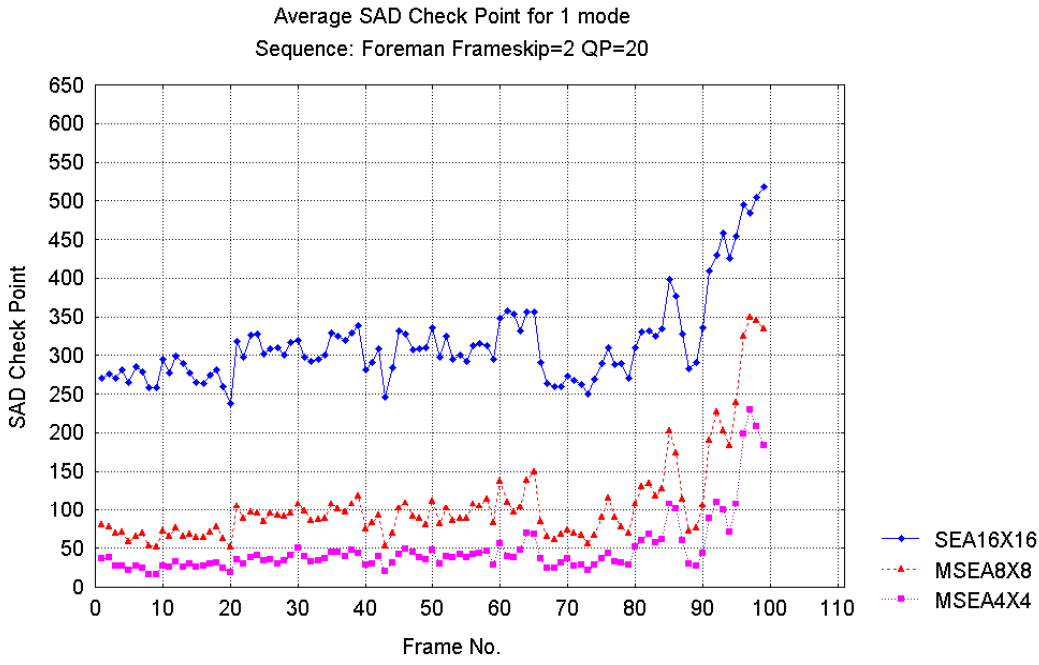


Figure 4.5 使用一個16x16模式的狀況下，16x16SEA、8x8MSEA及4x4MSEA所需要檢查的點數

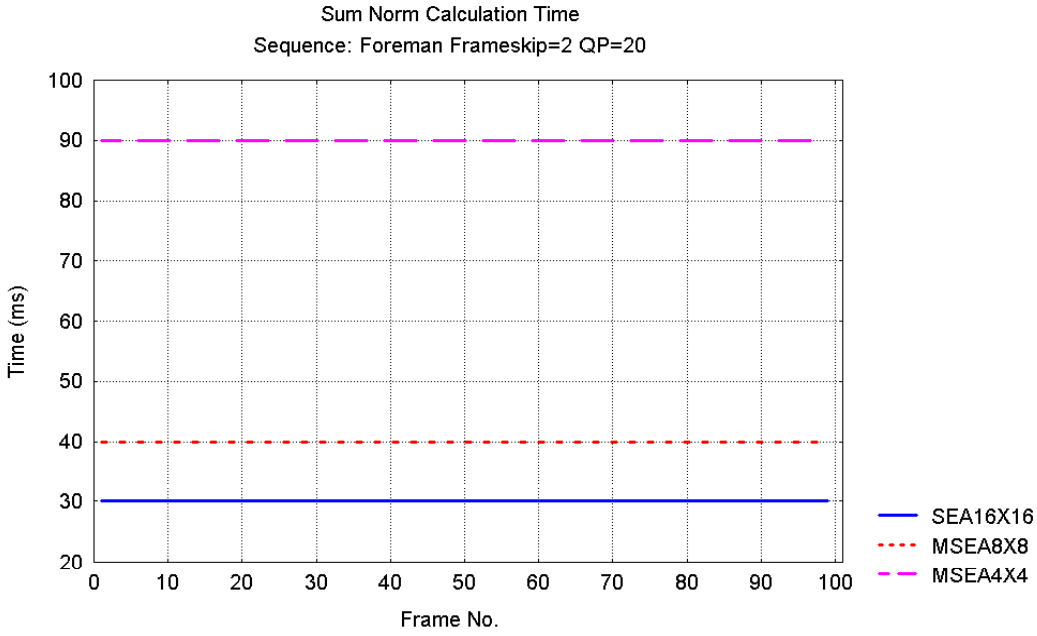


Figure 4.6 使用一個16x16模式的狀況下，16x16SEA、8x8MSEA及4x4MSEA算Sum Norm的時間

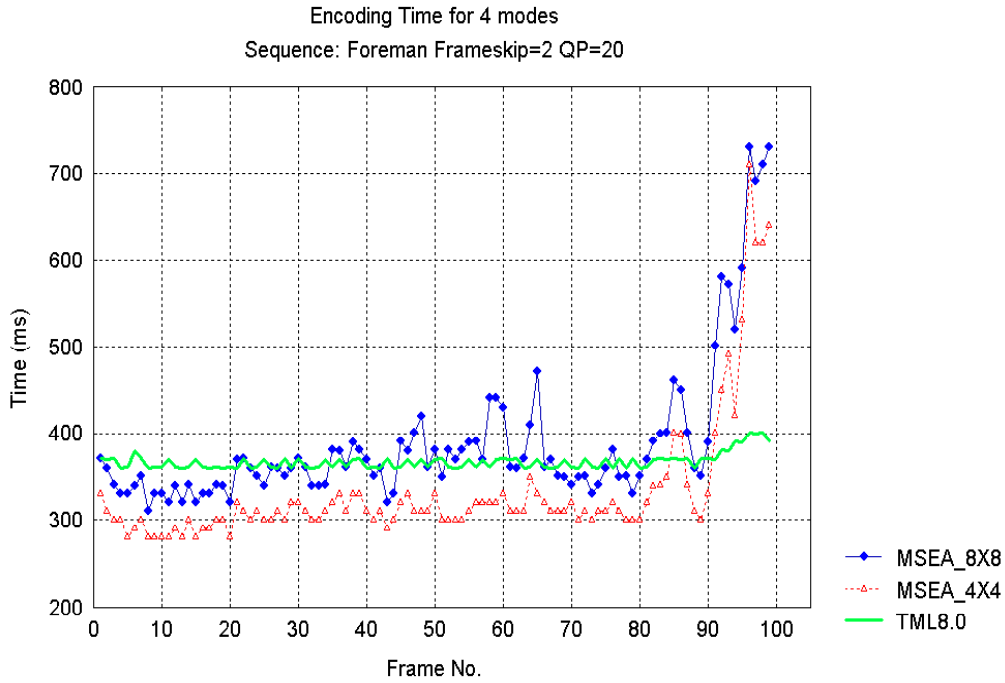


Figure 4.7 使用四個模式16x16-8x8的狀況下，TML8.0、8x8MSEA及4x4MSEA壓一張所需要花的時間

B. 啟動四個區塊模式(16x16-8x8)編碼

而若以四個模式來編碼，由於 16x16 SEA 並沒有小區塊的 Sum Norm 值，所以無法比較，所以只能用 8x8 MSEA 和 4x4 MSEA 來比

較，由 Figure 4.7 所示，4x4 MSEA 所花的時間比較短，其次是 TML8.0 的 Full Search，最後是 8x8 MSEA，並且由 Figure 4.8 及 Figure 4.9 所示，並沒有增加位元率或降低畫面品質。

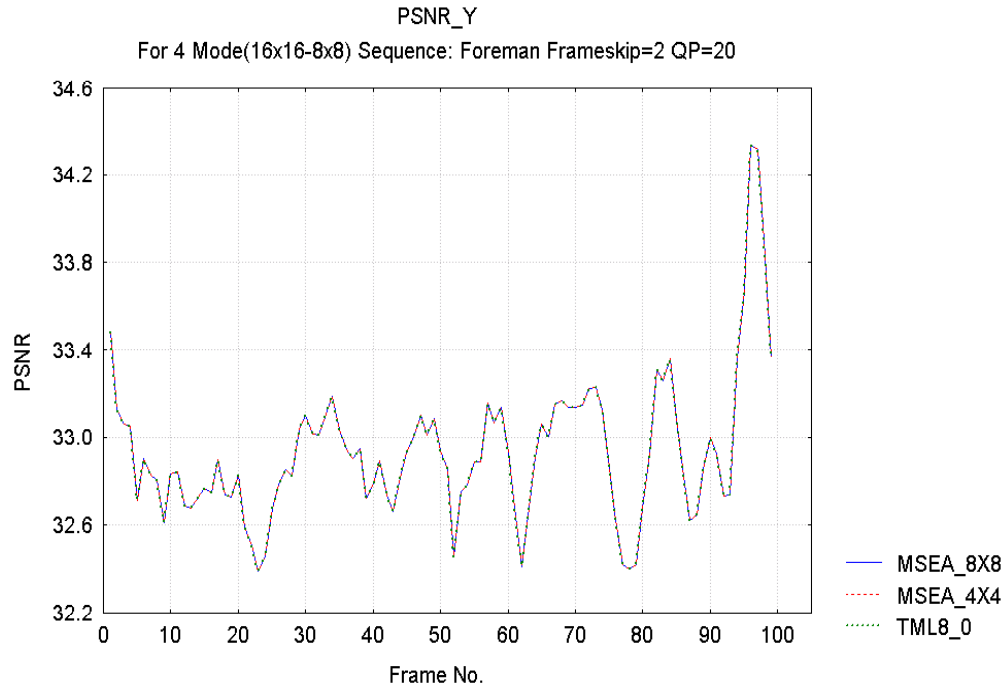


Figure 4.8 使用四個模式 16x16-8x8 的狀況下，TML8.0、8x8MSEA 及 4x4MSEA 畫面品質的比較

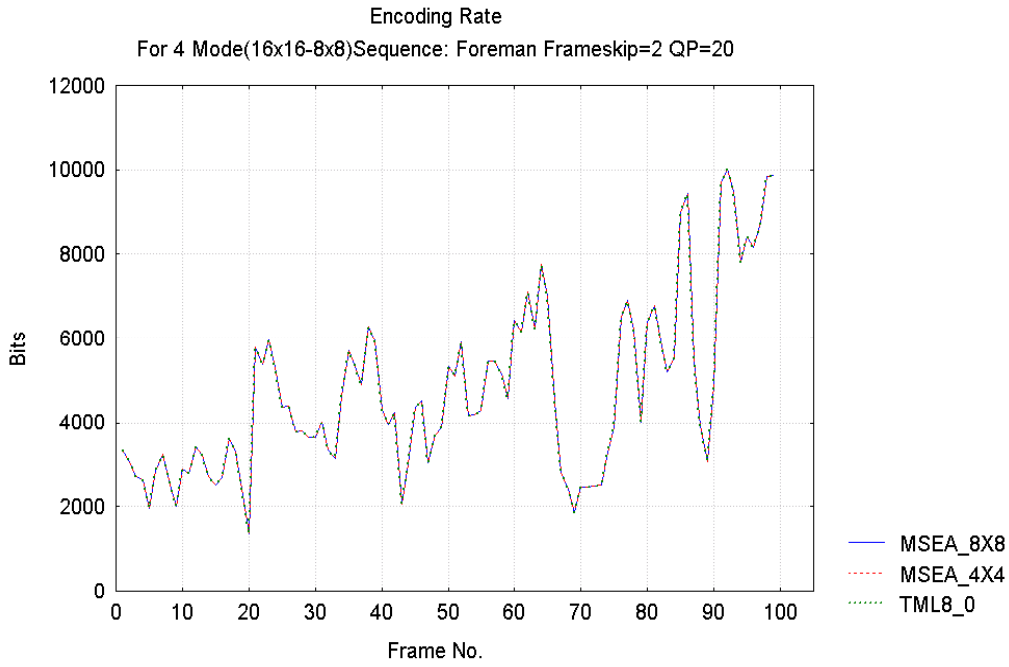


Figure 4.9 使用四個模式16x16-8x8的狀況下，TML8.0、8x8MSEA及4x4MSEA位元率的比較

從 Figure 4.10 中，可觀察到 8x8 MSEA 用在四個模式下的編碼速度蠻差的，主要是因為愈往小區塊去縮窄，直到區塊 8x8 後，8x8 MSEA 的情況就會和 16x16SEA 的情況一樣，所要算的 SAD 值會增多，而在這個情況下，雖然 4x4 MSEA 所要花費在計算 Sum Norm 的時間比較多，但是其所要計算 SAD 值的數目會比 8x8 MSEA 少很多。

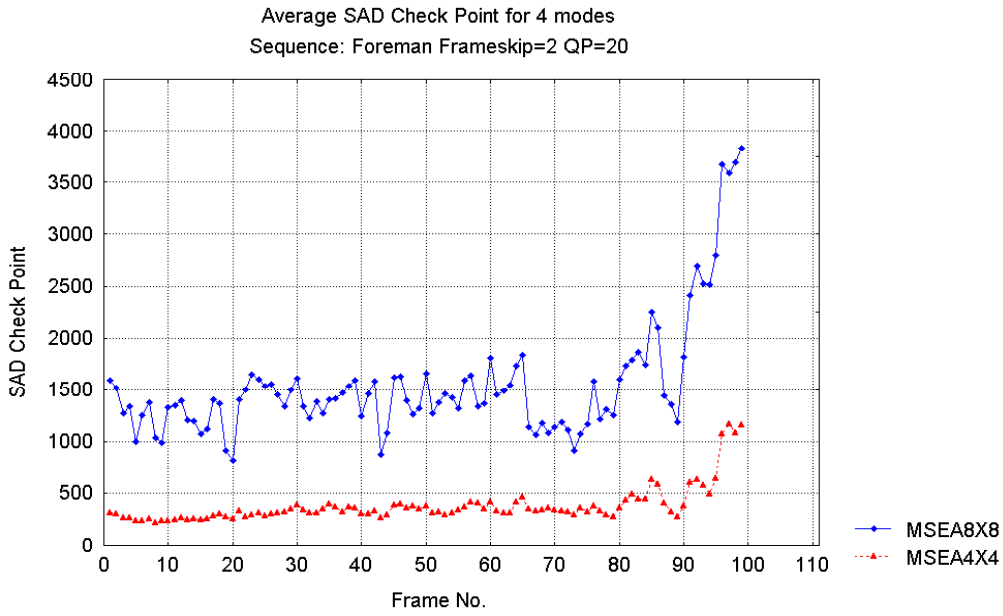


Figure 4.10 使用四個模式16x16-8x8的狀況下，8x8MSEA及4x4MSEA所需要檢查的點數

C. 啟動七個區塊模式(16x16-4x4)來編碼

由前面所分析出的結果，當所使用的區塊模式與算 Sum Norm 的區塊單位愈接近時，MSEA 快速演算法在這個情形下，就會沒有效率。所以在七個區塊模式時，4x4 MSEA 一定也會比 TML8.0 的速度慢，可由 Figure 4.11 所示。Figure 4.12 及 Figure 4.13 是表示 MSEA4x4 的位元率和畫面品質 PSNR 是和 TML8.0 Full Search 是一樣的。

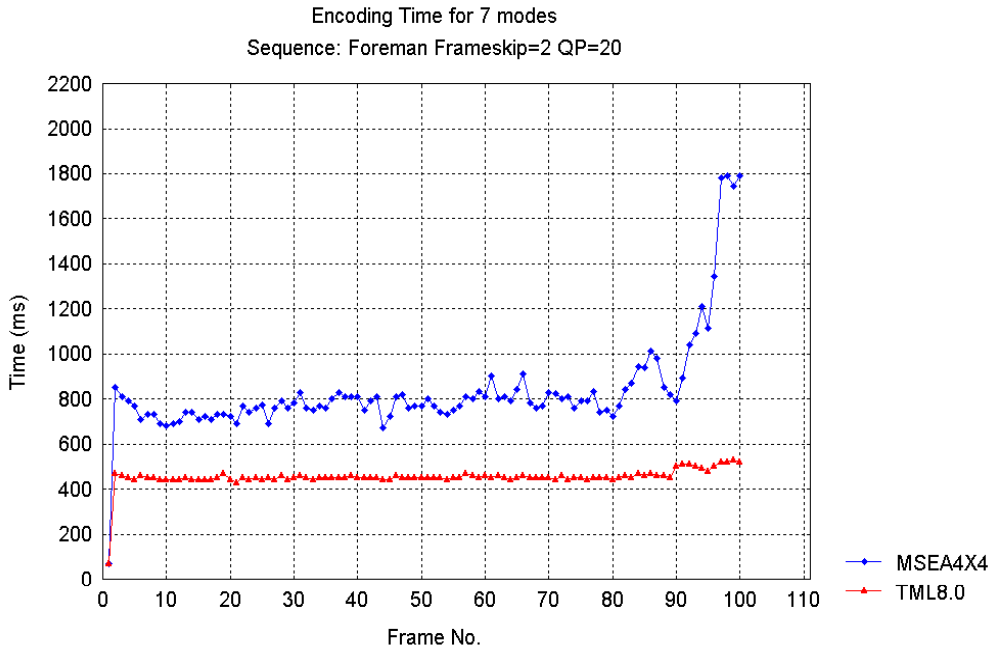


Figure 4.11 使用七個模式 16x16-4x4 的狀況下，TML8.0、及 4x4MSEA 壓一張所需要花的時間

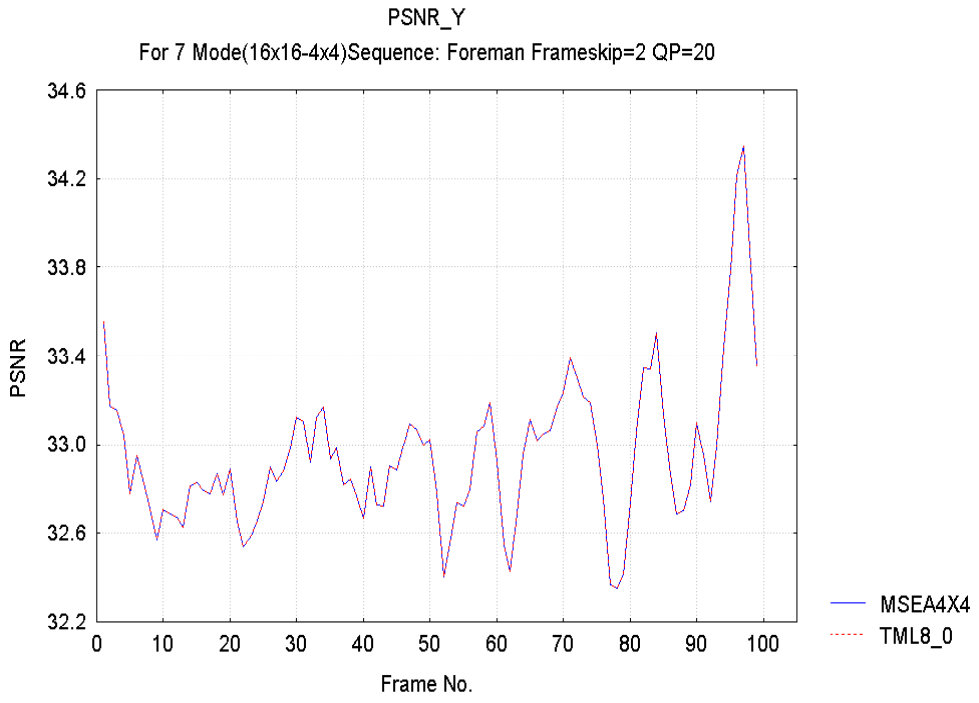


Figure 4.12 使用七個模式16x16-4x4的狀況下，TML8.0、8x8MSEA及4x4MSEA畫面品質的比較

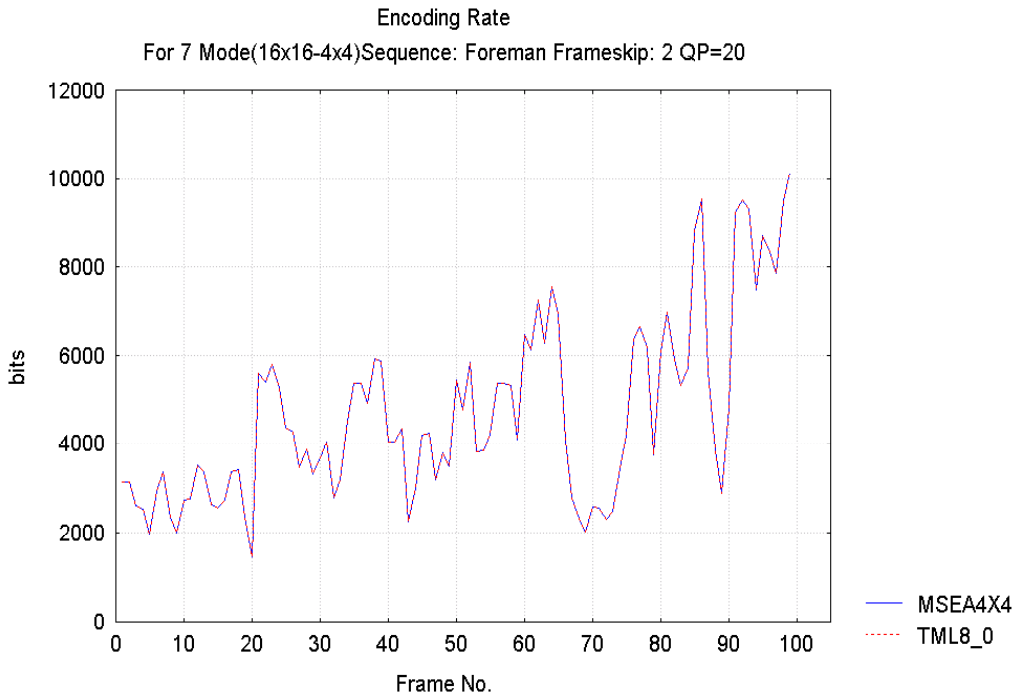


Figure 4.13 使用七個模式16x16-4x4的狀況下，TML8.0、8x8MSEA及4x4MSEA位元率的比較

所以在本節中下一個小小的結論，就是只要使用的區塊模式愈多，多階連續消除演算法的效能就會因為要算的 SAD 值過多而無法達到加速的功效，並且所使用計算 Sum Norm 區塊大小，也會因區塊逐漸縮小而增長要計算的時間。然而是否可用大區塊有算出 Suboptimal SAD 值的點，加以重覆利用，將也是本論文研究重點。

4.2 H.26L 的模式分析

本節中，主要要分析的部份，是分析在各種 Sequence、不同 Frame skip 的張數、不同的 QP 值、不同的格式大小(QCIF、CIF)，Inter mode 的 7 個模式的機率分佈是如何？首先先將要壓縮位元率環境設定在中位率，即 QP 值為 17 時，沒有任何 Frame Skip。

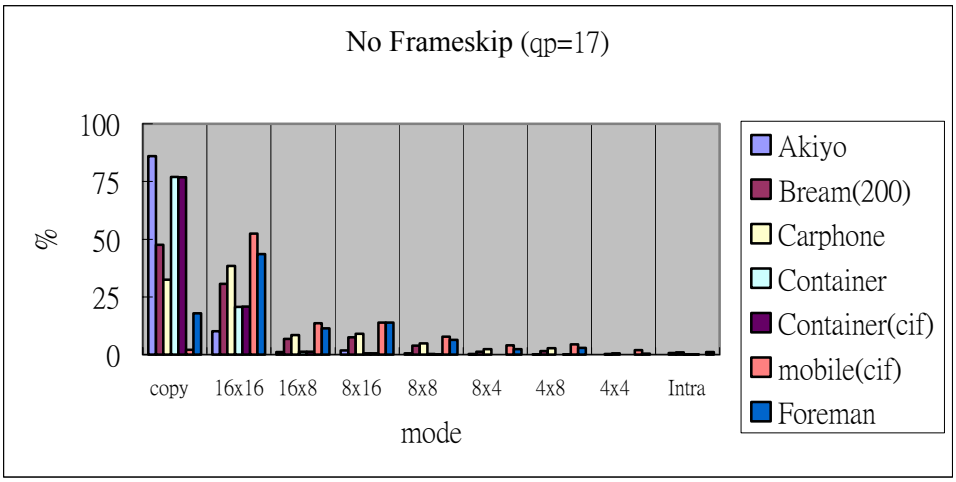


Figure 4.14 模式分佈 - 中位元率 (No Frameskip)

由 Figure 4.14 所示，各種的 Sequence，包含了 Akiyo (300 張) QCIF、Bream (200 張) QCIF、Carphone (300 張) QCIF、Container (300 張) QCIF、Container (300 張) CIF、Mobile (300 張) CIF 及 Foreman (300 張) QCIF。而由 Figure 4.14 所示，16x16 及 Copy(Copy 就是當 16x16 區塊位移向量為 0 時)比率佔了絕大多數，依序才是 16x8、8x16、8x8、8x4、4x8 及最少 4x4 及 Intra。

讓我們來觀察若 Skip 張數增多時，又會如何？

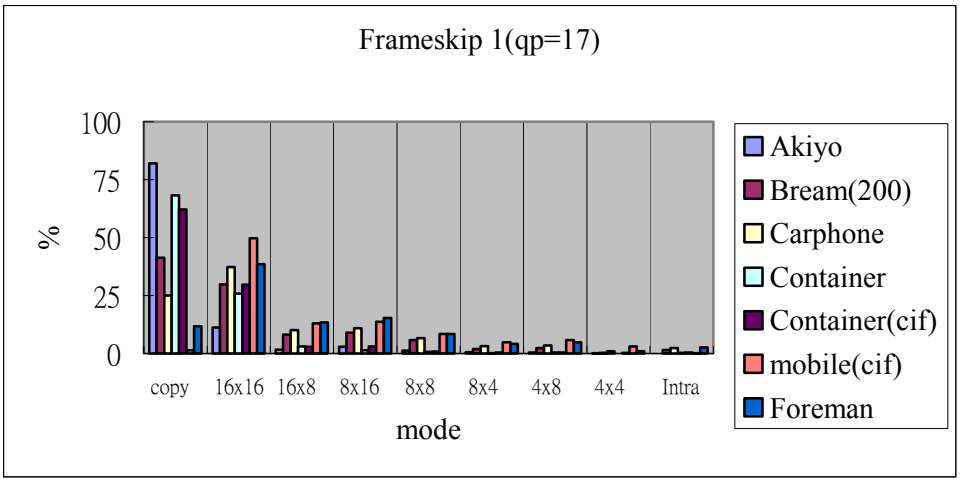


Figure 4.15 模式分佈 - 中位元率 (Frameskip 1)

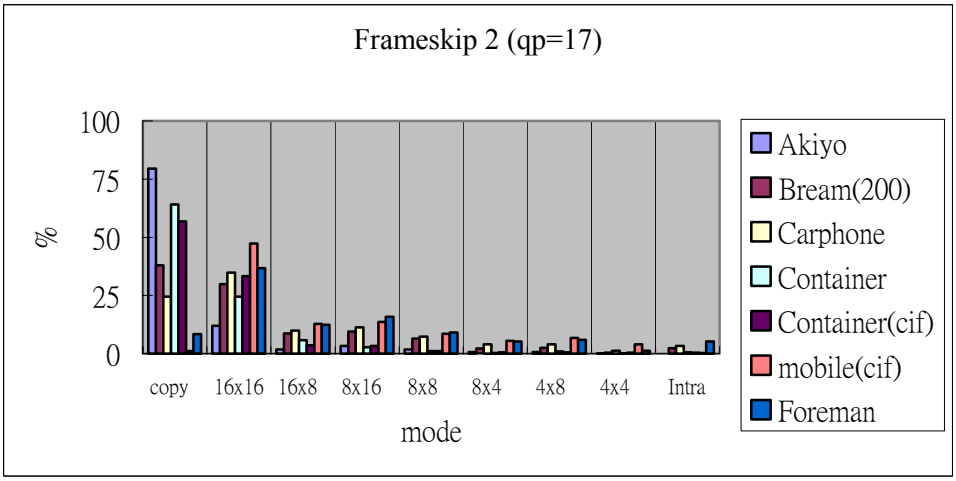


Figure 4.16 模式分佈 - 中位元率(Frameskip 2)

由 Figure 4.15 及 Figure 4.16 所示，Copy 及 16x16 還是佔了絕大多數，但可以由圖中注意到，當 Frameskip 張數增多之後，copy 及 16x16 的數目會減少，主要是因為跳的張數增多之後，位移的機率增加並且位移的量也增大，所以會使用到較小的區塊來編碼。

而在低位率時模式的選擇(QP 值等於 30)又是如何，可由 Figure 4.17、Figure 4.18、Figure 4.19 所示，和中位元率模式選擇

不同的地方是，16x16 模式增多了，並且在 8x8 以下區塊的模式幾乎沒有了。同樣的，當有 Frame skip 時，位移量增多，就比較需要用較小的區塊去編碼，但在低位元率只需要用到 8x8 以上的區塊模式就已經足夠了。

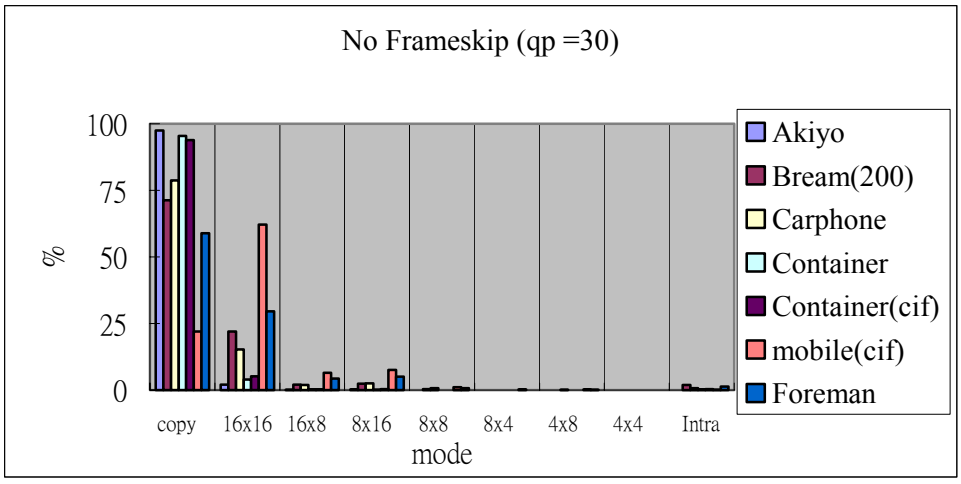


Figure 4.17 模式分佈 - 低位元率 (No Frame skip)

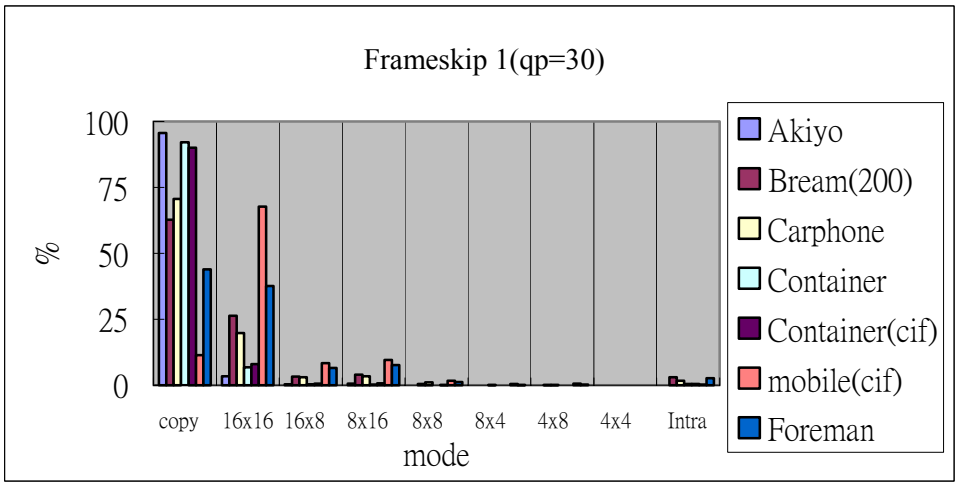


Figure 4.18 模式分佈 - 低位元率 (Frameskip 1)

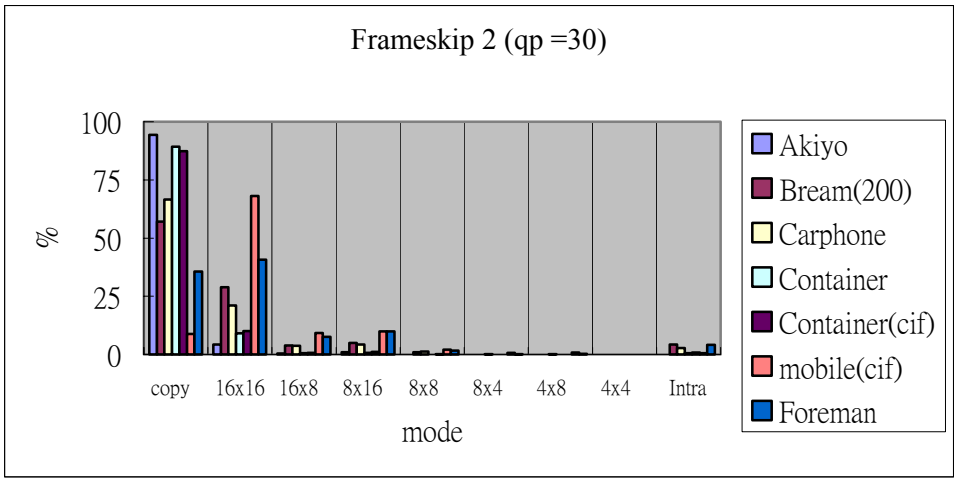


Figure 4.19 模式分佈 - 低位元率 (Frameskip 2)

而從以上的圖，所選用的模式幾乎都用較大的區塊，那究竟何時才會有大量使用較小的區塊來編碼，原來可由 Mode selection motion vector cost 看出，因為在估測每一個模式時，都會加上 Mvcost，如下式：

Mvcost=QP * MVBitUse(x,y)

所以當要使用較小的區塊編碼時，其 Motion Vector 數量會增加，而增加其 SAD 值的 Weighting，所以在中低位元率中，也就是 QP 值較大時，不輕易使用小的區塊編碼。但在高位率中，尤其是在 QP 值小於 5，小的區塊的效應就會明顯的增強，如 Figure 4.20、Figure 4.21、Figure 4.22 所示。如較靜態的 Akiyo、Container 及 Container(cif)多數還是在 16x16-8x8 之間，但在較動態的 Bream、Carphone、Mobile、Foreman，就會有出現用比較小的區塊去編碼。

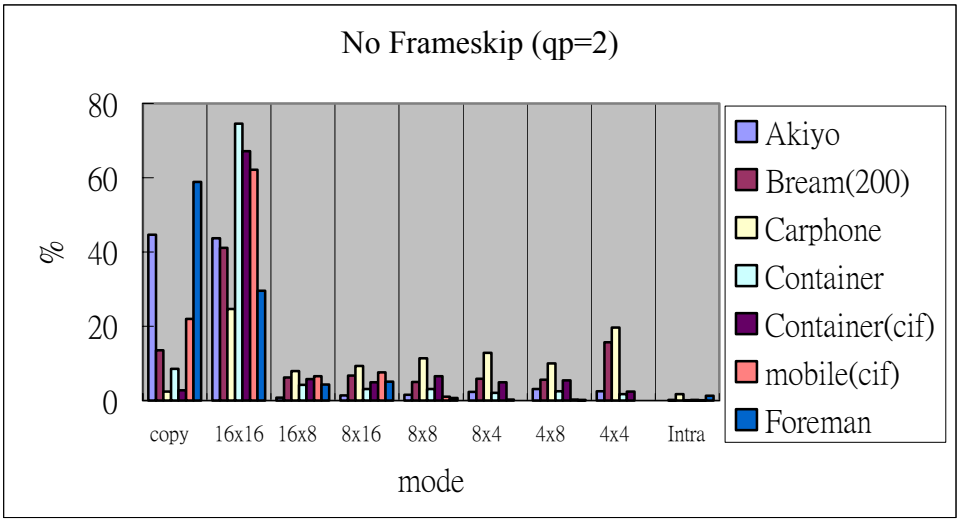


Figure 4.20 模式分佈 - 高位元率 (No Frameskip)

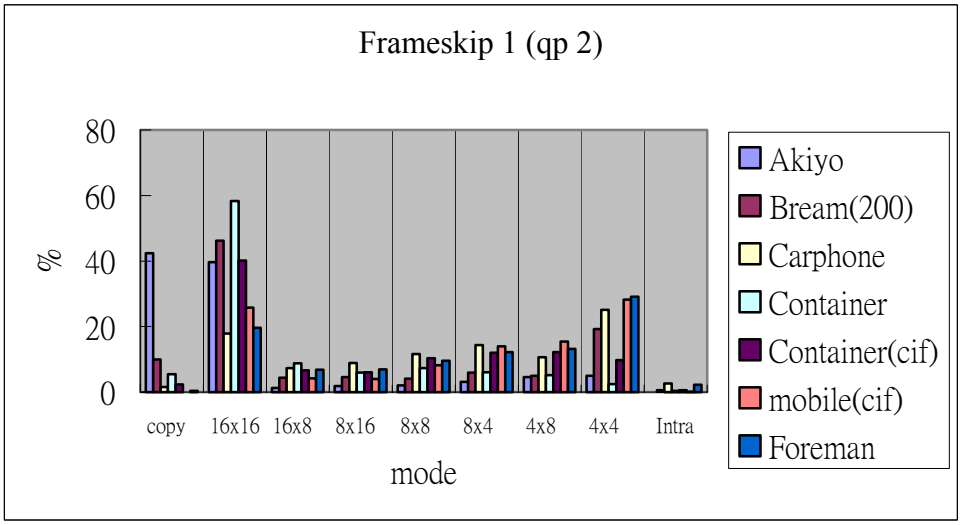


Figure 4.21 模式分佈 - 高位元率 (Frameskip 1)

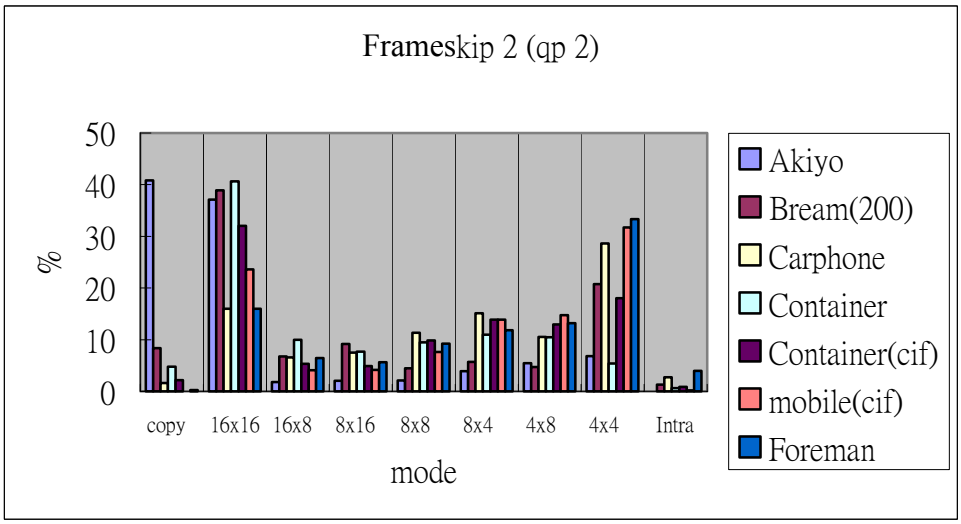


Figure 4.22 模式分佈 - 高位元率 (Frameskip 2)

所以同樣也在此下一結論，1)愈多張的 Frameskip 會造成位移量增多，使用大區塊(16x16)模式編碼的機率降低。2) 另外一方面，在低位元率時，使用 16x16-8x8 模式已經是幾乎足夠了，並且不大需要 8x8 以下的模式，但在中高位元率時，就需要七個模式都必須開啟，因為會出現一部份使用 8x4、4x8 及 4x4 等模式。

然而在各種情況下，模式的分佈都不一樣，但在一般的情況中，用大區塊(16x16)來編碼，佔了一定的使用機率，所以若能將大區塊的位移向量，做良好的預測，並且利用其搜尋後的一些特性，來預測小區塊的位移向量，這樣就或許可以達到加速的效果，並且可以有一定的畫面品質及減少位元率的增加。

4.3 運用既有連續消除消除演算法所算出的 16x16 SAD 值來降低複雜度

由 4.1 及 4.2 節所得的結果，可知 MSEA 會因為模式的增加，造成壓縮時間的增長，以及在模式的分佈中，以 16x16 區塊模式為最主要使用的模式。所以我們先分析用 TML8.0 (Full Search)，在單一 16x16 區塊模式來進行編碼，並與四個模式及七個模式的差別。採用的 Sequence 是 Foreman。

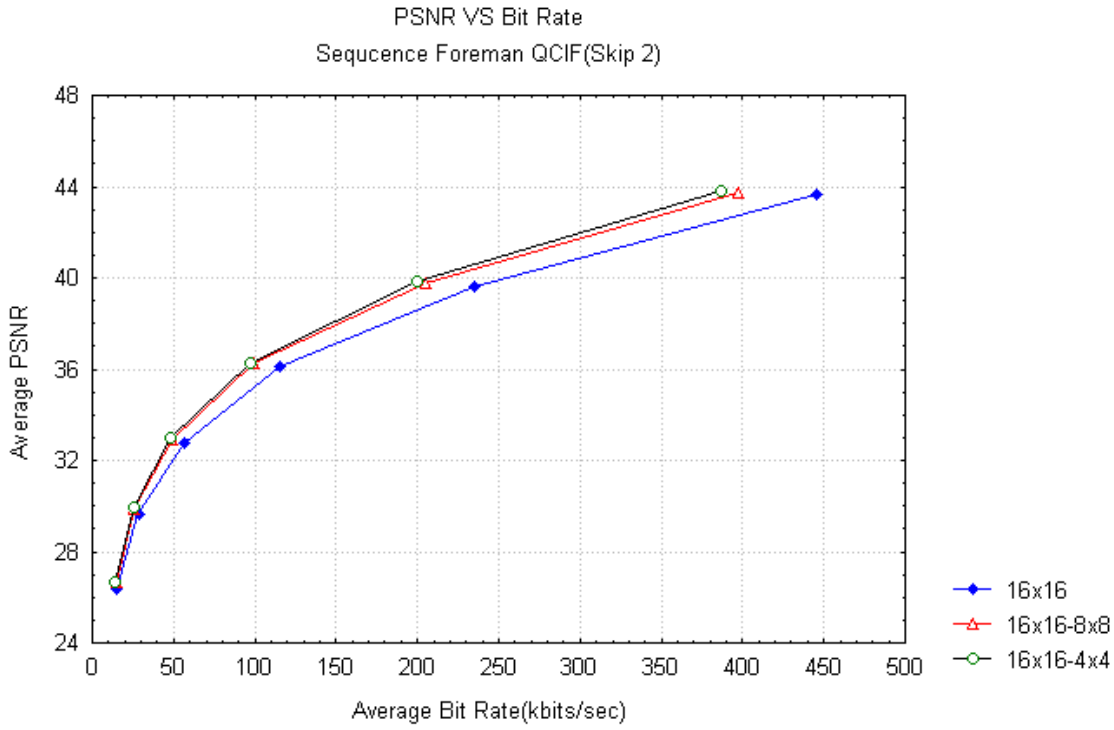


Figure 4.23 以Foreman Sequence使用1 Mode、4 Mode及7 Mode的Rate Distortion分析

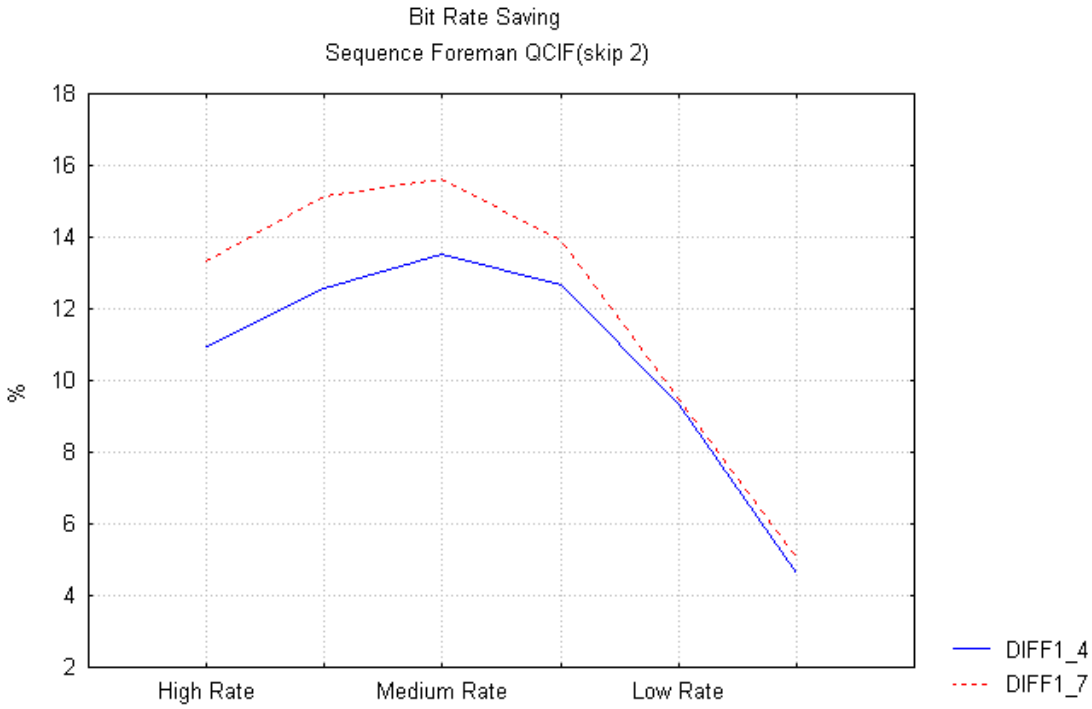


Figure 4.24 以Foreman Sequence在各種位元率下使用1 Mode、4 Mode及7 Mode所得到降低位元率比例圖

由 Figure 4.23 及 Figure 4.24 可知，subblock motion search 可降低位元率，並且可以在一定的位元率下，有更好的畫面品質。另一方面也可觀察到，誠如在 4.2 節所分析的，subblock 只佔了一些部份，若能將 16x16 預估準確，其餘的 subblock 可利用在 16x16 區塊模式下，所算出的 SAD 值，並且將此 SAD 值以 4x4 區塊為儲存的單位，這樣在小區塊的搜尋中，只需要將 4x4 區塊的 SAD 值，如同疊積木的，算出該模式大小的 SAD 值，來做為該區塊模式找尋精細位移向量(Motion Refinement)的起始點比較來源。

所以我們先觀察使用 SEA (or MSEA) 在 16x16 區塊搜尋後，所

算得的 suboptimal、optimal 位移向量的 SAD 值，是否正確命中 subblock 的最佳位移向量。那麼如何快速得到 16x16 區塊模式的最佳位移向量及殘餘的 SAD 值呢？如 4.1 節所得的結論，8x8 MSEA 在 16x16 區塊模式中可以得到較好的加速並與 FS 有同樣的位元率及 PSNR 值，所以本論文在 16x16 區塊的位移估計採用 8x8 MSEA。回

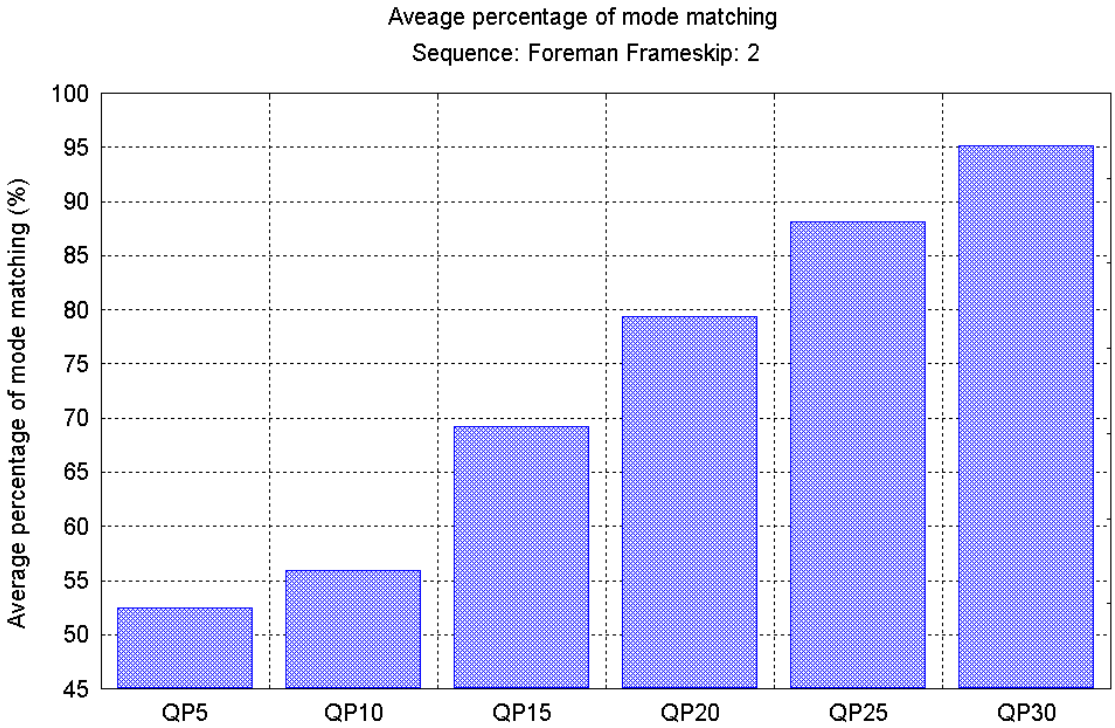


Figure 4.25 使用8x8 MSEA搜尋16x16區塊模式，並使用所得的SAD值繼續搜尋16x8 – 4x4區塊模式，並和TML 8.0區塊模式的準確度

到之前所談的，在 16x16 區塊利用 8x8 MSEA 所儲存下來的 SAD 值中，是否可以命中以 FS 在 subblock 所找到的 optimal 的位移向量？由 Figure 4.25 和 4.26，表示利用 8x8 MSEA 及其殘留資訊可找到最佳模式及最佳位移向量的百分比。由此兩圖中，在低位元率(QP 值大)時，所得到的效果不錯，可達九成，但當 QP 值慢慢變小時，也就位

元率昇高時，利用殘餘的資訊，就無法有效並準確的找到最佳的模式及位移向量。主要的原因可由 4.2 節所描述的，在中高位元率時，小區塊模式增加，所以更需要在小區塊中搜尋，因此必須加入精確(Refine)位移向量的動作。

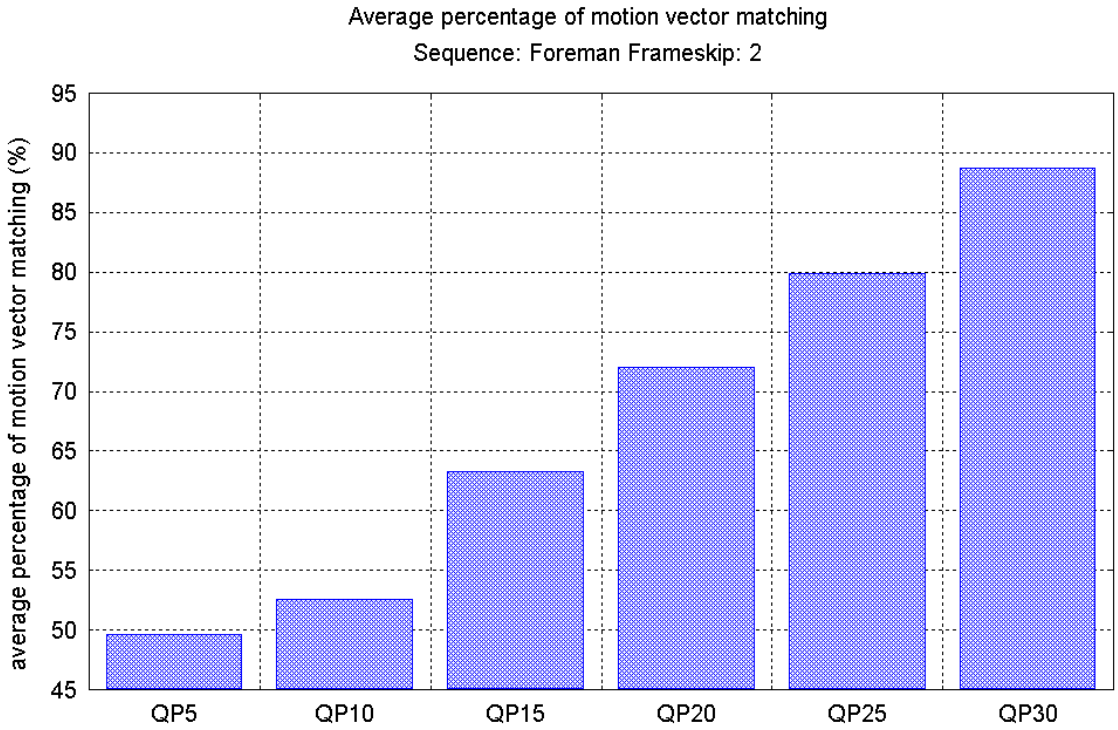


Figure 4.26 使用8x8 MSEA搜尋16x16區塊模式，並使用所得的SAD值繼續搜尋16x8 – 4x4區塊模式，並和TML 8.0位移向量的準確度

再由 Figure 4.27 來比較 Full Search 和利用 8x8MSEA 所殘留下 SAD 搜尋結果的模式分佈，可知 16x16 模式和 Intra 模式增加，而 16x16 以下的模式所佔的比率都降低，愈小區塊模式降低愈多，主要是因為利用 16x16 所搜過的 SAD 值，來求得小區塊的位移向量因資訊不足而不夠精確，所以必須要作局部的 Refinement，以增進其準確度。

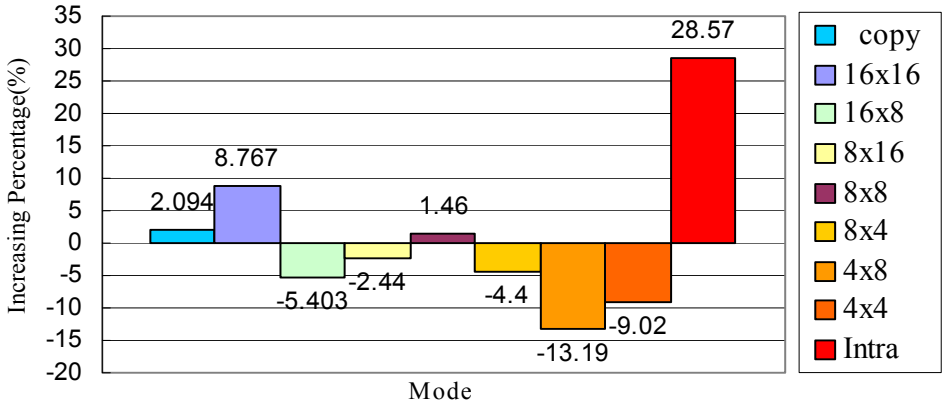


Figure 4.27 以 8x8MSEA 來搜尋 16x16 區塊模式，再利用其殘餘的 SAD 值，往下搜尋小區塊的位移向量，與 TML8.0 所產生模分佈的差異圖

4.4 精細小區塊的位移向量(Motion Refinement)

誠如 4.3 節所述，雖可以重覆利用 16x16 區塊所計算出來的 SAD 值，但所找到的各區塊模式的 Suboptimal 位移向量不夠精準，所以還是有必要局部搜尋，以增進位移向量的精確度。而要找的各區塊模式的最佳位移向量離所找到的 Suboptimal 位移向量有多遠，可由 Figure 4.27 看出，0x0 代表完全命中，3x3 表示離開此點的一個 Pixel 距離的位置，以此類推。並且為了確保正確，我們利用不同的 QP 值來測試其在高位元率及低位元率是否如此。位移向量完全相同的百分比，會因為高低位元率(QP 值的大小) 而有不同，但由其周圍附近可找到最佳位移向量機率很大，尋找周圍八個點就可以有百分之七十至九十的機會，可以找到最佳的點，所以在此印證，有必要對小區塊位移向量做精細的動作，以得到最佳的位移向量。

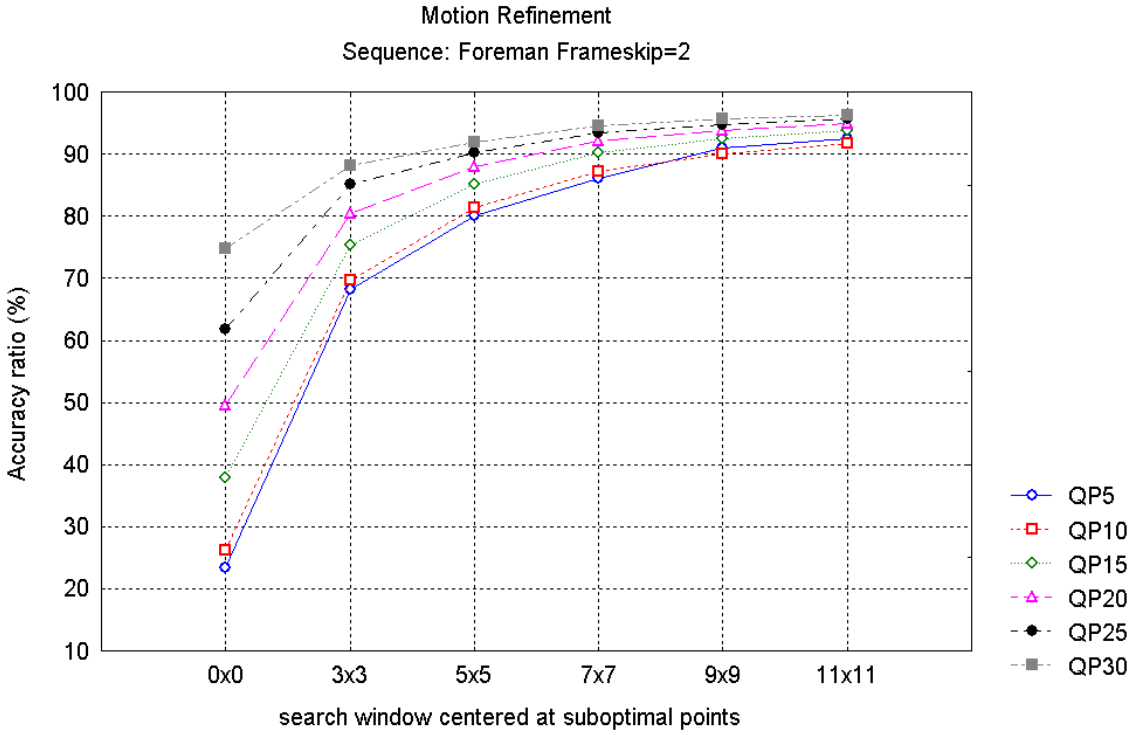


Figure 4.28 以16x16所搜尋過的點，利用其SAD值，往下搜尋小區塊的位移向量，得到其中最小的一個，圖中的距離就是TML8.0和上述方法的距離統計圖

由於是局部的搜尋，所以想取得增加搜尋點數及增加效能之間的最佳平衡點，所以本論文使用了三個不同的方法來進行精確的動作。分別是以 3x3、5x5 為搜尋視窗及二步搜尋法，如 Figure4.29 ~ Figure4.31 所示。

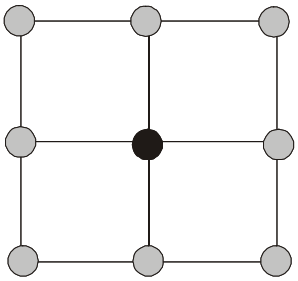


Figure 4.29 3x3 搜尋視窗

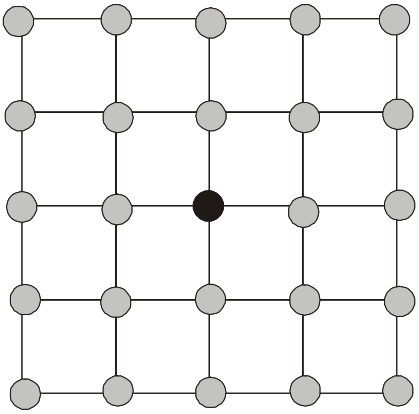


Figure 4.30 5x5 搜尋視窗

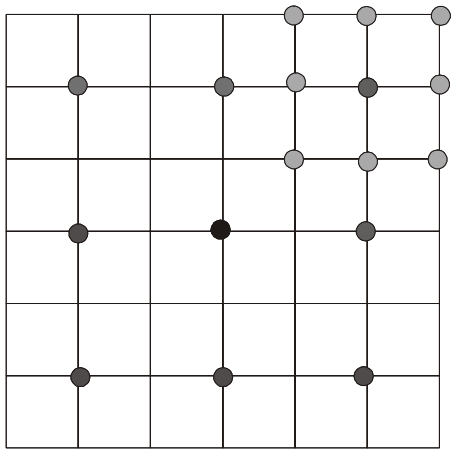


Figure 4.31 二步搜尋法

再由 Figure 4.32 中，可以觀察到，在中高位元率下，使用 3x3 搜尋視窗來做為精確小區塊位移向量的效能，可使位移向量相似度從百分之五十增至百分之七十。而以 5x5 搜尋視窗及二步搜尋法 (類似三步搜尋法)，所提昇的百分比，效能會以線性的上昇，也就是所搜尋的點數，會因為所找的點數而有增加，但時間花費也會因此增加。所以本論文會以 3x3 搜尋的範圍，來精細小區塊的位移向量。

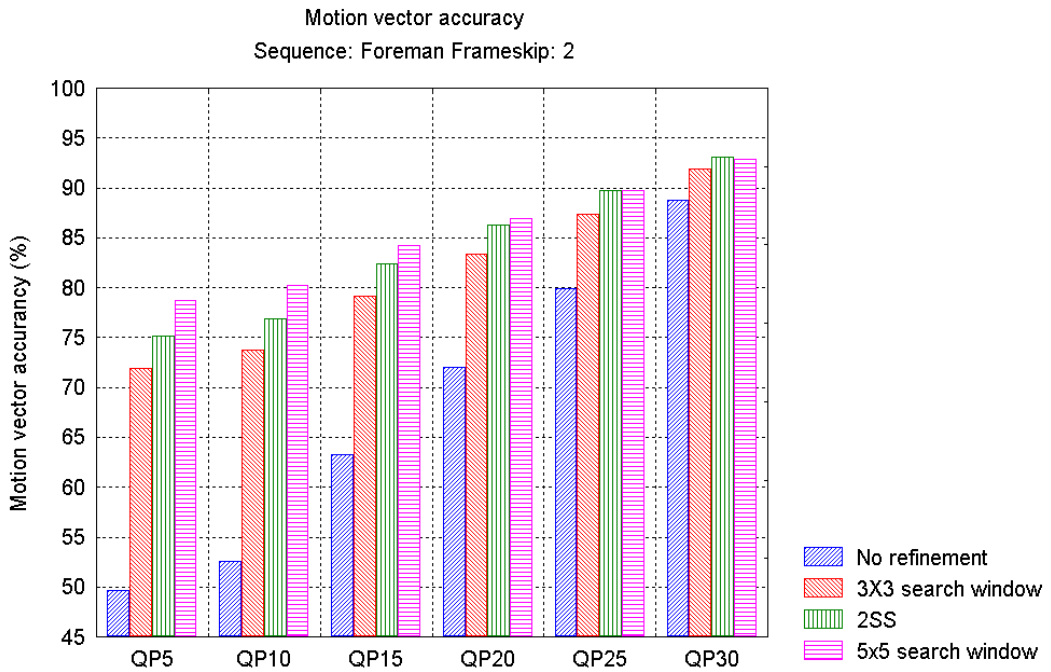


Figure 4.32 利用周圍3x3 、5x5 search window及2SS(two step search)的方式來進行精確小區塊中的位移向量相似度的比較

4.5 中斷決策(Half Stop Decision)

由 4.2 節中模式分析中，可知一般模式的分佈是以 16x16 為最多，其次是 16x8、8x16、8x8，再來 4x8、8x4，及 4x4 的數目就是少數了，所以是否可以減少要搜尋的模式，成為本節中的重點。一般而言，最佳的模式若是小區塊的話，代表區塊中有細部的微動量，並且程式在搜尋時都以大區塊為第一優先之後，才去搜尋小的區塊，所以我們可以利用當區塊漸漸縮小搜尋時，去檢查最佳移動向量是否有往小區塊模式發展的趨勢，若沒有的話，程式就立刻停止並且換下一個 macroblock 去編碼。由 Figure 4.33，代表若搜尋 8x8 區塊時，搜尋完之後，若是最佳的模式，這代表區塊要往下分的趨勢，此時我們就繼續往小區塊搜尋，否則，8x4、4x8 及 4x4 這三個模式的搜尋，則可省略，以增加編碼效能。再從 Figure 4.34 看到，用中斷決策可達到八成的準確率，所以可用這個機制來減低搜尋小區塊的計算量。

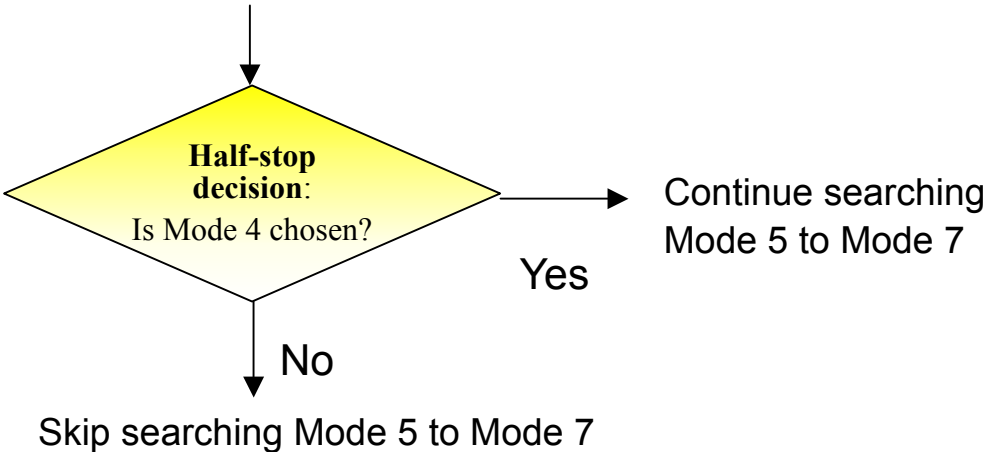


Figure 4.33 Half stop decision 流程圖

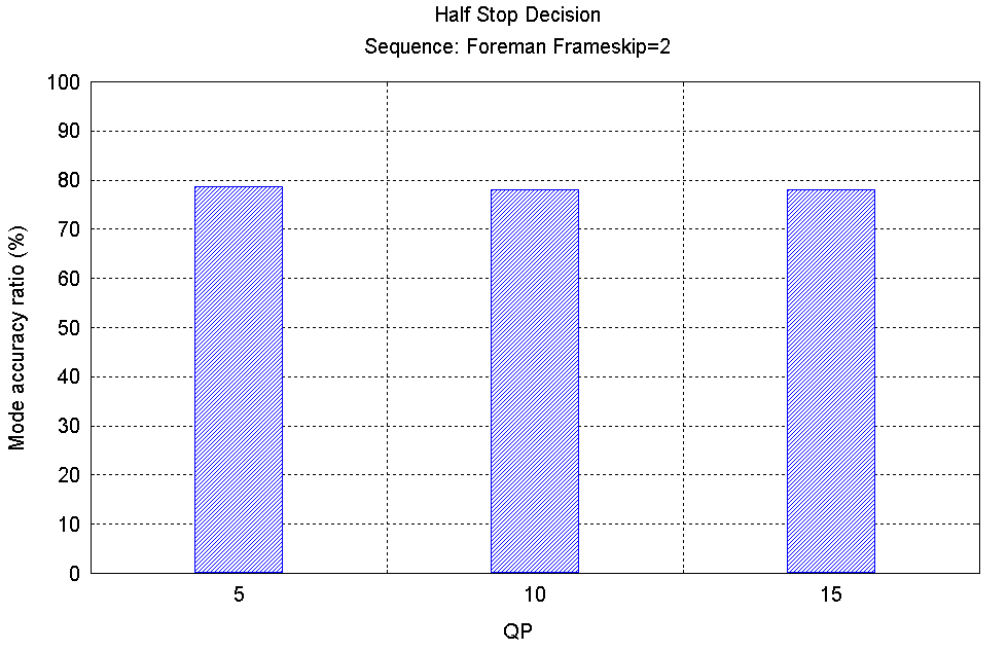


Figure 4.34 利用Half stop decision 判斷成功的機率

4.6 快速多階連續消除演算法流程圖

本論文所提出的方法，是結合多階連續演算法、Motion Refinement 及 Half-Stop Decision，其最主要是要改善如 4.1 節中，(多階)連續消除演算法在 H.26L 多重模式移動預估的環境中，所造成加速效果不佳的狀況、所算的 SAD 值太多的情況。主要利用的方法，有四：1)利用 8x8 多階連續演算法來尋找大區塊(16x16)的最佳的位移向量，2)再利用尋找大區塊位移向量時在中間過程所算出的 SAD 值，求出其它小區塊中的 Suboptimal 的位移向量。3)再將 2)中，所找出的位移向量進行 Motion Refinement，Refinement 的方法是使用 3x3 搜尋視窗法 4)利用較大區塊的搜尋結果，來決定是否要繼續搜尋小區塊的位移向量，即所謂的 Half-Stop Decision，整個 FMSEA 的完整流程圖，

請見 Figure 4.35。

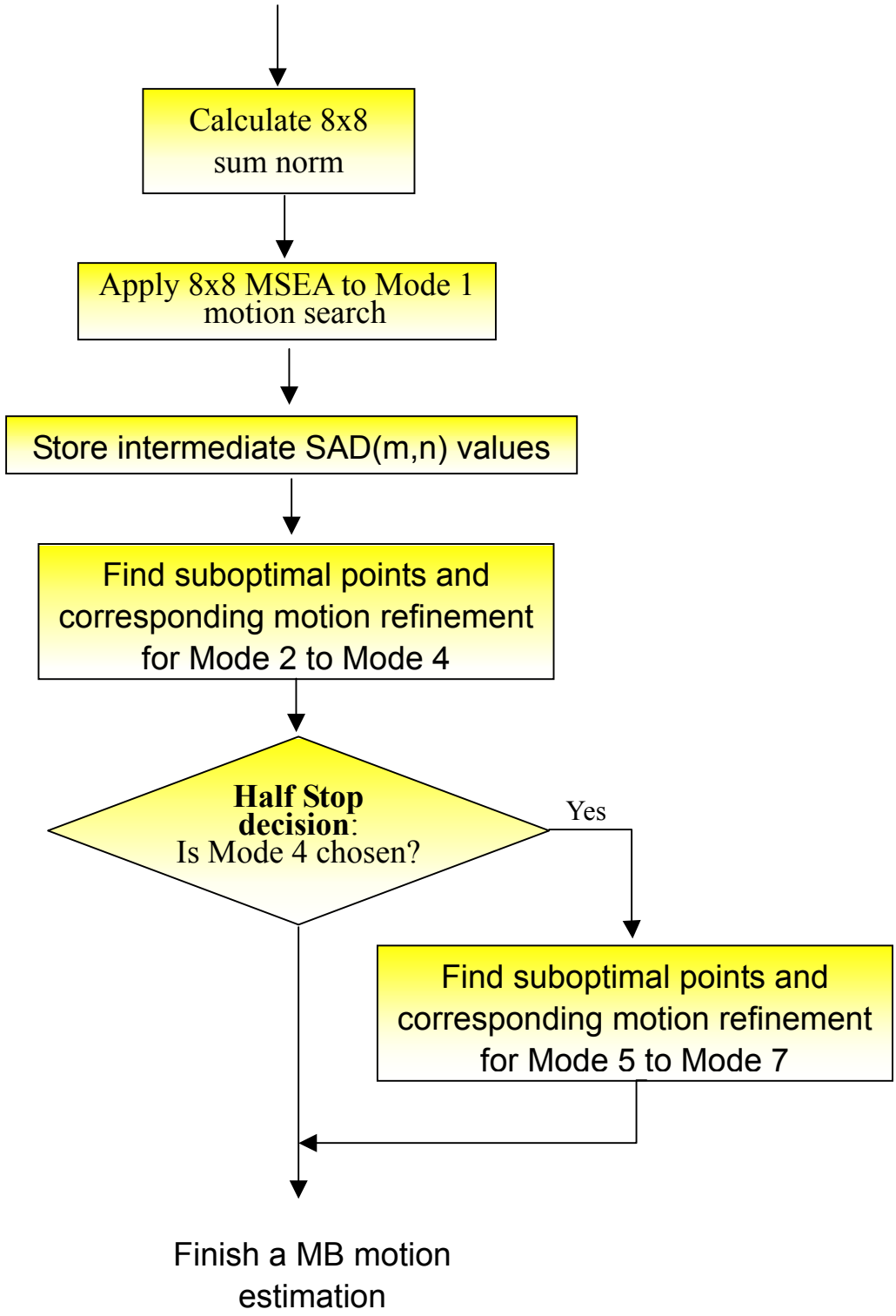


Figure 4.35 快速多階連續消除移動估計演算法流程圖

第五章 實驗結果與討論

在前面我們詳細地介紹了已發展出來的快速演算法及本論文所提出的快速多階連續消除演算法。接著本章要進行模擬，並且且討論所得到的數據。

5.1 環境參數與所使用的視訊樣本

視訊樣本的種類有很多種，可由 Table 5.1 得知樣本的分類，其中有動作較慢的、較快的及在空間域上較複雜的。

Table 5.1 視訊樣本的分類表

Sequence Name	Class	Sequence Name	Class	Sequence Name	Class
Akiyo	A	Carphone	B	Fun Fair	D
Container Ship	A	Table Tennis	C	Bream	E
Foreman	B	Stefan	C	Weather	E
News	B	Mobile & Calendar	C		
Silent Voice	B	Football	C		
Coastguard	B	Tunnel	D		

Class A: Low spatial detail and low amount of movement
Class B: Medium spatial detail and low amount of movement or vice versa
Class C: High spatial detail and medium amount of movement or vice versa
Class D: Stereoscopic
Class E: Hybrid natural and synthetic

本論文所使用的視訊樣本，有 Class A Akiyo、Class B News、

Carphone、Foreman、Class C Stefan、Class E Bream。由於手上並沒有 Class D 的視訊樣本，所以暫時無法測試。

Class A Akiyo

Class B News

Class B Carphone

Class B Foreman

Class C Stefan

Class E Bream

系統模擬環境，列於以下：

CPU：1GHz Pentium Ⅲ

Memory：256MB SDRAM

OS：Microsoft 2000 Server + Service Pack 2

Compiler：Intel C++ 6.0

H.26L(TML8.0)上的參數：

Sequence type：IPPP

Image Format：QCIF

Frame Rate：10 fps

Reference frames used in P prediction： 1

Blocktytype: 7 modes

Hadamard transform： Not used

Search range restrictions: none

MV Resolution: 1/4-pel

RD-optimized mode decision: Low Complexity

Search Range in motion estimation：16 for 10fps

Entropy coding method: UVLC

QPs used：9、12、15、18、20 and 23

所有測試編碼器執行的時間是使用 Ansi C clock() funtion。

5.2運用快速多階連續消除移動預估演算法加速的效果

在本節中，主要目的是要顯示本論文所提出的快速多階連續消除演算法的加速效果，分別會以全域搜尋(Full Search)、多階連續消除演算法(MSEA)、三步搜尋法(3SS)、四步搜尋法(4SS)、鑽石搜尋(DS)與本論文所提出之方法做比較。比較基準是以每壓一張 Frame，做為比較的標準，至於為何不使用檢查點(check point)，作為比較的標準，是因為多階連續消除演算法，所要算的 sum norm 量無法轉換成檢查點，並且本論文所提出的中斷模式(Half Stop)，會使得較小區塊的三個模式區塊不做，而會使實驗的比較，較不為客觀。以下是實驗結果的分析，由 Figure 5.1 至 Figure 5.6，分別依序是 Akiyo、News、Carphone、Foreman、Stefan 及 Bream，在不同 Rate 之下壓縮所花費時間圖。最後再由 Table 5.2，來總結各種快速演算法加速的效果。

Akiyo

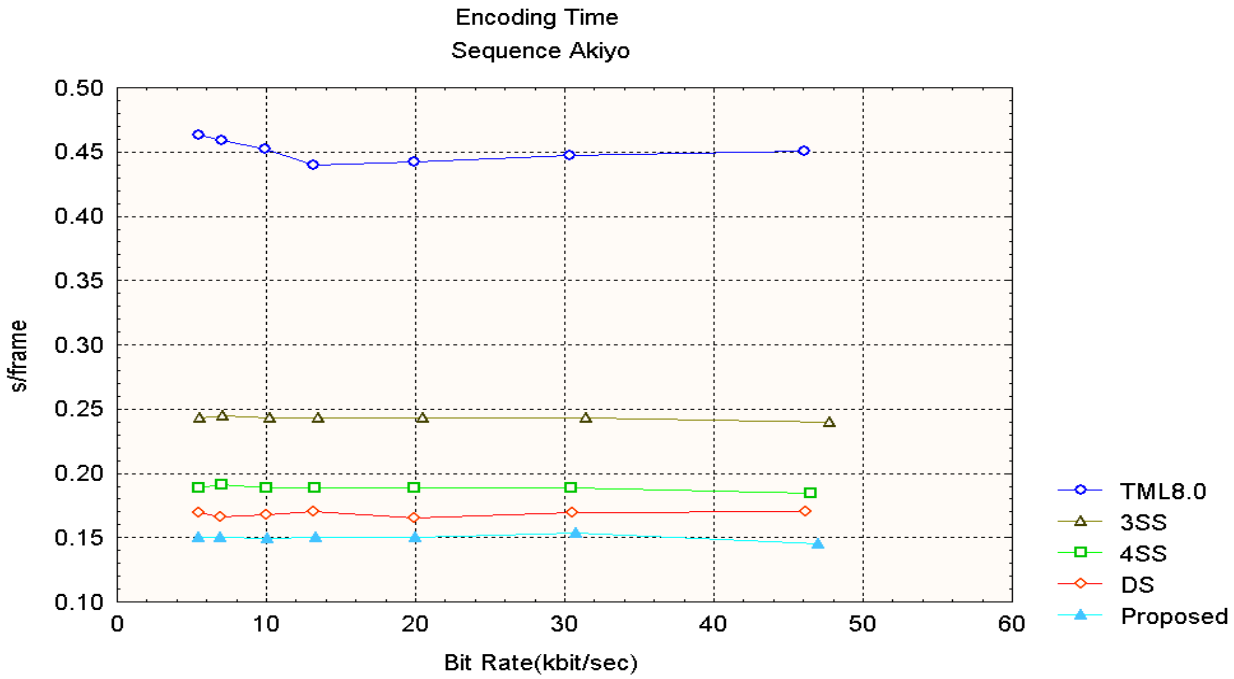


Figure 5.1 各種快速演算法用在 Akiyo Sequence 編碼時間的比較

News

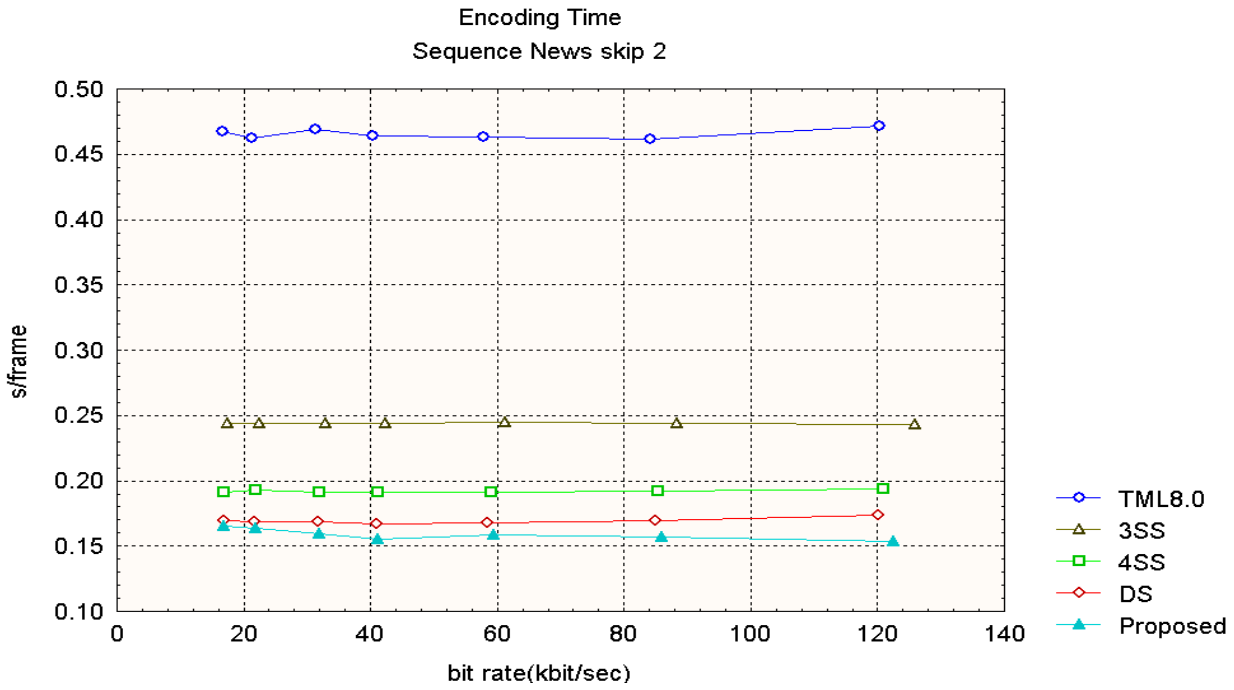


Figure 5.2 各種快速演算法用在 News Sequence 編碼時間的比較

Carphone

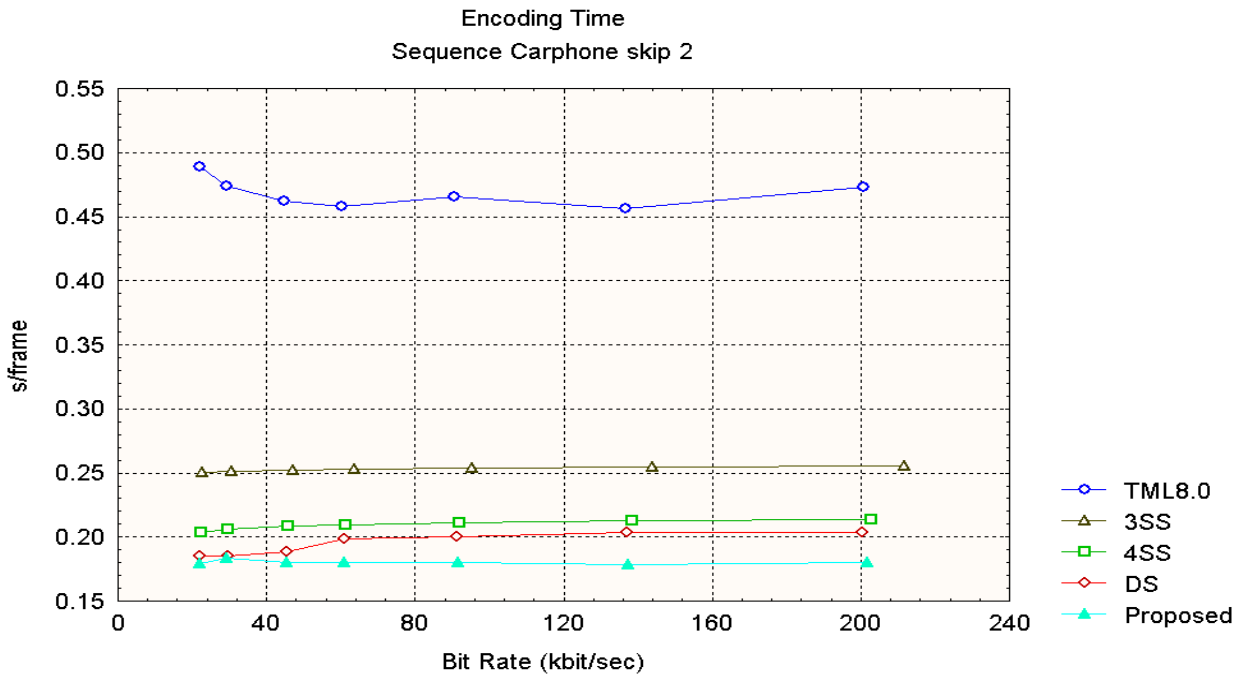


Figure 5.3 各種快速演算法用在 Carphone Sequence 編碼時間的比較

Foreman

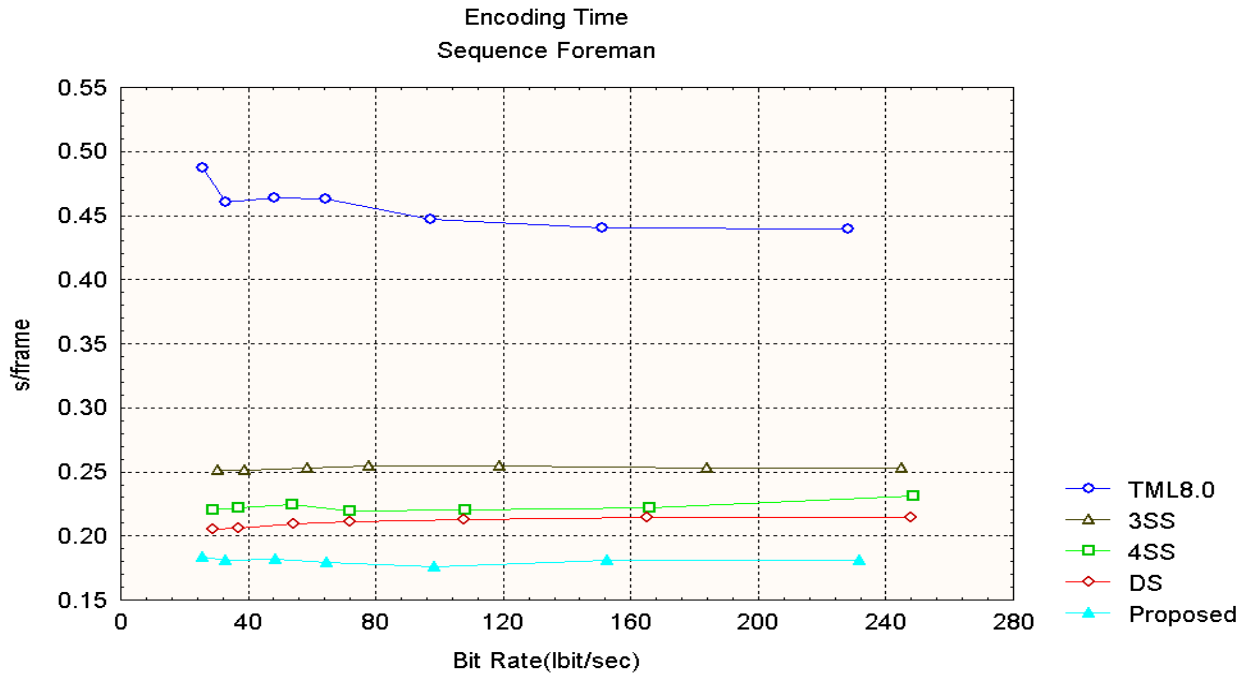


Figure 5.4 各種快速演算法用在 Foreman Sequence 編碼時間的比較

Stefan

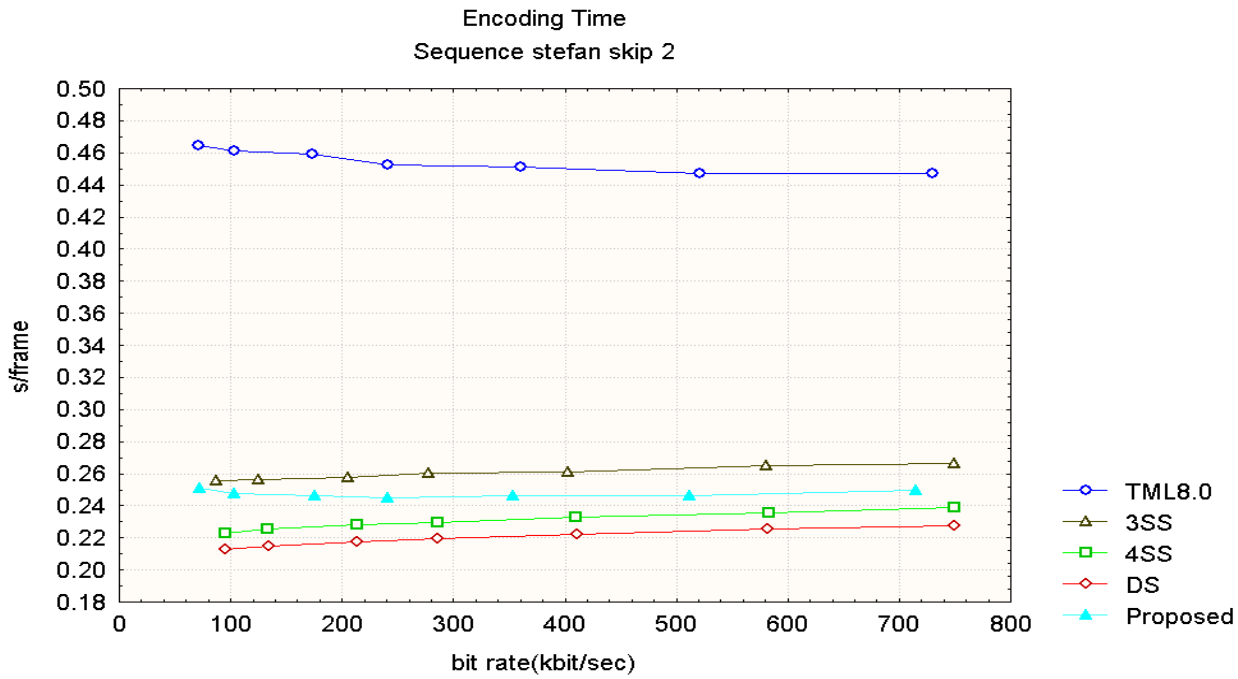


Figure 5.5 各種快速演算法用在 Stefan Sequence 編碼時間的比較

Bream

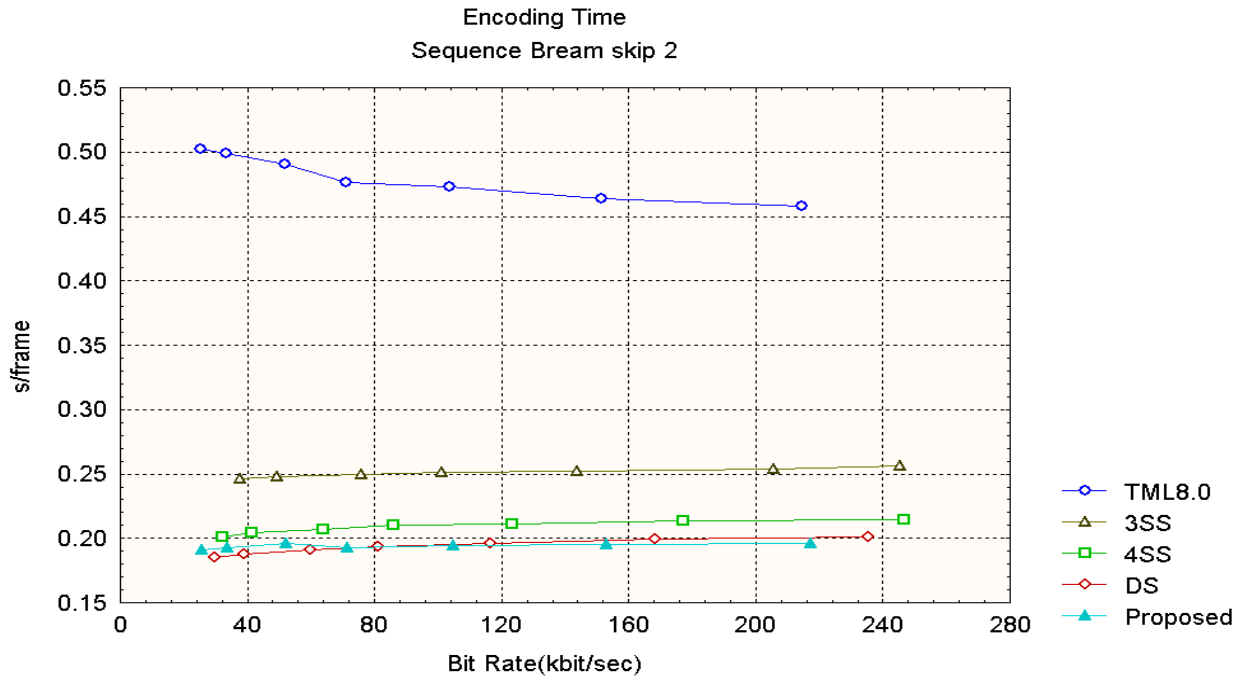


Figure 5.6 各種快速演算法用在 Bream Sequence 編碼時間的比較

Table 5.2 各種快速演算法編碼張數比較表

Algorithm / Sequence	TML 8.0	MSEA (4x4)	3SS	4SS	DS	FMSEA	Ratio (FMSEA/TML8.0)
Akiyo	2.22	1.54	4.12	5.3	5.96	**6.48**	291%
News	2.15	1.49	4.1	5.21	5.9	**6.28**	292%
Carphone	2.13	1.47	3.95	4.78	5.12	**5.55**	260%
Foreman	2.19	1.33	3.95	4.49	4.75	**5.54**	253%
Stefan	2.20	1.26	3.84	4.34	**4.54**	4.04	184%
Bream	2.08	1.30	3.98	4.78	**5.17**	5.15	248%
Average	2.16	1.40	3.99	4.81	5.24	5.51	255%

由 Figure 5.1 到 5.5 及 Table 5.2 實驗數據顯示，在 Class A 及 B 的視訊樣本中，本論文所提出的方法，皆有較快的加速效果，而在 Class D、E 中加速的效果稍微差了一點，主要的原因，是因為 Stefan 和 Bream 是屬於較動態的視訊樣本，而使得在 16x16 區塊模式，要多增加一些檢查點，來算 SAD 值，而所增加的檢查點數是否會增加編碼器的效能，答案是肯定的，可由下一節的實驗結果發現，本論文所提出的方法，雖然在較動態視訊樣本下，加速沒有這麼的快，但在位元率比對畫面品質 (Rate distortion) 的比較上有極佳的表現。

5.3 Performance 評估

在上節中，討論用不同方法所得到的加速效果，本節中則以探討不同的方法，所造成位元率昇高及畫面品質的降低為主。測試的標準，是以 Rate-Distortion 的方式來評比。由 Figure5.7 到 5.16，依序是 Akiyo、News、Carphone、Foreman、Stefan 及 Bream。

以 Class A 的 Akiyo 及 Class B 的 News 來說，由於這兩個視訊的樣本是較靜態的，所以所有比較的方法的效能差異不會很多。但從 Class B Carphone 開始到 Class E Bream 中，就可以看出在各快速演算法增加編碼器的速度之際，所犧牲壓縮效能及畫面品質為如何。以較明顯的 Class B 的 Foreman、Class C Stefan、及 Class E Bream 來說，在相同的位元率下，三步搜尋法、四步搜尋法及鑽石搜尋法，分別使畫面的品質下降了 1-2dB (3SS)、0.5-1dB(4SS)、0.5-0.6dB(DS)，而由本論文所提出的方法，其效能和全域搜尋 (Full Search)演算法的效能幾乎相同，最主要是原因是本論文有效掌握 H.26L 在多個模式的特性而使得搜尋到的位移向量比其他的快速演算法更為精確，並極為趨近於 TML8.0(Full Search)的搜尋結果。

Akiyo

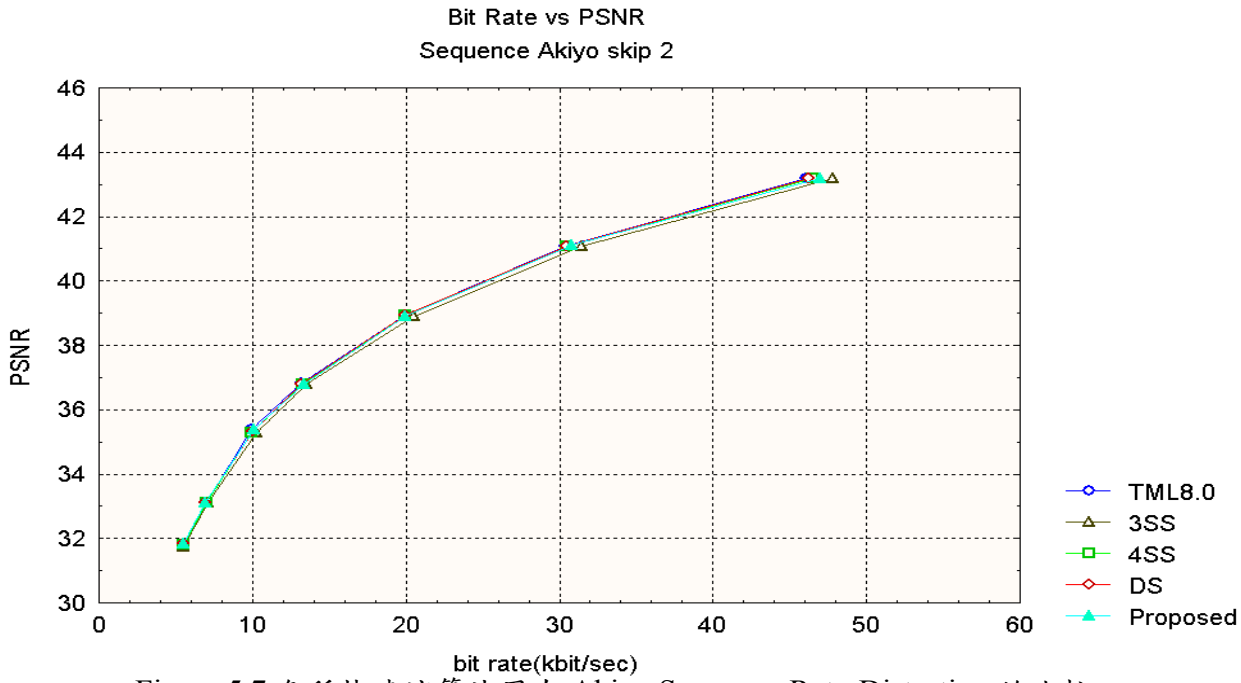


Figure 5.7 各種快速演算法用在 Akiyo Sequence Rate-Distortion 的比較

News

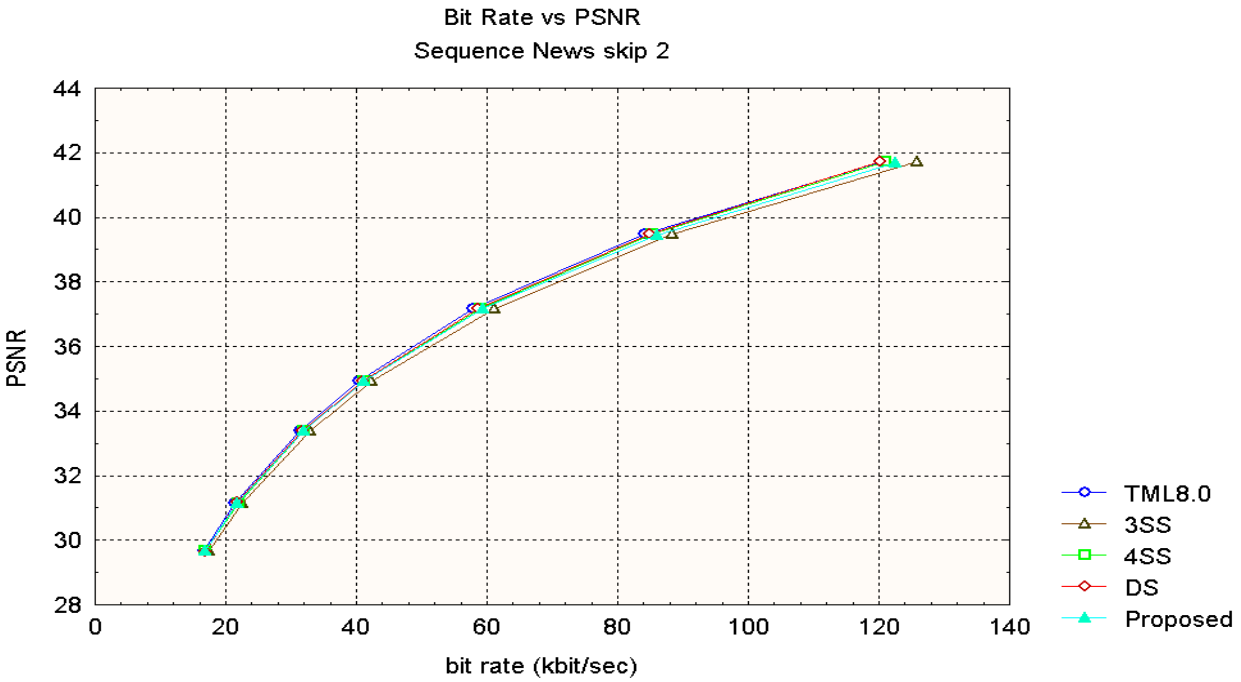


Figure 5.8 各種快速演算法用在 News Sequence Rate-Distortion 的比較

Carphone

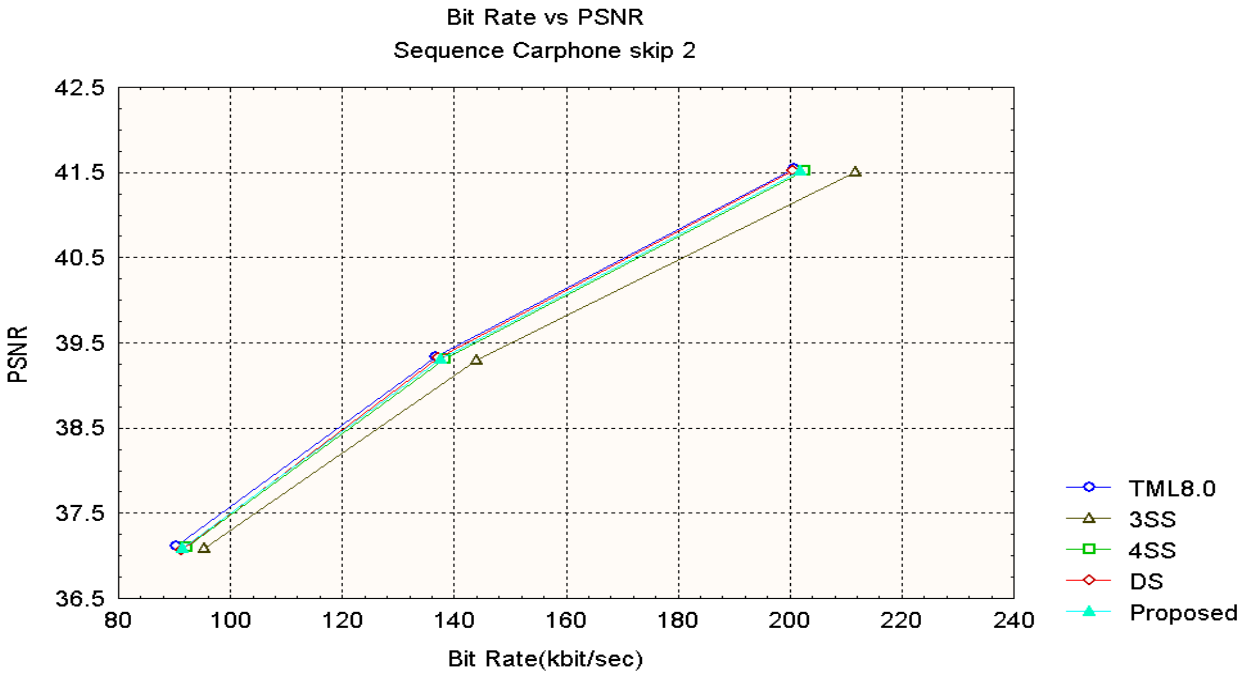


Figure 5.9 在高位元率，各種快速演算法用在 Carphone Sequence Rate-Distortion 的比較(1)

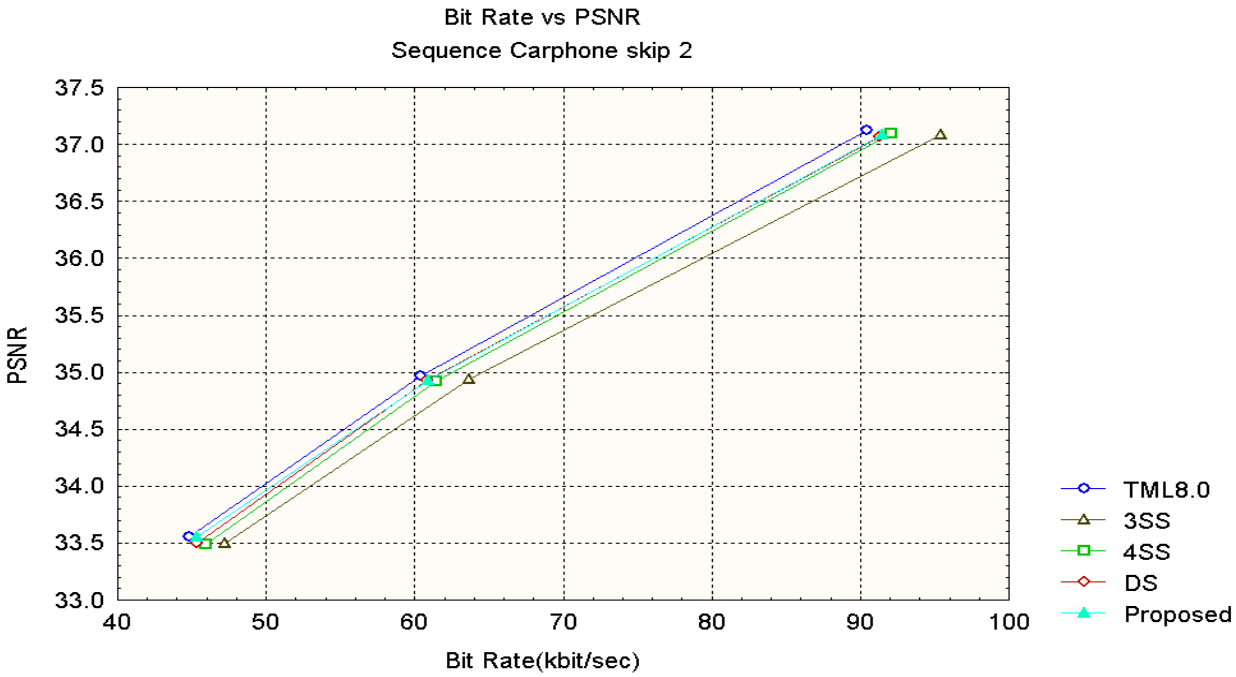


Figure 5.10 在中位元率，各種快速演算法用在 Carphone Sequence Rate-Distortion 的比較(2)

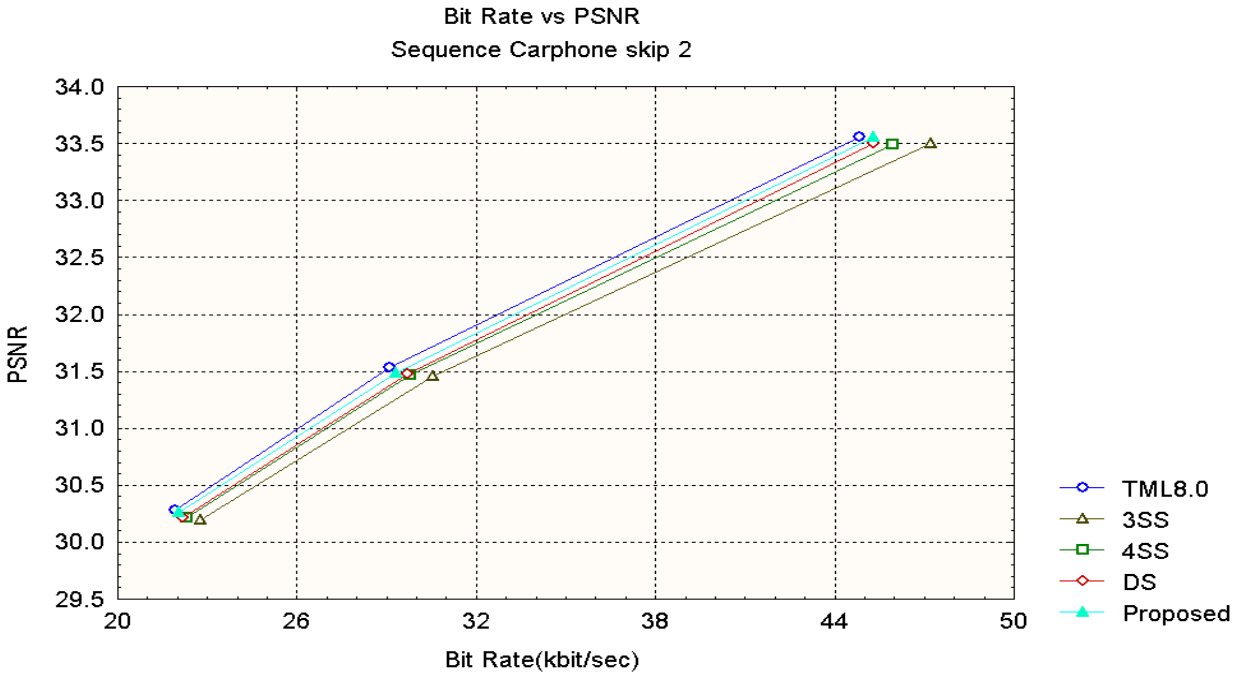


Figure 5.11 在低位元率，各種快速演算法用在 Carphone Sequence Rate-Distortion 的比較(3)

Foreman

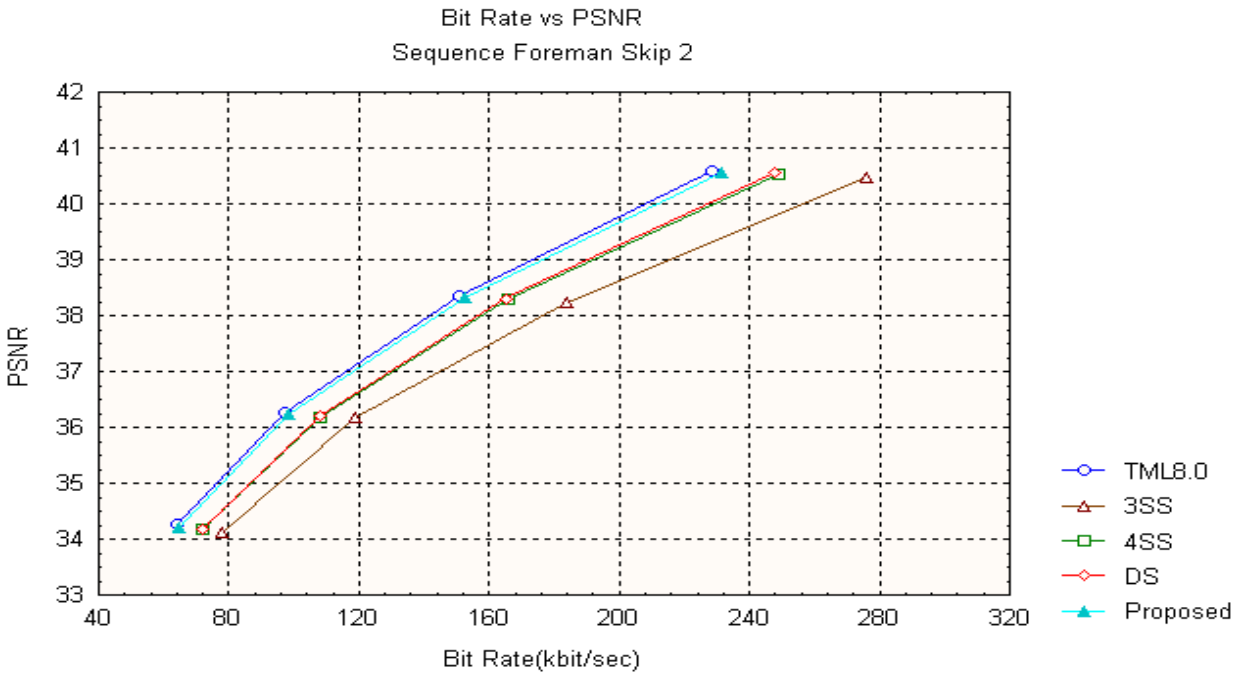


Figure 5.12 在中高位元率，各種快速演算法用在 Foreman Sequence Rate-Distortion 的比較(1)

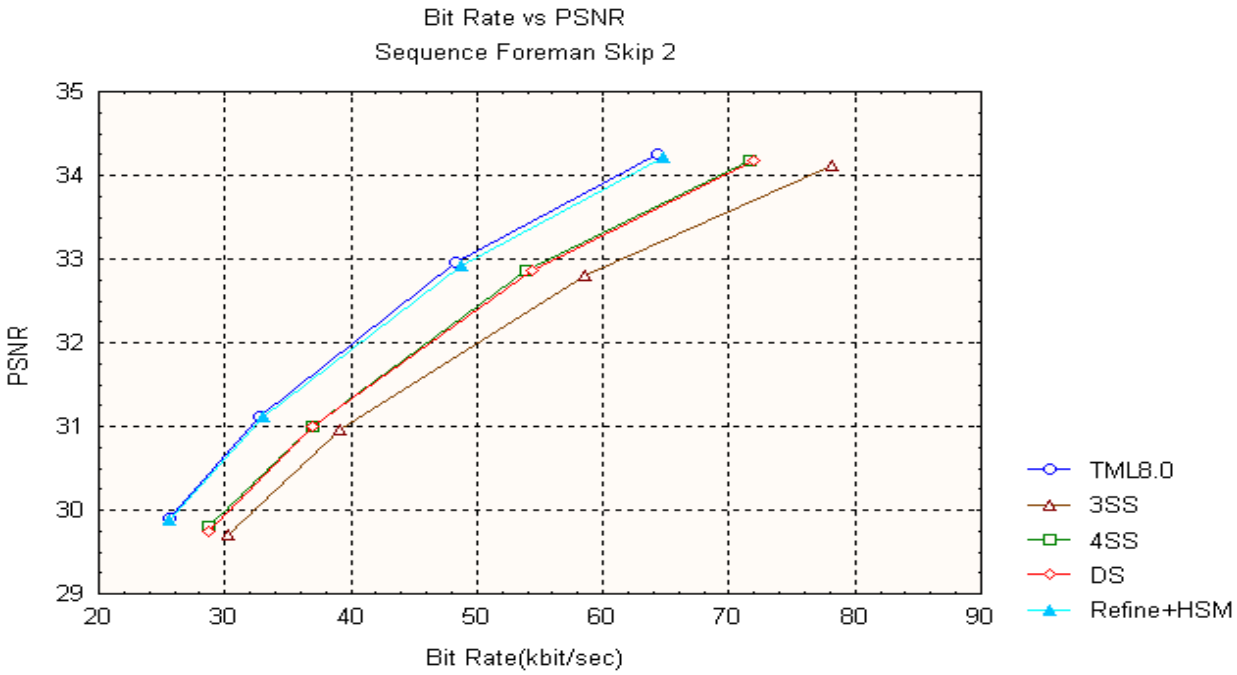


Figure 5.13 在中低位元率，各種快速演算法用在 Foreman Sequence Rate-Distortion 的比較(2)

Stefan

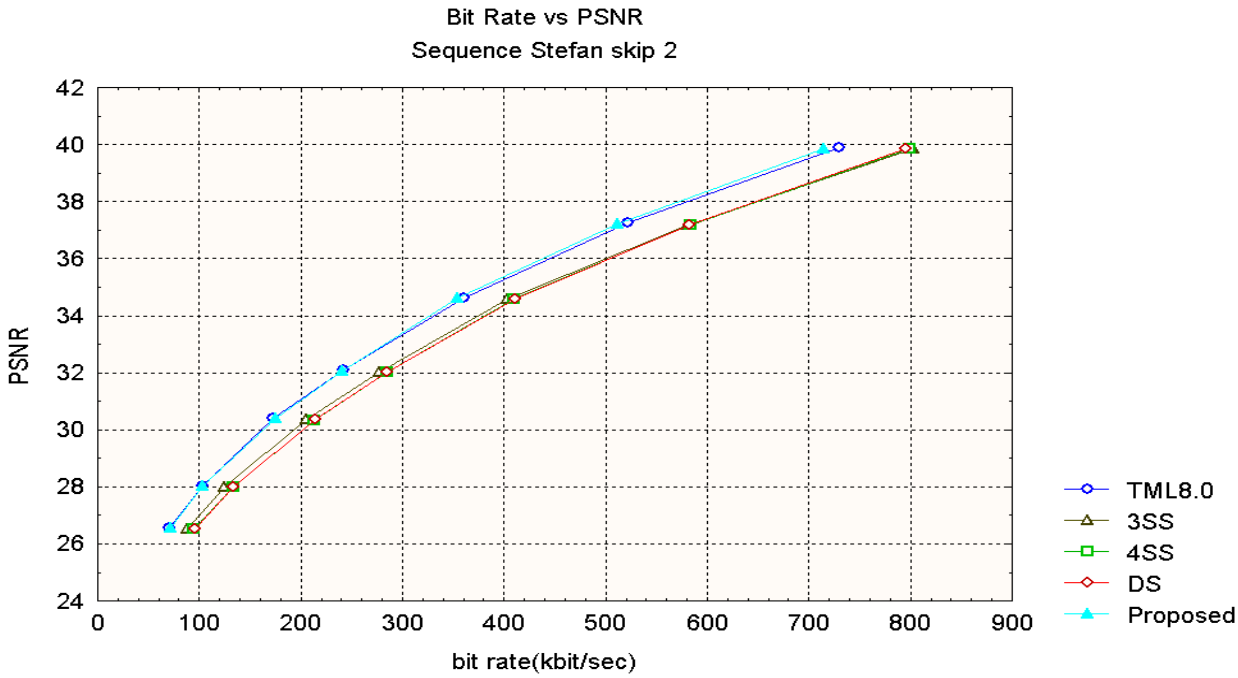


Figure 5.14 各種快速演算法用在 Stefan Sequence Rate-Distortion 的比較

Bream

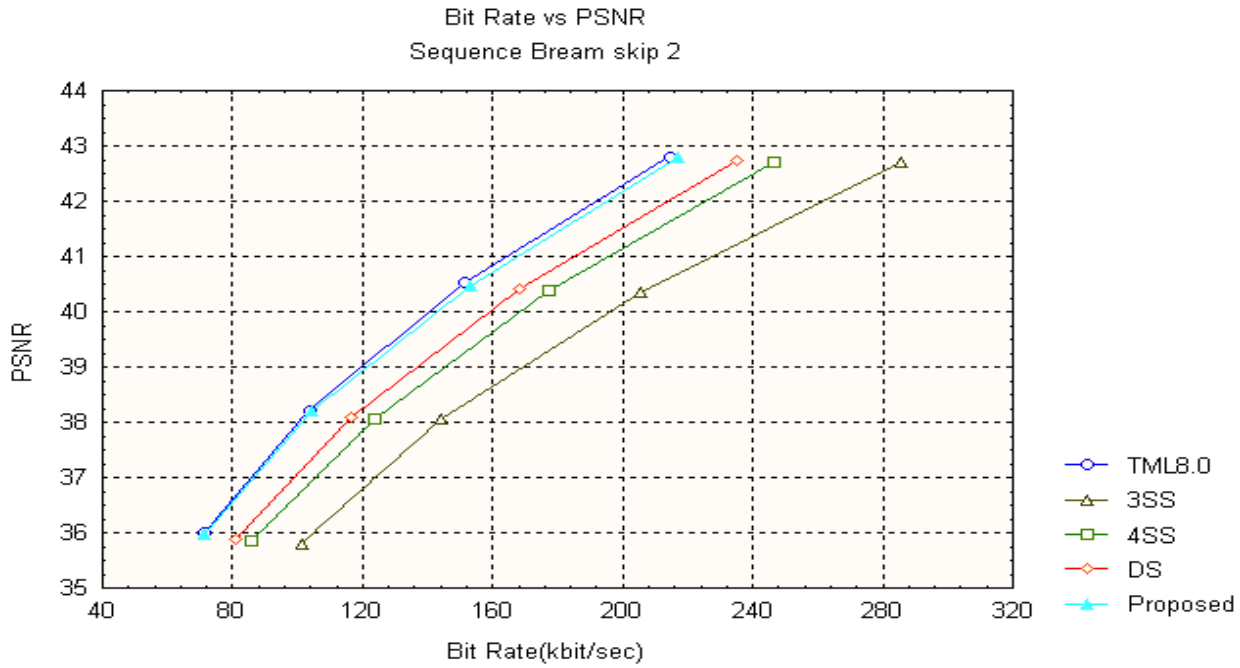


Figure 5.15 在中高位元率，各種快速演算法用在 Bream Sequence Rate-Distortion 的比較(1)

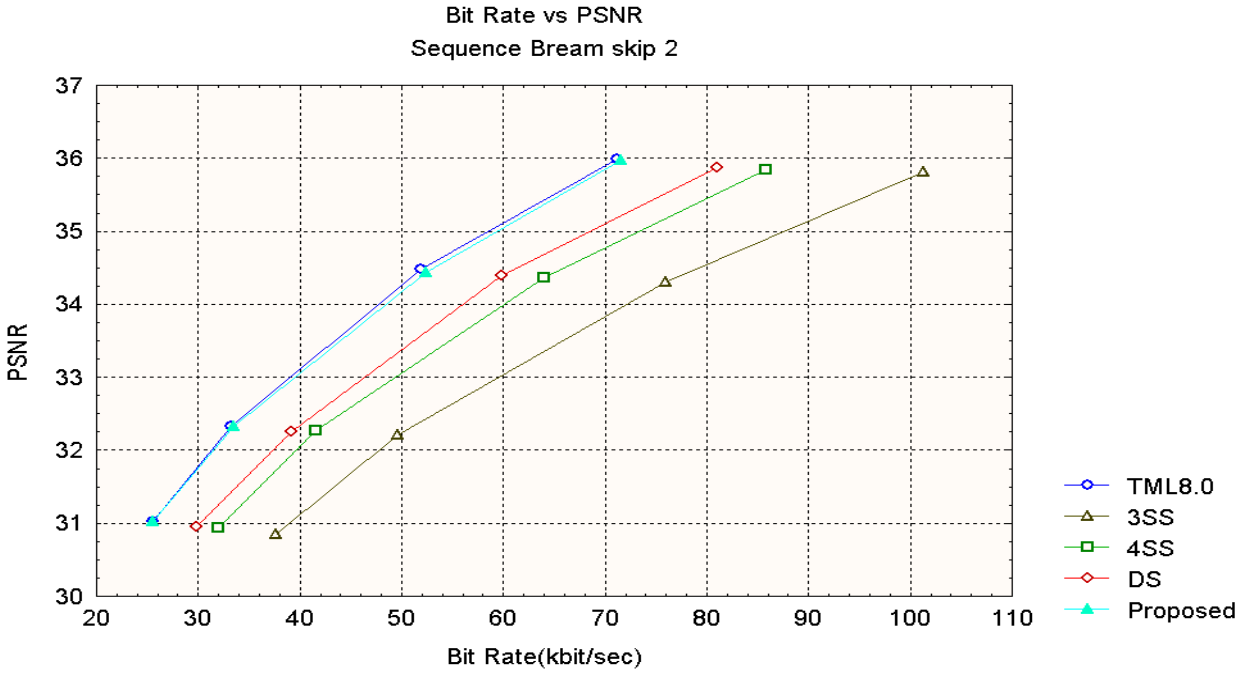


Figure 5.16 在中低位元率，各種快速演算法用在 Bream Sequence Rate-Distortion 的比較(2)

一般來說，失真性快速演算法會造成位元率昇高及畫面品質的降低，但由以上的實驗數據可知，本論文所提出快速多階連續消除移動預估演算法，比對全域搜尋法 (Full Search) 所要增加的位元率及畫面品質的降低極少，並且加速編碼器的速度，到達平均 2.5 倍，是比其它的快速演算法更適合應用在低位元率高畫質的視訊標準 H.26L 上，所以本論文所提出的方法，是兼具效能及速度的要求。

第六章 結論與未來展望

本論文以多階連續消除演算法為基礎，搜尋大區塊 16x16 區塊，並配合著本論文所特有的 Motion Refinement 及 Half Stop Decision 的方法，在損失極小壓縮效能及畫面品質的狀況下進行加速。

雖然，本論文所提出的方法，無法在實驗的數據上，達到即時壓縮的目標，但與其它的快速演算法比較，是值得用在視訊標準 H.26L 上。接下來，我們由 Table 6.1 來觀察哪些地方，還需要加速及改進。

Table 6.1利用FSEA在H.26L上所得到各Function的時間

Function	Time(ms)
Interpolation	50
Motion Search	70-100
Sum Norm	30
Integer Search	9-39ms
Half-Pel Search	16
Quarter-Pel Search	15
Intra prediction	10-11
Others	30

最重要的部份是要改進，Interpolation 和 Motion Search 這兩個部份，而這兩個部份可利用硬體的方式來加速，若你使用的系統是 DSP，則可使用平行處理的指令集，若你用的是個人電腦，則可選擇使用 MMX 的指令集，來進行加速，方可使 H.26L 真正運用於即時的壓縮上。

參考文獻

[1] ITU-T Recommendation H.261: Video Codec for Audiovisual Services at Px64 Kbits, ITU, 1993.

[2] Draft ITU-T Recommendation H.263: Video Coding for Low Bitrate Communication, ITU, May 1996.

[3] Draft ITU-T Recommendation H.263+: Video Coding for Low Bitrate Communication, ITU, July 1997.

[4] ISO/IEC JTC1, Generic Coding of Audiovisual Objects –Part 2: Visual(MPEG-4 Visual), ISO/IEC 14496-2, Version 1: January 1999; Version 2: January 2000; Version 3: January 2001

[5] ITU-T/SG 16/VCEG, Video Codec Test Model Long Term Number 8(TML-8). Doc, VCEG-N10, July. 2001.

[6] S. Wenger, M. Hannuksela, T. Stockhammer, “Identified H. 26L Applications, “ITU-T SG 16, Doc. VCEG-L34, Eibsee, Germany, Jan. 2001.

[7] Itoh, Y., Ngai-Man Cheung , “Universal variable length code for DCT coding” IEEE 2000 International Image Processing Conference , Volume: 1, pp. 940 -943, 2000

[8] G. J. Sullivan and T. Wiegand, ”Rate-Distortion Optimization for Video Compression,” in IEEE Signal Processing Magazine, vol. 15, no.6, pp.74-90, Nov.1998.

[9] Wiegand, T.; Girod, B., ”Lagrange multiplier selection in hybrid video coder control,” in IEEE Conference on Image Processing, vol.3, pp. 42 –545, 2001

[10] W. Li and E. Salari, “Successive elimination algorithm for motion estimation,” IEEE Trans on Image Processing, vol.4, no.1, pp.105-107, Jan. 1995.

[11] X.Q. Gao, C.J. Duanmu, and C. R. Zou, “A multilevel successive elimination algorithm for block matching motion estimation,” IEEE Trans. On Image Processing, vol.9, no.3, pp.501-504, Mar. 2000

[12] J. Kim and R. Park, “A fast feature-based block matching algorithm using integral projections,” IEEE Jour. Sel. Areas in Comm., vol.10, no.5, pp. 968-971, Jun. 1992.

[13] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding ," IEEE Trans Commun., vol. COM-299, no.12, pp.1799-1808, Dec. 1981.

[14] Reoxiang Li, Bing Zeng and Liou, M.L, " A new three-step search algorithm for block motion estimation" IEEE Transactions on Circuits and Systems for Video Technology, Vol. 4,pp.438-442, Aug. 1994

[15] Jianhua Lu, Liou, M.L," A simple and efficient search algorithm for block-matching motion estimation" IEEE Transactions on Circuits and Systems for Video Technology, Vol 7, pp.429 –433, April 1997

[16] Lai-Man Po, Wing-Chung Ma," A novel four-step search algorithm for fast block motion estimation" IEEE Transactions on Circuits and Systems for Video Technology, Vol 6, pp.313 –317, June 1996

[17] Shan Zhu、Kai-Kuang Ma,”A New Diamond Search Algorithm for The Fast Block Matching Motion Estimation” IEEE International Inference on Information Communications and Signal Processing,pp.9-12 Sep 1997

[18] H. K. Chow and M. L. Liou, “Genetic motion search algorithm for video compression ”, IEEE Trans. Circuits and System for Video Technology,vol. 3,no. 6,1993,pp.440-445

[19] Hyun Mun Kim; Acharya, T.,“ CAS: Context Adaptive Search for motion estimation“, International Conference on Information Technology: Coding and Computing, pp.202 -206 , 2001

[20] Lurng-Kuo Liu、Ephraim Feig,” A Block-Based Gradient Descent Search Algorithm for Block Motion Estimation in Video Coding” IEEE transactions on circuit and system for video technology,VOL. 6,NO.4，AUGUST 1996

[21] Lappalainen V., Hailapuro, A., Hamalainen, T.D., “Performance of H.26L video encoder on general-purpose processor” ,IEEE International Conference of Consumer Electronics(ICCE), pp.266 –267, 2001.

[22] Stephan Wenger, " H.26LComplexity Analysis according to VCEG-L36 section 2.1.4," document VCEG-M23, ITU-T Video Coding Experts Group (VCEG) Meeting, 2-4 April, 2001.

[23] Pankaj Topiwala, Gary Sullivan, Anthony Joch and Faouzi Kossentini, " Performance Evaluation of H.26L, TML 8 vs. H.263++ and MPEG-4," document VCEG-N18, ITU-T Video Coding Experts Group (VCEG) Meeting, 20 Sep, 2001.

[24] ITU-T Recommendation H.263 software implementation, Digital Video Coding Group at Telenor R&D, 1995