# 國 立 交 通 大 學

# 電 機 與 控 制 工 程 研 究 所

# 碩 士 論 文

## 以適應性門檻改進快速移動估測演算法與硬體架構設計

## An improved fast motion estimation algorithm based on adaptively thresholding and architecture design


研究生：簡芳彥

指導教授：董蘭榮 博士



中華民國九十四年六月

# 以適應性門檻改進快速移動估測演算法與硬體架構設計

研究生：簡芳彥　　指導教授：董蘭榮 博士

國立交通大學 電機與控制工程研究所

## 摘　要

在目前的影像壓縮規格中如 H.264 與 MPEG 4 等，移動估測的計算是佔總計算量比例最大的部分，所以許多為了減少計算量的快速演算法持續便被設計出來，而其中相對於使用全部搜尋區塊匹配演算法的快速搜尋演算法能減少大量的計算量，目前這一類的快速演算法中以鑽石搜尋演算法為基礎的區塊匹配演演算法能有較佳的效能，但是其中還是含有不必要的多餘計算量，所以我們設計一套提早終止的機制來節省更多的計算量，主要是藉由預測出可能的移動估測結果來避免無效的搜尋步驟，而預測出的數值即為提早終止機制的門檻值。我們從連續畫面中時間域上的關連性來預測出移動估測的結果，以決定出在每一張畫面的移動估測計算前都會更新的門檻值，這個門檻值能依照不同連續影像而改變數值以達到良好的適應性質，而能在不影響移動估測結果的情況下來節省計算量，這樣的提早終止技術可適用於各種快速區塊匹配演算法，而我們將之應用在鑽石搜尋法上設計出一套全新的快速移動估測演算法。為了希望能即時的完成壓縮編碼，所以我們也針對新演算法設計出硬體架構以實現移動估測，在設計上的目標為了減少外部記憶體的讀取量及功率消耗，所以此架構中利用平行處理的方式與設計特殊的記憶體位址產生器來實現此快速移動估測演算法。

# An improved fast motion estimation algorithm based on adaptively thresholding and architecture design

Student : Fang-Yen Chien Advisor : Dr. Lan-Rong Dung

Department of Electrical and Control Engineering
National Chiao Tung University

## Abstract

In present video compression standard, such as H.264 and MPEG4, the computation of motion estimation is most of total encoding computation. Thus, many fast algorithms for saving computation load have been proposed. The fast search algorithm saves a lot of computation load comparing full search block matching algorithm. In this type of fast algorithm, the block matching algorithm based on diamond search has better performance. But there is unnecessary computation in those fast block matching algorithms. We have designed an early termination technique to save more computation load. This technique tries to avoid unnecessary search step with predicting result of motion estimation. This predictive result is the threshold of early termination technique. We use the relation of temporal domain to predict result in sequence frames. The threshold is set before starting motion estimation of next frame. Thus, this threshold has good adaptability in those different video cases. Because this early termination technique can save computation load without destroying motion estimation, it should be applied to different fast search block matching algorithm. We proposed a novel diamond search algorithm based on early termination. For encoding in real time, we also designed architecture of this novel algorithm to implement. To cope with low off-chip memory utilization and low power consumption, the proposed architecture has parallel schedule and special memory address generator.

# 目錄

# 圖目錄

## 表目錄

# 第一章 緒論

## 1.1 研究動機

移動估測（motion estimation）在幾種影像壓縮的編碼標準中是不可或缺的重要部分，例如 MPEG-2 、MPEG-4 及 H.263，還有目前被廣為重視的 H.264 影像編碼標準。其中主要是利用區塊匹配演算法（block matching algorithm），得到連續影像中時間域的相關性，進而減少多餘的資訊量來達到高品質的影像壓縮結果。

區塊匹配移動估測演算法（block matching motion estimation algorithm）是先將未壓縮連續影像中的個別畫面（frame）分割成若干微小區塊(macro block)，再估計出兩張連續畫面區塊間的移動向量（motion vector），以此來求得影像中時間域的相關性。但是演算法運作過程中所需的計算量相當龐大，大約是佔所有壓縮編碼總計算量的 60%至 70%，所以能夠減少此演算法的計算量並求得準確的移動向量是非常值得深究的題目。

在眾多已發表的區塊匹配移動估測演算法中，鑽石搜尋演算法（diamond search algorithm）是目前經常被引用並被改進的一種快速演算法，但是所改進的方式大都無法根據不同的被壓縮影像內容做調整，所以我們提出了一個改善演算法的方式，除了可以針對不同性質的影像做適應性的調整，也可以套用在現今許多已發表的區塊匹配移動估測演算法上，主要是能除去其中不必要的步驟以節省移動估測所需的計算量，此論文中我們以基本的鑽石搜尋演算法為基礎來證實新的演算法可以有效的針對影像適應性地減少計算量。

此外，由於影像編碼過程中移動估測的計算量相當龐大，所以為了能在即時（real time）狀況下做影像壓縮就必須要用硬體來實現此一部份，因此我們也完成新演算法的硬體設計。在此硬體架構中，除了追求減少功率消耗及低硬體成本的目標外，也針對鑽石搜尋演算法在資料讀取上設計了一個特殊的快速記憶體位址產生器(fast memory address generator)，以減少讀取記憶體所需的時間。

## 1.2 相關研究

眾多影像壓縮規格中，MPEG4[1]及 H.264[2]是目前熱門的壓縮技術，其中移動估測（motion estimation）演算法也已發表許多相關的的研究，都是因為本來區塊匹配移動估測演算法（block matching motion estimation algorithm）中所使用的全部搜尋（full search）技術雖然可以估測到正確的動作，但是卻需要大量的計算量，所以近年來發展出了快速搜尋法，相對於全部搜尋法（full search）的有三步搜尋法（3-step search）[3]、新三步搜尋法（new 3-step search）[4]、四步搜尋法（four-step search）[5]及較新的鑽石搜尋法（diamond search）[6][7]，由以上幾種的演算法為基礎，許多演算法也被持續發表出來[8~13]。

在接下來的章節中，將詳細介紹幾種不同種類的快速演算法及其效能的比較，並證明我們所發展的新演算法能適用各種演算法而達到更節省計算量的目的，並以硬體設計來說明此新演算法對於硬體負擔上的增加也十分有限。

## 1.3 章節簡介

第一章 緒論

提出論文的主題與基本目的，並對相關的研究做簡介。

第二章 研究背景

介紹目前影像壓縮中移動估測的技術，並比較各式不同論文中的演算法及其設計理論的基礎相關研究。

第三章 演算法介紹及實驗結果

針對我們所提出的新演算法做一詳細的介紹，包括開發此演算法的歷程與不同適應性門檻值的決定方式，以設計出具有硬體實現上的考量所改進的鑽石搜尋演算法，並經由實驗結果證明其效能。

第四章 硬體實現及結果

將前一章的演算法利用硬體描述語言設計出硬體，並說明其硬體架構與所使用的設計規格，還有介紹為了此演算法硬體實現所額外設計的記憶體產生器。最後是此硬體設計的各項數據結果。

第五章 結論

對於此論文的主要貢獻做總結，並提出各項未來的展望與建議事項。

# 第二章　研究背景

目前較常見影像壓縮的規格有 MPEG1、2、4 及 H.263、H.264 等，而又以 H.264 及 MPEG4 為現今持續研究並改進的高畫質、高壓縮比的影像壓縮技術，本章首先介紹的即是 H.264 影像壓縮技術，並說明移動估測技術（motion estimation, ME）在其中的重要性。

## 2.1 影像壓縮技術

在 H.264 影像壓縮技術中，包含影像中單一畫面（inter frame）的壓縮技術之外，及利用連續影像畫面（intra frame）中時間域的多餘資訊（temporal redundancy）來達到壓縮的目的，圖 2.1 及圖 2.2 即為連續影像中的兩張連續畫面，欲壓縮後一張畫面（圖 2.2）時，我們不直接對整張畫面作壓縮而是找出與前一張畫面差異的地方，這樣可以減少大量重複的資訊量，而在編碼器（encoder）只要針對前一張和相對於後一張的相異處即可完成壓縮，經由各式傳輸技術送到解碼器（decoder）後，只要利用前一張畫面及相異處就可以重建出後一張畫面，這個編碼步驟就是對連續影像作壓縮的關鍵技術。

圖 2.1 參考畫面

圖 2.2 現時畫面

但是兩張畫面的差異處如果直接用兩張畫面作相減還是存在重複的多餘資訊量，如圖 2.3 所示，白球的位置只是往上移動，但是用相減的方式卻無法表現出來，所以移動估測（motion estimation）技術的目的就是要能估測出後一張畫

面相對於前一張畫面的所有相關移動位置，如此就能得到如圖 2.4 的畫面，我們稱之為剩餘畫面（residual frame），這張畫面上的資訊量越少，代表所做的移動估測減少越多時間域中重複的資訊量。

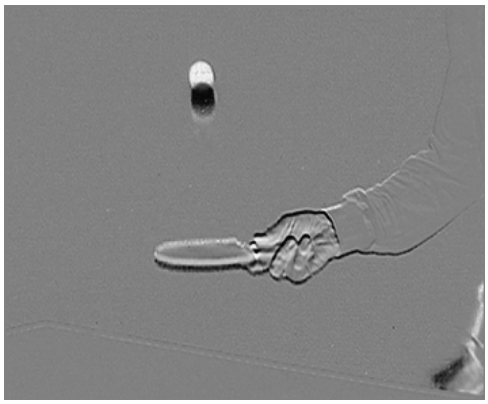


圖 2.3 差異畫面

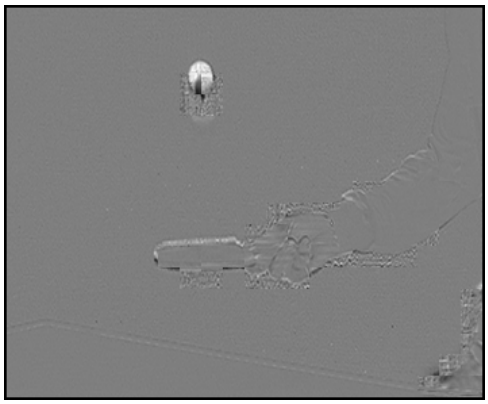


圖 2.4 剩餘畫面

圖 2.5 是 H.264 中編碼器（encoder）的示意圖，圖中 ME 區塊即是移動估測的部分，被壓縮影像中所有畫面都會經過此處，所以如何準確、快速的完成移動估測的動作便是編碼器設計中的重點。

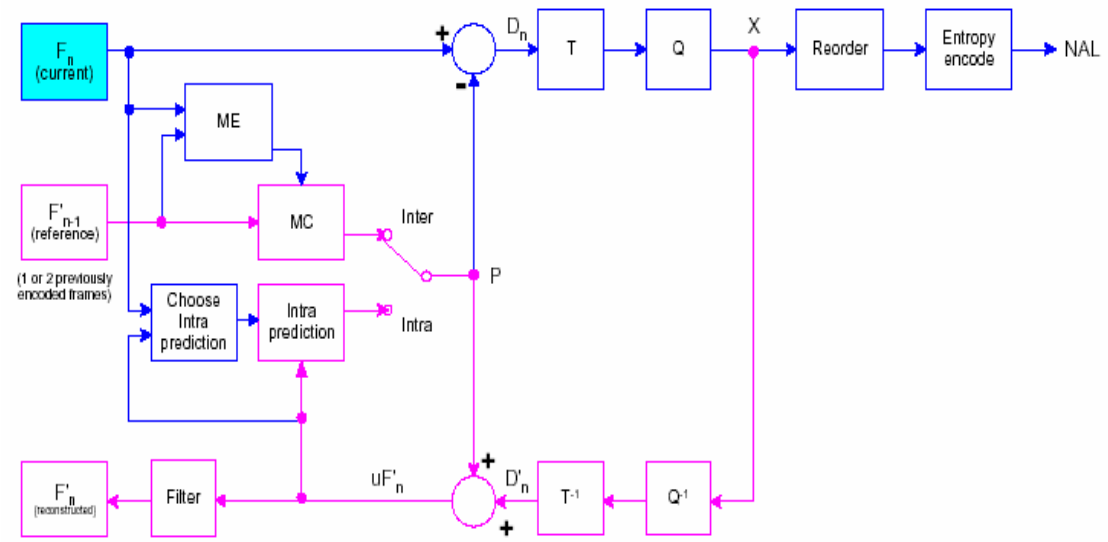



圖 2.5 H.264 編碼器區塊示意圖

## 2.2 移動估測技術

在移動估測技術中（motion estimation），目前一般廣為使用的演算法稱為區塊匹配移動估測演算法（block matching motion estimation algorithm），先將要處理的單一畫面（frame）分為若干大區塊（macro block）來作計算，在 H.264 的規格中區塊大小有從 4×4、4×8、8×4、8×8、8×16、16×8 到 16×16 畫素（pixels），而使用越小的區塊雖然可以得到較好的估測結果，但處理一張畫面也會花費較多的計算量。決定區塊大小 N×N 後，便在現時畫面（current frame）中以此區塊為基準，去判斷此區塊位於參考畫面（reference frame）中所在的位置，通常參考畫面就是現時畫面的前一張畫面，而區塊的相似程度是利用計算絕對差值總和（sum of absolute difference, SAD）來作判斷的，如下式：

$$SAD(v_x,v_y)=\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\left|I_t(x+m,y+n)-I_{t-1}(x+v_x+m,y+v_y+n)\right| \qquad (2\text{-}1)$$

這裡 $I_t$ 表示現時畫面、$I_{t-1}$ 表示參考畫面，其 SAD 值越小代表兩個區塊越相似，如果為零可以認定是完全相同的區塊。而相似區塊的位置是用移動向量(motion vector)來表示，見圖 2.6，在參考畫面中相對於原來現時畫面原區塊所在的位置即是所估測的移動向量，而一般作移動估測時，會限定在一個搜尋窗口（search window）或稱搜尋區域（search area）的範圍中去估測，這是假設在此區域中即能找到所要的移動向量而避免搜尋參考畫面中每一個位置。

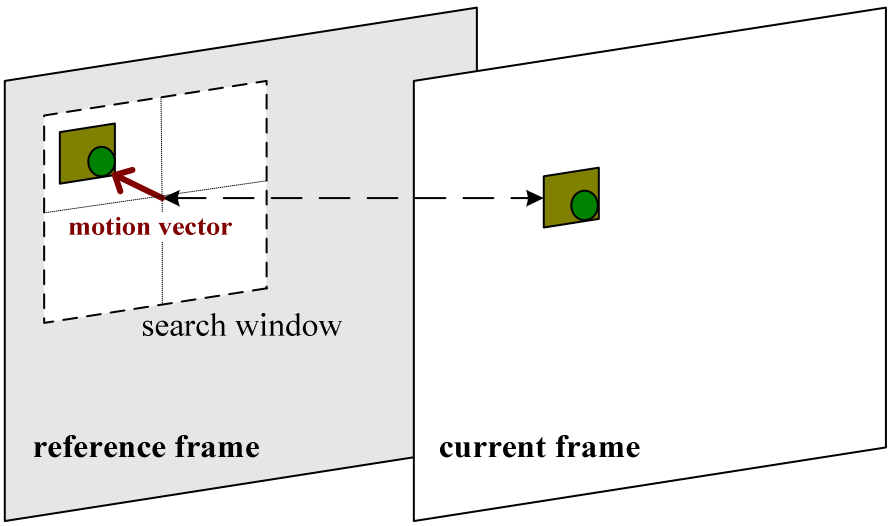



圖 2.6　移動估測示意圖

而最基本的估測方式採用全部搜尋（full search）的方式計算，全部搜尋移動估測（full search ME）就是在參考畫面的搜尋窗口中每一個位置都跟現時畫面中的區塊作比較，見圖 2.7，一個±2 的搜尋窗口共有 25 個位置要去比較。

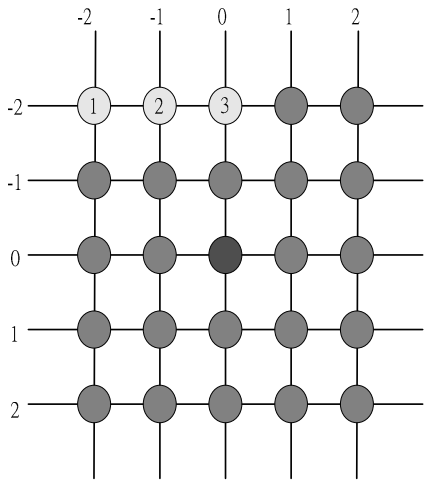



圖 2.7 全部搜尋法

所以搜尋窗口（search window）的大小與區塊（block）的大小決定演算法所需的計算量，而全部搜尋雖然可以找到搜尋窗口中 SAD 值最小的位置，但是在較大的搜尋窗口中會耗費過多計算量，因此許多快速演算法便為了節省計算量的目的而被設計出來，接下來便是要介紹各式不同的快速演算法。

## 2.3 快速區塊匹配移動估測演算法

此節所介紹的快速演算法都是相對於區塊匹配移動估測演算法中所使用的全部搜尋（full search）技術，主要的目的都是為了能使用較少的計算量來估測出移動向量，所付出的代價就是可能會得到較差的移動估測結果。

其中具代表性的可以分三類作法，第一類是利用取樣（sub-sample）的方式或低通濾波器（low-pass filter）減少所要處理的畫面資訊已達到減少計算量的方式，例如階級式移動估測（hierarchical ME）[14]。第二類是利用少數特定位置來取代全部位置的搜尋方式，包括有三步搜尋法（3-step search）[3]、新三步搜尋法（new 3-step search）[4]、四步搜尋法（4-step search）[5]及鑽石搜尋法（diamond search）[6][7]等，第三類是預先判斷移動向量或提早結束移動估測以節省計算量，例如連續消除演算法（successive elimination algorithm）[15]及預知移動向量搜尋技術（predictive motion vector search technique）[18] 。

第一類的作法在作移動估測之前先減少畫面的資訊量，所以可以減少在區塊匹配移動估測演算法所花費的計算量，並非直接改變搜尋的方式，而第二類及第三類的作法是接下來要詳細介紹的部分，此類作法便是直接去改變全部搜尋的方式而得到減少計算量的加速效果，而我們所發展出來的新演算法也可以歸類在此類作法之中。

### 2.3.1 三步搜尋法

三步搜尋法（three-step search）是一種簡單而且能有效減少移動估測中所需計算量的演算法，見圖 2.8，一開始做區塊匹配的計算時，與全部搜尋不同的是只先從圖中九個位置開始，再利用九個位置所算出的 SAD 值決定接下來第二步的位置，以圖中為例，當發現第一步中（4, 4）位置所得的是最小的 SAD 值時，第二部便從此位置四周再做九個位置的區塊匹配，但是此九個點間的距離只有第一步所的一半，做完第二步後再以相同的方式完成第三步的搜尋，即可得到最後的搜尋結果，這也是三步搜尋法名稱的由來。從圖例中，可以發現每完成一次區塊的移動搜尋，在搜尋窗口為±7 的大小中只需要 25 個位置的計算，相較於全部搜尋需要 255 個位置的計算只需 1/9 的計算量，所以此三步搜尋法到目前還是加

快區塊匹配移動估測演算法常見的快速演算法之一。

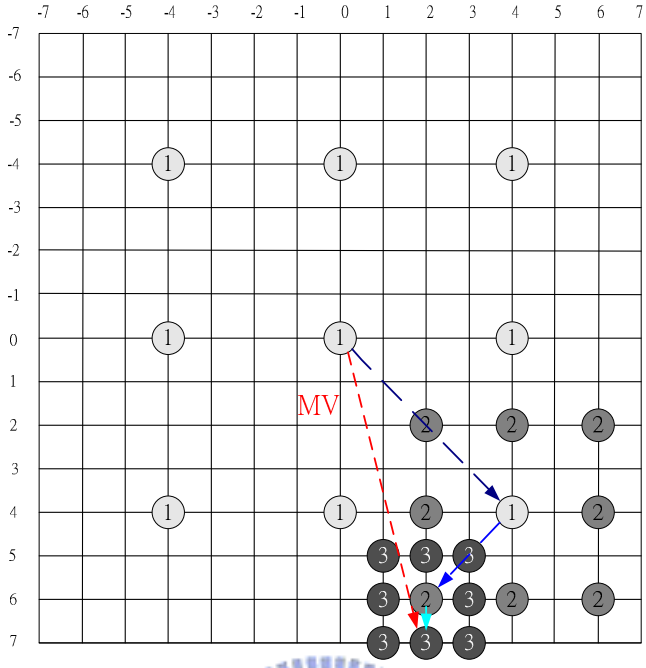



圖 2.8 三步搜尋法

但是三步搜尋法由於第一步就決定接下來搜尋的位置，所以有時會因為些微 SAD 值的差異或是一開始位置的差異過大，而導致接下來第二、第三步所做的區塊匹配沒有機會找到較好的移動向量，導致移動估測的效果不佳，所以為了改善此一缺點，就改進演算法而有了新三步演算法（new three-step search）。

### 2.3.2 新三步搜尋法

新三步搜尋法（new three-step search）的演算法是改進原來三步演算法在小幅度移動的缺點，所以在原來九個位置的搜尋點外再加上中心（0,0）位置四周八個點，如圖 2.9，在第一步的 17 個區塊匹配中，如果最小的 SAD 值位於外圍的八個位置，那接下來的第二步、第三步與原來的三步搜尋法作法一樣，但是要是最小的 SAD 值位於中心周圍八個位置，那下一步就只需作一個緊鄰其位置四周的區塊匹配，所以此演算法不但較適用於移動量較小的連續影像，在總計算量上也比三步搜尋要小。

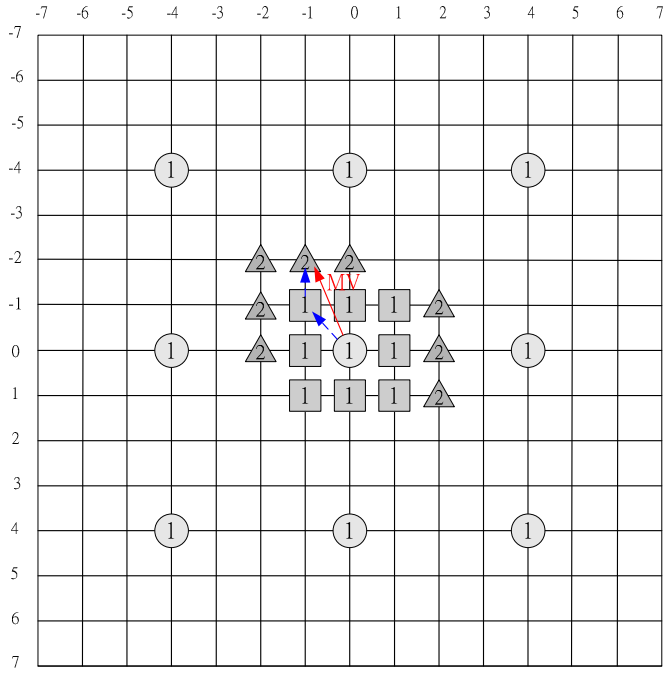



圖 2.9 新三步搜尋法

但在之後的研究中發現無論是三步搜尋法或新三步搜尋法都有還是含有多餘的計算量，原因就在於在第一步搜尋時，有一些位置的區塊匹配是不必要的，所以後來為了更節省計算量，針對連續影像移動估測結果的統計數據而設計出了新的演算法。

### 2.3.3 四步搜尋法

根據許多連續影像的移動估測的統計結果，可以得知大部分的移動向量都是位在於中心點及緊鄰中心點的位置，從相關研究論文[9]的統計數據中可以畫出圖2.10，可以很清楚的發現在±7的範圍中可以得到移動向量的機率都是落在中心處，遠高於在其它較遠的周圍機率，而此類連續影像共有的特質被稱為偏中心的移動向量分佈（center-biased motion vector distribution），以此為出發點所設計的演算法是近年來常見的改進方式，這裡先介紹的是四步搜尋法（four-step search）[5]。

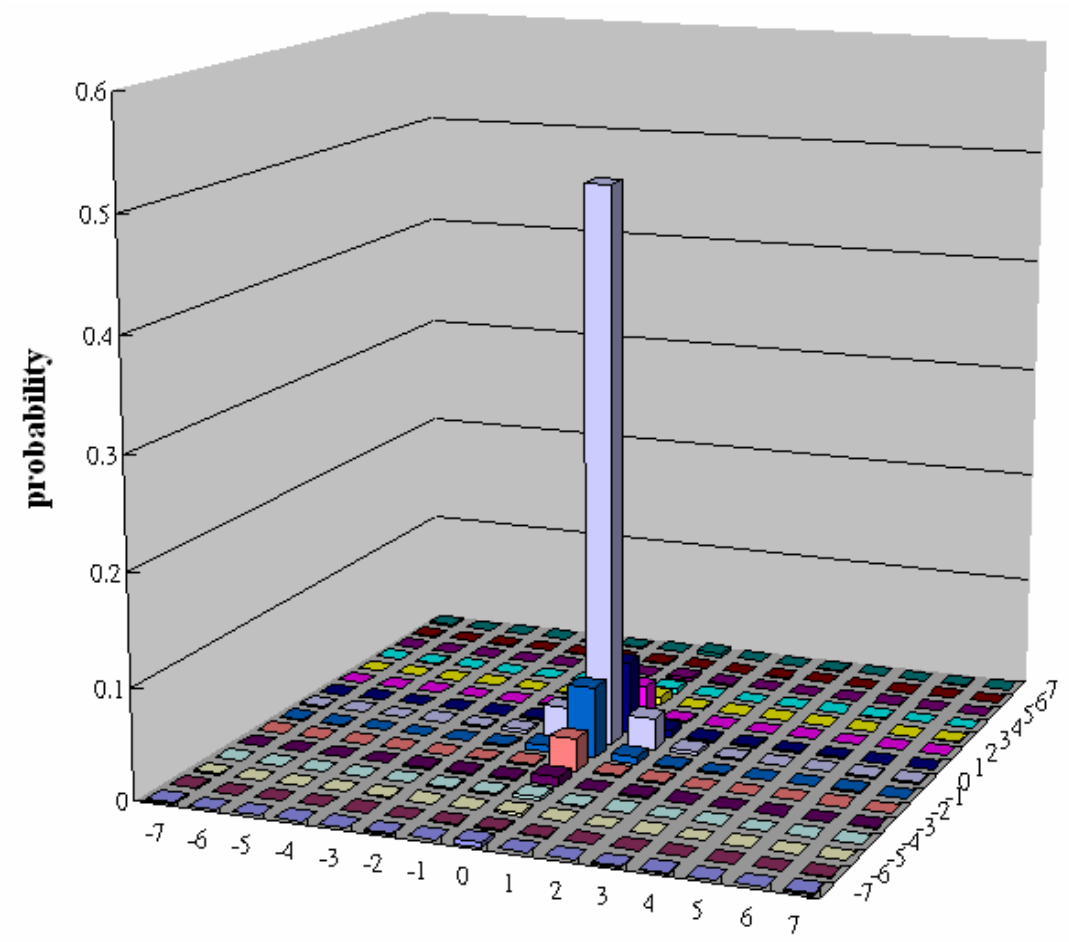



圖 2.10 移動向量統計結果

參考圖 2.11，由於是考量到偏中心的移動向量分佈，所以四步搜尋法的第一步的區塊匹配位置比三步搜尋法還靠近中心點，當比較出最小的 SAD 值的位置後，接下來的第二、第三步便是以同樣的格式移動去增加新搜尋的位置，到了第四步便結束估測，而且此步的區塊匹配改為計算最小的 SAD 值的緊鄰八個位置，此演算法在搜尋窗口為±7 的情況中最多只需要作 27 個位置的計算，最少可以只作 17 個位置的計算，而從使用四步搜尋演算法所做的實驗結果很明確的證實在大多數情況下，這種作法不但可以比三步搜尋法節省更多計算量，還能估測出較精準的移動向量，所以證明以偏中心的移動向量分佈特質來設計的區塊匹配演算法是具有明顯優勢的快速演算法，從此便啟發了之後鑽石演算法及其它類似演算法的快速發展。

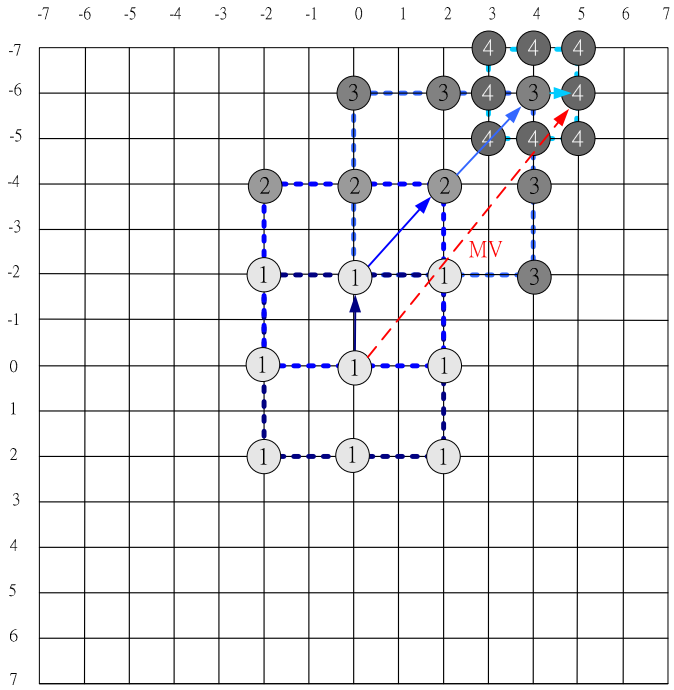



圖 2.11 四步搜尋法

### 2.3.4 鑽石搜尋法

鑽石搜尋法（diamond search）同樣是基於偏中心的移動向量分佈特質而設計的一種快速演算法，參考之前移動估測的統計資料可以發現在垂直與水平軸上移動向量的分佈機率較高，所以簡化了方形而使用了菱形的位置圖樣做區塊匹配的計算，其名稱便是來自於此類似鑽石狀的圖樣。此演算法執行的步驟如下：

步驟一：針對中心點（0,0）與四周鑽石圖樣的八個位置作區塊匹配計算，比較出其中最小的 SAD 值，若是位於中心點便跳至步驟三，位於其它八個位置則執行步驟二。

步驟二：把原來 SAD 最小值的位置當作新鑽石圖樣的中心點，並對相同周圍八個位置作區塊匹配計算，並比較出其中最小的 SAD 值，若是位於中心點便跳至步驟三，位於其它八個位置則重複執行步驟二。

步驟三：當中心點即是 SAD 值最小的情況下，便搜尋上下左右緊鄰的四個位置是否有更小的 SAD 值，連中心點總共五個位置其中 SAD 值最小的便是估測出的移動向量，結束此區塊搜尋。

見圖 2.12 之例，當第一步到第三步 SAD 值最小的都不是在中心點，直到執行第四步時發現多增加的三個位置中無法得到更小的 SAD 值才進入第五步。

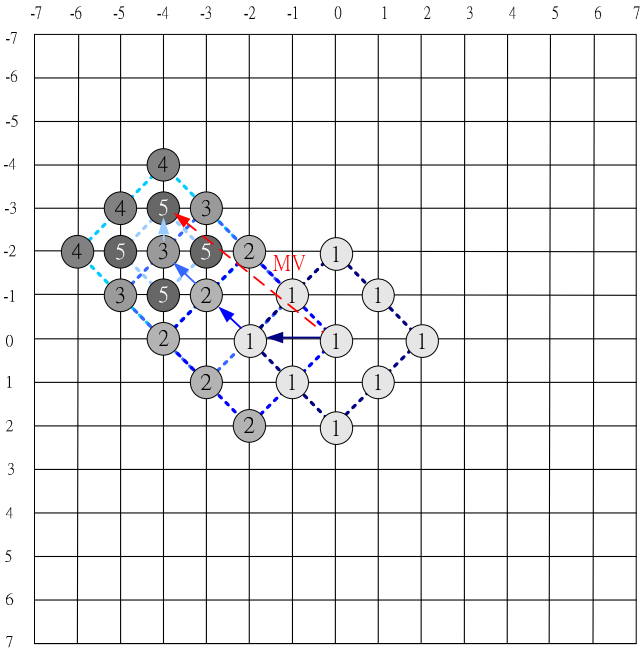



圖 2.12　鑽石搜尋法

此演算法的特色就是沒有固定的執行步驟數目，也就是當沒有滿足進入步驟三的條件下會不斷重複步驟二搜尋動作，除非因超出搜尋窗口的限制而無法在執行下去而止，這樣的演算法雖然有可能會花費許多計算量在得到最小的 SAD 值，但是相較於固定區塊匹配執行步驟的作法更有機會得到較準確的移動向量，而且單純考慮所需要的最少計算量，鑽石演算法只需要計算 13 個位置，比四步演算法的 17 個位置還要精簡，而且在大多數影像都有的偏中心的移動向量分佈特質下，鑽石演算法的總計算量是可以有效減少的，此結論也被移動估測模擬的實驗數據所證明，在移動估測結果與加速程度上都能有所提升。

由於鑽石演算法的出現，許多相關的改進演算法也被設計出來，大多是改變搜尋位置的樣式而希望得到更好的移動估測結果或是更少的計算量，例如菱形可以針對垂直或水平拉長的樣式，或是在第一步先用十字形狀的樣式開始，還有六角型樣式的變化等等[8-13]。但大致上觀察實驗數據能發現改善的程度有限，而且無法在所有種類的實驗影像中都得到最好的結果，所以目前還無法判斷哪一種改進演算法是最適合作區塊匹配，但是到目前所發展的這幾種快速演算法都有一個缺點，就是在連續影像中出現快速、幅度大的動作時，移動向量的估測就相較於全部搜尋法要增加許多誤差，在搜尋窗口越大的情況此現象也較為嚴重，不過當搜尋窗口定的越大，相較於全部搜尋的方式可以省下更多計算量，這也是鑽石搜尋法中比固定搜尋步驟有彈性的地方。

## 2.4 其它移動估測加速技術

除了減少區塊匹配所需要計算的位置也就是搜尋點數（search points）的方式外，還有許多減少移動估測總計算量的演算法，都是在盡可能不損失移動向量的準確度的前提下所設計的，大約可分成兩類作法，一類是預知移動向量搜尋技術（predictive motion vector），此類技術是利用已知的資訊以預先估計可能的移動向量位置，這樣就能盡快得到想估測的移動向量。另一類便是提早終止計算技術（early termination），就是提早放棄沒有必要的區塊匹配計算以節省計算量的方式，主要是預先設立控制條件，當發生符合條件的情況下就終止現在的計算而執行下一步驟。我們所提出的演算法也可以歸類為這一種的快速演算法。

### 2.4.1 預知移動向量搜尋技術

預知移動向量（predictive motion vector）的技術主要是利用連續影像畫面中空間域（spatial）及時間域（temporal）上的相關性預估可能的移動向量，雖然不能完全預估出正確的位置，但是只要能找出可能性較高的區域再利用上述的快速區塊匹配演算法作計算，就可以成功達到節省計算量的目的。

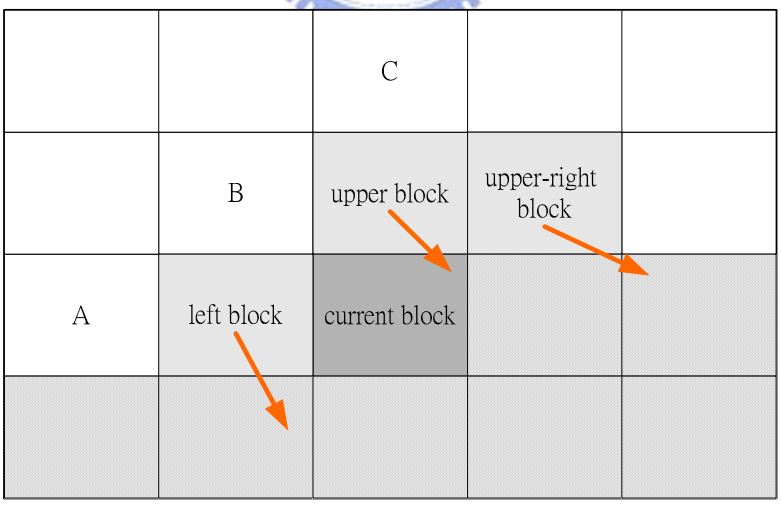



圖 2.13 預測區塊示意圖

空間域預知移動向量技術利用在畫面上緊鄰的大區塊（macro blocks, MB）間的相關性作預測移動向量的依據，參考圖 2.13，現時區塊（current block）是正要進行區塊匹配前的深色方塊，假設四周的區塊會一起往相同的方向移動，為

了得到現時區塊的預知移動向量，所以利用緊鄰的三個淺色區塊也就是左方、上方與右上方的區塊中已計算出的移動向量作預測的依據，在不同研究中也有多加上圖中 A、B、C 三個區塊的移動向量[19]，要注意的是區塊是從左上方往右下方作移動估測，所以斜線區塊中都還未得到移動向量，所以不能用來當預測依據。而預測的方式一種是已知的移動向量的 X、Y 軸分量作平均而得預知移動，另一種方式是將已知的三個向量都當作可能的向量而視為此區塊的預知向量，以這些向量再做區塊匹配便能判斷現時區塊真正的移動向量，當然兩種方式也能一起使用[23]，而實驗結果也證實比單一使用要好，但是若是過多預知向量對於節省計算量並沒有好處，所以需要先作區塊匹配再利用 SAD 值判斷最可能的位置，再從此位置為中心點利用之前的快速搜尋演算法就能比從（0,0）更早得到準確的移動向量。

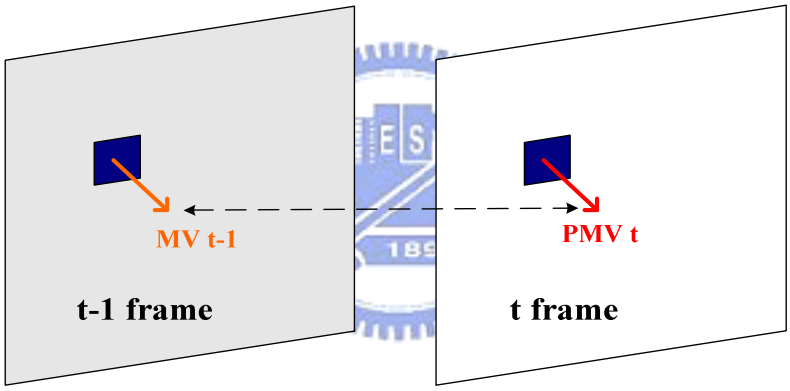



圖 2.14 時間域預測示意圖一

時間域預知移動向量技術是從前後連續畫面的同一個位置的區塊作預測的依據，見圖 2.14，考量前一張畫面的同一個位置可能有同樣的移動向量，所以直接將前一張畫面中所得到的移動向量作為現在目標區塊的預知移動向量，這是比較簡單的一種方式，也有從前一張畫面中數個鄰近的區塊作預測依據的作法。

還有一種方式是不止考慮時間上的相關性，還採用了移動速度上的觀念來預測移動向量[20]，見圖 2.15，利用前兩張連續畫面中相同位置區塊中已知的移動向量作為依據，利用時間與距離的對應關係來作現時畫面的移動向量預測，只是根據實驗結果來看，兩種方式並沒有絕對的優劣差異，會依據連續影像中的動作類型而有不同的結果。

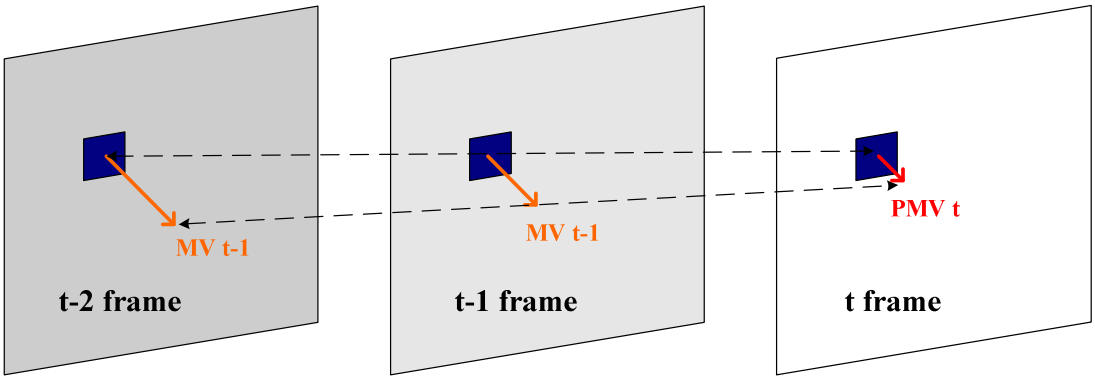



圖 2.15　時間域預測示意圖二

上述所提到的空間域及時間域的移動向量預測方式是可以合併使用的，雖然預知移動向量的數目會增加，但提高了預測正確的可能性，當然所得到的預知移動向量都只能當作參考位置，對於鄰近的位置都還要再做區塊匹配來驗證是否為更準確的移動向量，這樣的作法才能的得到較佳的移動估測結果。

### 2.4.2 提早終止計算技術

提早終止計算技術這一類快速演算法對移動估測的結果影響很小，因為主要節省計算量的方法是避免不必要的計算步驟，而非改變區塊匹配的搜尋方式，所以即使加速效能比不上許多快速演算法，但在實用性及適應性上都較好。接下來介紹三種這一類的技術：

**連續消除演算法（Successive Elimination Algorithm）[15]**

在 2.2 節中所介紹過的 SAD 值計算是區塊匹配中主要的計算步驟，此演算法便是節省此計算過程的計算量，利用式(2-2)與式(2-1)的關係簡化判斷的過程，

$$\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} I_t(x+m,y+n) - \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} I_{t-1}(x+v_x+m,y+v_y+n) \le SAD \qquad (2\text{-}2)$$

用意在區塊匹配前先計算出個別的總和以判斷是否有機會得到小於目前所得的 SAD 值，若總和相減大於之前區塊匹配得到 SAD 值就略過此位置而不作區塊匹

配，這樣就能減少在 SAD 值上的計算量。此演算法還發展出多層次連續消除演算法（multilevel successive elimination algorithm）[16]與其它類似的快速演算法[17]，其中多層次連續消除演算法是利用類似的機制但是將區塊分成不同層次來作判斷，都是為了增加判斷的準確度及節省更多的計算量，而在軟體上的實驗結果也得到證明，因為 SAD 值計算中的絕對值相減式是相當耗費計算量的，所以用其它算式去節省 SAD 值的計算可以得到明顯的加速效果。

**節省判斷時間技術（optimization of decision-timing）[21]**

與連續消除演算法相似的是節省判斷時間技術也是為了改進計算 SAD 值的方式節省計算量，但此演算法是利用機率的觀念去排除 SAD 值可能會過大的區塊匹配計算，也就是說當在計算 SAD 值時，不必全部計算完成便可以判斷此位置的區塊匹配是否能得到準確的移動向量，利用一部份的 SAD 值的結果去判斷全部 SAD 值有多少的機率不會是最小的 SAD 值，若是機率太小就提早放棄計算，這樣一來便能節省下區塊匹配演算法的計算時間。

**運算量調節系統（computation-aware scheme）[22]**

此運算量調節系統是參考影像壓縮規格中的資料傳輸量控制（rate control）的方式去調節計算量，由使用者先訂出總運算量後在此條件下盡量做到最好的移動估測，大致的步驟是利用所限制的總運算量訂出一個指標，在作移動估測的計算時，便利用這個指標來限制所做的步驟多寡，為了要得到較好的結果，所以一開始就要同時考量數個區塊而不是一個區塊接一個區塊的執行方式，當然同時考量越多區塊效果越好，這樣就能先把計算量用在 SAD 值比較大的那些區塊去作其它位置的區塊匹配，而 SAD 值一開始就比較小的可以不必耗費計算量去多做計算，這樣就能在保持估測結果的情況下節省計算量，只是此系統需要預先作一些運算來決定指標跟判斷程序，所以並不是單純為了節省計算量而設計的快速演算法。

本章中幾類快速演算法雖然加速的程度不一，但是因為節省計算量的方式互相獨立，只要能結合這幾種演算法就能得到相當理想的加速效能，但是在硬體設計中，在 2.4.2 節中的三種演算法雖然都能有效的節省計算量，但對於軟體執行

上的節省計算量的效能比較明顯，而硬體角度上來看節省的量就比較有限，因為其中許多判斷跟運算式都比較適合在軟體上執行，尤其當中某些運算的硬體實現較不容易，可能因此而增加硬體負擔或是功率消耗，這樣一來就降低了節省計算量的優勢，所以我們先提出一個關於提早終止計算技術的演算法，並在考量硬體設計的角度下使其成功的改進鑽石搜尋演算法，這些設計原理在接下來的章節中會有詳盡的介紹。

# 第三章 演算法介紹及實驗結果

此章要介紹所開發的新演算法應用在鑽石搜尋區塊匹配演算法（diamond search block matching algorithm）的設計構想及實驗結果，主要目的是節省在鑽石搜尋法中不必要的計算部分，類似的構想在相關論文[24]中有提出過，但是其中節省計算量的效果與針對不同連續影像的適應性都不是很好，所以我們利用連續影像中時間域的相關性及簡單的門檻（threshold）判斷方式去改進上述這兩點，根據軟體模擬實驗所得到的結果也證明此演算法的確能達到我們所訂出的目的。此章節也會詳細說明設計此演算法的過程中所利用的幾種不同方式及最終選擇的理由。

## 3.1 鑽石搜尋法的缺點及改進的方式

在上一章 2.3.4 節中所介紹的鑽石搜尋法是目前所發展的快速區塊匹配演算法中常被使用的一類，包括許多類似的改進及變化都是以鑽石搜尋法為基礎而設計出來的，雖然改進的方式有很多，但大多是針對搜尋的位置作改變，基本精神是相同的，所以這裡就以原始的鑽石搜尋法來作設計基礎。基本上鑽石搜尋法及上一章介紹的許多快速區塊匹配演算法都有一個相同的缺點，就是搜尋過程中包含有多餘的計算量，雖然鑽石搜尋法相較之下已經節省了大量多餘的計算量，但是我們還是可以發現在某些狀況下會有不必要的區塊匹配計算。

參考圖 3.1 舉例來說，在搜尋中心從（0,0）移到（-4,-2）時，雖然（-4,-2）就是鑽石搜尋中 SAD 值最小的結果，但是還是搜尋步驟還是需要做到第五步，也就是第四步先確定（-4,-2）此中心點還是 SAD 值最小的位置，然後在作第五步去確定（-4,-2）的周圍沒有 SAD 值更小的點，而實際上搜尋到第三步的時候便已得到（-4,-2）的位置了，所以從結果來看，圖中斜線位置的區塊匹配計算都是多餘的。途中第二個例子也是相同的，當搜尋到第二步的（4,0）位置時還要在多做第三步和第四步來確認（4,0）是不是 SAD 值最小的位置，所以在斜線位置所做的區塊匹配計算對動作估測結果來說並沒有增加準確度，而兩個例子中的斜線位置共有 16 個，若兩個例子就是兩個大區塊（macro-block）的移動估測過程，其計算位置總共有 47 個，所以相較於全部 47 個的計算量就有 16 個多餘的

計算，要是節省下這些計算量的話相當於能加速 1.5 倍。

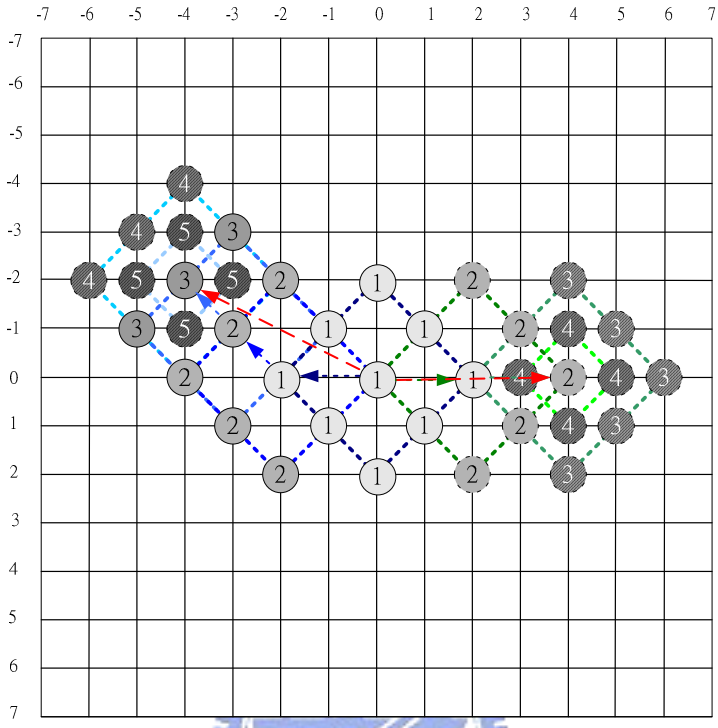



圖 3.1　搜尋過程範例示意圖一

再舉一個比較極端的例子來說明，參考圖 3.2，當鑽石搜尋法沒辦法找到這一步的中心點就是 SAD 值最小的時候，也就是最小的 SAD 值一直是落在周圍菱形上的位置時，這樣就會不會停止下來而一直到搜尋窗口的邊界才會停下來，雖然因而找到 SAD 值最小的位置就是動作估測的目的，但是在 SAD 值並沒有明顯減少的情況下，這樣的搜尋步驟可以視為是沒有必要，就像圖 3.2 中的搜尋步驟中可能在第五步就已經可以得到一個相當小的 SAD 值，而後面所做的步驟相較此 SAD 值減少的量卻很小，但是卻還是必須一直計算下去，到了第二十二步可能得到的 SAD 值相較於第五步的結果減少不到 1%，這對於移動估測的結果影響是相當有限的，但是搜尋步驟總計卻多付出三倍以上的計算量，當然這種情況發生的機會很小，不過要是能有控制機制來避免這種情況，就能再更節省鑽石搜尋法的計算量。

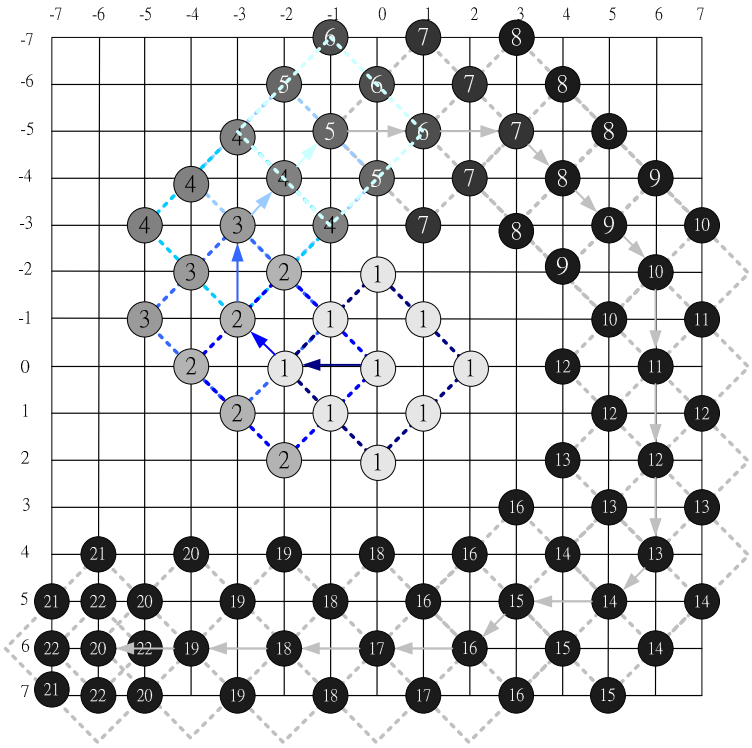


圖 3.2　搜尋過程範例示意圖二

綜合以上所有的例子，可以知道所要設計的控制機制就是一個提早終止的技術（early termination），作法便是訂出一個門檻（threshold）來決定目前這個位置得到的 SAD 值是否是夠小的而可以提早結束鑽石搜尋法的計算，所以這個技術的重點便是放在如何決定此門檻的數值，而其中困難的地方是因為其值定的過大會影響動作估測的準確度，而定的過小則對於節省計算量沒有幫助，但只要能定出符合所需要的數值大小，就能有效的節省計算量而不會影響到最後動作估測的結果，此門檻的決定方式便是我們這項研究中的重點。

## 3.2 適應性門檻技術

上一節中已經說明對 SAD 值作判斷的門檻（threshold）大小是提早終止鑽石搜尋步驟的關鍵指標，但是還有一個重點就是此門檻對不同影像的適應性，尤其是在不同影像中在移動量、複雜性上都有很大的差異，所以可以想見動作估測時所計算出的 SAD 值會有很大的差異。而且即使在同一段連續影像中，因為畫面的改變與物件的移動也不是事先可以預期的，所以能夠隨時因應畫面的變化情形而改變此門檻就是我們希望達成的目標。為了利用現有的資訊去達到此目標，這就跟 2.4.1 節中所介紹的預知移動向量搜尋技術是類似的預測動作，也就是利用之前區塊匹配時所計算出的 SAD 值來預測現在要做的區塊匹配可能會有的 SAD 值，利用這樣的觀念所發展出的方式我們稱之為適應性門檻技術（adaptive threshold），我們在研究過程中發展了五種方式來比較，加上相關研究中的一種方式，接下來就介紹這六種方式以及之間的比較。

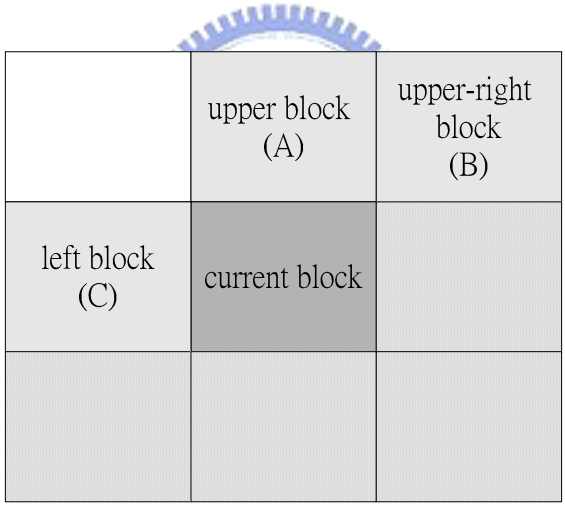



圖 3.3　現時區塊匹配示意圖

首先在參考資料的快速演算法研究[24]中提供了一種訂立門檻的方式，主要精神是利用畫面中空間域的相關性來做判斷 SAD 值是否夠小的依據，參考圖 3.3 所示，在計算到現時區塊（current block）時，利用四周相鄰的區塊匹配結果來估測現時區塊匹配可能會得到的 SAD 值，這個值便訂為提早終止計算的門檻，估測的方式就是選擇其中最小的 SAD 值，可以表示為式(3-1)。

$$Threshold = \min(SAD_A, SAD_B, SAD_C) \qquad (3\text{-}1)$$

因為周遭區塊中，斜線部分是尚未動作估測的部分，所以只能採用圖 3.3 中 A、B、C 這三個區塊所得的 SAD 值來作預測。因為這個方式是建立在空間域的相關性上，通常在區塊較小時例如 4×4 的情況相關性是很大的，但是在區塊大小為 16×16 的情況下 SAD 值的差異會相差較大，而且取三個之中最小值的作法雖然可以減少對動作估測結果的影響，但是卻可能會使計算的節省有限，還有一個缺點就是會因為現時區塊落在邊界時，能參考的周遭區塊數量就不足三個，這也會降低預測的準確度。雖然在論文中有提到可以再乘上係數或加上常數來增加節省計算量的效果，但是並沒有提出一個很完整的解決方案，而且此方式從硬體設計上考量還有不利的地方，就是需要在計算到每一個區塊前都要比較之前的數據，這樣會造成額外的硬體負擔，也會造成額外的時間與功率上的消耗，因此我們希望能有一個更好的定值方式。所以我們分析搜尋步驟的區塊匹配過程以作為設計考量，並嘗試用其它作法來訂出適應性的門檻。

### 3.2.1 適應性門檻分析

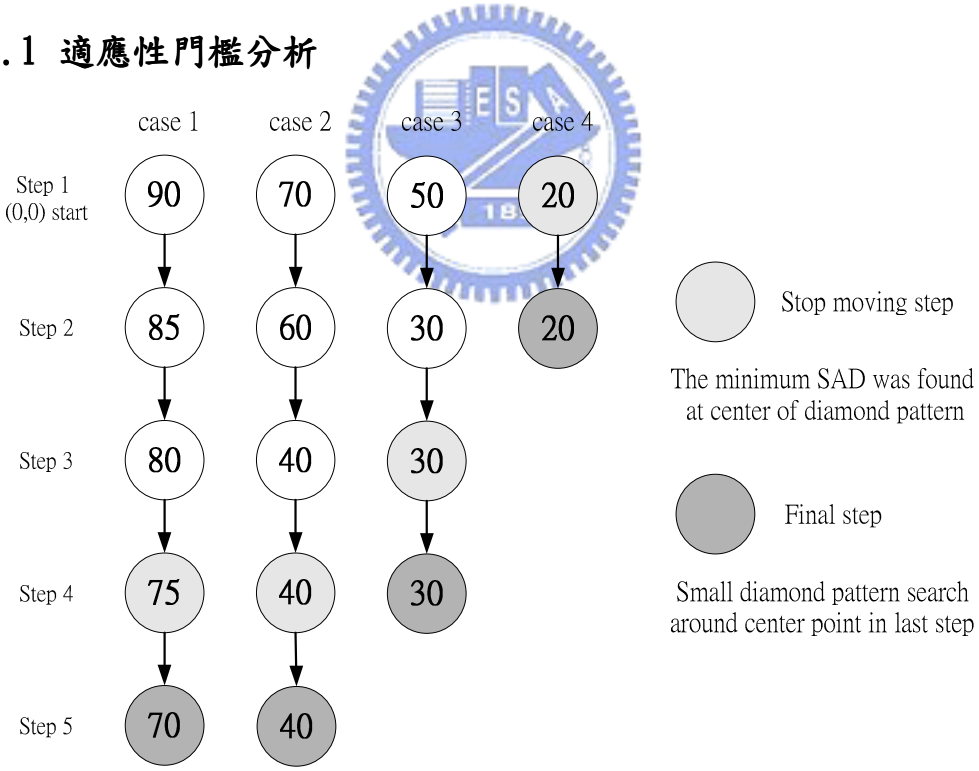



圖 3.4　區塊匹配範例示意圖一

首先我們先要探討的是在動作估測過程中可能發生的狀況，如同 3.1 節中所說明的在鑽石搜尋法中含有的缺失，這裡我們用區塊匹配演算法所算出的 SAD 值作例子，參考圖 3.4 中的四個例子，第一個例子表示一個完整的鑽石搜尋步驟

在第五步完成計算，並找到最小的 SAD 值。而第二到第四個例子就有發生多餘計算的步驟，第二個例子中，很明顯的第四步與第五步是不必要的計算步驟，同樣的情況在第三個例子中，第三、第四步也可以省略，而第四個例子中因為第一步搜尋步驟就已經找出最小的 SAD 值落在（0,0）的位置上，所以第二步的搜尋是可以不必計算的

從圖例中的數值看起來，如果門檻定的太大的話，如 T＝100 的情況就會使這四個例子都只作第一步就停止了，完全沒有動作估測的動作，但是決定的數值過小的話，如 T＝10 的情況，就沒有利用提早終止的方式到去節省計算量。考慮到這四個例子，我們很明顯的可以推測出如果要節省不必要的計算量，將提早終止的門檻定為 40 的話可以得到最好的結果，雖然要從已知的資訊中得到這樣的數值是不容易的，但參考預測移動向量的方式中，我們知道在連續影像中時間域的相關性是一個很好的參考指標。

在連續影像中，考慮前後張畫面在時間域的相關性是很明顯的，所以我們可以利用前一張畫面所做的區塊匹配結果，參考其 SAD 值以決定現在要做的區塊可能的 SAD 值，但因為 SAD 值會因為畫面動作的改變及物件的複雜度而有相當的差異，所以光是參考前一張畫面中相同位置區塊的 SAD 值並無法得到好的指標，尤其是當此區塊已經移動到其它位置的時候，再考慮到原先門檻值的用意是為了節省計算量，所以我們決定從前面一張畫面中所有區塊計算出的的 SAD 值中得到一個數值來作為現時畫面中所有的區塊所共用的門檻值，這樣就不需針對個別現時區塊去決定門檻的大小，如此不論是硬體或軟體上的實現都更能加快計算速度。

參考圖 3.4 的例子，我們假設前一張畫面中得到的 SAD 值也是 70、40、30、20 左右，所以只要能根據之前的已知數據預先將門檻值定為 40，就可以得到圖 3.5 的結果，因為算出 SAD 值小於 40 時便不再計算下去，所以節省了五個位置的區塊匹配計算，雖然這樣的例子是十分理想化的情況，但是從接下來的五種決定門檻的方式都是利用這樣的基本精神所演進的，再觀察實驗模擬的結果去修正定值的方式，而希望能得到一種可以不需要另外乘上係數或加上常數就能有效節省計算量的適應性門檻決定方式。

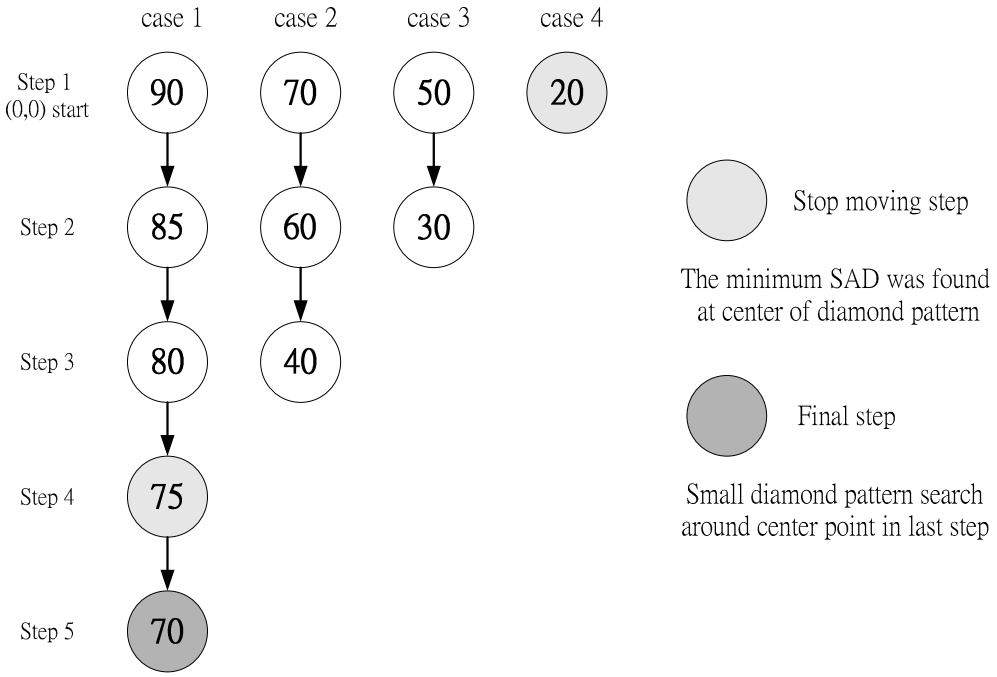



圖 3.5 區塊匹配範例示意圖二

### 3.2.2 適應性門檻決定方式

最先使用的方式是參考前一張畫面所做的區塊匹配結果來作門檻的估測，先將之前得到的所有 SAD 值取平均值，再加上一個 256 的常數增加計算量的節省，

$$Threshold = average(SAD_1, SAD_2, ..., SAD_n \ in \ last \ \ frame) + 256 \qquad (3\text{-}2)$$

這個方式因為還需要常數所以在適應性上並不好，因為增加 256 這個值可能會影響有些影像的動作估測結果，而且影響的程度也無法事先預估。所以後來分析連續畫面的 SAD 值情形，我們發現不加常數加速能力較差是因為一般在背景的部分因為沒有移動也沒有變化的原因而計算出的 SAD 值都是零，當前一張畫面出現大量 SAD 值為零的時候，取平均值的方式會降低門檻去節省計算量的能力。

我們採取另一個方式來計算門檻的大小，就是取前一張畫面的所有區塊 SAD 值的中位數，先將所有 SAD 值由大到小依序排列，再從數列中挑選位於中間的那個數，如果是偶數就取中間兩個數的平均，這個方式是中位數 A 的作法。

$$Threshold = median\_A(SAD_1, SAD_2, ..., SAD_n \ in \ last \ frame) \quad (3\text{-}3)$$

還有另一個中位數 B 的作法是將重複的 SAD 值視作一個數，將數字從大到小排列後，再取數列中間的數，這種方式可以避免當全部 SAD 值有超過一半為零的情況會使門檻值為零，這樣的門檻決定方式不需要其它係數或常數的修正就能得到相當好的結果，也就是與希望達到的目標接近。

$$Threshold = median\_B(SAD_1, SAD_2, ..., SAD_n \ in \ last \ frame) \quad (3\text{-}4)$$

但是在適應性門檻的硬體設計考量上，因為取中位數的方式需要耗費相當多的計算量，這就違背一開始我們所希望的設計目標。我們又考慮是否還有其它的計算方式，因為在之前做取平均值的計算方式中會因為 SAD 值為零的情況而得不到好的結果，所以我們去除掉為零的 SAD 值在做平均，希望能夠因此改善節省計算量的能力，而且能夠不必依靠乘上係數或是加上常數的方式去改變原來的已知資訊，所以就設計了非零 SAD 值的平均值計算式（3-5），這樣的作法對於硬體設計上的負擔也很小，當然我們連除法器也避免去使用，所以又在改進這個式子為一個能用更簡單的硬體實現的方式。

$$Threshold = average(SAD_1, SAD_2, ..., SAD_n \ non-zero \ in \ last \ frame) \quad (3\text{-}5)$$

考量到減少硬體上的實現負擔，所以我們利用平移除法器（shift division）來代替原來的取平均的除法，也就是將一般除數為任意正整數的平均值作法改為只能使用以 2 為底數的次方數，當然計算結果會有誤差，但是因為能利用此誤差向下修正平均值的值，以避免門檻值過大，這樣的計算方式也有益於得到較好的動作估測結果。

$$Threshold = average\_SD(SAD_1, SAD_2, ..., SAD_n \ non-zero \ in \ last \ frame) \quad (3\text{-}6)$$

當然以上這六種決定方式（3-1～3-6）不只是能應用在鑽石搜尋法上，在第二章所介紹的快速區塊匹配演算法都能適用，因為這些演算法都會存在多餘的計算步驟，尤其是 2.3 節中提到的四步演算法及新三步演算法，假設第一步搜尋到四周的八個位置時可能就能決定最後的移動向量，但是這兩個演算法還是要執行到第四步才會停止計算，這樣一來多做的第二步、第三步等對於動作估測的結果完全沒有幫助卻需要多作許多個位置的區塊匹配運算，這種情況都是可以避免的，所以雖然都是應用在鑽石演算法的改進上，但可以預期的是如果能準確的預估出提早終止計算的門檻大小，對於所有區塊匹配演算法計算速度的提升都能有明顯的改善，尤其是以鑽石演算法為基礎而改進的快速搜尋法。

## 3.3 實驗步驟及結果比較

此演算法的模擬實驗完全是依據第二章介紹的動作估測流程所設計的，並使用鑽石搜尋演算法來搭配提早終止技術來計算移動向量，其中的提早終止控制機制請參考圖 3.6，門檻（threshold, T）的大小由前一節中六種方式決定，第一張畫面因為沒有可以參考的數值所以直接把 T 值訂為零，也就是在 SAD 值為零的時候才會進入提早終止狀態。進入鑽石搜尋區塊匹配動作後，第一步從中心點為（0,0）的位置開始，計算出九個的 SAD 值再判斷其值最小的位置，再進入判斷機制，最小的 SAD 值若是小於門檻值 T 就提早終止搜尋進入下一個區塊的動作估測，否則繼續進行鑽石搜尋的下一步，直到進入最後一步搜尋步驟才會停止，搜尋過程中除了判斷提早終止的機制外其餘都與原來的鑽石搜尋演算法相同。

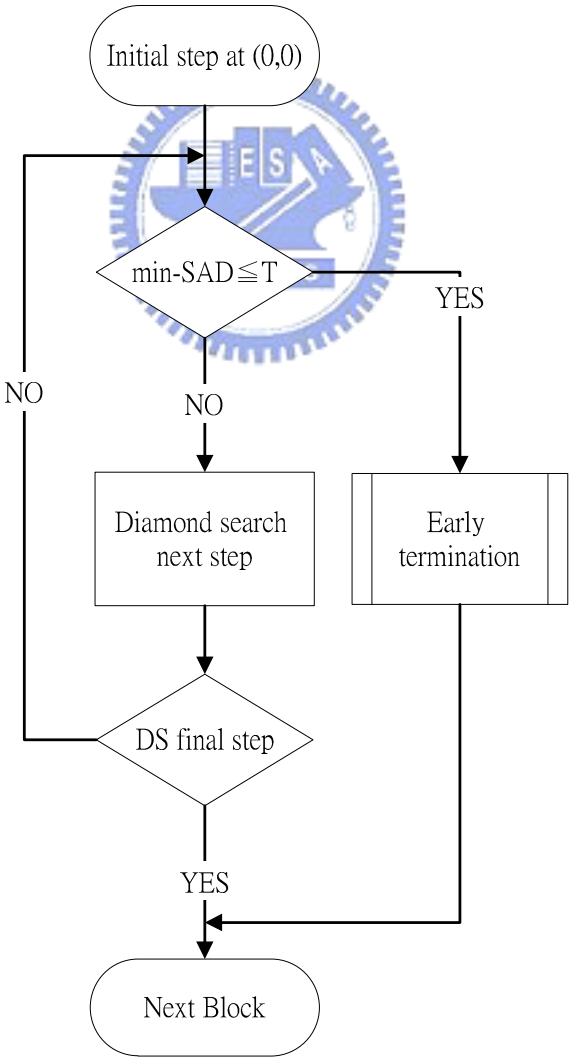



圖 3.6　提早終止控制流程圖

### 3.3.1 模擬環境

在軟體模擬環境下，採用四段不同的連續影像去作動作估測，每段影像都是每秒 30 張畫面（frame），長度都是十秒也就是共有 300 張畫面，每張畫面大小為 352×288 也就是採用 CIF 格式，區塊匹配演算法所選用的大區塊（macro-block）大小為 16×16 畫素（pixels），搜尋窗口（search window）的大小是訂為±7，也就是使用全部搜尋（full search）的方式會有 225 個位置（15×15）要作區塊匹配的計算。圖 3.7 是所選用的四段影像，分別是 Akiyo、Table、Foreman、Stefan。


圖 3.7　實驗用連續影像單張畫面

我們所比較的重點放在動作估測計算量的節省上，所以用搜尋位置的點數多寡來表示，也就是使用越少的搜尋點數（search points）完成動作估測代表速度越快，而動作估測結果的優劣則用均方和誤差（Mean Square Error, MSE）表示，也就是現時畫面中所有區塊跟移動向量位置上的參考畫面區塊作誤差值平方總合的平均值，這個值直接表示動作估測後會產生的剩餘畫面的大小，也就是壓縮前的資訊量，所以 MSE 值越小就可以視為動作估測的誤差越小。

$$MSE = \frac{1}{N \times N} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} (I_t(x+m, y+n) - I_{t-1}(x+v_x+m, y+v_y+n))^2 \qquad (3\text{-}7)$$

### 3.3.2 實驗結果

表 3.1 中的數據即是所有模擬實驗的結果，除了基本的鑽石搜尋法（DS）與全部搜尋法（FS），利用（3-1）到（3-6）的方式訂出的門檻所做的提早終止鑽石演算法也依序排列在表中，搜尋點數（search points）表示全部做完動作估測後的共作了幾個位置的區塊匹配，均方和誤差（MSE）是每一張畫面做完估測後的平均誤差。表格中的四段影像數據是特別依照畫面中物件的移動量大小排列，所以很明顯的可以發現移動量越大的影像動作估測的結果越差，也需要越多搜尋點數才能完成動作估測，這與所有相關的研究論文中都是一致的，所以可以知道當處理移動量很大的影像都很難有好的估測結果，而計算量的節省也較為困難，這是所有快速區塊匹配演算法都會得到的結論。

在全部搜尋法（full search, FS）中，搜尋點數都是一樣的也就是去搜尋的全部位置，而搜尋結果可以確定找到搜尋視窗中 SAD 值最小的區塊匹配位置，所以表 3.2 就是所有快速演算法相對於全部搜尋法的比較數據，加速倍數（speed up）就是搜尋點數的倍數，數值越大代表計算量節省越多，而均方和誤差百分比（MSE(%)）表示相對全部搜尋法的誤差百分比，數值越小就是代表動作估測的結果越接近全部搜尋法的 MSE 值，從表 3.2 中可以發現，單純的鑽石搜尋法就有很好的加速能力，可以有八倍到十六倍的加速效果，但在搜尋結果上就會有較大的誤差，這也是必須付出的代價，但是除了 Stefan 的影像外，其餘的誤差值都不是很大，尤其在 Akiyo 的影像中，因為此影像中只有極少數的物件在移動，所以可以有很好的估測結果，類似這種的連續影像使用全部搜尋法作動作估測很明顯的是沒有必要的。

表 3.3 是將使用適應性門檻技術執行提早終止動作與單純的鑽石演算法的比較結果，與前面的結果相同的是都是移動量越少的影像能有較好的結果，甚至能有兩倍的加速能力，但是在 Stefan 的影像中，只能增加 20%到 30%的速度，不過能證明的是只要應用提早終止技術就能更節省區塊匹配演算法的計算量，而且幾乎是不會增加誤差，從表 3.3 中可以發現除了用（3-3）的方式在 Foreman 的影像中會有較大的誤差，其餘增加的誤差都遠低於 10%，許多情況甚至連 1%都不到，這與我們的假設是符合的，但是在表中並不容易看出哪種門檻決定方式效果較好，所以我們針對個別影像再作比較。

| | Akiyo | | Table | | Foreman | | Stefan | |
|---|---|---|---|---|---|---|---|---|
| | search points | MSE | search points | MSE | search points | MSE | search points | MSE |
| FS | 24187904 | 3.7933 | 24187904 | 67.73 | 24187904 | 70.54 | 24187904 | 462.01 |
| DS | 1466611 | 3.8114 | 1999244 | 75.19 | 3007322 | 76.39 | 2915110 | 556.07 |
| adjacent block(3-1) | 718172 | 3.8206 | 1793011 | 76.78 | 2634308 | 77.58 | 2545717 | 574.83 |
| ave+256(3-2) | 597087 | 3.8125 | 1351586 | 77.91 | 1843634 | 86.25 | 2102153 | 581.23 |
| median_A(3-3) | 752463 | 3.8114 | 1723689 | 75.39 | 2414109 | 78.42 | 2456368 | 561.43 |
| median_B(3-4) | 639363 | 3.8115 | 1578081 | 75.65 | 2225439 | 79.93 | 2229913 | 573.17 |
| nonzero_ave(3-5) | 613825 | 3.8121 | 1417071 | 76.88 | 2034370 | 83.13 | 2157430 | 577.34 |
| nonzero_ave_SD(3-6) | 612657 | 3.8122 | 1426273 | 76.53 | 2054983 | 80.15 | 2163729 | 574.11 |

表 3.1　模擬結果數據

| | Akiyo | | Table | | Foreman | | Stefan | |
|---|---|---|---|---|---|---|---|---|
| | speed up | MSE(%) | Speed up | MSE(%) | speed up | MSE(%) | speed up | MSE(%) |
| DS | 16.49 | 0.48 | 12.1 | 11.01 | 8.04 | 8.29 | 8.3 | 20.34 |
| adjacent block(3-1) | 33.68 | 0.72 | 13.49 | 13.36 | 9.18 | 9.98 | 9.5 | 24.4 |
| ave+256(3-2) | 40.51 | 0.51 | 17.9 | 15.03 | 13.12 | 22.27 | 11.51 | 25.78 |
| median_A(3-3) | 32.14 | 0.48 | 14.03 | 11.31 | 10.02 | 11.17 | 9.85 | 21.5 |
| median_B(3-4) | 37.83 | 0.48 | 15.33 | 11.69 | 10.87 | 13.31 | 10.85 | 24.04 |
| nonzero_ave(3-5) | 39.41 | 0.5 | 17.07 | 13.51 | 11.89 | 17.85 | 11.21 | 24.94 |
| nonzero_ave_SD(3-6) | 39.48 | 0.5 | 16.96 | 12.99 | 11.77 | 13.62 | 11.18 | 24.24 |

表 3.2　與全部搜尋法之比較

| | Akiyo | | Table | | Foreman | | Stefan | |
|---|---|---|---|---|---|---|---|---|
| | speed up | MSE(%) | speed up | MSE(%) | speed up | MSE(%) | speed up | MSE(%) |
| adjacent block(3-1) | 2.04 | 0.24 | 1.12 | 2.11 | 1.14 | 1.56 | 1.15 | 3.37 |
| ave+256(3-2) | 2.46 | 0.03 | 1.48 | 3.62 | 1.63 | 12.91 | 1.39 | 4.52 |
| median_A(3-3) | 1.95 | 0 | 1.16 | 0.27 | 1.25 | 2.66 | 1.19 | 0.96 |
| median_B(3-4) | 2.29 | 0 | 1.27 | 0.61 | 1.35 | 4.63 | 1.31 | 3.07 |
| nonzero_ave(3-5) | 2.38 | 0.02 | 1.41 | 2.25 | 1.48 | 8.82 | 1.35 | 3.82 |
| nonzero_ave_SD(3-6) | 2.39 | 0.02 | 1.4 | 1.78 | 1.46 | 4.92 | 1.35 | 3.24 |

表 3.3　與鑽石搜尋法之比較

### 3.3.3 適應性門檻決定方式比較

表 3.3 中雖然可以看出應用提早終止技術在鑽石演算法上可以有很好的結果，但是因為不同門檻決定的方式的結果沒有絕對的好壞差異，所我們利用在 3.2.1 節的分析方式來判斷那種門檻決定方式能在搜尋點數與均方和誤差中會有較好的取捨。我們先將不同影像針對不同的門檻值作動作估測，也就是說先利用手動的方式去定出許多不同的 T 值，利用此結果出找出理想的門檻值可能的位置，在判斷哪種方式較接近那個位置以評估適合的適應性門檻決定方式。

圖 3.8 到圖 3.11 分別是是四段影像對於不同 T 值的搜尋點數與均方和誤差趨勢圖，在圖 3.9 的 Akiyo 的影像中因為靜止背景的部分相當大，所以會有很明顯的趨勢變化，而當計算量減少到一個程度才會出現均方和誤差大量上升，其它影像雖然沒有那麼明顯的變化，但是也可以看出在趨勢線的斜率會在計算量減少到一定程度時有急速改變，這證明了減少同樣數量的計算量所付出的代價會隨當時計算量的值而不同。在我們所設計的門檻決定方式中，很明顯的在計算量上能比用鄰近區塊預測的方式（adjacent block）要節省得多，而且均方和誤差的增加也相當有限，而在這其中幾種門檻決定方式以取中位數的方式對均方和誤差的影響最小，但是這種方式節省的計算量比去掉零以外的 SAD 值取平均值的方式要差，當然可以看得出門檻決定方式（3-6）是可以得到接近趨勢斜率變化的那一個區段的門檻值，這在四段影像中都能看到這樣的情況，這就是我們所希望的高適應性特質，在這四段影像中不管是物件的移動方向、速度與畫面的複雜度都有相當大的差異性，所以這樣的結果證明我們所得到的門檻決定方式能有相當好的適應能力，已經不需要再利用乘上係數或加上係數的方式來決定門檻值。

而且考慮到在硬體上（3-6）這個方式遠比取中位數的方式要容易實現，所以接下來的硬體實現移動估測演算法中，我們便是以這個方式作為提早終止技術的判斷機制。雖然以後所使用的門檻值都是從零以外的數值取平均值而產生的，但是不代表取中位數（3-5）的決定方式不適用提早終止技術，只是在硬體實現上會造成過多額外負擔，但是在均方和誤差百分比的增加上，用此方式可以抑制誤差在相當小的量上，而每一張畫面才計算一次門檻值在軟體執行上也不會增加過多計算量，所以（3-5）的決定方式仍有其價值。

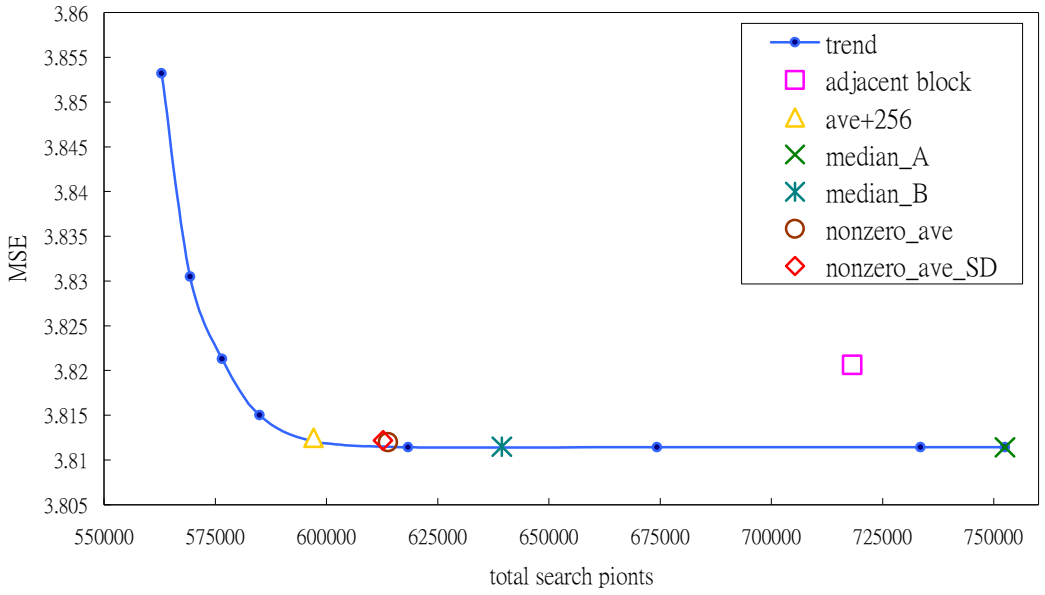



圖 3.8 Akiyo SP/MSE 比較圖

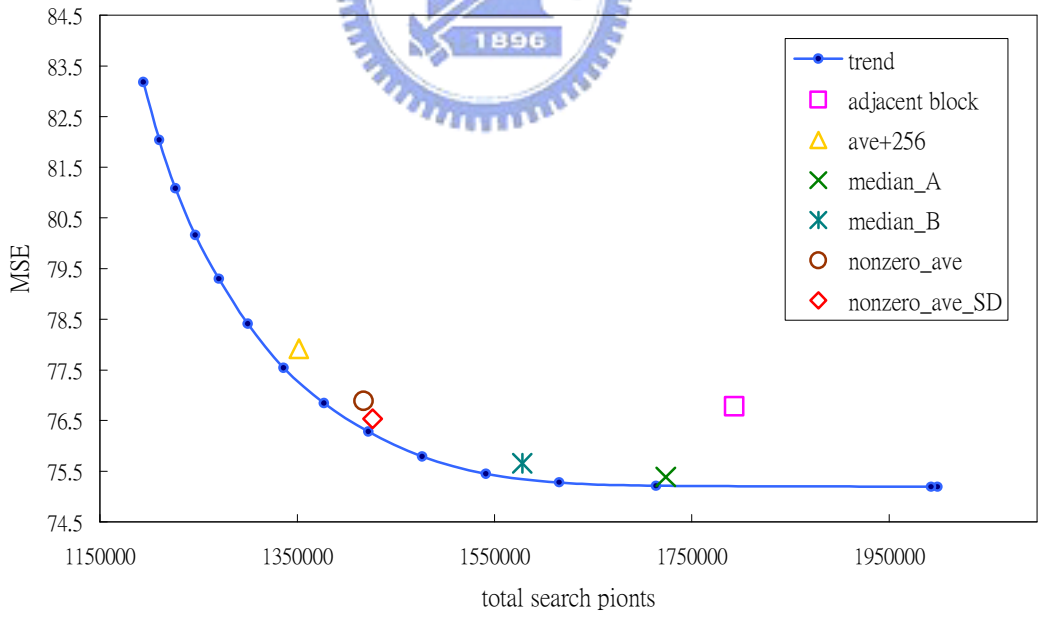



圖 3.9 Table SP/MSE 比較圖

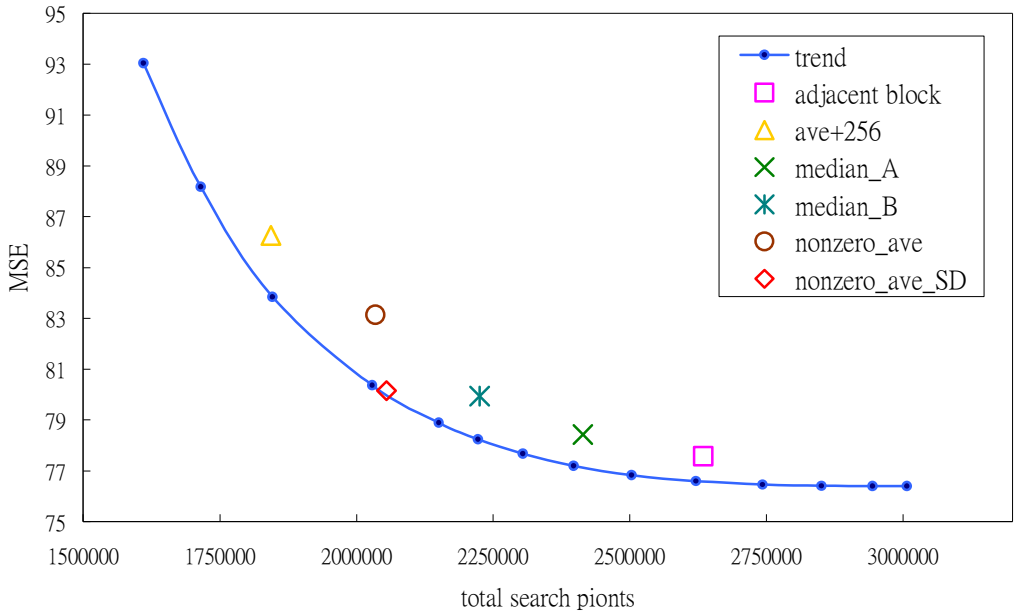



圖 3.10 Foreman SP/MSE 比較圖

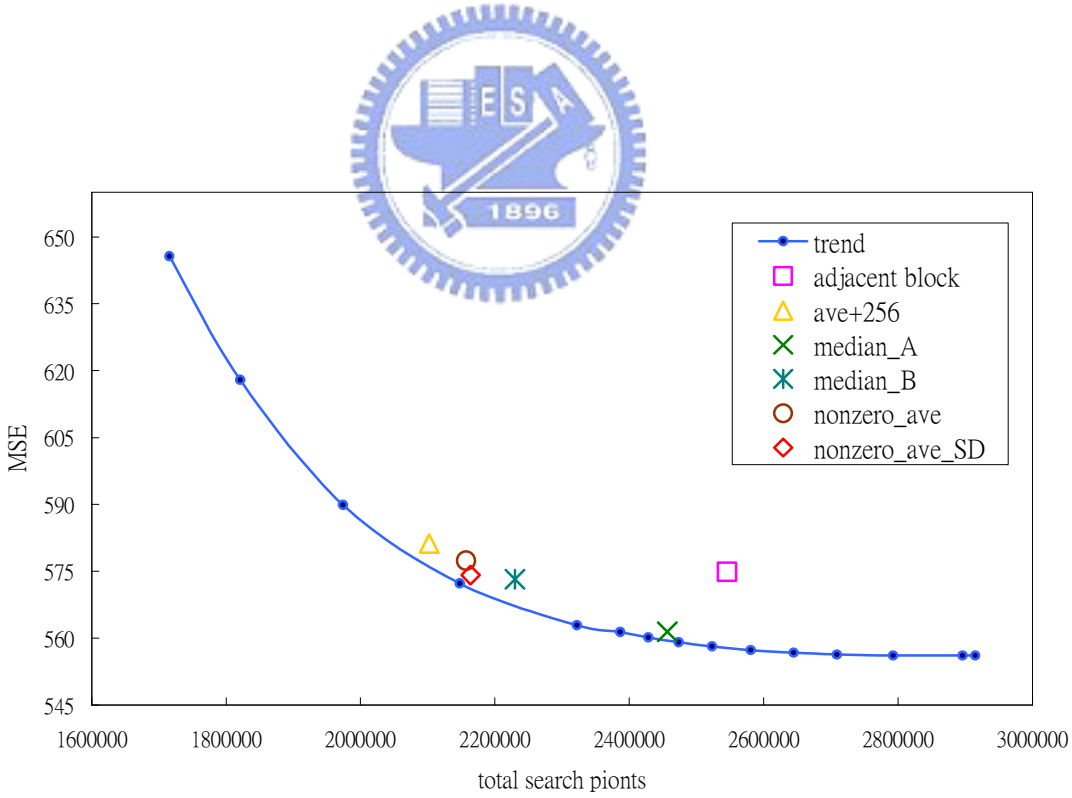



圖 3.11 Stefan SP/MSE 比較圖

### 3.3.4 應用在不同快速搜尋法上的比較

接下來將我們所設計的提早終止技術應用到參考資料中[8]、[11]、[12]的快速區塊匹配演算法中，這三種快速搜尋法都是根據鑽石搜尋法所改進的類似演算法，同樣是在搜尋位置上有所更改而非鑽石演算法的固定九個位置，雖然都較鑽石搜尋法更能節省移動估測所需的計算量，但是彼此的加速能力與造成的估測誤差都沒有明顯的優劣差異，我們可以發現因為同樣都是採用沒有固定搜尋步數的作法，所以如同鑽石搜尋法一樣會有多餘的搜尋步驟來判斷是否可以結束搜尋動作，在這裡我們利用前一節設計的判斷機制來提早終止不必要的搜尋步驟，模擬實驗結果如表 3.4 所示，我們將鑽石演算法定為基準，將其餘三種快速演算法及包括鑽石演算法在內的四種應用了提早終止技術（ET）的改進演算法放在一起比較，可以證明我們所選定的適應性門檻值的確可以節省更多的計算量，而且也很容易應用在這三種類似鑽石搜尋法的快速搜尋法，雖然效果會隨著不同影像內容而有相當的差異，但是對於誤差的增加都相當有限，這表示此門檻值在不同連續影像上的適應性不會因為不同快速搜尋法而有太大的影響，這也是因為我們在針對鑽石搜尋法設計時所希望能具備的特色。

表 3.5 是比較應用了提早終止技術與原來單純的快速搜尋法的差異，很明顯可以發現在所有項目中，誤差的增加絕大多數都在 5%以下，而在前幾個動作較慢、畫面低複雜度的影像中，誤差的增加甚至遠小於 1%，這也是因為此類的連續影像比較容易得到好的移動估測結果，尤其使用是類似鑽石搜尋法的快速搜尋法，所以當節省下搜尋步驟之後也不容易使誤差增加，全部平均來看誤差也都增加不到 2%，可以視為對估測結果只會造成相當小的影響。而在加速能力上就有較大的差異，雖然可以確定都能節省多餘的計算量，但是平均加速能力還是以應用在鑽石演算法上能有最大的效果，而在其它三種快速搜尋法中對於某些影像的加速能力就不大，但是對於 Akiyo、News、Weather 及 Children2 這四個影像不管是哪種搜尋法都能有很好的加速能力，從全部平均來看也還是有 1.37～1.69 的加速能力，這證明了我們所設計的適應性門檻值應用在這幾四種快速搜尋法都能有相當好的效果，只是在加速能力上對於不同影像內容的效果是不同的，這與物件移動速度與畫面的複雜度有相當的關係，而我們所設計的提早終止技術大幅提升了這四種快速搜尋法能因應不同影像內容去節省計算量的能力。

| | | DS | | AMMS[12] | | KCDS[11] | | NCDS[8] | |
|---|---|---|---|---|---|---|---|---|---|
| | | speed up | MSE(%) | speed up | MSE(%) | speed up | MSE(%) | speed up | MSE(%) |
| Akiyo | normal | 1 | 0 | 2.42 | 0.42 | 2.47 | 1.1 | 1.4 | 0.94 |
| | ET | 2.39 | 0.02 | 4.09 | 0.43 | 4.2 | 1.11 | 3.52 | 0.96 |
| News | normal | 1 | 0 | 2.42 | -0.46 | 2.48 | -0.43 | 1.4 | -0.44 |
| | ET | 2.36 | 0.05 | 4.2 | -0.45 | 4.22 | -0.43 | 2.65 | -0.44 |
| Weather | normal | 1 | 0 | 2.23 | 6.29 | 2.3 | 6.46 | 1.41 | 6.4 |
| | ET | 2.81 | 0.69 | 4.45 | 7.06 | 4.49 | 6.91 | 3.35 | 6.88 |
| Container | normal | 1 | 0 | 2.39 | 0.81 | 2.44 | 0.61 | 1.41 | 0.82 |
| | ET | 1.35 | 0.02 | 2.6 | 0.82 | 2.52 | 0.61 | 1.46 | 0.83 |
| Mother and daughter | normal | 1 | 0 | 1.75 | 2.5 | 1.92 | 2.7 | 1.37 | 3.77 |
| | ET | 1.35 | 0.26 | 2.3 | 2.68 | 2.17 | 2.76 | 1.59 | 3.88 |
| Silent | normal | 1 | 0 | 1.77 | 6.72 | 1.97 | 5.16 | 1.42 | 6.97 |
| | ET | 1.24 | 0.09 | 1.86 | 6.78 | 2.01 | 5.19 | 1.46 | 7 |
| Paris | normal | 1 | 0 | 2.13 | 2.41 | 2.23 | 2.27 | 1.4 | 2.66 |
| | ET | 1.3 | 0.1 | 2.21 | 2.47 | 2.28 | 2.29 | 1.42 | 2.68 |
| Table | normal | 1 | 0 | 1.52 | 8.57 | 1.55 | 9.24 | 1.41 | 9.6 |
| | ET | 1.4 | 1.78 | 2.03 | 12.11 | 1.95 | 10.91 | 1.63 | 11.7 |
| Children2 | normal | 1 | 0 | 1.78 | 7.21 | 1.95 | 6.39 | 1.42 | 8.85 |
| | ET | 2.23 | 1.61 | 2.81 | 9.24 | 3.09 | 7.5 | 2.79 | 10.24 |
| Flower | normal | 1 | 0 | 1.35 | 0.83 | 1.51 | 0.32 | 1.38 | 0.06 |
| | ET | 1.35 | 2.42 | 1.99 | 2.48 | 1.87 | 0.97 | 1.8 | 0.93 |
| Foreman | normal | 1 | 0 | 1.17 | 4.67 | 1.33 | 2.26 | 1.5 | 5.11 |
| | ET | 1.46 | 4.92 | 1.78 | 13.16 | 1.83 | 7.19 | 1.99 | 11 |
| Mobile | normal | 1 | 0 | 1.37 | 0.04 | 1.82 | 0.03 | 1.14 | 0.06 |
| | ET | 1.14 | 5.25 | 2.07 | 0.32 | 1.87 | 0.11 | 1.44 | 0.25 |
| Dancer | normal | 1 | 0 | 1 | 4.1 | 1.27 | 2.37 | 1.61 | 5.31 |
| | ET | 1.87 | 4.81 | 1.98 | 10.36 | 2.29 | 6.44 | 2.61 | 10.2 |
| Stefan | normal | 1 | 0 | 1.15 | 6.66 | 1.25 | 5.43 | 1.58 | 15.11 |
| | ET | 1.35 | 3.24 | 1.59 | 11.38 | 1.61 | 8.26 | 1.89 | 17.73 |

表 3.4　各式快速演算法與鑽石搜尋法之比較

| | diff. with DS | | diff. with AMMS[12] | | diff. with KCDS[11] | | diff. with NCDS[8] | |
|---|---|---|---|---|---|---|---|---|
| | speed up | MSE(%) | speed up | MSE(%) | speed up | MSE(%) | speed up | MSE(%) |
| Akiyo | 2.39 | 0.02 | 1.68 | 0.005 | 1.7 | 0.002 | 2.5 | 0.02 |
| News | 2.36 | 0.05 | 1.73 | 0.01 | 1.7 | 0.002 | 1.89 | 0.005 |
| Weather | 2.81 | 0.69 | 1.99 | 0.72 | 1.95 | 0.42 | 2.38 | 0.45 |
| Container | 1.35 | 0.02 | 1.09 | 0.01 | 1.03 | 0.002 | 1.04 | 0.006 |
| Mother* | 1.35 | 0.26 | 1.32 | 0.17 | 1.13 | 0.06 | 1.15 | 0.11 |
| Silent | 1.24 | 0.09 | 1.04 | 0.05 | 1.02 | 0.03 | 1.03 | 0.03 |
| Paris | 1.3 | 0.1 | 1.04 | 0.06 | 1.02 | 0.02 | 1.02 | 0.03 |
| Table | 1.4 | 1.78 | 1.33 | 3.26 | 1.25 | 1.52 | 1.16 | 1.91 |
| Children2 | 2.23 | 1.61 | 1.58 | 1.89 | 1.59 | 1.05 | 1.96 | 1.28 |
| Flower | 1.35 | 2.42 | 1.48 | 1.64 | 1.24 | 0.64 | 1.3 | 0.87 |
| Foreman | 1.46 | 4.92 | 1.51 | 8.1 | 1.38 | 4.82 | 1.32 | 5.6 |
| Mobile | 1.14 | 5.25 | 1.51 | 0.28 | 1.03 | 0.08 | 1.27 | 0.19 |
| Dancer | 1.87 | 4.81 | 1.99 | 6.01 | 1.8 | 3.97 | 1.62 | 4.65 |
| Stefan | 1.35 | 3.24 | 1.37 | 4.41 | 1.29 | 2.68 | 1.19 | 2.28 |
| AVG | **1.69** | **1.8** | **1.48** | **1.9** | **1.37** | **1.09** | **1.49** | **1.24** |
| Mother* = Mother and daughter | | | | | | | | |

表 3.5　提早終止技術的效能比較

# 第四章 硬體實現及結果

本章是介紹區塊匹配動作估測的硬體實現架構，因為使用了鑽石演算法與提早終止技術，所以比單純使用全部搜尋法的硬體增加了一些硬體負擔，因此我們硬體架構的設計目標是希望能減少硬體上的負擔，加上原本快速演算法能節省計算量的優點，就能實現出一個低功率消耗的動作估測硬體。

## 4.1 硬體實現架構

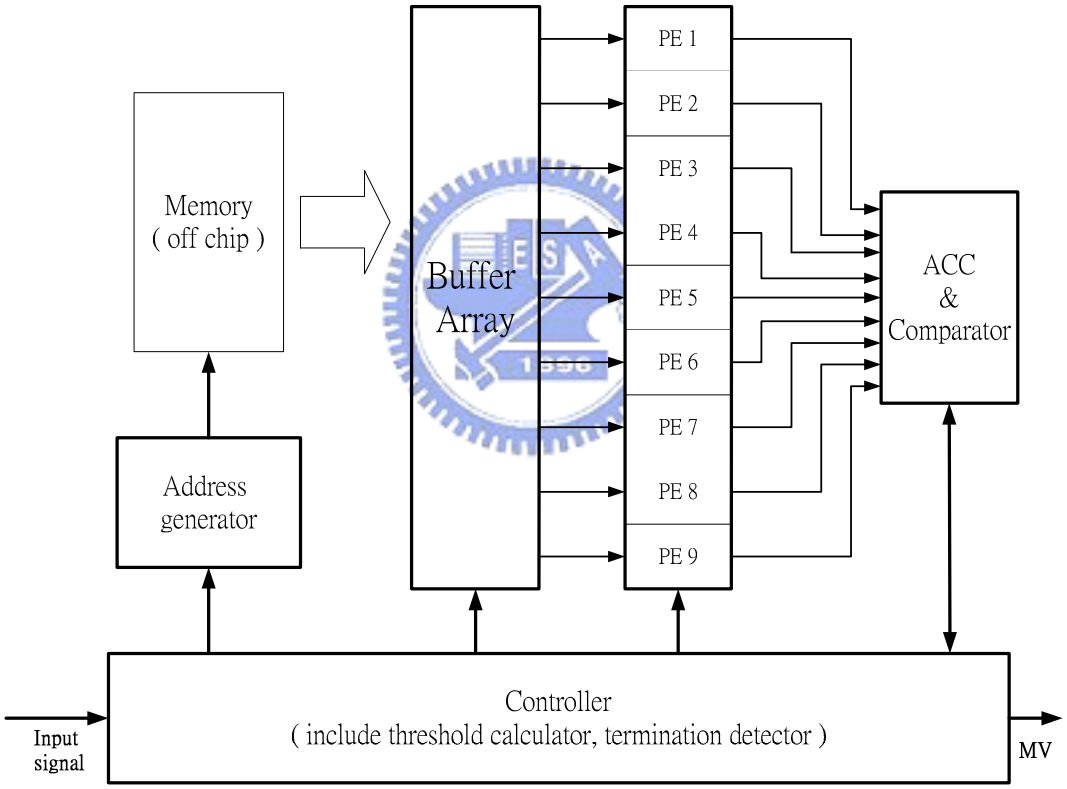



圖 4.1　硬體實現架構圖

動作估測的硬體實現分為幾個部分，參考圖 4.1，計算絕對差值的運算單元（PE），暫存畫面資料的緩衝存儲器陣列（buffer array），累加和比較絕對差值總和的累加器、比較器（ACC、comparator），還有讀取外部記憶體所需要的記憶體位址產生器（address generator），都由一個控制器（controller）來控制所有的資料運算流程。其中控制器的部分還包含計算門檻值的平移除法器及提早終止的

偵測器，參考圖 4.2，控制器的狀態流程詳細說明如下。

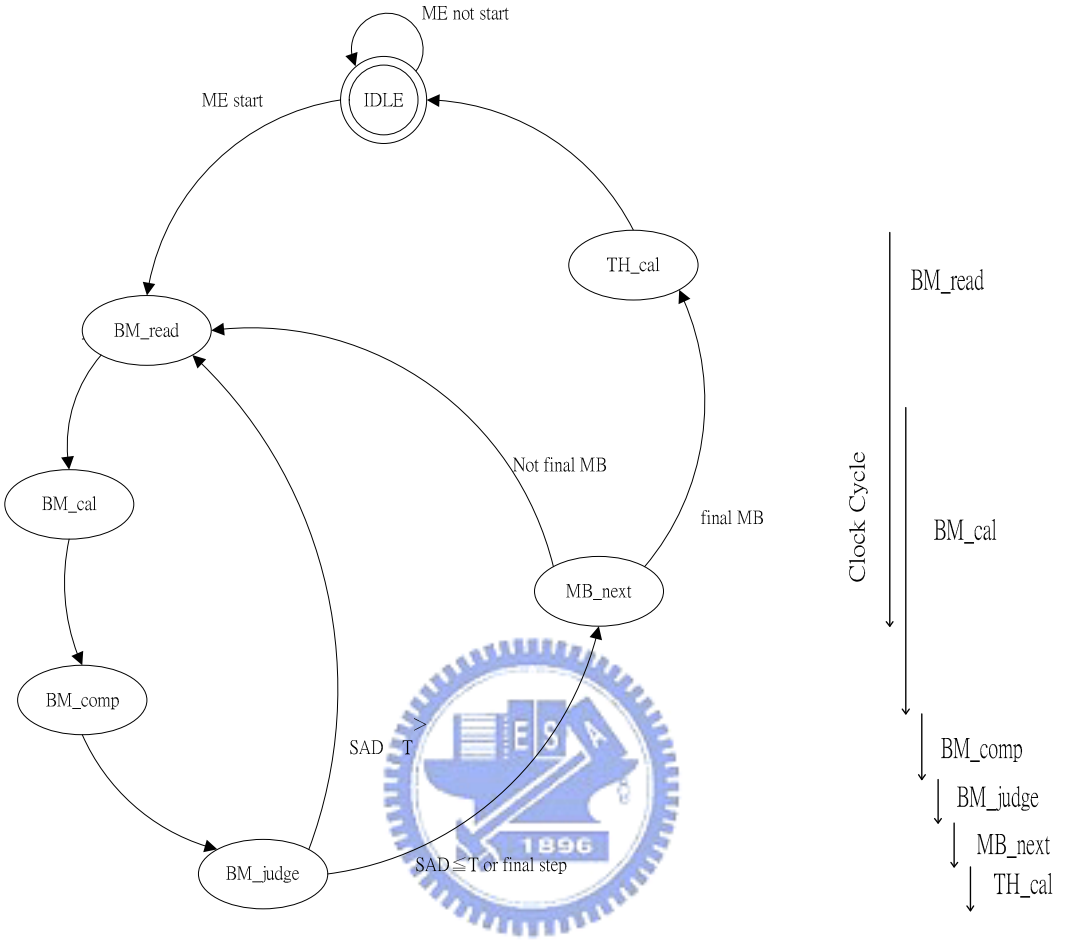



圖 4.2　控制狀態流程

在給予動作估測開始的訊號後，進入讀取狀態（BM_read），利用位址產生器產生所需的記憶體位址將記憶體中的畫面資料讀進硬體存儲器中，讀取過程中進入計算狀態（BM_cal），由計算單元來計算存儲器中的資料並送到累加器，得到九個位置的絕對差值總和（SAD）後，進入比較狀態（BM_comp），在此狀態比較出最小的 SAD 值及其位置，再進入判斷狀態（BM_judge），此狀態中就是決定是否此區塊匹配還需作下一步，如果還未執行到鑽石搜尋的最後一步或是所計算出的 SAD 值大於門檻值，即回到讀取狀態（BM_read）開始下一步的鑽石搜尋，否則就結束此一區塊的搜尋，進入下一個區塊的狀態（MB_next），在此狀態會先判斷是否已經執行到畫面的最後一個區塊，尚未計算到最後一個區塊就更新區塊的起始位置並進入讀取狀態（BM_read），開始此區塊的鑽石搜尋區塊匹配動作，當已經計算到最後的區塊時，便進入計算門檻值狀態（TH_cal），此

狀態的動作是利用之前所累計的 SAD 值來計算下一張畫面的動作估測所需要的新門檻值，計算出來後就回到起始的閒置狀態（IDLE），等待下一張畫面動作搜尋的開始，此硬體控制器的架構就是依照這樣的控制流程設計的。

此硬體規格也都是依照所設計的演算法所設計的，採取 16×16 畫素的區塊大小，每個畫素（pixel）用 8-bits 來表示，因為畫素沒有負值只需用正值來表示數值，所以數值範圍是從 0 到 255，而搜尋窗口的大小則定為±7。使用鑽石搜尋快速區塊匹配法，所以平行使用九個運算單元來計算第一步搜尋中的九個位置，比較出最小的 SAD 值再進行下一步的搜尋步驟，在此我們特地設計了一個實現鑽石搜尋法的硬體架構，希望能方便的套用於任意的搜尋窗口大小及連續影像的畫面大小。

## 4.2 記憶體位址產生器

在鑽石搜尋法中，因為需要根據上一步的搜尋結果來決定新的搜尋位置，所以需要因應變動的位置產生記憶體位址來讀取存取畫面的記憶體，在參考硬體實現動作估測演算法的研究[25]後，我們設計了一個專門應用於鑽石演算法的記憶體位址產生器，將搜尋窗口中的所有位置用 1 到 9 標記上去，見圖 4.3(a)，第一步就是從中心點(1)與四周(2~9)的位置去對照出所需的記憶體位址，並將畫素的數值存到緩衝存儲器陣列中，這樣就能平行去計算與現時區塊的 SAD 值，進行下一步時再從 SAD 最小值的那個位置去判斷接下來的新位置，去產生新的記憶體位址來存到緩衝存儲器中。見圖 4.3(b)，發現(2,0)是 SAD 值最小的位置，所以移到 6 的標記處為鑽石搜尋的中心，而 1、2、5、6 的結果已經得到了，所以只需在讀取標記為 9、7、8、4、3 位置的新資料，而直接將這些資料儲存到存儲器中，這樣只要利用運算單元 9、7、8、4、3（PE9、PE7、PE8、PE4、PE3）來計算新的 SAD 值即可完成新的鑽石搜尋區塊匹配動作。第二個範例見圖 4.3(c)，在第一步中若發現（1,-1）是 SAD 最小值的位置，下一步便只需讀取標記為 8、9、7 的位置的資料到存儲器中，這樣接下來就只需利用到運算單元 8、9、7 的計算就能完成新的區塊匹配動作，這樣對於對於產生新的記憶體位址只需用查表的方式就能完成，而對於運算單元的使用也很方便，只需要依照標記去決定那些運算單元需要計算，而之前重複的資料便暫存在之後的比較器中跟新的資料再比較出最小的 SAD 值，總共有 72 種組合，請參考表 4.1。

| | -7 | -6 | -5 | -4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -7 | 3 | 7 | 5 | 3 | 7 | 5 | 3 | 7 | 5 | 3 | 7 | 5 | 3 | 7 | 5 |
| -6 | 6 | 1 | 8 | 6 | 1 | 8 | 6 | 1 | 8 | 6 | 1 | 8 | 6 | 1 | 8 |
| -5 | 9 | 4 | 2 | 9 | 4 | 2 | 9 | 4 | 2 | 9 | 4 | 2 | 9 | 4 | 2 |
| -4 | 3 | 7 | 5 | 3 | 7 | 5 | 3 | 7 | 5 | 3 | 7 | 5 | 3 | 7 | 5 |
| -3 | 6 | 1 | 8 | 6 | 1 | 8 | 6 | 1 | 8 | 6 | 1 | 8 | 6 | 1 | 8 |
| -2 | 9 | 4 | 2 | 9 | 4 | 2 | 9 | 4 | 2 | 9 | 4 | 2 | 9 | 4 | 2 |
| -1 | 3 | 7 | 5 | 3 | 7 | 5 | 3 | 7 | 5 | 3 | 7 | 5 | 3 | 7 | 5 |
| 0 | 6 | 1 | 8 | 6 | 1 | 8 | 6 | 1 | 8 | 6 | 1 | 8 | 6 | 1 | 8 |
| 1 | 9 | 4 | 2 | 9 | 4 | 2 | 9 | 4 | 2 | 9 | 4 | 2 | 9 | 4 | 2 |
| 2 | 3 | 7 | 5 | 3 | 7 | 5 | 3 | 7 | 5 | 3 | 7 | 5 | 3 | 7 | 5 |
| 3 | 6 | 1 | 8 | 6 | 1 | 8 | 6 | 1 | 8 | 6 | 1 | 8 | 6 | 1 | 8 |
| 4 | 9 | 4 | 2 | 9 | 4 | 2 | 9 | 4 | 2 | 9 | 4 | 2 | 9 | 4 | 2 |
| 5 | 3 | 7 | 5 | 3 | 7 | 5 | 3 | 7 | 5 | 3 | 7 | 5 | 3 | 7 | 5 |
| 6 | 6 | 1 | 8 | 6 | 1 | 8 | 6 | 1 | 8 | 6 | 1 | 8 | 6 | 1 | 8 |
| 7 | 9 | 4 | 2 | 9 | 4 | 2 | 9 | 4 | 2 | 9 | 4 | 2 | 9 | 4 | 2 |

(a)

| | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|
| -3 | 8 | 6 | 1 | 8 | 6 | 1 | 8 | 6 |
| -2 | 2 | 9 | 4 | 2 | 9 | 4 | 2 | 9 |
| -1 | 5 | 3 | 7 | 5 | 3 | 7 | 5 | 3 |
| 0 | 8 | 6 | 1 | 8 | 6 | 1 | 8 | 6 |
| 1 | 2 | 9 | 4 | 2 | 9 | 4 | 2 | 9 |
| 2 | 5 | 3 | 7 | 5 | 3 | 7 | 5 | 3 |
| 3 | 8 | 6 | 1 | 8 | 6 | 1 | 8 | 6 |

(b)

| | -2 | -1 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|---|
| -4 | 5 | 3 | 7 | 5 | 3 | 7 | 5 |
| -3 | 8 | 6 | 1 | 8 | 6 | 1 | 8 |
| -2 | 2 | 9 | 4 | 2 | 9 | 4 | 2 |
| -1 | 5 | 3 | 7 | 5 | 3 | 7 | 5 |
| 0 | 8 | 6 | 1 | 8 | 6 | 1 | 8 |
| 1 | 2 | 9 | 4 | 2 | 9 | 4 | 2 |
| 2 | 5 | 3 | 7 | 5 | 3 | 7 | 5 |

(c)

圖 4.3（a）標記圖（b）範例圖一（c）範例圖二

鑽石搜尋法最後一步的動作也只要利用上一步中 SAD 最小值的位置四周緊鄰的標記來抓取資料，需要使用到硬體中存儲器陣列與運算單元，來完成最後一步的區塊匹配計算，但在後端比較器的設計上還是比較九個數值的結果，只是其中五個數值是從以上一步的中心點為中心的相鄰五個標記位置中取出來的。請參考表 4.1 中，小鑽石搜尋的九種組合。

| LD* shift up | | LD* shift up-right | | LD* shift right | | LD* shift down-right | |
|---|---|---|---|---|---|---|---|
| center | next position | center | next position | center | next position | center | next position |
| 1→4 | 2、6、7、8、9 | 1→5 | 8、9、7 | 1→6 | 9、7、8、4、3 | 1→2 | 4、3、8 |
| 2→5 | 3、4、8、9、7 | 2→6 | 9、7、8 | 2→4 | 7、8、9、5、1 | 2→3 | 5、1、9 |
| 3→6 | 1、5、9、7、8 | 3→4 | 7、8、9 | 3→5 | 8、9、7、6、2 | 3→1 | 6、2、7 |
| 4→7 | 5、9、1、2、3 | 4→8 | 2、3、1 | 4→9 | 3、1、2、7、6 | 4→5 | 7、6、2 |
| 5→8 | 6、7、2、3、1 | 5→9 | 3、1、2 | 5→7 | 1、2、3、8、4 | 5→6 | 8、4、3 |
| 6→9 | 4、8、3、1、2 | 6→7 | 1、2、3 | 6→8 | 2、3、1、9、5 | 6→4 | 9、5、1 |
| 7→1 | 8、3、4、5、6 | 7→2 | 5、6、4 | 7→3 | 6、4、5、1、9 | 7→8 | 1、9、5 |
| 8→2 | 9、1、5、6、4 | 8→3 | 6、4、5 | 8→1 | 4、5、6、2、7 | 8→9 | 2、7、6 |
| 9→3 | 7、2、6、4、5 | 9→1 | 4、5、6 | 9→2 | 5、6、4、3、8 | 9→7 | 3、8、4 |
| LD* shift down | | LD* shift down-left | | LD* shift left | | LD* shift up-left | |
| center | next position | center | next position | center | next position | center | next position |
| 1→7 | 5、6、4、8、3 | 1→9 | 4、5、6 | 1→8 | 2、7、6、4、5 | 1→3 | 6、2、7 |
| 2→8 | 6、4、5、9、1 | 2→7 | 5、6、4 | 2→9 | 3、8、4、5、6 | 2→1 | 4、3、8 |
| 3→9 | 4、5、6、7、2 | 3→8 | 6、4、5 | 3→7 | 1、9、5、6、4 | 3→2 | 5、1、9 |
| 4→1 | 8、9、7、2、6 | 4→3 | 7、8、9 | 4→2 | 5、1、9、7、8 | 4→6 | 9、5、1 |
| 5→2 | 9、7、8、3、4 | 5→1 | 8、9、7 | 5→3 | 6、2、7、8、9 | 5→4 | 7、6、2 |
| 6→3 | 7、8、9、1、5 | 6→2 | 9、7、8 | 6→1 | 4、3、8、9、7 | 6→5 | 8、4、3 |
| 7→4 | 2、3、1、5、9 | 7→6 | 1、2、3 | 7→5 | 8、4、3、1、2 | 7→9 | 3、8、4 |
| 8→5 | 3、1、2、6、7 | 8→4 | 2、3、1 | 8→6 | 9、5、1、2、3 | 8→7 | 1、9、5 |
| 9→6 | 1、2、3、4、8 | 9→5 | 3、1、2 | 9→4 | 7、6、2、3、1 | 9→8 | 2、7、6 |

LD* = Large Diamond search

| Small Daimond search | |
|---|---|
| center | next position |
| 1→1 | 7、6、8、4 |
| 2→2 | 8、4、9、5 |
| 3→3 | 9、5、7、6 |
| 4→4 | 1、9、2、7 |
| 5→5 | 2、7、3、8 |
| 6→6 | 3、8、1、9 |
| 7→7 | 4、3、5、1 |
| 8→8 | 5、1、6、2 |
| 9→9 | 6、2、4、3 |

表 4.1　鑽石搜尋記憶體位址讀取表

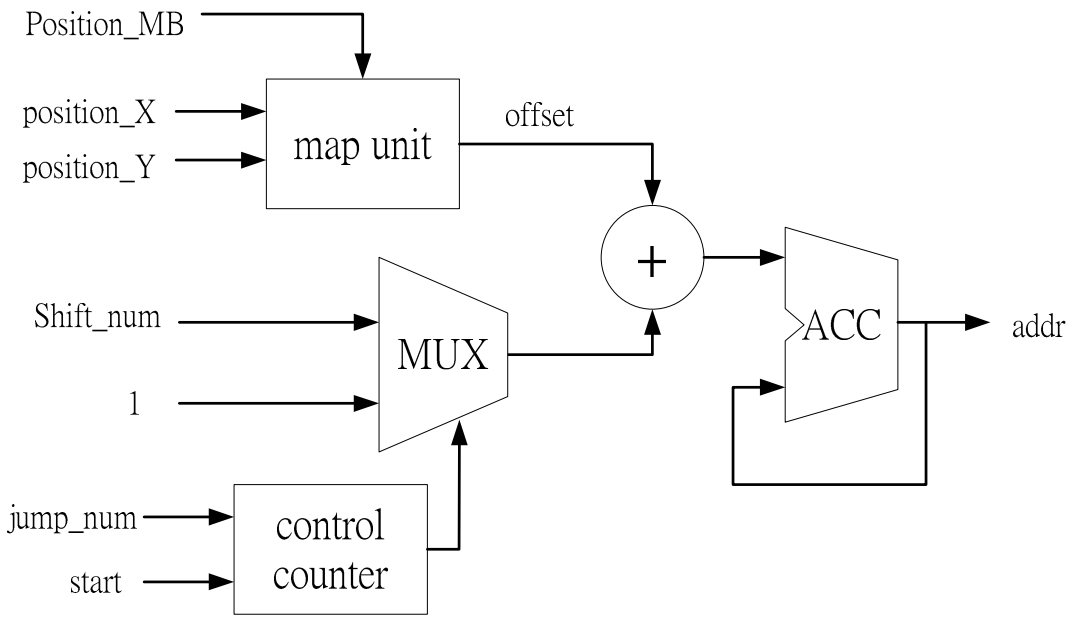



圖 4.4 記憶體位址產生器

參考圖 4.4 所示，此硬體架構包含 6 個輸入 1 個輸出訊號，Position_MB 表示現在所要搜尋的現時區塊位置，從此訊號就能知道現在區塊匹配的搜尋窗口位置，position_X 與 position_Y 表示現在鑽石搜尋位置的中心點，每移動到下一步就會更新到新的中心點，從這三個訊號就能對應到現在的外部實體記憶體的起始位址（offset）。Shift_num 與 jump_num 依照表 4.1 查出所需要更新的相對位置去設定，當 start 訊號啟動記憶體位址的計算後，從起始位址加上 Shift_num 開始輸出外部實體記憶體的讀取位址，之後就用累加器去累加 1 也就是讀取下一個畫素，一直到控制計數器數到 jump_num 的數值時，就在加上 Shift_num 去更新下一行需要的記憶體位址，舉例來說，輸出 addr 訊號從 $1002_{16}$ 到 $1011_{16}$ 是抓取記憶體中區塊第一行的 16 筆資料，接下來輸出就跳到 $1162_{16}$ 到 $1171_{16}$ 以抓取區塊第二行的 16 筆資料。

上述是利用查表的方式去產生現在起始的實體記憶體位址，也就是從目前所在的區塊位置與要計算的新搜尋位置來查出所要的畫素資料所存放實體記憶體的位址，再利用一個計數器去控制接下來要產生的位址，因為區塊是連續的畫素所組成，再利用一個平移數去改變位址到下一列的畫素，這樣就能得到我們所需要的新畫素資料，將這些放到存儲器中就能進行接下來的區塊匹配動作。在記憶體讀取的次數上相較於全部搜尋法需要全部記憶體位址讀取，使用了提早結束機制的鑽石演算法，在搜尋窗口為±7 的條件下只需讀取 1/3 至 1/2 的資料量，即使

在搜尋移動量很大的 stefan 連續影像中，還是只需讀取到 4/5 左右的資料，因為讀取外部記憶體是相當消耗功率的動作，遠大於硬體本身所消耗的功率，所以在這一方面能再節省對外部記憶體的讀取次數是有益處的。

而考量到搜尋窗口越大，全部搜尋法所需讀取的次數就會線性增加，所以使用快速演算法的優勢便會更明顯，從節省功率消耗的角度來看，實現快速演算法來減少對記憶體的讀取次數是更利於增加計算速度，雖然在目前的積體電路製程以足夠實現全部搜尋法的即時（real time）區塊匹配動作，不過為了處理能更大的畫面，例如 SXVGA(1280×1024pixels)以上的影像規格就還是需要快速演算法才比較有機會能達到即時的影像壓縮動作。

## 4.3 其它硬體架構設計

### a.門檻值計算器

在這個硬體設計中，因為要計算門檻值的大小，所以我們設計一個門檻值計算器，利用上一章中所提出的決定方式 3-6 作為設計基礎，所以我們設計一個先考慮 SAD 值為零的個數，再決定應該平移幾位的方式來實現一個除法器，最後則是額外增加一個由外部輸入門檻值的功能，所以再用一個多工器選擇不同來源的門檻值，架構如圖 4.5。

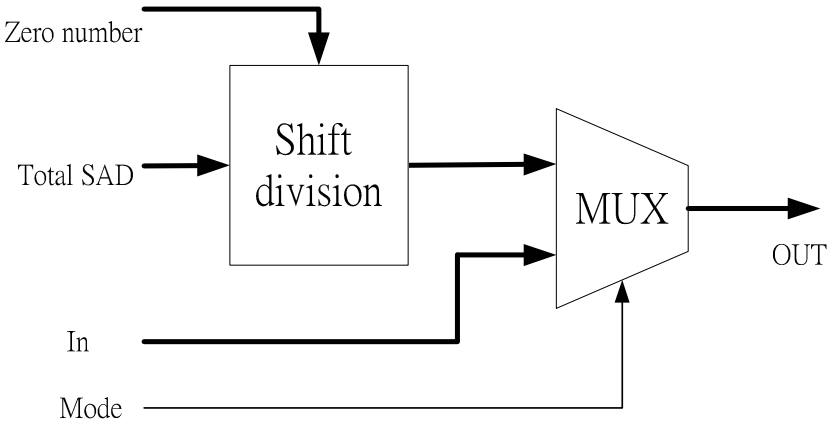



圖 4.5　門檻值計算器

平移除法器依 SAD 值為零的個數去分為十段平移方式，在大多數的連續影像中 SAD 為零的個數並不多，可以忽略所以都當做需要除以所有區塊的個數，我們所定的規格共有 396 個區塊，我們採取將上一章畫面 SAD 總和的二元數平移 9 位元與平移 11 位元的結果相加，得到的結果相當於除以 409.6 就很接近除以 396 的結果，其餘就是看零的個數來決定平移多少位元。此硬體實現的全部邏輯閘數（gate count）只需約 400 個，相較於全部的硬體所佔得比例相當低，所以對硬體負擔的增加遠小於使用取中位數的方式(3-4)，這也是我們在設計此門檻決定方式的目的。輸入端 Zero number 即為所累計 SAD 值為零的個數，Total SAD 是上一張畫面全部的 SAD 值總和，而 In 與 Mode 是為了可以由外界輸入門檻值所使用的訊號，輸出端 OUT 就是這一張畫面所要使用的適應性門檻值，由控制器中的暫存器所記錄下數值。

**b.運算單元**

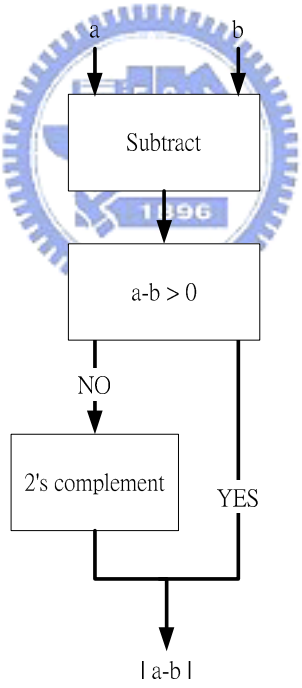



圖 4.6　運算單元

此硬體的運算單元（PE）是由相減取絕對值的計算所設計而成，參考圖 4.6，a 跟 b 是各輸入參考區塊跟現時區塊的畫素，我們先計算出 a-b 的數值，再判斷是否為負值，是的話便做二元補數轉為正值，不是負值的話就直接輸出，輸出與輸入都是 8-bits，這樣遠比先比較大小再相減的方式要節省硬體與功率消耗，此

硬體實現總共需要的邏輯閘數（gate count）約為 94 個，PE1~PE9 都是相同的設計，所以九個運算單元總共約 850 個邏輯閘數，所以需要的硬體負擔並不大。

**c. 比較器**

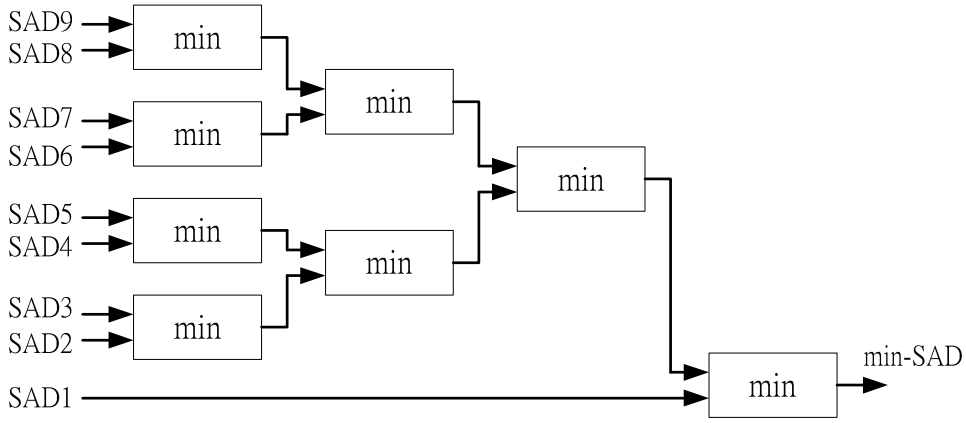



圖 4.7　比較器

圖 4.7 為比較器的部分，因為有平行九個輸入的 SAD 值來比較數值大小，所以需要做四次相互比較後即可得到最後的最小值，之後便輸入到控制器中執行接下來的動作，判斷接下來的搜尋動作是否還要繼續以及儲存下來去計算新的門檻值。這裡因為輸入的 SAD1～SAD9 是由運算單元計算出 16×16 筆畫素的絕對差值總和，min-SAD 輸出的結果是的九個其中最小的數值，所以都需要 17 位元的頻寬。

## 4.4 結果

| | Gate count(k) | Area ($um^2$) |
|---|---|---|
| PE array | 0.85(0.89%) | 8502 |
| Acc & comparator | 5.5(5.7) | 54752 |
| Controller(with THC) | 2.6(2.9) | 26207 |
| THC(threshold computer) | 0.4(0.41%) | 3971 |
| Address generator | 1.0(1.1%) | 10244 |
| Buffer | 86.5(89%) | 865120 |
| Total | 96.5(100%) | 964852 |

表 4.2 硬體邏輯閘數及面積

利用 TSMC 0.18um 的製程所實現的硬體，其結果如表 4.1，可以發現針對門檻值設計的架構對於整個移動估測的硬體負擔增加不大，而操作在 66.6MHz 下功率消耗約為 102mW，但是要注意的是因為不同影像會有不同的搜尋步驟，所以在移動估測過程中功率消耗與估測的時間都不是固定的，如在 Akiyo 影像中，平均不到 400 個時鐘週期（clock cycle）就能輸出一個移動向量，在其它影像中就需要較多時間，但平均也都不到 2000 個時鐘週期（clock cycle）就能完成一次移動估測，這樣對於每秒 30 張的 352×288 畫面輸入是足以在即時條件下完成移動估測的動作。但是考慮到演算法沒有在一般情況下終止的最長時鐘週期，若做出 50 步的搜尋動作，就會需要高達 20000 時鐘週期，可能就需要設定搜尋步數的上限來限制這種情況的發生來達到即時（real time）完成的要求。我們利用所需得搜尋步驟來估測可能需要的時鐘週期，如表 4.2 所示，將讀取記憶體所需的時鐘週期包括在內，如果在最多步的搜尋動作下，每估測一張畫面就需要到 8,000,000 個時鐘週期，這遠大於 stefan 這個影像所需的時鐘週期，雖然發生這種情況的機會很小，但是這就需要另外設計控制機制來避免這種情況的發生。

| | Usual case | worst case(50 steps) |
|---|---|---|
| clock cycles per MB | 1,800 | 20,000 |
| clock cycles per frame | 720,000 | 8,000,000 |

表 4.3 所需時鐘週期之估測

圖 4.8 是使用 SOC Encounter 所佈局出來的晶片佈局圖，在 TSMC 0.18um 製成之下核心的部分面積為 1.08×1.08mm$^2$，加上 IO PAD 之後，實體晶片大小為 1.26×1.26mm$^2$，最快操作頻率超過 80MHz。參考圖中框出的大約位置，主要的面積都是存儲器（Buffer）的部分，是因為此部分需要用暫存器（register）來放置 388 個 8 位元的參考區塊與 256 個 8 位元現時區塊的資料。其它硬體部分則集中放置在上方，而圖 4.9 則是此晶片的輸出與輸入腳位的定義圖，主要都是與外部記憶體做資料的傳輸與控制訊號。

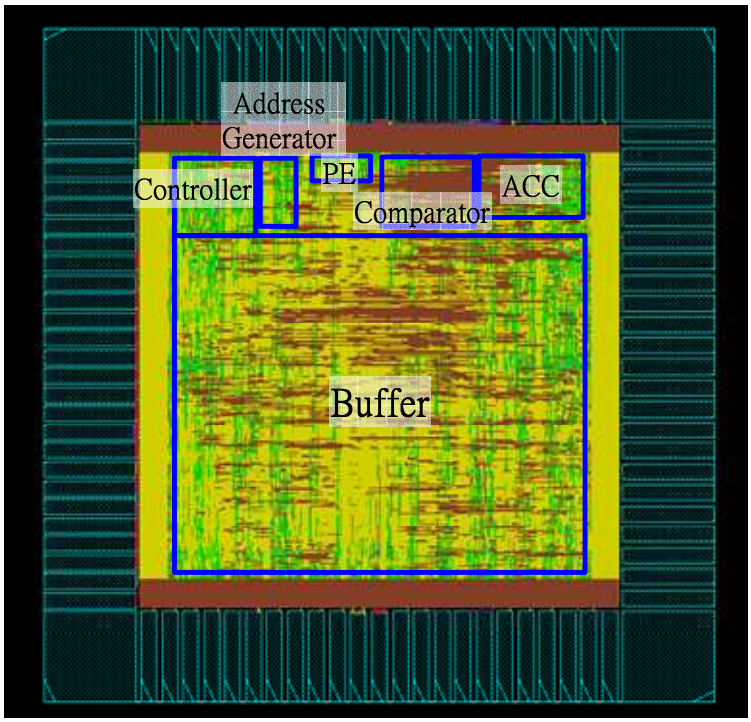



圖 4.8 晶片佈局圖

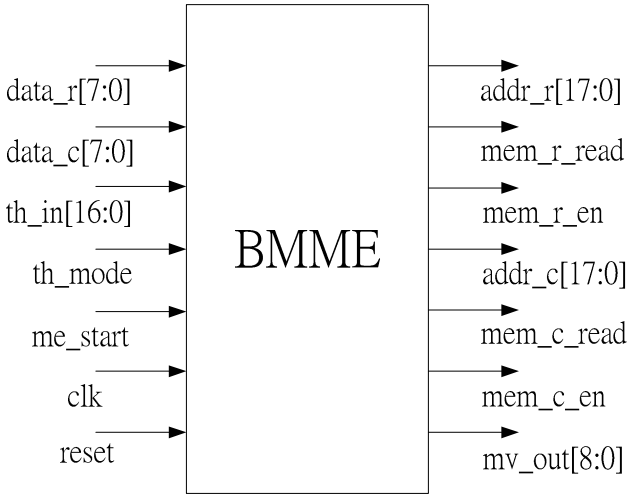



圖 4.9 晶片輸出輸入腳位圖

# 第五章 結論

## 5.1 主要貢獻

在影像壓縮技術中，因為動作估測的計算量佔總計算的比例超過六成，不管是在軟體或硬體上都是很大的負擔，所以藉由快速區塊匹配演算法的設計，我們追求能盡量在不影響估測結果的條件下去節省更多的計算量，雖然現在有許多快速區塊匹配演算法能達到此目的，但是因為影像內容上的差異而還沒有一個全部適用的演算法，但是在提早終止技術上，雖然節省的計算量並非很可觀，但是對於動作估測的結果影響十分有限，所以是一種能適合各種不同連續影像的快速演算法。我們在這個研究中設計了一種簡易的判斷機制去節省不必要的計算步驟，而能有效的節省區塊匹配過程中多餘的計算量。此提早終止技術中主要的門檻值是根據前面畫面計算的數值求得的，利用此門檻值判斷出現在要處理的畫面中可能會得到的區塊匹配數值，以節省下對動作估測結果無益的計算量，這樣的提早終止技術能套用在大多數的快速區塊匹配演算法而得到更好的加速能力，這個研究中我們先將之應用在鑽石搜尋法上，並分別針對全部搜尋法及未加此判斷機制的原始鑽石搜尋法做比較，軟體模擬的結果與我們所希望達成的目標是符合的。而且實驗結果顯示應用在以鑽石搜尋法為基礎改進的其它快速演算法也都能有很好的效果，代表許多不必要的搜尋步驟都可以利用我們所設計的適應性門檻值來避免，因此我們提出的提早終止技術中的適應性門檻值的確可以有效改進這類快速區塊匹配演算法的加速能力。

在設計門檻決定方式的過程中，我們從不同連續影像的結果中發現，在影像中有包含大量背景與少量移動物件的新聞影像下，這樣的提早終止技術能有相當優秀的加速能力，因為在要處理區塊是位於大量背景中的時候，所得的數值都會小於所定的門檻值，所以只需一步的區塊匹配動作即可結束，這與影像壓縮中利用時間域的相關性相同，當時間域的相關性越高、物件移動量小的時候，壓縮比越高而動作估測也越不需要大量的計算量，所以應用了此提早終止技術的鑽石搜尋法才可以更忠實的得到這樣的結果。而我們在模擬結果中證實我們所設計的門檻決定方式能適應性的針對不同影像而改變，不需要再去加上常數或乘上係數以

達到目的，這就是我們所希望的門檻決定方式，也因為這種低複雜度的提早終止方式，所以在硬體實現也不會增加太多硬體負擔。

容易在硬體上實現的快速演算法也是我們的設計重點，我們利用硬體描述語言實現出此我們所設計的快速動作估測演算法，不單是考慮區塊匹配計算量上的節省，也為了鑽石演算法的記憶體讀取額外設計快速位址產生器，不單是可以簡化讀取參考畫面資料的過程和硬體內部的低功率消耗目的，也減少對於外部記憶體的讀取次數，這樣也都能降低整體的功率消耗。而本研究中經由快速演算法的設計到硬體架構的實現，其主要的目的都是為了大幅增加動作估測的速度，進而達到即時的影像壓縮。

## 5.2 未來展望

在本論文的研究過程中，衍生了許多值得再探討的題目，可以作為將來加速區塊匹配演算法或應用在不同層面的動作估測上，雖然不一定都能有很好的效果，但是可以確定在動作估測上還能有許多改進的可能性。本節中會詳細討論與本研究動作估測相關的改進方式。

### 門檻值的決定方式

首先要提出的是門檻值的決定方式，我們可以知道此門檻值的大小決定了提早終止的判斷條件，所以直接造成計算量與移動估測結果上的差異，而本研究中只是找出在對移動估測結果影響小而盡可能節省計算量的方式，但是並沒有對整體的影像壓縮結果做考量，也就是說在資料傳輸量控制（bit-rate control）的條件下，我們可以再節省更多的計算量而故意犧牲估測結果使壓縮比下降，因為我們知道在處理某些影像的壓縮比本來就可以很高，如 Akiyo 這類的畫面，所以我們有機會藉由提高門檻值到資料傳輸量符合所需即可，而不必要求動作估測的準確度，這也是在本論文硬體設計中，會設計門檻值由外部輸入的機制，這樣可以由使用者自己判斷是否要減少區塊匹配的步驟。

從本研究中已知門檻值的估測是可以由多種方式決定，雖然我們為了硬體上

的考量而採取了一個門檻值處理一張畫面的適應方式，不過如果從不同區塊不同門檻的角度來設計應該更能增加提早終止的適應性，尤其是在模擬中發現針對鄰近方塊的估測方式雖不如我們所設計的方式，但是在區塊較小的空間域相關性是較為明顯，所以可以結合我們所考量的時間域相關性便能準確估出現時區塊的門檻值，這樣應該可以得到更好的適應性，但是並不一定能省下更多的計算量，尤其還需要比較多筆數據這一點可能會增加額外的計算時間。空間域與時間域結合可以參考以下的方式，例如利用前一張畫面的相同位置區塊與其鄰近的區塊的資訊，還有現時區塊的鄰近區塊，這些數據還可以用比較大小或是取平均的方式去預估現時區塊的門檻值，估測的準確度應該會較高，但是這也會受影像的內容在空間域與時間域是否有明顯的相關性而影響。

## 鑽石搜尋演算法

雖然本研究並非討論快速區塊匹配演算法的搜尋方法，但是從鑽石搜尋演算法的搜尋步驟上可以看出，對於真正的移動向量很大時，鑽石搜尋法能準確估測出此向量的機會並不高，尤其是搜尋步驟沒有往那個位置移動的趨勢時，所以也有研究是希望能改進鑽石搜尋法的估測準確度，這裡有一個方式是反過來利用提早終止的技術，也就是變成增加計算步驟的判斷機制，來要求鑽石演算法去搜尋更精確的移動向量，這樣的方式雖然會增加計算量，不過對於移動量較大的連續影像來說會較能避免只能發現鄰近區域的 SAD 最小值，而錯失搜尋到較遠區域的 SAD 最小值，當然這樣的方式也適用於類似鑽石搜尋法的變形搜尋法。

## 影像壓縮格式

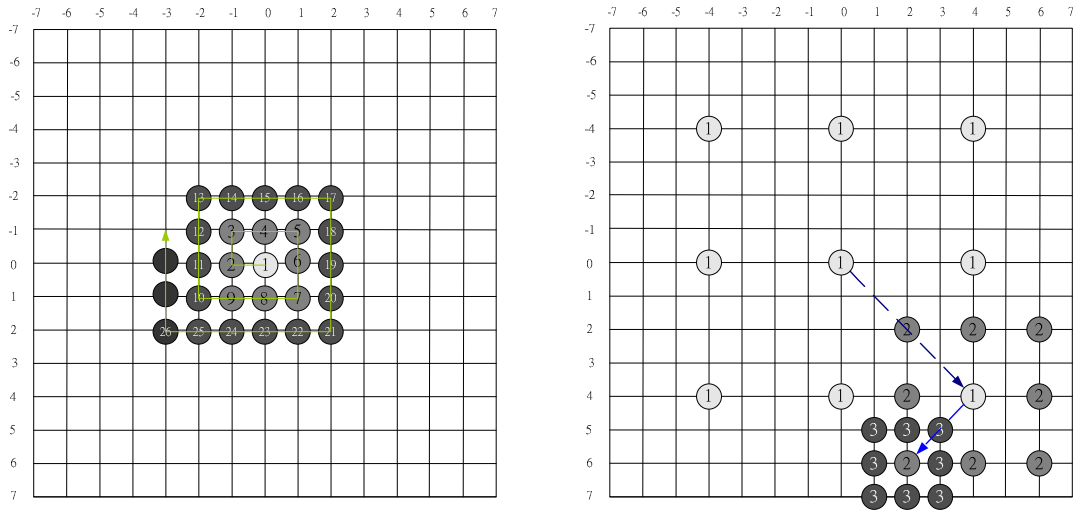
在不同影像壓縮格式中，動作估測的方式大都是相同的，所以本論文中所提出的快速估測演算法能適用於許多影像壓縮格式，如 MPEG4 等。但在 H.264 中因為每個區塊的大小不是固定的，也就是有些區塊是使用 16×16 的大小，有些是用 8×16、8×8 到 4×4 不等的大小組合，所以在門檻值的決定方式以 16×16 的方式決定出來之後，要是必須處理 4×4 的區塊大小時，就除以 16 以得到此區塊的門檻值，同理當處理 8×8 的大小就除以 4，這樣就能應用在 H.264 的壓縮格式上，只是要注意的是一開始如果是從 4×4 的區塊開始處理，那就需要用累加的方式去求較大區塊的門檻值，所以這樣的提早終止技術也適用於 H.264 壓縮格式中可變

區塊（variable block）的動作估測技術。

## 應用範圍

在本論文中雖然只有應用在鑽石演算法及類似的快速演算法上，但是其實還有許多獨立的動作估測加速技術，如第二章中所提到的預估移動向量的方式，都能在應用在這個快速演算法之上，尤其是快速演算法還有許多獨立的改進方式，我們並沒有應用在此搜尋法上，如果都能把彼此獨立的加速技術結合在一起相信能得到非常好的加速能力，如同[23]研究中所組合出的快速演算法，因為我們所設計的提早終止技術也是獨立於其中所使用的方式，再套用我們設計的提早終止技術上去相信就能得到更好的加速能力。

此外，不止是類似鑽石搜尋法這類的快速搜尋法，在圖 5.1 所顯示的兩種快速搜尋法也適合我們所設計的適應性門檻值來提早終止搜尋步驟，圖中的螺旋全部搜尋法（spiral full search）是從中心點開始向外搜尋，雖然類似全部搜尋法，但可以利用提早終止的技術去停止後面的搜尋步驟，這樣就比一般的全部搜尋要較節省計算量，而三步搜尋法中也可以利用適應性門檻值來判斷是否要完成三步的搜尋，還是只需一到二步就停止搜尋動作，這兩個方式都應該都能加快原始的搜尋法的搜尋速度，但是對於誤差的增加就要看搜尋過程中是否能在一開始就得到較準確的 SAD 值，這樣才能預估出較準確的適應性門檻值。



(a)　　(b)

圖 5.1 (a)螺旋全部搜尋法(b)三步搜尋法

# 參考文獻


[1] MPEG4 specification, “INTERNATIONAL STANDARD ISO/IEC 14496-2: Information technology — Coding of audio-visual objects — Part 2: Visual,” ISO/IEC 2001

[2] H.264 specification, “Draft Text of Final Draft International Standard for Advanced Video Coding, ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC,” ISO/IEC 2003

[3] T. Koga, K. Iinurna, A. Hirano, Y. Iijima, and T. Ishiguro, “Motion-compensated interframe coding for video conferencing,” Proc. of NTC 81, pp. C9.6.1-9.6.5, New Orleans, LA, Nov./Dec. 1981.

[4] Reoxiang Li, Bing Zeng; Liou, M.L., ”A new three-step search algorithm for block motion estimation,” IEEE Trans. Circuits Syst. Video Technol., Vol. 4, pp. 438-442, Aug. 1994.

[5] Lai-Man Po, Wing-Chung Ma, **“**A novel four-step search algorithm for fast block motion estimation,” IEEE Trans. on Circuits and Systems for Video Technology, Vol. 6 , Issue 3, pp. 313 - 317 June 1996

[6] S. Zhu and K.-K. Ma, **“**A new diamond search algorithm for fast block matching motion estimation,” Proc. of Int. Conf. Inform., Communication, Signal Process., Singapore, pp. 292 - 296 ,Sept. 1997

[7] S. Zhu and K.-K. Ma “A new diamond search algorithm for fast block matching motion estimation”, IEEE Trans. Image Process., pp. 287-290, Feb. 2000

[8] Hongiun Jia; Li Zhang , “A new cross diamond search algorithm for block motion estimation,” IEEE International Conference on Acoustics, Speech, and Signal Processing, 2004. (ICASSP '04). vol.3 pp. iii - 357-60 May 2004

[9] Chun-Ho Cheung, Lai-Man Po “A novel cross-diamond search algorithm for fast block motion estimation**”** IEEE Trans. Circuits System Video Tech., Vol.12, pp. 1168-1177, Dec. 2002

[10] Chun-Ho Cheung, Lai-Man Po, “A novel small-cross-diamond search algorithm for fast video coding and videoconferencing applications,” IEEE International Conference on Image Processing. 2002. Proceedings. 2002 Volume 1, pp. I-681-I-684 Sept. 2002

[11] Chi-Wai Lam, Lai-Man Po, Chun Ho Cheung, “A novel kite-cross-diamond search algorithm for fast block matching motion estimation,” IEEE International Symposium Circuits and Systems, ISCAS’04, Proceedings of the 2004 International Symposium on Volume 3, pp. 729-32 May 2004

[12] Yilong Li, Oraintara S., “A novel adaptive multi-mode search algorithm for fast block-matching motion estimation,” IEEE International Symposium Circuits and Systems, ISCAS’04. Proceedings of the 2004 International Symposium on Volume 3, pp. III-977-III-980 Vol.3, May 2004

[13] Ce Zhu, Xiao Lin; Lap-Pui Chau, “Hexagon-based search pattern for fast block motion estimation,” IEEE Trans. on Circuits and Systems for Video Technology, Volume 12, Issue 5, pp.349 - 355, May 2002

[14] Lopes F., Ghanbari M., ”Hierarchical motion estimation with spatial transforms,” IEEE International Conference on Image Processing, Proceedings. 2000, Volume 2, pp.558 - 561, Sept. 2000

[15] W. Li, E. Salari, “Successive elimination algorithm for motion estimation,” IEEE trans. Image processing, Vol.4 pp.105 - 107, Jan. 1995

[16] S.-M. Jung, S.-C. Shin, H. Baik, M.-S. Park, “Efficient multilevel successive elimination algorithms for block matching motion estimation,” IEE Proc. -Vis Image signal process .Vol.149 No. 2, pp.73 - 84, April 2002

[17] M. Yang, H. Cui and K. Tang, “Efficient tree structured motion estimation using successive elimination,” IEE Proc. -Vis. Image Signal Process., Vol. 151, No. 5, pp.396 - 371, October 2004

[18] Tourapis, A.M., Au, O.C.; Liou, M.L., “Highly efficient predictive zonal algorithms for fast block-matching motion estimation,” IEEE Trans. on Circuits and Systems for Video Technology, Volume 12, Issue 10, pp.934 - 947, Oct. 2002

[19] Ismaeil I., Docef A., Kossentini F., Ward R, “Efficient motion estimation using spatial and temporal motion vector prediction,” Proc. of Image Processing, ICIP’99. 1999 International Conference on Volume 1, pp.70 - 74, 1999

[20] Zhibo Chen, Peng Zhou, Yun He, “Fast Motion Estimation for JVT,” Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG (ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6) 7th Meeting: Pattaya II, Thailand, March 2003

[21] Atsushi Hutubu, Takashi Miyazuki, and Ichiro Kurodu, “Optimization of decision-timing for early termination of SSDA-based block matching,” IEEE International Conference on Acoustics, Speech, & Signal Processing. Hong Kong; pp.821 - 824, April 2003

[22] Pol-Lin Tai, Shih-Yu Huang, Chii-Tung Liu, Jia-Shung Wang, “Computation-aware scheme for software-based block motion estimation,” IEEE Trans. on Circuits and Systems for Video Technology, Volume 13, Issue 9, pp.901 - 913 Sept. 2003

[23] Jianfeng Xu, Zhibo Chen, Yun He, “Efficient fast ME predictions and early-termination strategy based on H.264 statistical characters,” Information, Communications and Signal Processing, 2003 and the Fourth Pacific Rim Conference on Multimedia. Proceedings of the 2003 Joint Conference of the Fourth International Conference on Volume 1, pp.218 - 222, Dec. 2003

[24] Tourapis, A.M., Au, O.C., Liou, M.L, “Predictive motion vector filed adaptive search technique (PMVFAST) enhancing block based motion estimation,” Proc. of SPIE conf. on visual communication and image processing, Vol.4310, 2001

[25] Her-Ming Jong, Liang-Gee Chen, Tzi-Dar Chiueh, “Parallel architectures of 3-step search block-matching algorithm for video coding,” IEEE International Symposium Circuits and Systems, 1994. ISCAS'94., 1994 on Volume 3, pp.209 - 212 June 1994

[26] Abbas, M.; Talha, B.; Khan, S.; Abbas, A. ”A motion estimation chip for block based MPEG-4 video applications” IEEE International Multi Topic Conference, 2003. INMIC 2003. 7th International pp. 253 - 257. Dec. 2003

[27] S. Dutta and W. Wolf **"**A flexible parallel architecture adapted to block-matching motion-estimation algorithms,**"** IEEE Trans. on Circuits and Systems for Video Technology, Vo. 6, No. 1, pp. 74 - 86, Feb. 1996.

[28] Hsieh, C.-H.; Lin, T.-P., "VLSI architecture for block-matching motion estimation algorithm,**"** IEEE Trans. on Circuits and Systems for Video Technology, Volume 2, Issue 2, pp.169 - 175 June 1992

[29] Vasudev Bhaskaran, Konstantinos Konstantinides "Image and video compression standards ― algorithms and architectures second edition," Kluwer Academic Publishers sixth printing 2003

[30] Lain E.G. Richardson, "H.264 and MPEG-4 Video Compression ― Video Coding for Next-generation Multimedia," John Wiley & Sons, Ltd. 2003