行政院國家科學委員會補助專題研究計畫成果報告

比例式視訊編碼技術及視訊通訊終端機技術之研究(3/3)

Research in Scalable Video Coding Techniques and Visual Communication Terminal Technologies

計畫編號：NSC 91-2219-E-009-045
執行期限：91 年 8 月 1 日至 92 年 7 月 31 日
主持人：林大衛 交通大學電子工程學系 教授
計畫參與人員：詹益鎬、郭沛昀、陳彥福、林岳賢 交通大學電子工程學系 研究生

摘要

本計畫從事兩方面之研究：其一是比例式視訊編碼法，主要為物件域之比例式編碼相關技術；其二是視訊通訊終端機技術，主要為國際標準視訊編解碼器之實現與相關網際網路視訊通訊終端系統之實作。在物件域之比例式編碼方面，我們主要研究視訊內容分割法，迄今已提出數項技術。模擬結果顯示：萃取得之物件符合人類常態知覺。現亦在繼續研究改進視訊分割技術。在視訊通訊終端機方面，我們過去已發展了一個以個人電腦及數位信號處理器為平台的點對點視訊編碼與傳輸系統，其中的視訊編碼採用 H.263 標準。在本計畫中，我們繼續改進此系統之功能，並採用相似的平台進行 MPEG-4 simple profile 及 fine-granularity scalable 即時編解碼之實作。

關鍵詞：視訊分割、H.263、MPEG-4 視訊編碼、軟體即時編碼、網際網路視訊通訊終端機

Abstract

This project conducts research in two subject areas. The first is scalable video coding, primarily on technology related to object-oriented scalable coding; and the second is visual communication terminal technologies, primarily the implementation of international standard video codecs and the implementation of an internet visual communication terminal system. In object-scalable coding, we mainly research into methods for video content segmentation. Thus far we have proposed several segmentation techniques. Simulation results show that the extracted objects are in accord with common sense of human perception. We are continuing the research and improvement of video segmentation techniques. Concerning visual communication terminal technologies, we have previously developed a point-to-point video coding and transmission system employing personal computers and digital signal processors as the platform. The video codec in the system employs the H.263 standard. In this project, we have continued to improve the functionalities of the system, and we have conducted real-time implementation of MPEG-4 simple profile and fine-granularity scalable encoders employing a similar platform.

Keywords: Video Segmentation, H.263, MPEG-4 Video Coding, Real-time Software Coding, Internet Visual Communication Terminal

目錄 Table of Contents

一、計畫緣由與目的 1
二、結果與討論 3
A. 視訊分割 3
B. 視訊通訊終端系統實作之研究 4
三、參考文獻 6
四、圖表 7
五、計畫成果自評 10
六、附錄 11

一、計畫緣由與目的

視訊通訊領域在近數年來有兩個重要的發展方向，一是視訊壓縮與傳輸之理論與技術上的創新與進步，二是即時及儲存式視訊通訊系統的實用化。前者除可自近數年來相關學術期刊及會議中的論文窺其梗概，亦可由 MPEG-4 標準制定過程中所考慮的各種壓縮與傳輸技術中見其一斑。後者在儲存式系統方面，由 VCD、DVD、及數位動靜態攝影機等數位視訊產品的出現與風行，可得例證；在即時系統方面，則於近數年來，已有不少桌上型視訊會議產品出現。這兩方向的發展，都與數位信號處理硬軟體技術在近年來的迅速發展有關，使得研發人員一方面可做複雜理論與技術的演算與模擬，另方面可將其最終之設計做有效之實現。本計畫即考慮以上兩個方向，一面研究 MPEG-4 相關視訊編碼技術，另一面從事視訊通訊終端系統實作之研究。

在 MPEG-4 相關視訊編碼技術方面，我們知道 MPEG-4 的一項重要創新，是採用物件導向型式(object-oriented)的視訊編碼，以求能對畫面作更有彈性的編碼或組合(composition)等處理，也就是所謂的物件比例能力(object scalability)。因此，一個很重要的課題就是視訊區域的分割(video segmentation)，而此分割結果應該在人類常態視覺的角度看來是有意義的(semantically meaningful)。視訊區域的分割，要考慮到物件的運動，這是與傳統靜態影像分割的一個主要不同。因為物件的運動，一個物件在不同的畫面中可能會改變形狀，也可能會是不完整的出現(例如：畫面的背景，在所有的圖框中均可能被其他物件所部分遮掩)；這些情形就會影響視訊分割方法的設計與其分割品質。視訊區域的分割，在近年來雖有不少研究，但其技術仍未臻理想，而有相當大的改進空間。視訊分割的方法可分兩大基本途徑。其一是將視訊視為一馬可夫隨機過程(或馬可夫隨機場，Markov random field)，以貝氏估測(Bayesian estimation)等最佳化估計方法來做分割。此途徑一個代表性的參考文獻為[1]，其他文獻還有不少，茲不一一列舉。其二是對畫面做直覺的運動與紋理(texture)分析，參考文獻亦多。以其可獲得的結果而言，兩途徑之間難謂孰優孰劣，但前者一般使用疊代(iterative)計算，以直覺分析的結果作為其初始狀況(initial condition)，並需要很高的計算量。此外，亦有將兩途徑做某種結合者，如[2], [3]。我們過去曾對上述兩個基本途徑分別進行研究，但第一途徑的成效未如理想(其部分結果可見[4])，第二途徑則有較佳之結果。本年之計畫專注後者。

在視訊通訊終端系統實作之研究方面，我們研究國際標準視訊編碼法的即時軟體實現及網路視訊通訊系統的實現，其中我們以個人電腦及裝置於其上的數位信號處理器(DSP)插板為實現平台。好的視訊編碼法(壓縮比高而視訊品質好)的運算複雜度是很高的。以 H.263 而論，過去數年間發表的學術論文顯示，一些較快的 DSP 及微處理器(μP)每秒鐘約可處理十幾張至數十張 QCIF 大小的畫面之編解碼。MPEG-4 simple profile 視訊之編解碼速率亦相當。之前我們曾使用個人電腦及 Texas Instruments 的 TMS320C6201 定點數位訊號處理器(裝置在 Blue Wave Systems 的 PCI/C6600 電腦插板上)為平台，實現一個簡單的 H.263 視訊編解碼與網際網路視訊傳輸系統[5], [6]。其後我們亦在持續改進其功能。在本年之計畫中，我們除繼續此一努力外，另採用相同之數位訊號處理器(但改用 Innovative Integration 公司的電腦插板，因 Blue Wave Systems

已轉移事業方向)，從事 MPEG-4 simple profile 及 fine-granularity scalable (FGS)兩種視訊編碼法的即時實現研究。

二、結果與討論

茲分二小節分別討論視訊分割與視訊通訊終端系統實作兩方面之研究。

A. 視訊分割

要達成物件比例式視訊編碼(object-scalable video coding)，一個很重要的課題就是視訊分割，這對自然景像視訊(natural scenes)的編碼而言，尤其為然。由於 MPEG-4 標準中對於物件的定義及視訊分割的方式均無明確的規範，因此就留給研究者極大的餘裕。如前述，我們在此專注於採用直覺的運動與紋理分析來做視訊分割。此類分割方法，通常包括四個基本功能方塊，即紋理分析、運動分析(運動估計)、初始分割、及區域追蹤。我們近幾年來提出了幾個分割方法，其細節頗有不同，但基本架構如是[7]-[10]。圖一呈示我們最近提出的一個方法，其中 Edge Analysis 及 Change Detection 屬紋理分析，Forward Tracking 及 Backward Validation 用到運動分析，Mask Refinement 則完成初始分割與區域追蹤。在視訊分割的研究中，兩大議題是物件邊界的精確認定及運算量的降低。此方法在這兩方面都有特別的設計。以下我們就概略介紹此方法。其詳細討論可參附錄 A，該附錄為一欲發表之論文之初稿。

此方法中的 Edge Detection 目的在於較精確的找到物件的邊界位置。這是因為一般而言，物件的邊界有較大的亮度或色彩變化。我們所用的 Edge Detection 方法為 Canny edge detector。Change Detection 常被用來獲得移動物件的大致位置。我們所使用的 Change Detection 方法與近來若干學術論文所用的方法相似，就是透過 interframe difference 的分析來估計視訊畫面中的攝影機雜訊大小，然後設定一個門檻值，以檢驗 interframe difference。大於此門檻值的畫面位置就算是 changed，所有算是 changed 之像素就形成移動物件位置的一個粗估。

Forward Tracking, Backward Validation, 及 Mask Refinement 是此方法主要創新之所在，其中又尤以 Mask Refinement 為然。Forward Tracking 是用以估計已分割出來的物件的運動並做粗略的追蹤。由於後續的 Backward Validation 及 Mask Refinement 會更精確的確認物件邊界的位置，所以 Forward Tracking 中的運動估計不必非常精確，也因此可以降低其運算量。我們為此設計了一個特別的運動估計法。Forward Tracking 在跟據所估計得的運動作過初步的物件追蹤後，將其結果與 Change Detection 的結果相結合，作為 Forward Tracking 方塊的輸出。Backward Validation 是將 Forward Tracking 的輸出中，屬於 Change Detection 的結果而不屬於初步物件運動追蹤結果的像素，做反向運動估計，並檢測其是否屬於或鄰接於前張畫面中所分割出來的運動物件。若是，則保留，否則刪去。Mask Refinement 的主要精神，是假設 Backward Validation 的結果中，最外緣的 edge 像素，大多應是物件的邊界所在。透過一些 morphological 處理步驟，我們確認這些邊界像素的位置、針對其斷裂不連續的部分做內插以連接之、並填滿物件的內部。

實驗顯示此方法可得相當符合主觀視覺的分割結果。圖二及圖三分別呈示對 Mother and Daughter 及 Salesman 兩個影像序列進行分割的部分結果。

B. 視訊通訊終端系統實作之研究

本部分研究主要係使用個人電腦及其上裝置之數位訊號處理器插板來進行軟體視訊編解碼器及視訊壓縮與網路傳輸終端系統之實作。本項研究分兩子題，一是既有 H.263 編解碼與傳輸系統的改進，二是 MPEG-4 軟體視訊編解碼器的實作。以下分別討論之，但重點在第二項，因其為本部分研究之主要項目。

如前述，我們之前已經完成一個簡單的 H.263 視訊編解碼與網路傳輸系統[5], [6]。該系統結構如圖四所示。傳輸端的個人電腦是 server，接收端的則為 client。接收端不須數位訊號處理器，由個人電腦逕行做視聲訊的解碼與播放。傳輸端的個人電腦，其視訊輸入經個人電腦轉交數位訊號處理器插板做編碼。本年的工作主要為系統功能的改進。為免大幅更動系統架構導致意想不到的問題，我們沿用之前使用的數位訊號處理器插板，即 Blue Wave Systems 的 PCI/C6600，其上裝置 Texas Instruments 的 TMS320C6201 定點數位訊號處理器二顆，工作速率為 200 MHz。但我們的視訊編碼器僅用其中一顆。編碼方法為 H.263，但沒有配置所有的功能，使其簡化以利即時實現。聲訊以外之系統功能，大體上可參[6]。聲訊部分，未做壓縮，僅由個人電腦將之與壓縮後的視訊組成封包，交由網路卡透過 UDP 規約傳出。原始之實現係針對 subQCIF (128x96)之畫面，上年度已改為可處理 QCIF (176x144)畫面，本年則改進為可處理 CIF (352x288)畫面，但編碼速率則成比例下降：subQCIF 每秒約可編 20 張畫面，CIF 則僅 2-3 張。經分析程式，發現其資料輸出入部分可做一些改進，但對程式加速的幫助極有限。其他改進則尚須做更多分析，才能確定其效用。不過以上經驗將有助於新年度(下年度)之 MPEG-4 研究。

在 MPEG-4 軟體視訊編解碼器部分，我們考慮了其 simple profile 及 FGS 編碼器二者，並分別使用一個數位訊號處理器平台(含個人電腦及數位訊號處理器插板)來實現。所用的數位訊號處理器仍是 TMS320C6201，但插板則為 Innovative Integration 公司的 Quatro62。該插板共裝置四顆 TMS320C6201，但我們的二種編碼器實現則各使用二顆。以下分別討論之。

B.1. MPEG-4 Simple Profile 視訊編碼器

MPEG-4 simple profile 視訊編碼器的實現，係以 MoMuSys C 語言軟體為本，加以修改以適數位訊號處理平台之用。主要工作內容可分程式縮小與程式加速兩方面，簡述於下。詳可參附錄 B (會議論文稿)。

在程式縮小方面，由於一顆 TMS320C6201 的內建程式記憶體僅 64 KB，故若無法縮至此大小，則會影響程式設計與執行速率。我們所用的方法有以下幾項：

1. 去除流率控制：MoMuSys 在此使用浮點運算。除去流率控制後可大幅縮小程式並加快執行速率。
2. 使用 macros 來取代簡單的 functions。
3. 用#include 來設定控制參數，而非使用 file reading。
4. 使用 conditional compilation (即#if 與#else 等指令)，使 compiler 依據控制參數之設定來進行程式編譯，以免去無須的程式段落。
5. 去除多物件(objects)及多 layers 之功能：MPEG-4 simple profile 僅一個 object

及一個 layer。

6. 其他如 functions 之合併、無用之函式呼叫之去除、及使用直接運算以取代對函式庫內功能簡單之函式的呼叫。

以上除流率控制之去除外，基本上對 simple profile 功能之完整性並無影響。共減少程式大小約 85%。最後的程式約 108 KB，可放在二顆數位訊號處理器內。

在程式加速方面，我們做了以下幾項更動：

1. DCT 與 IDCT：MoMuSys 使用浮點運算。我們改為定點運算。
2. 改用較快速的 16x16 整數位移運動估計法。
3. 改用較快速的 8x8 整數位移運動估計法。
4. 改用較快速的半點位移運動估計法。
5. 改進為半點位移運動估計而做的像素內插計算程序。
6. 改進運動估計中須做的絕對差和(SAD)計算程序。

整個運算速率的提昇可達 4 倍以上。

圖五顯示個人電腦(host)與兩顆數位訊號處理器(CPU 0 及 CPU 3)如何共同運作。實驗顯示編碼速率約為每秒 6-8 張 QCIF 畫面，其 PSNR 值與原始程式相差不遠。

B.2. MPEG-4 FGS 視訊編碼器

MPEG-4 FGS 視訊編碼器的基本架構如圖六所示。我們採用一個既有的 H.263+編碼器[11]為 base layer，而 enhancement layer 則使用 MoMuSys 軟體修改而得。其中 base layer 佔一顆數位訊號處理器，而 enhancement layer 則使用另一顆；整個系統架構如圖七所示。

我們發現，在 FGS 軟體編碼中，兩種最耗時間的運算是編好之碼的輸出與可變長度編碼(VLC coding)。故除了系統整合外，FGS 軟體實現的主要議題在於程式加速。其方法簡述於下，詳可參附錄 C (會議論文稿)：

1. 選擇適當的 compiler 選項。
2. 改寫程式段落，減少迴圈與指令數目。
3. 促成 software pipelining。
4. 使用 intrinsics (C 語言可呼叫之特殊函式，可有效使用數位訊號處理器資源)。
5. 將短的數據集輯(pack)在一起，使一個 load 或 store 動作可以處理數個數據。
6. 儘量將數據放在數位訊號處理器之內建記憶體，並在使用外部記憶體中之數據之前，使用 DMA 將之先行移入內建記憶體。
7. 使用“restrict” keyword 來告知 compiler 若干變數之間的相關性，以幫助 compiler 將程式平行化而有效使用數位訊號處理器的資源。
8. 使用 macros 來取代一些簡單的 functions。
9. 儘量使用 short 型式之數據來做乘法。

實現的結果，在沒有省略任何 bitplanes 之情形下，視比較基礎之不同，約可加速為原程式的 2.4-2.7 倍，或 6.4-7.6 倍。編碼速率約每秒 11.5-13.5 張 QCIF 畫面。若省略最後二個 bitplanes，則速率可達每秒 17-19 張畫面。

三、參考文獻

[1] A. M. Tekalp, *Digital Video Processing*, Prentice Hall, 1995, ch. 8.

[2] I. Patras, E. A. Hendriks, and R. L. Lagendijk, “Video segmentation by MAP labeling of watershed segments,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 23, no. 3, pp. 326-332, Mar. 2001.

[3] Y. Ysaig and A. Averbuch, “Automatic segmentation of moving objects in video sequences: a region labeling approach,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 7, pp. 597-612, July 2002.

[4] Y. Chou, “Video segmentation via iteratively enhanced spatial-temporal analysis,” M.S. thesis, Dept. Electronics Engineering, National Chiao Tung University, June 2002.

[5] S.-W. Chen and D. W. Lin, “H.263 video codec implementation on a TMS320C62xx digital signal processor,” in *Proc. Workshop on Consumer Electronics*, pp. 1-4, Taipei, Oct. 1999.

[6] J.-R. Wu and D. W. Lin, “DSP-based realtime video encoding and transportation for videoconferencing system,” in *Proc. Workshop on Consumer Electronics*, pp. 181-184, Taipei, Oct. 2000.

[7] Y.-H. Jan and D. W. Lin, “A method for video segmentation based on object tracking,” in *Proc. Int. Symp. Commun.*, paper 10.4, Tainan, Nov. 2001.

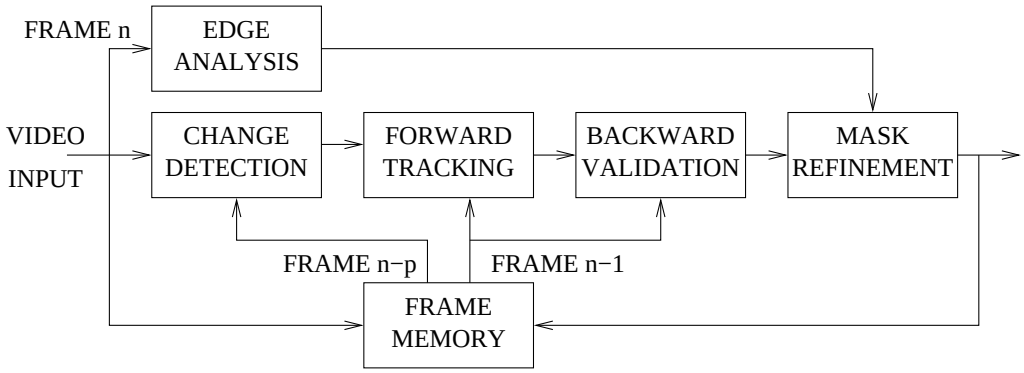
[8] Y.-H. Jan and D. W. Lin, “Image sequence segmentation via heuristic texture analysis and region tracking,” in *SPIE vol. 4671, Visual Commun. Image Processing*, pt. 2, pp. 543-551, Jan. 2002.

[9] Y.-H. Jan and D. W. Lin, “Extraction of video objects by combined motion and edge analysis,” in *Proc. IEEE Int. Symp. Circuits Syst.*, pp. V-677—V-680, May 2002.

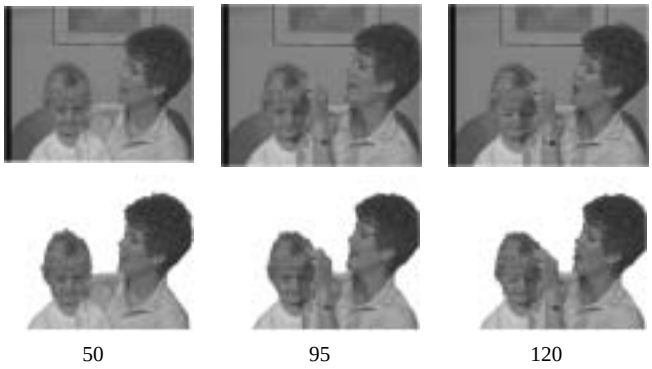
[10] Y.-H. Jan and D. W. Lin, “Automatic video objects segmentation and tracking employing tiered spatio-temporal analysis,” to appear in *Proc. Int. Symp. Commun.*, Taoyuan, Dec. 2003.

[11] M.-L. Woo, “Real-time implementation of H.263+ using TI TMS320C62x,” M.S. thesis, Dept. Electronics Engineering, National Chiao Tung University, June 2000.

四、圖表

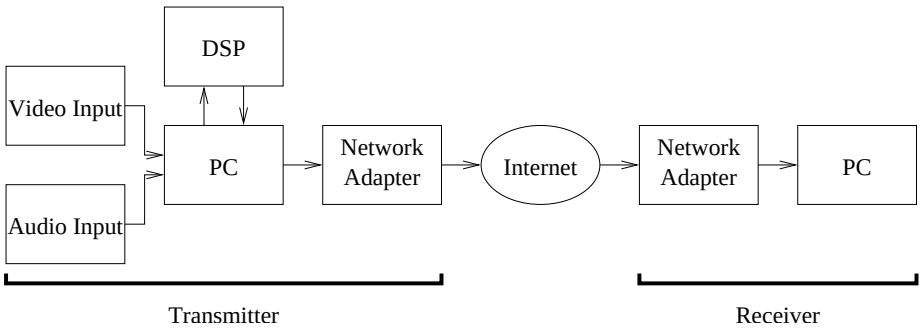


圖一：直覺分析視訊分割法之一

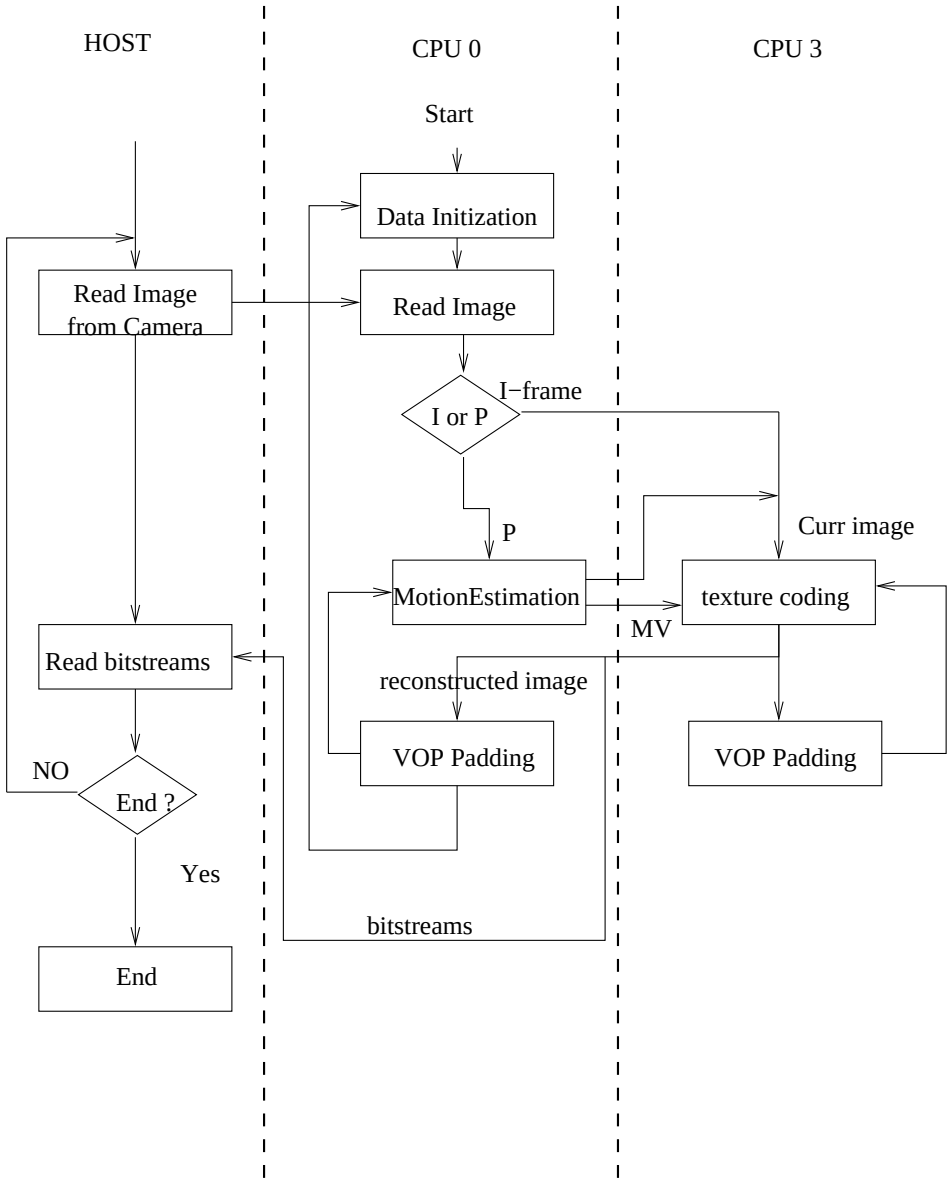


圖二：對 Mother and Daughter 影像序列做分割的部分結果。頂排為原始畫面，中排為分割出之移動前景物件，底排為畫面序號

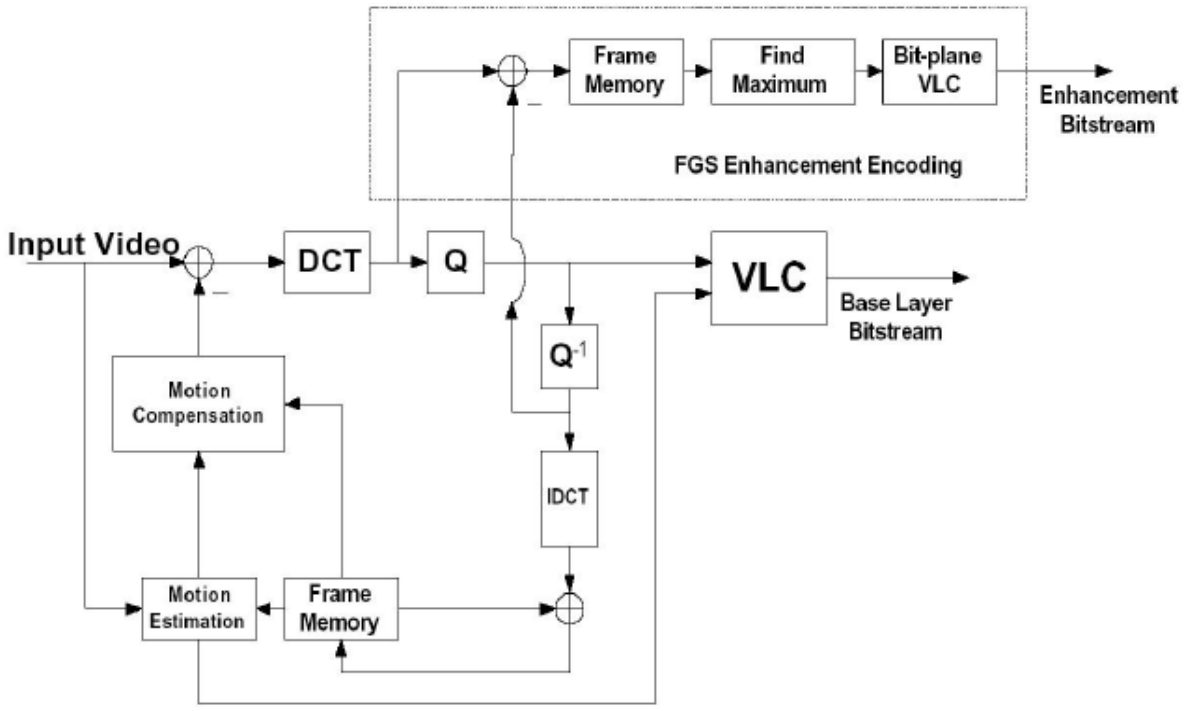
圖三：對 Salesman 影像序列做分割的部分結果。頂排為原始畫面，中排為分割出之移動前景物件，底排為畫面序號



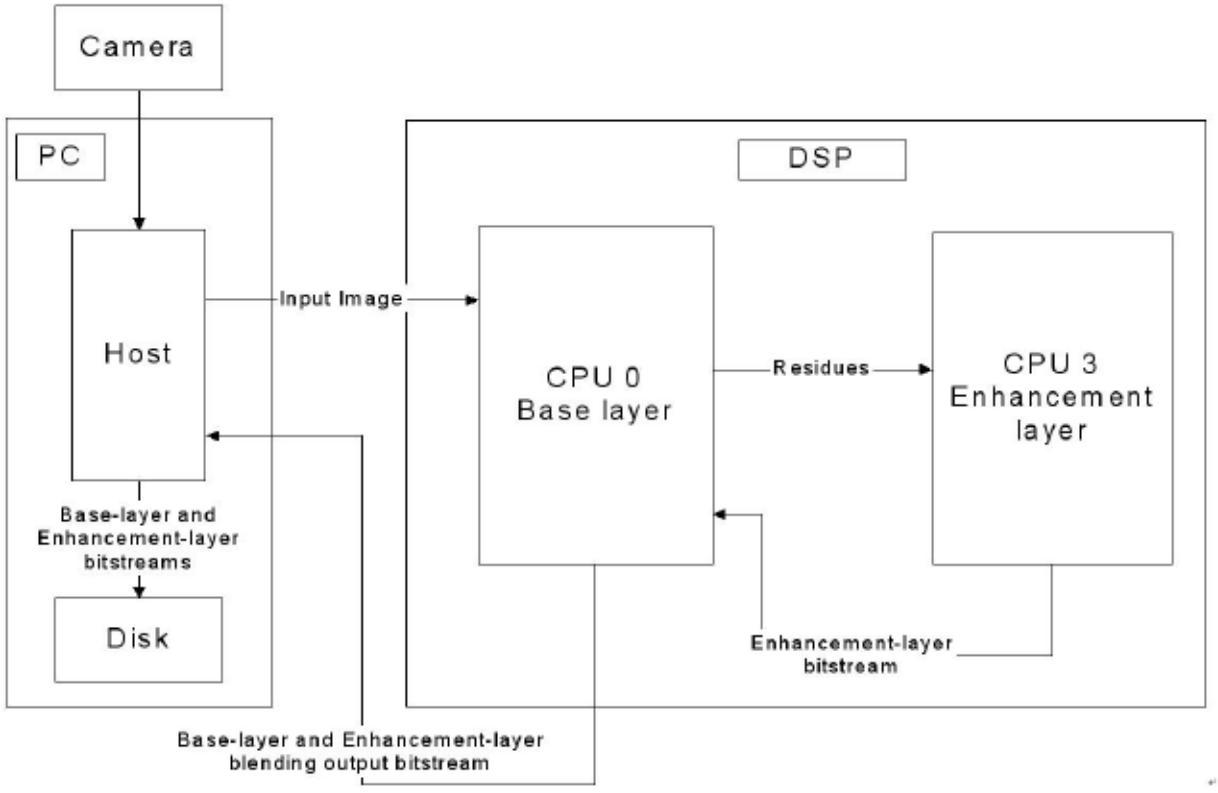
圖四：H.263 視訊編解碼與網路傳輸系統架構



圖五：MPEG-4 simple profile 軟體視訊編碼器之架構



圖六：MPEG-4 FGS 視訊編碼器之基本架構



圖七：MPEG-4 FGS 軟體視訊編碼器之架構

五、計畫成果自評

研究內容與原計畫相符程度：達成計畫名稱所揭示之研究標的，即比例式視訊編碼技術(特別是物件比例式視訊編碼所需之視訊分割技術)及視訊通訊終端機技術(特別是建構於數位訊號處理器上、基於主要國際視訊標準之軟體編碼器)之研究。

達成預期目標情況：本子計畫達成之貢獻形式，含技術上之創新、實驗系統之建立、人才培育。

成果之學術與應用價值等：視訊分割方面之若干成果已發表為會議論文，並在進行期刊與其他會議論文之撰稿與投稿。視訊終端系統實作方面之若干成果也已發表為國內會議論文，或在投稿中；其經驗也將成為我們後續相關研究的參考。以上成果亦皆可供相關業界參考，惟其性質可能不適合做專利申請或技術移轉之用。

綜合評估：本計畫獲得一些具有學術與應用價值的成果，並達人才培育之效。成效良好。

六、附錄

本附錄共含三篇論文稿，如下列：

A. Y.-H. Jan and D. W. Lin, “Automatic video segmentation with novel motion analysis and edge processing for accurate identification of object boundaries” (7 pages).

B. P.-Y. Kuo and D. W. Lin, “Real-time implementation of MPEG-4 video encoder on digital signal processors,” to appear in *Proc. Int. Symp. Commun.*, Taoyuan, Dec. 2003 (6 pages).

C. Y.-F. Chen and D. W. Lin, “Real-time implementation of MPEG-4 fine-granularity-scalable video encoder on digital signal processors” (6 pages).

附錄 A

Automatic Video Segmentation with Novel Motion Analysis and Edge Processing for Accurate Identification of Object Boundaries

Yih-Haw Jan and David W. Lin, Senior Member, IEEE

Abstract — *We consider automatic segmentation of natural video for content-based video applications. A critical problem in this area is accurate identification of object boundaries. We present an algorithm designed to achieve this goal at reasonable complexity. The algorithm employs change detection and motion estimation to identify and approximately track deformable moving objects. The primary novelty of the algorithm consists in a function block termed mask refinement, which does morphological edge-oriented processing to delineate the object boundaries with accuracy, using the edges found from a suitable edge detector. The motion estimation is object-based. For robustness in object tracking, both forward and backward motion estimation are conducted, but the method has relatively low complexity. We present experimental results to illustrate the subjective segmentation performance of the algorithm. The required computational time for the algorithm is seen to be appropriate for real-time desktop or portable multimedia applications* [1].

Index Terms — MPEG-4, object tracking, video segmentation.

I. Introduction

The past ten years have witnessed phenomenal growth in the generation and consumption of digital video in consumer applications, either over the internet or by employing local devices such as digital still cameras, digital video cameras, and DVD players. The recently enacted MPEG-4 standard (which is still evolving) promises to bring additional convenience and functionalities to such digital video, and its impact is only starting to be felt.

Compared to previous video coding standards, a major novelty in the MPEG-4 standard is the introduction of object-based video representation and coding which facilitate differential treatment in coding of different video objects and object-based manipulation of video contents. Many conceivable consumer applications of these capabilities would involve natural video, for example, object-based editing of home video, scene composition for interactive video games, and desktop virtual conference room—videoconferencing system that shows a composited conference room scene of participants' images. To be able to support object-based representation and coding of natural video, one must be able to segment the video into semantic video objects. We consider automatic video segmentation in this work. While there has been much research on this subject in the last few years, the technology is yet to be improved in performance and in complexity for expected user satisfaction.

Techniques for video segmentation can be divided into two basic approaches: probabilistic and heuristic. The probabilistic methods model the video as a random process and attempt to maximize a certain goodness measure in the segmentation process. An example is [1, ch. 8]. The optimization usually requires an iterative procedure and is thus computation-intensive. In contrast, the heuristic methods employ heuristic motion and texture analysis and can be designed to have lower complexity. Some probabilistic methods do probabilistic optimization only for a subset of the parameters characterizing the segmentation, leaving the others obtained through heuristic means. This way, they can have a lower complexity than fully probabilistic methods. Some examples are [2] and [3]. We consider the purely heuristic approach.

The heuristic video segmentation methods can be further divided into two main categories: those starting with an initial spatial segmentation and those starting with an initial motion-based analysis or segmentation. Either way, both spatial and temporal (motion) analyses are needed for subsequent object tracking. The proposed algorithm of this paper falls in the latter category, as our experience indicates that this approach can yield relatively good results at reasonable complexity. Before presenting the proposed algorithm, we briefly review some reported research concerning this category of methods.

Chen and Shirai [4] and Neri *et al.* [5] are two examples of motion-based segmentation. However, accurate identification of object boundaries is not considered and the segmented region boundaries can be quite far from actual object boundaries. Since object boundaries in an image are often characterized by high intensity variation, edges (high-gradient image sections) provide important cues to object contours. A few algorithms try to identify object boundaries by analyzing the edges found in the image regions that show significant motion [6]-[8]. Still, accurate identification of object boundaries remains an issue to be fully resolved in video segmentation. This is the case especially when the object boundaries are grossly nonconvex.

The key technological aim of this work is, therefore, novel segmentation techniques that can identify object boundaries accurately at acceptable computational complexity. This paper is organized as follows. Section II describes the proposed segmentation algorithm with its novelties. Section III presents some experimental results. And Section IV draws the

[1] This work was supported by the National Science Council of R.O.C. under Grant No. NSC 91-2219-E-009-045 and by Lee and MTI Center for Networking Research at National Chiao Tung University.

The authors are with Department of Electronics Engineering and Center for Telecommunications Research, National Chiao Tung University, Hsinchu, Taiwan 30010, R.O.C. (e-mails: yhjan.ee86g@ nctu.edu.tw, dwlin@mail.nctu.edu.tw).

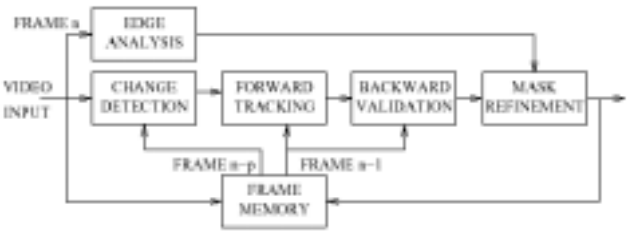


Fig. 1. Structure of the proposed algorithm.

conclusion.

II. The Proposed Algorithm

Figure 1 shows the overall structure of the proposed algorithm. Roughly speaking, semantic objects are detected initially with "change detection," tracked in time with motion-compensated "forward tracking" and "backward validation" that allow object shape changes, and their boundaries delineated more accurately with "mask refinement." The block "edge analysis" provides useful information for the last function. The primary novelties of the algorithm consist in the mask refinement function, as well as the motion-based forward tracking and backward validation.

In video signal analysis and segmentation, motion information is helpful for tracking of moving objects. However, conventional motion estimation methods are usually very computation-intensive and need not yield reliable motion information. As a result, one often would like to minimize its use or its complexity. By simply detecting the changed areas, change detection is a favored low-complexity mechanism to roughly identify image regions that contain moving objects [7], [8]. However, if an originally moving object comes to a standstill, change detection will lose track of the object unless a long-term memory is provided, such as in [9]. In this work, we choose to develop a low-complexity, object-based motion estimation technique for convenience of object tracking. Nevertheless, change detection is also employed to identify new moving objects and to help capture fully the possible object shape changes with time.

The edge analysis block in the algorithm employs the well-known Canny detector [10]. Below we describe the remaining functional blocks in greater detail, namely, change detection, forward tracking, backward validation, and mask refinement.

A. Change Detection

Change detection is a frequently employed technique to roughly locate the moving regions in consecutive video frames [5], [11]-[13]. It requires statistical modeling of the background noise (due at least in part to camera noise), and it typically determines whether a pixel is changed (e.g., moving) or unchanged by comparing the frame difference with a threshold calculated from the statistical model.

Let the video have stationary background. Similar to others' work, we assume the background part of the frame difference follows a zero-mean Gaussian distribution

$$p(\delta_n(x,y)\,|\,H_0) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\delta_n^2(x,y)/2\sigma^2}, \qquad (1)$$

where H_0 denotes the null hypothesis that the pixel at location (x, y) is unchanged, n is time index, $\delta_n(x, y)$ is the difference at pixel (x, y) between frames I_n and I_{n-p}, and σ^2 is equal to twice the camera noise variance. In slow-motion video (such as conference video), we may let p>1 to capture the moving objects more easily.

For robustness, instead of making the changed-unchanged decision based on the single-pixel frame difference $\delta_n(x, y)$, we base our decision on its mean-square value in a window as in [11]. For independent Gaussian random variable, their mean-square value obeys a χ^2 distribution. The decision threshold TH_α is parametrized on a significance level α evaluated from

$$\alpha = p\{V(x,y) > TH_\alpha \,|\, H_0\} \qquad (2)$$

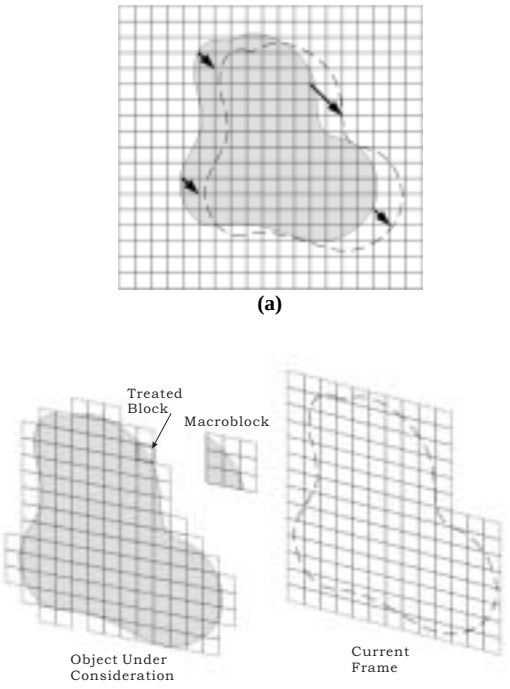
where $V(x, y)$ is the mean-square frame difference in the test window, normalized by dividing it by the variance of the background noise. The background noise variance is evaluated using the method in [14]. Experimentally, we find that an observation window of size 5×5 or 7×7 and a significance level α between 10^{-2} and 10^{-3} lead to good results. For convenience, the set of changed pixels is denoted CD_n.

By itself, change detection does not fully delineate a moving object. Normally, some background pixels will be declared as changed. On the other hand, a moving object with smooth interior may have many of its pixels declared as unchanged. In our algorithm, change detection is used, in association with motion estimation, for deformable object tracking. It is also used for initial detection of new moving objects, especially in the first two frames. The tasks of accurate boundary delineation and filling-in of object interior are relegated to the mask refinement function.

B. Forward Tracking

A main purpose of the forward tracking function in the proposed algorithm is to find the footprint of each object of the previous frame (say I_{n-1}) in the current frame (say I_n). Some inaccuracy is tolerated, because the subsequent backward validation and mask refinement will localize the object boundary more precisely. By integrating the footprint with change detection's output, we can accommodate new appearance of moving objects and enhance the ability to deal with object shape changes.

Key to the forward tracking function is a low-complexity motion estimation method, which is illustrated in Fig. 2. Fig.



(a)

(b)

Fig. 2. Object-based motion estimation. (a) An object under consideration (shaded region) in the previous frame has moved to a different position in the current frame (dashed contour). Each small square is $BW \times BW$ in size. (b) Illustrating the idea that motion estimation is carried out on the 3×3 macroblock centered at the treated block.

2(a) shows that an object under consideration in frame I_{n-1} has moved to a different position in I_n. The object is divided into square blocks of size $BW \times BW$ pixels. (We have used $BW = 4$ in this work.) To save computation, we first consider all blocks on the object boundary. For each such block, we perform block-matching motion estimation for a "macroblock" of 3×3 blocks centered at it, as illustrated in Fig. 2(b). For simplicity, we only consider translational motion. However, the motion vectors may point out of the frame as in the unrestricted motion vector (UMV) mode of the ITU-T H.263 standard. The required out-of-frame pixels are obtained by repeating the pixel values at frame boundaries. The obtained motion vector is assigned to the pixels of the treated block. We then proceed inwards from the boundary blocks until all the interior blocks are treated. In each step, we treat one "layer" of blocks that are immediately inside the blocks that were treated in the previous step. For each of these blocks, we perform macroblock-based motion estimation similar to the case of boundary blocks, but the candidate motion vectors are highly limited: only the motion vectors of its treated neighbors and the zero vector are tested, and the best is taken.

To continue, the mask (i.e., pixel map) of each object in I_{n-1} is projected forwardly onto I_n using the motion vectors obtained above. For convenience, let $O_{i,n-1}$, $i = 1,2,3...,S$, denote the ith object in I_{n-1} and let $P_{i,n}$ denote the corresponding projected footprints. The forward-tracked mask $PCD_{i,n}^F$ of each object is obtained by taking the union of $P_{i,n}$ and CD_n, retaining the largest connected set of pixels, and fill up all small isolated "holes" that may show up in the pixel map due to slight difference in the estimated motion of nearby blocks.

C. Backward Validation

Backward validation is conducted to trim each mask $PCD_{i,n}^F$ for better accuracy, since the forward motion estimation need not be very accurate and since CD_n may contain contributions from more than one object. For this, backward motion estimation from I_n to I_{n-1} is performed for the pixels in $PCD_{i,n}^F$ that are not in $P_{i,n}$. The method is similar to the forward motion estimation described above. Only those pixels whose backward motion-compensated projections lie inside or touch $O_{i,n-1}$ are retained. As in forward tracking, if any small isolated holes show up in the trimmed $PCD_{i,n}^F$, they are filled up. The resulting set of validated pixels is denoted $PCD_{i,n}^B$ and referred to as the *rough mask*.

In typical videophone scenes, the pixels needing backward validation are relatively few. Hence the required backward motion estimation does not add exorbitant complexity.

D. Mask Refinement

Due to the nature of our change detection and motion estimation methods, the rough mask $PCD_{i,n}^B$ may contain background pixels beyond the actual object boundary. Besides, holes may still appear in the object's interior where there are not. The mask refinement function attempts to rectify these problems.

Since, as noted previously, object boundaries are often characterized by high intensity variation, our mask refinement procedure does edge-based processing. For natural video, typical edge detectors (such as the Canny detector that we use) often do not obtain closed or connected contours at object boundaries. Therefore, in addition to finding the correct edges that mark object boundaries, a way to link up the edge gaps must be devised. An equivalent problem is the exact identification of the interior mask for the object.

We illustrate the procedure using the arbitrary rough mask example shown in Fig. 3(a) (the gray pixels). The mask is highly nonconvex on the outer side to signify that a distinguishing feature of the proposed method, in comparison to some other work, is the ability to handle highly nonconvex object shapes well. Assume that an object is located

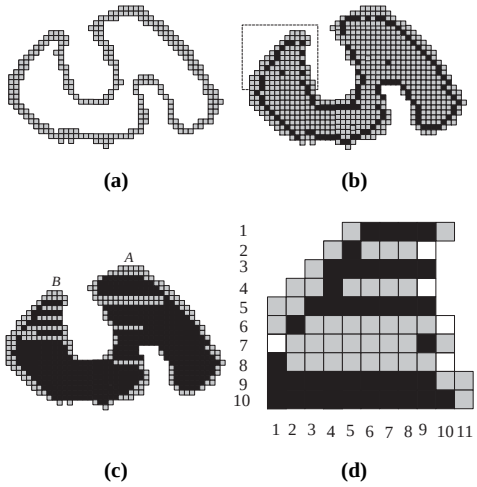


Fig. 3. Illustration of the mask refinement method. (a) An arbitrary example of the rough mask. (b) FG, with edge pixels in black. (c) Row map from edge-scan (black pixels). (d) Zoomed-in plot of upper-left part to illustrate some points.

completely within the rough mask. In many cases, a rough mask may enclose a half-open area. A common example is a person's image in a videophone scene, which may show the person's upper body butted to the bottom side of the frame and result in a rough mask that is half open in the lower side. In such cases, the following procedure is modified slightly to take care of the open side, much similar to [8]. The details are omitted.

To start, we fill in the interior of the rough mask by orthogonal scans. Specifically, a horizontal scan is performed over each row in the rough mask to fill in the space between the leftmost and the rightmost pixels. Corresponding vertical scans are performed to obtain another result. The union of the two results is taken, similar to [15]. The above closes any inner "holes," but the obtained union map may be larger than the actual object. (For the example of Fig. 3(a), it is easily seen that this will be the case.) To cut away the overgrowth, we erode the union map from outside in, so that every outer pixel that is not in the rough mask nor an eight-connected neighbor of the rough mask is removed. This will also stop any single-pixel "cracks" on the outer side of the rough mask, but will in general cause the rough mask to grow by one pixel around. We thus further examine the outermost "slice" of pixels and remove all that are not in the original rough mask. The result, denoted FG, is a solid area enclosed by the rough mask, with single-pixel cracks filled up.

Now we consider the edge pixels in FG. (Fig. 3(b) gives an example, where the edge pixels are shown in black.) We assume that most edge pixels close to FG's perimeter define the object boundary. The problem is to identify and connect them properly. For this, we first examine each connected horizontal and vertical line segment in FG. For each horizontal line segment, if the outermost edge pixels are close to the ends of the segment (say within several pixels of an end and sufficiently distant from the segment's center), then the space between them is filled. Call the result a row map. A similar operation is carried out vertically to result in a column map. For example, Fig. 3(c) shows the row map (black pixels) obtained from Fig. 3(b). For ease of reference, the procedure is termed "edge scan."

The first use of the row and the column maps is to trim away some small overshoots in the FG. For this, we do a top-down scan of the row map and mark all areas that appear like caps. (In Fig. 3(c), A and B are such caps.) Experience shows that such caps often contain pixels outside the desired object and may even contain edges in the background. Thus for each cap, we examine the distance between the leftmost and the rightmost edge pixels in each line segment, from top down. If the distance is small compared to the length of the segment, say under 35%, then the line segment is deleted from both maps and the FG. This continues until we encounter a line segment wherein such distance is not small. Likewise, we conduct a bottom-up scan on the row map and process the caps on the bottom side. Corresponding left-to-right and right-to-left scans are also conducted on the column map and the caps in these directions are processed similarly. The above is termed "cap trimming" for convenience.

Next, we tighten up the footprint of FG in preparation for interpolation between the assumed boundary edges where gaps exist. (Accordingly, we shall refer to the process as "footprint tightening.") Specifically, we take the union of the cap-trimmed row and column maps and delete from FG any pixels that are not at least eight-connected to this union. The retaining of eight-connected neighbors leaves some additional leeway in placing the interpolated edge pixels. This is illustrated in Fig. 3(d), which shows a zoomed-in plot of a part in the upper left of Fig. 3(c). In the figure, black color denotes a pixel in the cap-trimmed row map, gray color a pixel in the reduced FG but not in the row map, and white color an end-of-line pixel on whose side a valid edge is missing. The white pixel in row 7, column 1 is one that would be deleted if we did not retain the eight-connected neighbors as above. On the other hand, from Fig. 3(d) it can also been seen that, by retaining these eight-connected neighbors, for a row (resp. a column) that contains a valid edge on one side, the edge pixel may be up to two pixels away from the end pixel of the reduced FG horizontally (resp. vertically) on this side. In preparation for subsequent processing, we now delete the extraneous pixels on the outer side of the valid edge pixels. This can be achieved by examining each connected horizontal and vertical line segment in the reduced FG. If a valid edge pixel is at most two pixels inward from the end of the line, then delete the one or two pixels on the outer side of the edge pixel.

Now, to interpolate and fill in the "missing" edge pixels in boundary edge gaps, we use an iterative procedure designed on a minimum-distance principle. In each iteration, we examine the footprint-tightened FG in the horizontal and in the vertical directions alternately. In the horizontal direction, we examine the left side of all the connected horizontal line

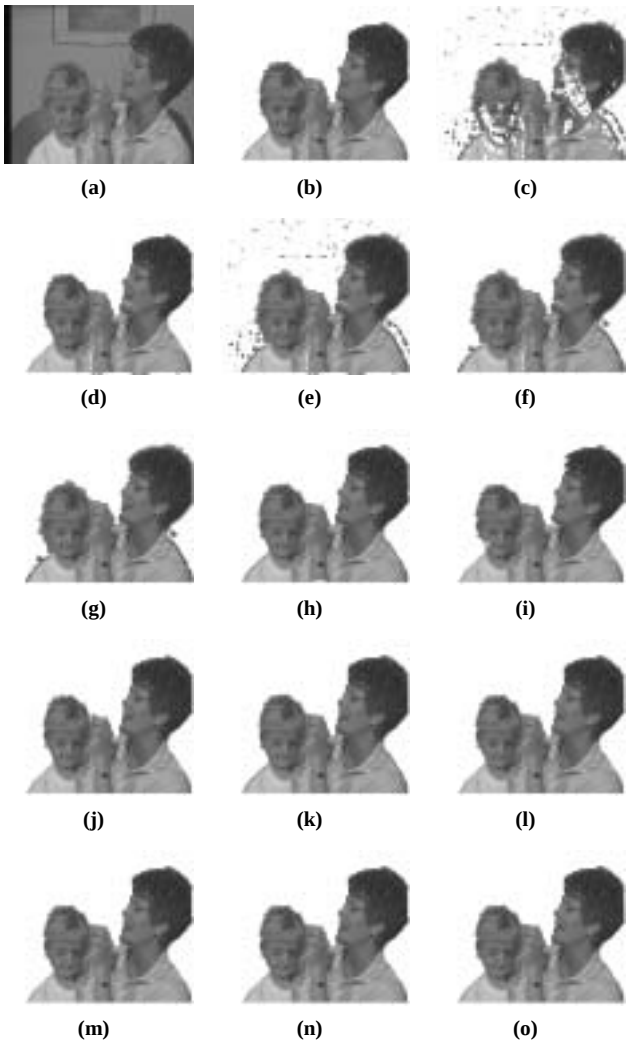


Fig. 4. Illustration of algorithm steps using frame 100 of the Mother and Daughter sequence as example. (a) Original frame 100. (b) Foreground object $O_{1,99}$ segmented from frame 99. (c) Change detection result CD_{100}. (d) Projected footprint $P_{1,100}$ of $O_{1,99}$ in frame 100. (e) Union of $P_{1,100}$ and CD_{100}. (f) Largest connected pixel set in the union. (g) Final forward-tracked mask $PCD^{F}_{1,100}$. (h) Rough mask $PCD^{B}_{1,100}$. (i) Row map from edge scan. (j) Column map from edge scan. (k) Cap-trimmed FG. (l) Footprint-tightened result. (m) Result of first iteration of edge filling. (n) Result of second iteration of edge filling. (o) Final extracted object $O_{1,100}$.

segments in sequence and then the right side in sequence. In the vertical direction, we examine the top side of all the connected vertical line segments and then the bottom side, both also in sequence.

In left-side processing, if a horizontal line segment, say H, misses the left edge, then we step rightwards as well as leftwards from its left end, where the left-end pixel is denoted Cx for convenience. At each step, we check if an edge pixel exists in the same column in any of the line segments above. If so, the distance with the closest of them is noted, where the distance is measured in number of pixels Cx has to step through to reach the edge pixel. However, if any pixel en route is not in the boundary-tightened FG, then that pixel is given a distance measure of 10 instead of 1. The column position of the nearest edge pixel above, say Cu, is recorded. In the same way, the nearest edge pixel below, say Cd, is found and its column position also recorded. Then the average column position between Cu and Cd is obtained, and the pixel in H at that position is denoted Cm. Now we step from Cx towards Cm and delete the pixels as we go, until we encounter an edge pixel or we reach Cm. Let this pixel where pixel deletion stops be denoted Ce. In some cases, there may not exist a valid Cu or Cd. In these cases, Cm is taken to be at the average column position between Cx and the valid Cu or Cd. If neither a valid Cu nor a valid Cd exists, then Cm is undefined and the iterative procedure skips to the next step. Ce is considered equivalent to a left edge pixel for the remaining line segments to be processed in the present iteration. If it is indeed an edge pixel, then it is considered the true left edge of the present line segment and treated as other existing left edge pixels in later iterations.

The processing at the right, top, and bottom sides is similar. The iteration continues until no more deletion of pixels is possible. For convenience in reference, the procedure will be termed “edge filling.”

As an example, consider row 7 in Fig. 3(d), which misses the left edge. The left-side processing would place the filled-in edge pixel in column 2. For rows 2, 4, 6, and 8 in Fig. 3(d), all missing the right edge, the right-side processing would place the filled-in edge pixels in column 9 for them all.

Lastly, we fine-tune the object boundary slightly. Specifically, we consider the edge-filled row and column maps. We shrink the former by one pixel on the left and on the right, and the latter on the top and on the bottom. Their intersection is taken. Pixels in the edge-filled FG that are not in the footprint of the intersection or are not eight-connected neighbors of the pixels therein are deleted. The resulting pixel mask defines the extracted object.

III. Experimental Results

To illustrate the working and the performance of the proposed video segmentation algorithm, we present some results on the common CIF test sequences Mother and Daughter, Salesman, and Akiyo. (Experience shows that the Salesman sequence is a quite difficult sequence for video segmentation.) We first give a walk-through of some key algorithm steps. Then we present some segmentation results. And finally, we provide some data regarding algorithm speed.

For a walk-through of the algorithm, Fig. 4(a) shows the original frame 100 of the Mother and Daughter sequence. Fig. 4(b) shows the foreground object $O_{1,99}$ segmented from frame 99. Fig. 4(c) shows the change detection result CD_{100}

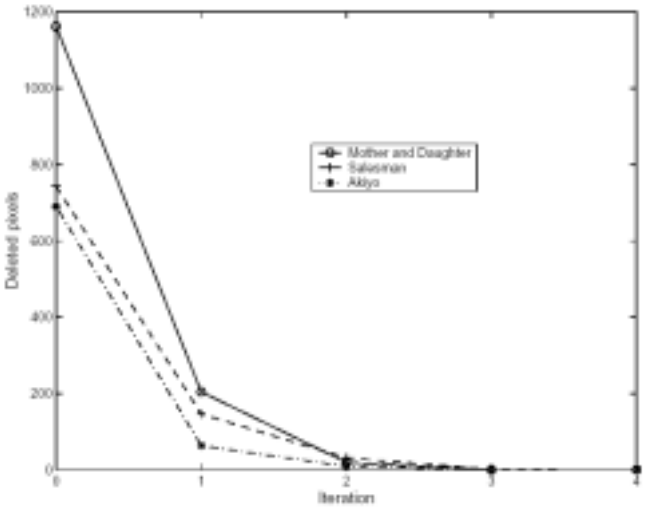


Fig. 5. Convergence behavior of edge filling for the first frame.

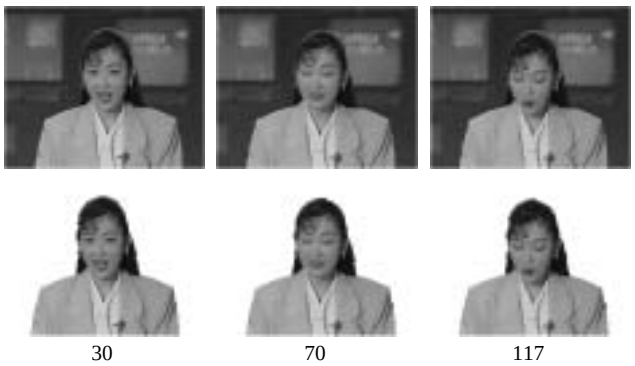
50 95 120

Fig. 6. Segmented foreground object of Mother and Daughter. Top row: original frames; middle row: segmentation results; bottom row: frame numbers.

10 20 30

Fig. 7. Segmented foreground object of Salesman. Top row: original frames; middle row: segmentation results; bottom row: frame numbers.

obtained from analyzing frames 100 and 97 (i.e., we have let p = 3). Figs. 4(d)-(g) illustrate the process of forward tracking. We have employed the well-known three-step motion estimation algorithm [16, Sec. 11.3] for the boundary blocks. Because the sequences are of the videophone type, the search



30 70 117

Fig. 8. Segmented foreground object of Akiyo. Top row: original frames; middle row: segmentation results; bottom row: frame numbers.

range is ±7 pixels. Fig. 4(d) shows the projected footprint $P_{1,100}$ of frame 99's foreground object in frame 100. Some small isolated holes have shown up due to slight difference in the estimated motion of nearby blocks, as noted before. Fig. 4(e) shows the union of CD_{100} and $P_{1,100}$. Fig. 4(f) shows the largest connected pixel set in the union. And Fig. 4(g) shows the final forward-tracked mask $PCD^{F}_{1,100}$ after filling up the small isolated holes. Fig. 4(h) shows the rough mask $PCD^{B}_{1,100}$ after backward validation.

The remaining pictures in Fig. 4 illustrate the process of mask refinement. Figs. 4(i) and (j) show the row map and the column map, respectively, from edge scan. Fig. 4(k) shows the cap-trimmed FG. Fig. 4(l) shows the footprint-tightened result. Fig. 4(m) and (n) show the results after the first and the second iterations of edge filling, respectively. And Fig. 4(o) shows the final extracted object $O_{1,100}$.

To further illustrate the convergence behavior of the edge filling procedure, we show in Fig. 5 the number of deleted pixels in each iteration for the first processed frame of each test sequence. Note that the procedure takes only a small number of iterations to converge.

Figs. 6, 7, and 8 show some segmentation results for the Mother and Daughter, the Salesman, and the Akiyo sequences. We see that the object boundaries are quite accurately identified.

Concerning the algorithm speed, we have used as test platform a personal computer with 1.4-GHz Pentium IV CPU. The program is not optimized. Excluding disk access time, the result is 26.9, 19.7, and 22.3 ms per frame for Mother and Daughter, Salesman, and Akiyo, respectively. A very large portion of the time is spent on forward and backward motion estimation, which amounts to 13.7, 10.6, and 11.6 ms, respectively, in the above cases. With suitable optimization of the program, it should be able to run significantly faster. Thus the algorithm is suitable for real-time desktop or portable multimedia and videoconferencing applications.

IV. CONCLUSION

We presented an algorithm for automatic object extraction and tracking for nature video scenes. The algorithm makes accurate determination of boundaries of moving objects with novel edge-based morphological processing. And it can handle grossly nonconvex object shapes, which are commonplace in typical natural video. A novel, low-complexity motion estimation technique was also designed to aid robust object tracking. Experimental results show good subjective performance. The required computational time for the algorithm is also seen to be appropriate for real-time desktop or portable multimedia applications.

REFERENCES

[1] A. M. Tekalp, *Digital Video Processing*. Prentice Hall, 1995.

[2] I. Patras, E. A. Hendriks, and R. L. Lagendijk, "Video segmentation by MAP labeling of watershed segments," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 23, no. 3, pp. 326-332, Mar. 2001.

[3] Y. Tsaig and A. Averbuch, "Automatic segmentation of moving objects in video sequences: a region labeling approach," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 7, pp. 597-612, July 2002.

[4] H.-J. Chen and Y. Shirai, "Segmentation based on accumulative observation of apparent motion in long image sequences," *IEICE Trans. Inf. & Syst*. vol. E77-D, no. 6, pp. 694-704, June 1994.

[5] A. Neri, S. Colonnese, G. Russo, and P. Talone, "Automatic moving object and background separation," *Signal Processing*, vol. 66, no. 2, pp. 219-232, Apr. 1998.

[6] T. Meier and K. N. Ngan, "Segmentation and tracking of moving objects for content-based video coding," *IEE Proc.-Vis. Image Signal Process.*, vol. 146, no. 3, pp. 144-150, June 1999.

[7] T. Meier and K. N. Ngan, "Video segmentation for content-based coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, no. 8, pp. 1190-1203, Dec. 1999.

[8] C. Kim and J.-N. Hwang, "Fast and automatic video object segmentation and tracking for content-based applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 2, pp. 122-129, Feb. 2002.

[9] S.-Y. Chien, S.-Y. Ma, and L.-G. Chen, "Efficient moving object segmentation algorithm using background registration technique," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 7, pp. 577-586, July 2002.

[10] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 8, no. 6, pp. 679-698, Nov. 1986.

[11] T Aach, A. Kaup, and R. Mester, "Statistical model-based change detection in moving video," *Signal Processing*, vol. 31, no. 2, pp. 165-180, Mar. 1993.

[12] M. Kim, J. G. Choi, D. Kim, H. Lee, M. H. Lee, C. Ahn, and Y.-S. Ho, "A VOP generation tool: automatic segmentation of moving objects in image sequences based on spatio-temporal information," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, no. 8, pp. 1216-1226, Dec. 1999.

[13] M. Kim and J. Kim, "Moving video object segmentation using statistical hypothesis testing," *Electron. Lett.*, vol. 36, no. 2, pp. 128-129, Jan. 2000.

[14] Y.-H. Jan and D. W. Lin, "Extraction of video objects by combined motion and edge analysis," in *Proc. IEEE Int. Symp. Circuits Syst.*, pp. V-677—V-680, May 2002.

[15] T. Meier, and K. N. Ngan, "Automatic segmentation of moving objects for video object plane generation," *IEEE Trans. Circuits Syst. Video Tehcnol.*, vol. 8, no. 5, pp. 525-538, Sep. 1998.

[16] Y. Q. Shi and H. Sun, *Image and Video Compression for Multimedia Engineering*. New York: CRC Press, 2000.

Yih-Haw Jan received the B.S. and M.S. degrees in Electrical Engineering from Chung Hua Polytechnic Institute, Hsinchu, Taiwan, ROC., in 1995 and 1997, respectively. He currently is working towards the Ph.D. degree at the Department of Electronics Engineering of National Chiao Tung University, Hsinchu, Taiwan, R.O.C. His fields of research include video segmentation and digital communication systems.

David W. Lin received the B.S. from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1975, and the M.S. and Ph.D. degrees from the University of Southern California, Los Angles, U.S.A., in 1979 and 1981, respectively, all in electrical engineering.

He was with Bell Laboratories during 1981-1983, and with Bellcore during 1984-1990 and again during 1993-1994. Since 1990, he has been a Professor in the Department of Electronics Engineering and the Center for Telecommunications Research, National Chiao Tung University. He has conducted research in digital adaptive filtering and telephone echo cancellation, digital subscriber line and coaxial network transmission, speech and video coding, and wireless communication. His research interests include various topics in communication engineering and signal processing.

附錄 B

Real-Time Implementation of MPEG-4 Video Encoder on Digital Signal Processors

Pei-Yun Kuo and David W. Lin

Department of Electronics Engineering and Center for Telecommunications Research
National Chiao Tung University
Hsinchu, Taiwan 30010, R.O.C.
E-mails: peggy.ee90g@nctu.edu.tw, dwlin@mail.nctu.edu.tw

Abstract

The MPEG-4 standard specified by ISO/IEC MPEG is a very efficient coding standard for multimedia data. In this work, we use digital signal processors (DSPs) to implement a real-time simple profile MPEG-4 video encoder. The platform employed is Innovative Integration's Quatro62 personal computer card, which houses four chips of Texas Instruments' TMS320C6201. It turns out that we need to use two chips to obtain an efficient encoder implementation. We do some trimming of the MoMuSys code to fit the program into the limited DSP on-chip memories. To speed up the execution, we modify the DCT/IDCT and the motion estimation algorithm. We also modify some program sections to help the compiler parallelize the compiled code for efficient utilization of the parallel functional units of the DSP during execution. Currently, the implementation can encode about 6 QCIF frames per second, but with uneven loads between the two DSPs. Compared to the original code, the speed-up is about four times, the code size is reduced by about 80%, and the loss is in PSNR (peak signal-to-noise ratio) is less than 0.5 dB.

1. Introduction

We consider implementation of an MPEG-4 simple profile video encoder on digital signal processors. The MPEG-4 video standard was originally for coding with high efficiency, very low bit-rate and universal access.

Figure 1 shows the detailed structure of video objects encoder. The key elements in simple profile are motion coder, discrete cosine transformation (DCT), quantization, DC predition, and variable-length coding (VLC). The motion coder consists of a motion estimator, motion compensator, previous/next VOPs Store and motion vector (MV) Predictor and Coder. It is used to reduce temporal redundancy. In the simple profile, the motion vectors include motion vector for 16×16 macroblock and four motion vectors for 8×8 blocks, and are specified to half-pixel accuracy.

This work was supported by the National Science Council of R.O.C. under grant no. NSC 91-2219-E-009-045.

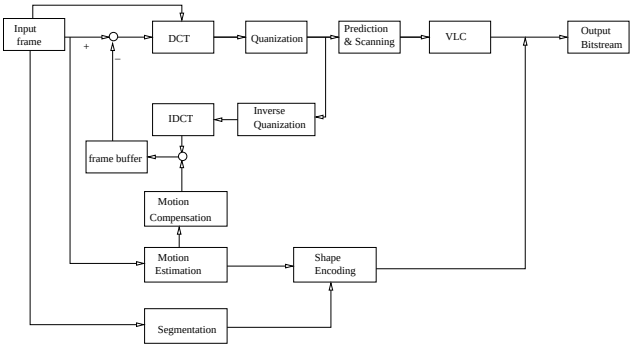


Fig. 1: Detailed structure of video objects encoder.

The purpose of DCT, quantization and DC predition is to reduce spatial redundancy for both intra and inter frames. The VLC technique reduces syntax redundancy.

The environment of DSP involves a host PC, DSP board and DSP chips on the board. The DSP chips are Texas Instruments (TI)'s TMS320C6201. The TMS320C62x is fixed-point DSP with eight operation units. The DSP board we use is Innovative Integration (II)'s Quatro6x. It is a PCI bus compatible DSP card housing four TI TMS320C62x processors in a symmetric multiprocessing relationship with high bandwidth inter-processor communication links.

The implementation is based on the code from [4]. It is a public source for MPEG-4 main profile encoding and decoding. Table 1 shows the functionalities that the MoMuSys software supports. However, to implement an MPEG-4 encoder on the DSP chip, the main profile appears too complicated on first attempt. Therefore, we implement the simple profile only.

The reduced MoMuSys source from [5] is a MPEG-4 source code with simple profile functions. The code size of the reduced source is decreased from 1.8 Mbytes to 740 Kbytes in CCS.

In order to implement the MPEG-4 video encoder on TMS320C62x, we reduce the code size and alter some functions to fit the limited program memory, 64 Kbytes. The altering of some functions, such as DCT/IDCT and motion estimation, also facilitates better parallelism after

Table 1: Functionalities of MoMuSys

Visual Tools	Simple	Main	MoMuSys
Basic *-I-VOP* *-P-VOP* *-AC/DC Prediction* *-4-MV,Unrestricted MV*	V	V	V
Error resilience	V	V	V
Short Header	V	V	V
B-VOP		V	V
Method 1/Method 2 Quantization		V	V
P-VOP based temporal scalability *-Rectangular* *-Arbitrary Shape*		V	V
Binary Shape		V	V
Grey Shape		V	V
rate control		V	V

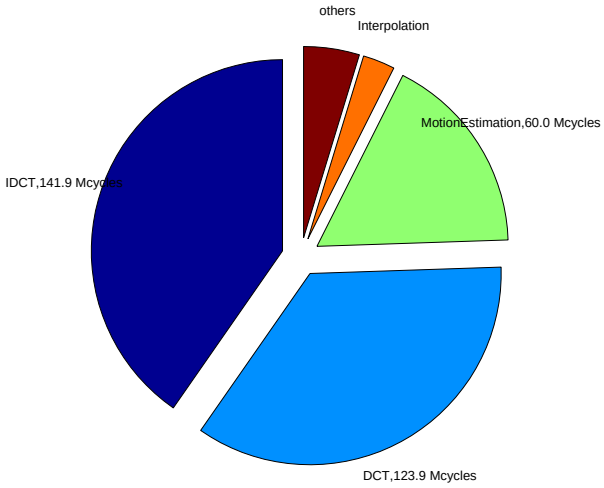


Fig. 2: Comparison of execution time in MoMuSys.

compilation. Therefore, with this features, we can achieve real-time performance.

In the section 2, we discribe the method of the code size reduction and code acceleration. The overall system design of the MPEG-4 encoder, which uses two DSPs working together with host PC, is described in section 3. The paper also presents experimental results on the speed and the rate-distortion performance of the implementation in the section 4. Finally, sention 5 contains the conclusion.

2. MPEG-4 Video Encoder Implementation and Optimization for DSP

2.1. Code Size Reduction

For the limited program memory, we use some methods to reduce the code size.

- Remove the rate control

  The rate control defined in MoMusys is composed of floating-point operations which is difficult to digital signal processor. For the code size and code speed, the rate control is removed first.

- Using macros to replace simple functions

  The C code in MoMuSys has many functions with just one or two instructions. Therefore, we can use the "define" macros to replace the simple function.

- changing method of setting control parameters

  The initial method of setting the control parameters is reading them from the file. we change the method to specifying their value in a *.h mode and use the instruction "#include" to include the *.h files.

- Using conditional compilation to compile needed functions

  Because the compiler knows these control parameters at the time of compilation by including the *,h file, We can use "#if", "#else" to separate mutually exclusive function calls based on settings of control parameters. Therefore, only the needed functions would be compiled.

- Sidestepping layers and objects

  In simple profile, there is just one layer and one object, hence muti-layers and muti-objects coding is not necessary. We rewrite some functions for just single layer and single object.

- Other methods for code size reducing

  Several other useful methods for code size reduction are function combination, removal of useless library function calls and replacement of library function calls by simpler operations.

Table 2 shows the summary of the code size reduction. With the techniques described above and some simple code reorganizatoin, the final code size becomes 108 Kbytes.

2.2. Code Acceleration

Figure 2 shows the comparsion of complexity (execution time) in MoMuSys with QCIF Foreman. The functions DCT, IDCT, motion estimation and interpolation occupy most of exection time. In order accelerate the encoder, we have to optimize those functions.

1. DCT and IDCT

   The DCT/IDCT in MoMuSys is Fast DCT using floating-point computation which is unsuitable for fixed-point digital signal processor. In converting the floating-point DCT/IDCT to fixed-point, all the floating-point numbers are represented by 16-bit integers. The SNR with integer DCT/IDCT is just a

Table 2: Summary of Code Size Reduction

method	reduction	percentage
Remove the rate control	257 Kbytes	34.7%
Using Macros to Replace Simple Functions	123 Kbytes	16.6%
Changing Method of Setting Control Parameters	26 Kbytes	3.5%
Using Conditional Compilation to Compile Needed Functions	76 Kbytes	10.7%
Sidestepping Layers and Objects	56 Kbytes	7.6%
Other Methods for Code Size Reducing	96 Kbytes	12.1%

little worse than the floating-point case, but the integer DCT/IDCT yields great improvement in timing, almost 98% timing reduction.

2. Fast 16×16 integer motion search

   We use the diamond motion search as the fast algorithm to replace the full-search. Considering code memory and DSP character, the diamond search is a best choice for slow-moving video. When coding fast-moving video, we use the motion vector in the left-hand macroblock to estimate current motion vector. The method is as simple as the initial diamond search but faster than the initial diamond search when coding fast-moving video.

3. Fast 8×8 integer motion estimation

   After searching for 16×16 motion vectors, additional search made for 8×8 vectors. Using the 16×16 motion vector as the search center, the search range of 8×8 motion estimation is ± 2 pixels. The same as the 16×16 motion estimation, the algorithm of 8×8 motion estimation change to diamond search to replace the full search.

4. Half-pel motion estimation

   The half-pel motion estimation searches over the eight nearest half-pel neighbors of the best full-pel vector to find the best solution. In this part, using the algorithm described in [6] can be beneficial. It separate the eight nearest half-pel neighbors to the perpendicular and oblique parts. Check the perpendicular part first. If the smallest SAD is in the center, stop the functions. If not so, check the oblique parts.

5. Fast Interpolation

   In half-pixel motion estimation, the half pixel values are computed using the bi-linear interpolation. Each four half pixels is combination of four integer pixels. Because each memory aceess instruction costs four cycles in TI's digital signal processor, the intepolation with huge memory access is time-consuming.

   Because the intepolated pixel is combinated by the two or four neighboring integer pixels, the pixels in the center of the two half pixels is loaded twice. To speed up the functions, we can decrease the repeated loads, and use the assembly instruction 'ldw' (load word) and 'sdw' (store word) to replace two 'ldh' (load short) and 'sdh' to access two neighboring short integers simulaneously.

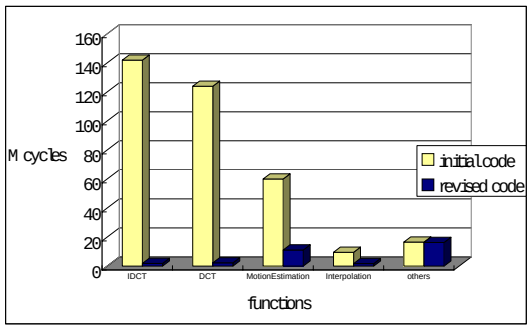


Fig. 3: Comparison between initial code and revised code in time complexity.

6. Fast SAD (sum of absolute differences)

   Because SAD is the main kernel of motion estimation, the speed of SAD directly influences the speed of motion estimation. Using the characteristics of the DSP chip, such as software pipelining and parallel processors, the speed of SAD can be increased.

Figure 3 shows the comparsion between initial code and revised code in timing. The clock cycle for IDCT/DCT is reduced from 141.8 and 123.9 Mcycles to 1.9 and 2.2275 Mcycles, which is 98.6% and 98.2% reduction. The clock cycle for motion estimation is reduced from 60.0 Mcycles to 11.1 Mcycles, which is 81.5% reduction.

3. Video Encoder Integration on DSP Board

Because the optimized code is still too big to be placed in one 64Kbytes program RAM, we separate the MPEG-4 encoder into two parts. One, called the main part, include file IO, data initial definition and motion estimation. The other, called the texture part, does texture coding, which primarily consists of the function "VopCodeShapeMorTextInter."

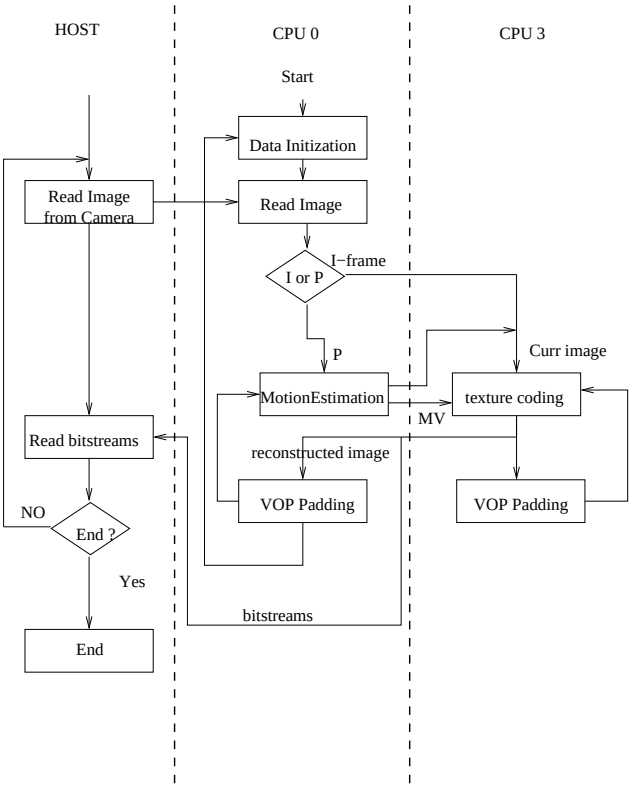


Fig. 4: Overall program flow on PC and DSP board.

Figure 4 shows the overall program flow on the DSP board and the host PC.The overall system employs three processors, namely, the host, DSP CPU0, and DSP CPU3. The host is only used for file IO. The input image is captured by the camera and transmitted to DSP board by host PC. The output bitstreams is writen to a file by host PC. CPU0 executes the main part of MPEG-4, handling the initial definitions for each video object and motion estimation. CPU3 executes texture coding for intra and inter frames.

when coding intra frames, the current image data are transferred to CPU3. After getting the image data, CPU3 can complete DCT, quantization, de-quantization,IDCT and run length coding and produce the reconstructed image and the encoded bitstream. The reconstructed image is fed to CPU0 for VOPPAdding and becomes reference frame in coding of next frame. Also, in CPU3, another VOPPadding reconstructed image.

When coding inter frames, motion estimation and motion compensation are needed. In the system, motion estimation and texture coding are executed in parallel. The Synchronization of CPU0 and CPU3 is achieved using messages over the FIFOLInk.

The final result, the bitstreams, are generated from both CPU0 and CPU3. Because the PCI interface control register and SRAM are only accessible from CPU0, only it can communicate with the host.Therefore, the bitstreams data must be transferred through CPU0 to the host.

4. Simulation Result

After optimization and parallelism,the function time is reduced. In the section, we show the overall performance of MPEG-4 encoder and R-D curve between initial MoMuSys and optimization mode.

4.1. Experimental Results

4.1.1. Texture Part

Table 3 shows a breakdown of the clock cycle for QCIF coding intra frame coding. The function MB_CodeCoeff implements zigzag scan and run length coding, which need many conditional branches and are difficult to reduce in optimization. Therefore, the speed-up of the function is less. The clock cycles of other functions with loops are reduced more significantly. The total clock cycles to encode one I frame in the texture parts are approximately 19.8 Mcycles, or 0.119sec.

When encoding a P frame, motion compensation is added. This takes almost 40 Kcycles for each loops. Therefore, the total cycles in encoding on P frame in the texture part are almost 21 Mcycles, or 0.131 sec.

4.1.2. Main Part

Table 4 shows a breakdown of the clock cycle of the main part. The primary complexity of the main part is due to motion estimation. Using the methods described preciously, the motion estimation is speeded up by 542.96%. The overall speed-up of the main part is 431.43%. In the final mode, the main part costs about 17.5 Mcycles, 0.109 sec per QCIF frame with data transmission.

4.1.3. Overall system

Using the clock functions defined in “time.h” to estimate the speed of system, we obtain the results shown in Table 5.

The execution time per a frame includes encoding in the main part and texture part and data transmission. Although the estimated execution time is only 0.13 seconds per frame, the pipelining of the two requires additional time. To sum up the performance, the resulting system can encode approximately six QCIF frames per second.

4.2. Rate-Distortion (R-D) Performance

Figures 5 and 6 show the R-D performance for the test sequences, foreman_qcif.yuv, trevor_qcif.yuv, suzie_qcif.yuv and akiyo_qcif.yuv. The original frame rate is 30 frame/sec in each case. The intra quantization step size is 15. The intra period is 4. As the figures show, because change in the motion estimation and DCT, the performance of two programs is about 0.3 dB decrease on the same bit-rate. Because the sequence Foreman has the Because the sequence “Foreman” has the global motion vector, using the first motion estimation can got the better performance. Therefore,

Table 3: Breakdown of Clock Cycles for QCIF Intra Frame Coding

seb_text	initial MoMuSys Code	Final optimized Code	number of execution
Process Initial	4,261	4,261	1
for each loop	2,819,090	151,119	11*9
CodeMB	2,740,428	66,535	11*9
DCT	208,549	3126	11*9*6
BlockQuantH263	4,409	2086	11*9*6
BlockDequantH263	4,967	2158	11*9*6
IDCT	238,815	3750	11*9*6
doDCACpred	20,671	16671	11*9
MB_CodeCoeff	57,991	47,981	11*9
total	278,734,416	15,705,870	
with Data transmission	282,869,570	19,841,024	

Table 4: Breakdown of Clock Cycles for Main Part in Coding of QCIF Frames

main	initial_MoMuSys	Final optimized Code	number of execution
initial (main & vop process)	113,135	113,135	1
vopProcess (without)	248,336	248,336	1
read image	2,833,299	2,833,299	1
Vopcode (without motionEstimation)	2,507,009	1,194,283	1
VopPadding	2,415,721	1,128,053	1
MotionEstimation	69,539,561	12,860,402	1
init (Interpolate Image and AllocImage)	9,522,936	1,741,803	1
MBMotionEstimation	303,126	43,666	99
halfpel motion estimation	219,155	39,713	99
SAD_MB	13,854	3168	$\geq$ 99*5
End	18,824	18,824	
total	75,508,500	17,516,615	

Table 5: Overall Coding Speed

	average time per QCIF frames (seconds)			
QP	akiyo_qcif.yuv	suzie_qcif.yuv	trevor_qcif.yuv	foreman_qcif.yuv
28	0.131	0.137	0.137	0.142
24	0.131	0.137	0.137	0.143
20	0.132	0.137	0.138	0.143
16	0.132	0.138	0.140	0.146
12	0.142	0.143	0.145	0.153
8	0.145	0.149	0.151	0.158
4	0.150	0.155	0.158	0.164

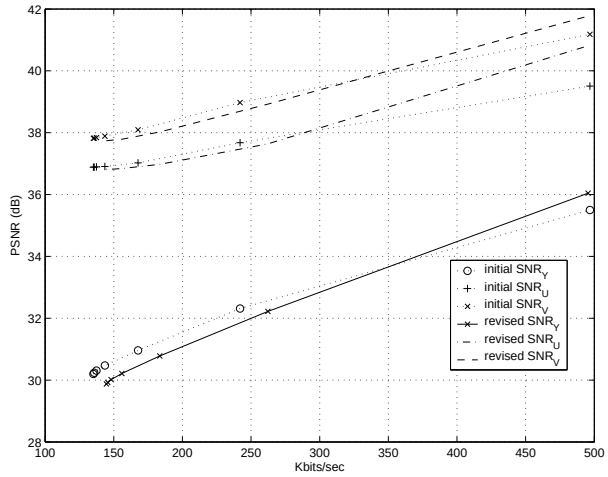


Fig. 5: R-D performance in coding for foreman_qcif.

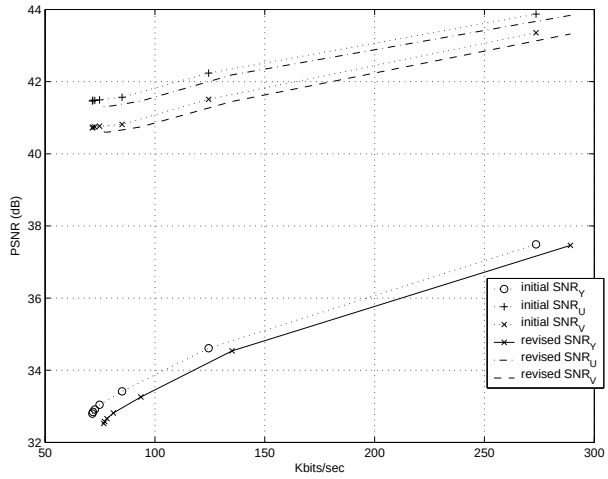


Fig. 6: R-D performance in coding for suzie_qcif.

we can get the better performance, with the small quantization step.

However, the final revised code achieves approximately 400% speedup comparing with initial simplified code for both processes.

5. Conclusion

We considered implementation of real-time MPEG-4 simple profile video codec on TI's C62xx DSPs in II's Quatro62 environment. The work was based on modifying the MoMuSys source code, which is a powerful but huge codec for MPEG-4 video. Current implementation employs two DSPs.

We did some code size reduction to fit the code to the limited DSP on-chip program memory, where we removed the rate control, used macros to replace simple functions, changed the method of setting control parameters, sidestepped layers and objects, and did some others simple program changes. To speed up the program, we replaced the motion estimation algorithm and the DCT/IDCT, which resulted in only a small performance loss. We also modified and rearranged some program sections, in particular the interpolation and the SAD functions for motion estimation and compensation, to help the compiler parallelize the compiled code for better utilization of the DSP's parallel functional units in encoder execution.

Currently, the implementation can encode about 6 QCIF frames per second, but with uneven loads between the two DSPs. This is almost four times speed-up compared to the original code. The code size is reduced by about 80%. The performance loss compared to MoMuSys is less than 0.5 dB.

6. References

[1] International Committee for Information Technology Standards, web site, http://www.ncits.org/.

[2] Texas Instruments, *TMS320C6000 Technical Brief*, literature number: SPRU197D, Feb. 1999.

[3] Innovative Integration, *Quarto6x Quatro6x Development Package Manual*, Jan. 16, 2001.

[4] Video Group, "Text of 14496-7 PDTR (Optimized Visual Reference Software)" ISO/IEC JTC1/SC29/WG11 N4057, Singapore, March 2001

[5] S. H. Wang, C. N. Wang, T. Chiang, and H. Sun "AHG report on editorial convergence of MPEG-4 reference software" ISO/IEC JTC1/SC29/WG11, MPEG2002/M 8041, March 2002

[6] W. Zheng, I. Ahmad, and M. L. Liou, "Real-time software based MPEG-4 video encoding, " *Proc. IEEE workshop Exhibition MPEG-4*, pp. 71–74, 2001,

[7] W. Zheng, I. Ahmad, and M. L. Liou, "Adaptive motion search with elastic diamond for MPEG-4 video coding" *Proc. IEEE Int. Conf. Image Processing*, vol.1, PP. 377 -380, 2001.

附錄 C

Real-Time Implementation of MPEG-4 Fine-Grainularity-Scalable Video Encoder on Digital Signal Processors

Yen-Fu Chen and David W. Lin

Department of Electronics Engineering and Center for Telecommunications Research
National Chiao Tung University
Hsinchu, Taiwan 30010, R.O.C.
E-mail: afu.ee90g@nctu.edu.tw, dwlin@mail.nctu.edu.tw

Abstract

Fine Granularity Scalability (FGS) is a technique specified in the Amendment of MPEG-4. It is developed to the growing need of video delivery over the Internet. Compared to conventional techniques, it offers a different way to optimize video quality over a range of bitrates. In this work, we implement a real-time MPEG-4 FGS encoder on digital signal processors (DSPs). The digital signal processing environment is Innovative Integration's Quatro62 personal computer plug-in card, which houses several Texas Instruments' TMS320C6201 DSPs. We use a formerly developed ITU-T H.263+ encoder as the base-layer encoder, which resides on one DSP. The FGS encoder works at the enhancement layer and resides on a second DSP. We base our FGS encoder on modifying the publicly available software MoMuSys. In order to achieve real-time encoding on DSP, we replace a few slow blocks in the original C program and further refine our code by taking into account the features of the DSP chip to produce a more efficient program. Overall, we speed up the MPEG-4 FGS encoder on DSP by several-fold. The final encoding speed is about 12 QCIF frames per second with all bitplanes encoded, and about 18 frames per second with two last bitplanes dropped.

1. Introduction

In response to the fast growing network video applications, the Amendment of MPEG-4 has specified Fine Granularity Scalability (FGS) coding to provide enhanced video deliverty capability for services such as Internet streaming video. Compared to conventional layered scalability techniques, FGS employs a different strategy to optimize video quality over a range of bitrates. Through FGS coding, the enhancement bitstream can be truncated to nearly any number of bits to provide partial enhancement according to the bits delivered or decoded for each frame.

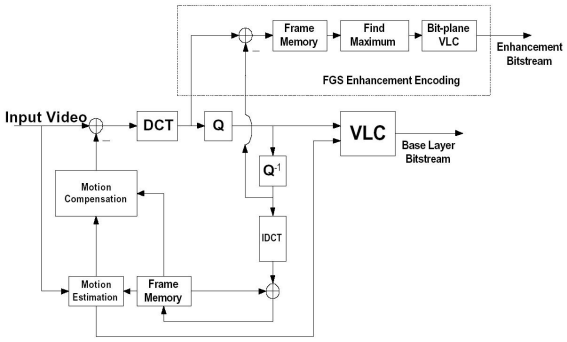


Fig. 1: Basic FGS encoder structure.

The basic FGS encoder structure is shown in Figures 1. In the encoder, the base layer bitstream is generated from motion compensation, DCT (discrete cosine transform), quantization, and VLC (variable-length coding) according to the MPEG-4 standard. The FGS enhancement encoder takes the original and reconstructed DCT coefficients as inputs. After obtaining all the DCT residues of a VOP (video object plane), the maximum absolute value of the residues is found and the maximum number of bitplanes for the VOP is determined. The enhancement bitstream is then generated after each bitplane is coded through the bitplane variable length coding. The bitstream of the FGS enhancement layer may be truncated to nearly any number of bits per picture after the encoding is completed.

Our goal of this work is real-time implementation of MPEG-4 Fine-Grainularity-Scalable video encoder on digital signal processors (DSPs). The environment of our DSP implementation involves a host PC, a DSP board and the DSP chips on the board. The DSP chips are Texas Instruments (TI)'s TMS320C6201. The TMS320C62x is a fixed-point DSP with 5 ns instruction cycle time. It employs the VelociTI Very Long Instruction Word (VLIW) architecture that enables sustained throughput of up to eight instructions in parallel

This work was supported in part by the National Science Council of R.O.C. under grant no. NSC 91-2219-E-009-045.

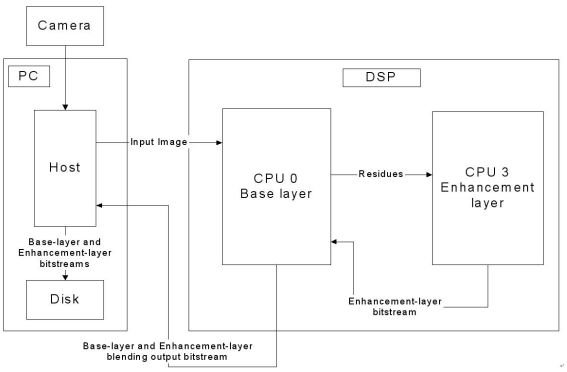


Fig. 2: Architecture of the overall video encoder system.

[1]. In addition, the C62x DSPs come with on-chip program and data memories, which may be configured as cache on some devices. The DSP board we use is Innovative Integration (II)'s Quatro6x. It is a PCI bus compatible DSP card housing four TI TMS320C62x processors in a symmetric multiprocessing relationship with high bandwidth inter-processor communication links.

For convenience, we use an H.263+ encoder as the base-layer encoder. The encoder is a result of earlier work [2]. In the development process, we first combine these two encoders on a PC and then convert the environment from PC to DSP. We make use of the features of the C62x chip to enhance the FGS encoder on the DSP. The resulting system achieves real-time coding speed for QCIF pictures.

2. Architecture of Overall Video Encoder System

Figure 2 shows the overall encoder system architecture and how it operates. Image data are captured by the camera and transmitted to the host PC. The host PC is in charge of the communication mechanism between PCI and DSP. Of the four DSPs on the Quatro6x card, only one, denoted CPU0, can communicate with the host directly. We let it implement the base-layer encoder. After CPU0 receives the image data from the host, the encoding processing begins and the base-layer bitstream is generated. In the middle of the base-layer encoding process, the residues, that is, the difference between the original and reconstructed DCT coefficients, are generated and transmitted to the enhancement-layer encoder through the FIFOLink on the Quatro6x between CPU0 and CPU3. As mentioned, we employ a previously developed H.263+ encoder as the base-layer encoder and employ the MPEG-4 FGS encoder as the enhancement-layer encoder. After the whole picture is encoded by both the base-layer encoder and the enhancement-layer encoder, the base-layer bitstream

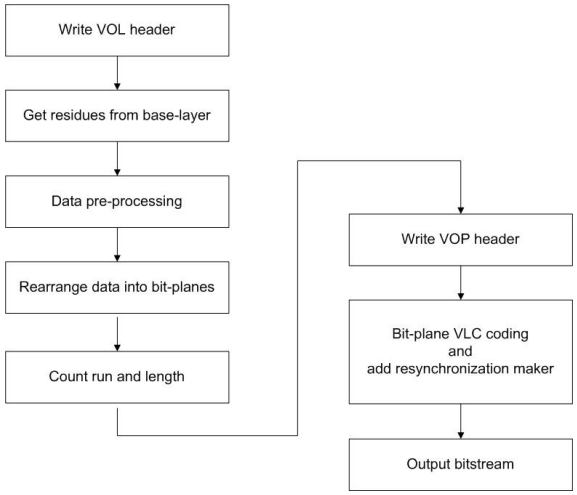


Fig. 3: Procedure of FGS coding.

and the enhancement-layer bitstream are packed together and transmitted to the host. The combined bitstream after post-processing in host is divided into two bitstreams and stored to the disk.

3. FGS Encoder Optimization

3.1. Profile of the Original FGS Encoder on DSP

Since an existing software H.263+ encoder on DSP is used for base-layer encoding, the main task of our real-time implementation work, besides system integration, is to obtain an efficient DSP implementation of the FGS encoder. This involves primarily studies to speed up the execution of FGS encoding on DSP.

Figure 3 shows the procedure of FGS coding, where "data preprocessing" (the third block) means the procedure where the residues are altered by the fgs_shift_matrix and the fgs_rectangular_shift_factor while either frequency weighting or selective enhancement is enabled. By profiling the FGS encoder on DSP without doing any optimization, the proportions of the execution time of these procedures are obtained. And they are shown in Figure 4. The bit-plane VLC coding dominates the execution time of encoding. A reason is that FGS does not do motion estimation, motion-compensated prediction, DCT, and IDCT, which often consume the majority of computation time in motion-compensated DCT coding. Moreover, typical general-purpose CPU architectures and compiler properties are at odds with what VLC requires for efficient execution.

Further, the temporal redundancy in video is not exploited in FGS coding as in predictive coding. Consequently, the size of the full FGS encoded bitstream is very larger than that of H.263+. We use akiyo_qcif.yuv as a test sequence. The H.263+ encoder only encodes it with I-frames. Figure 5 shows the sizes of the output bitstreams under different QPI (intra quantization parameter) for the H.263+ encoder while frequency

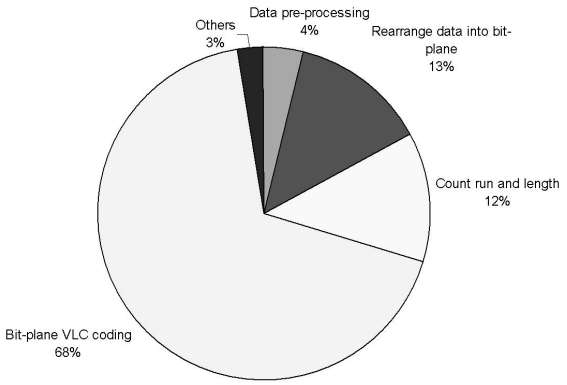


Fig. 4: Proportions of execution time of different procedures.

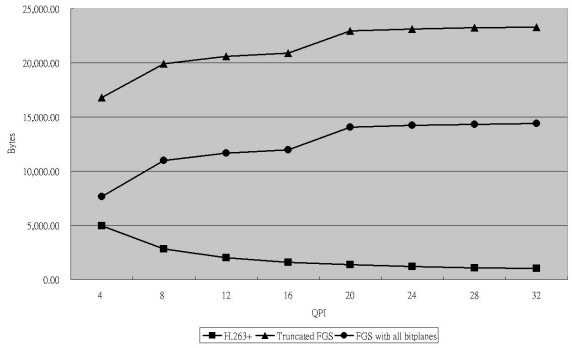


Fig. 5: FGS output bitstream sizes without frequency weighting or selective enhancement.

weighting and selective enhancement are disabled in the FGS encoder. And in Figure 6, we show the results where frequency weighting and selective enhancement are enabled.

3.2. Code Acceleration

To speed up FGS encoding, we make use of the features of the C62x chip as well as the relevant provisions of the compiler to optimize the FGS encoder.

1. Configuring of Compiler Options Setting
   TI's Code Composer Studio (CCS) is a useful GUI tool that helps engineers develop DSP codes. CCS compiles the C code and assembles it into the COFF file format. Compiler options control the operation of the compiler. Proper configuration of the compiler options helps the compiler generate efficient assembly codes.

2. Software Pipelining
   Software pipelining is a technique used to schedule instructions in a loop so that multiple iterations of the loop execute in parallel. Its realization consists of implementing parallel instructions, filling delay slots with useful instructions, loop unrolling, and maximizing usage of functional units. Software pipelining is an efficient way to improve performance.

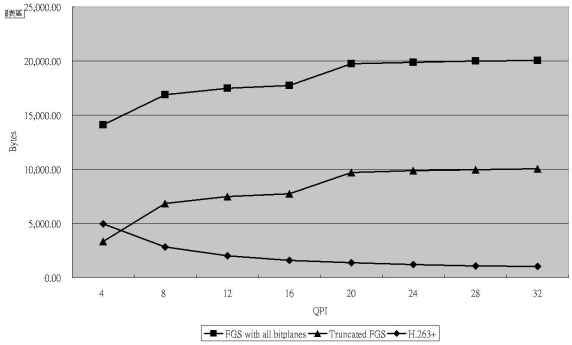


Fig. 6: FGS output bitstream sizes with frequency weighting and selective enhancement.

3. Using Intrinsics
   TI's C6000 compiler provides intrinsics, which are special functions that map directly to inlined C62x instructions, to optimize C code. Many efficient DSP instructions that are not easily expressed in C code are supported as intrinsics.

4. Packed Data Processing
   In order to maximize data throughput, it is often desirable to use a single load or store instruction to access multiple data values located consecutively in memory. When operating on a stream of 16-bit data, for example, we can use word (32-bit) accesses to read two 16-bit values at a time, and then use C62x intrinsics to operate on the data in parallel.

5. Memory Usage Strategy
   The C62x accesses to the external memory require more cycles than to the internal memory. The external memory access time also depend on what kind of RAM is used. The Quatro 62 board uses SDRAM and SBRAM as external memories. So it is good to use on-chip memory as much as possible to decrease the number of external memory accesses. If some data have to be put in the external memory, one should try to use DMA to load them into the on-chip memory before processing them.

6. Memory Model and Allocation
   To maximize the code efficiency, the compiler schedules as many instructions as possible in parallel. To schedule instructions in parallel, the compiler must determine the dependency between instructions, which means whether one instruction must be executed before another. For example, a variable must be loaded from memory before it can be used. Because only independent instructions can execute in parallel, dependency inhibits parallelism. To help the compiler

determine memory dependencies, we can qualify a pointer, reference, or array with the "restrict" keyword. This practice helps the compiler optimize certain sections of code because aliasing information can be more easily determined.

7. Using Macros
   Since it takes some clock cycles to complete a function call and since the compiler is such that software-pipelined loop cannot contain function calls, we may change functions into "define" macros under some conditions to speed up the execution. Because macros are expanded in the resulting code, the program size is usually bigger than using funciton calls.

8. Short Format for Multiplication
   The multiplication units of C62x performs 16-bit by 16-bit multiply operations. Multiplication of longer operands are broken into several such operations. So one should use the short data type for multiplication inputs whenever possible because this data type provides the most efficient use of the 16-bit multiplier in C62x. For loop counters, one should use int or unsigned int, rather than short or unsigned short, to avoid unnecessary sign-extension.

9. System Level Pipelining
   The basic data flow of the whole system is as shown in Figure 7. CPU 0 does the base-layer encoding and CPU3 performs the enhancement-layer encoding. The base-layer encoder encodes one macroblock and feeds the residues of one macroblock to the enhancement-layer encoder. That means, when the residues of one entire picture are generated, the base-layer encoding is almost done. After receiving all the residues and rearranging the residues to bit-planes, the enhancement-layer encoder begins the bit-plane VLC coding. As the profile in Figure 4 shows, the bit-plane VLC coding occupies most of the execution time. Therefore, the scheme depicted in Fig. 7 causes the CPU0 to idle and wait for the enhancement-layer encoding to finish. This is apparently not an efficient system design. Consequently, we reschedule the flow of the whole system as shown in Figure 8. After CPU0 finishes the encoding of the whole Picture 0, the residues are sent to CPU3 and the output bitstream of Picture 0 is buffered. Then the base-layer encoder continues to encode Picture 1 since it does not have to wait for the end of enhancement-layer encoding. In this manner, most of the CPU idle time is removed because CPU0 and CPU3 can work in parallel. The whole system is much more efficient than the non-pipelined design.

Figure 9 compares the initial code and the revised code in clock cycles. And figure 10 shows the popor-

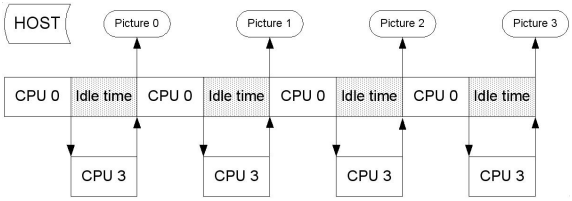


Fig. 7: System without pipelining.

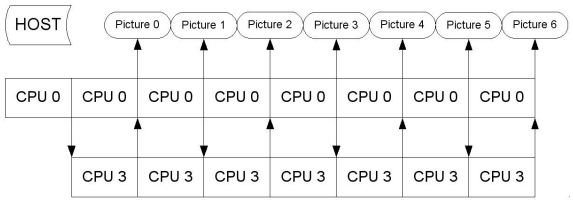


Fig. 8: System with system-level pipelining.

tions of execution time of different program sections after acceleration.

4. Additional Performance Results

We present some additional performance data of the implemented MPEG-4 encoder. We use the clock functions defined in "time.h" on PC to estimate the speed of our system. The test sequence is akiyo_qcif.yuv. We use different quantization step sizes to find out the speed of our system under different conditions.

4.1. With and Without Frequency Weighting and Selective Enhancement

Table 1 shows the overall coding speed under different quantization step sizes in the base layer. We consider three kinds of FGS options: "non-optimized" is the FGS encoder on DSP without any optimization, "software pipelining" means the encoder that is obtained by setting the compiler options properly and the compiler

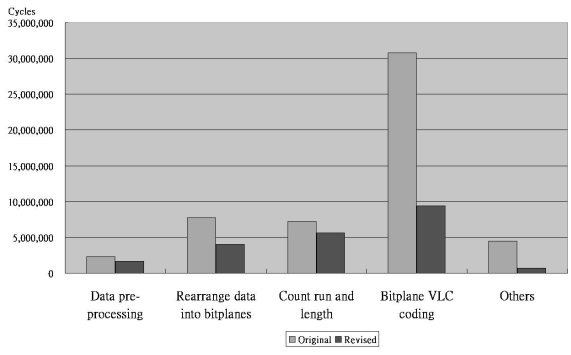


Fig. 9: Comparsion between initial code and revised code in clock cycles.

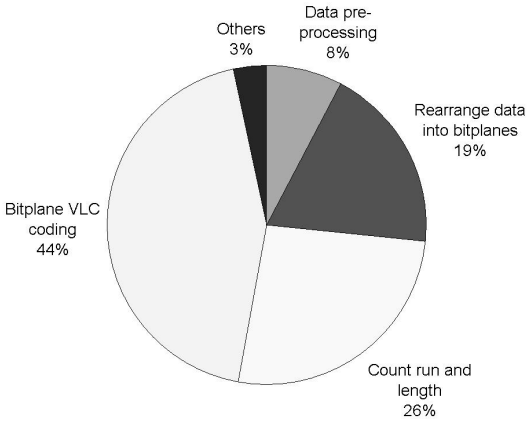


Fig. 10: Poportions of execution time of different program sections after acceleration.

has the ability of doing software pipelining, and "optimized" means the optimized final code. Frequency weighting and selective enhancement are both disabled in all three cases.

Now we enable frequency weighting and selective enhancement. The fgs_shift_matrix we use is as follows:

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

The value of fgs_rectangular_shift_factor is 3. The experimental result is shown in Table 2.

4.2. With and Without Encoding of the Last Two Bitplanes

In our implementation, the FGS output bitstream is transmitted to the base-layer encoder when all the bitplanes are encoded. In fact the channel bandwidth may be smaller than the bitstream. So the FGS stream may be truncated before being transmitted to the channel. Actually, it is the experience of some researchers that the last few bitplanes of the bitstream may be truncated without much effect on the subjective quality of the decoded video. Table 3 shows the performance of the FGS encoder when the last two bitplanes are not encoded. Since the last two bitplanes only affect the last two bits of the residues, the quality of the restructured pictures does not change significantly, while the improvement in speed is significant.

5. Concluding Remarks

We considered real-time implementation of MPEG-4 FGS video encoder on DSPs. We have used a previously implemented H.263+ encoder as the base-layer encoder. And the FGS enhancement-layer encoder is based on the FGS section of the MoMuSys software.

For DSP implementation, we have focused on the speed-up of the FGS encoder and the overall system design, since our system requires the working together of a host PC and two DSP chips. The use of two DSPs was for simplicity of system integration, where one DSP implements the H.263+ base-layer encoder and the other implements the FGS encoder. The code size of the FGS encoder was quite smaller than the DSP's internal memory size. Therefore, code size reduction was not a major point of our work as in some other implementation studies.

We profiled the FGS encoder and found out the bottlenecks in the encoder functions. We then sought to accelerate the code by utilizing the features of the C62x chip, as well as the provisions of the compiler. The bitplane VLC coding was found to take the majority of the program execution time. In particular, the function of outputting bits to the bitstream was found to cost an unexpected amount of complexity. Simply by rewriting this function, we gained proportionately the most improvement in all the work that we did.

In system integration, we scheduled the workflow so that the H.263+ encoder and the FGS encoder could work in parallel. Since the speed of the FGS encoder was slower than that of the H.263+, the speed of the overall system was dependent on the improvement of the FGS encoder. The final encoding speed of the implementation is about 12 QCIF frames per second at no video quality loss by bitplane dropping, which is about 650% speed-up compared to the original encoder with no optimization. With dropping of two last bitplanes, the speed can reach about 18 frames per second.

6. References

[1] N. Seshan, "High VelociTI processing," *IEEE Signal Processing Mag.*, vol. 15, no. 2, pp. 86–101, Mar. 1998.

[2] M.-L. Woo, "Real-Time Implementation of H.263+ Using TI TMS320C62x," M.S. thesis, Department of Electronics Engineering, National Chiao Tung University, June 2000.

Table 1: Overall Coding Speed Without Frequency Weighting or Selective Enhancement

	Average QCIF frames per second				
QPI	Non-optimized	Software pipelining	Optimized	Speed-up (non-optimized vs. optimized)	Speed-up (proper configuation vs. optimized)
4	2.10053	5.51755	13.50439	6.42904	2.44754
8	2.08065	5.45524	13.44447	6.46168	2.46451
12	1.98942	5.23286	12.98364	6.52636	2.48117
16	1.97656	5.23259	12.87830	6.51552	2.46117
20	1.92894	5.11169	12.70025	6.58406	2.48455
24	1.90360	5.06380	12.54863	6.59204	2.47810
28	1.88512	4.97661	12.38237	6.56847	2.48811
32	1.87403	5.01128	12.49688	6.66846	2.49375

Table 2: Overall Coding Speed with Frequency Weighting and Selective Enhancement

	Average QCIF frames per second				
QPI	Non-optimized	Software pipelining	Optimized	Speed-up (non-optimized vs. optimized)	Speed-up (proper configuation vs. optimized)
4	1.69497	4.73732	12.64702	7.46149	2.66966
8	1.67628	4.66810	12.49844	7.45607	2.67741
12	1.61283	4.51610	12.07438	7.48648	2.67363
16	1.60720	4.49438	11.98179	7.45507	2.66595
20	1.57480	4.42576	11.80638	7.49705	2.66765
24	1.55453	4.35996	11.74260	7.55378	2.69328
28	1.54552	4.33614	11.72058	7.58357	2.70300
32	1.53440	4.29516	11.55268	7.52911	2.68970

Table 3: Overall Coding Speed Without Encoding of Two Last Bitplanes

	Average QCIF frames per second	
QPI	Frequency Weighting and Selective Enhancement Disabled	Frequency Weighting and Selective Enhancement Enabled
4	19.33862	19.17178
8	19.28268	19.16076
12	19.22338	18.08318
16	18.88574	18.03101
20	18.33181	17.71793
24	18.17851	17.43983
28	18.13237	17.30104
32	17.63668	17.02128