行政院國家科學委員會補助專題研究計畫 □ 成 果 報 告
☑期中進度報告

支援 3-D 立體視訊的數位電視多媒體平台設計

計畫類別：□ 個別型計畫　☑ 整合型計畫
計畫編號：NSC　96－2220－E－009－038
執行期間：　96　年　11　月　1　日至　99　年　10　月　31　日

計畫主持人：蔡淳仁
共同主持人：張添烜、范倫達、彭文孝
計畫參與人員：

孫域晨、蘇冠年、廖珮晴、黃建峰、白佳芸、曾宇晟、廖元歆、陳奕均、洪瑩蓉、許博雄、陳宥辰、陳漪文、陳俊吉、楊復堯、許籐耀、林丞蔚、謝佳育、董盈里

成果報告類型(依經費核定清單規定繳交)：□精簡報告　☑完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
☑出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢

□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：國立交通大學資訊工程系、電子工程系

中　華　民　國　98 年　08 月　28 日

摘要

本計畫的目標在設計一個能支援下一代立體視訊及 3-D 人機介面的嵌入式系統多媒體應用程式平台。本平台的重點是它不是針對任何一個殺手級應用而設計，而是依照一個開放式的 Java 執行環境 DVB-MHP 來進行設計。這個平台因爲是架構在一個公開的國際標準之下，所以可以由協力商（third party）發揮創意來擴充其上的應用。這個平台的設計是透過中介軟體來包裝並整合底層的寬頻多媒體應用所需要的元件（如影音解碼、繪圖、以及 3-D 視訊）的加速功能。讓應用程式的開發者可以不用擔心底層的嵌入式系統所使用的處理器、作業系統是什麼，就能發揮出系統最豐富的功能。在這個計畫下，我們會開發出下列的關鍵元件：由 DVB-MHP 所擴充而來的 3-D 人機介面中介軟體、一個專爲支援 Java 執行環境而設計的極小型 OS、3-D Video 加速器、3-D Graphics 加速器、Java 處理器、和配合 3-D Video 的 MVC 多視角合成技術。

關鍵詞：中介軟體、Java 處理器、嵌入式作業系統、3-D Video、視差估測加速器、3-D Graphics 加速器、Multi-view Video 編碼

Abstract

The goal of this project is to design an embedded platform to support next generation stereo video and 3-D man-machine user interfaces for multimedia applications. The key design focus of this platform is that it is not designed for any particular killer applications. Instead, it is designed based on an open multimedia application middleware, namely DVB-MHP. Therefore, any third party designer can develop innovative applications for this platform without having to worry about the underlying architecture, such as the type of processor or the type of OS used to create the platform. More importantly, the middleware provides user application accesses to highly efficient feature-rich multimedia components, including audio-video decoding, graphics, and 3-D video. The key technologies that will be developed in this project include: DVB-MHP middleware with extension for stereo video and man-machine user interface, a deeply-embedded minimal OS designed specifically for Java runtime support, 3-D video accelerator, 3-D graphics accelerator, Java processor, and multiview video coding technology.

Keywords: Middleware, Java Processor, Embedded OS, 3-D Video, Disparity Estimation Accelerator, 3-D Graphics Accelerator, Multiview Video Coding

I、 計畫成果概述

本計畫預期目標是設計支援立體顯示的數位電視 DVB-MHP 標準的嵌入式平台，其中關鍵模組為：視差估測模組(*子計畫一*)、3D 繪圖加速器模組(*子計畫二*)、立體視訊顯像模組(*子計畫三*)以及嵌入式 JAVA 平台設計(*子計畫四*)。目前為止，總計畫已經階段性整合子計畫研發之模組，設計出兩個嵌入式平台，分別為：整合視差估測模組和立體視訊合成模組的立體視訊平台(符合 MPEG FTV 應用的嵌入式平台)、整合 3D 繪圖加速器模組和嵌入式 JAVA 平台提出支援 3D 人機介面的嵌入式 JAVA 平台。

以下將依序介紹總計劃的整合成果以及各子計畫關建模組成果：

總計畫整合成果

目前總計畫是分成兩階段進行整合，第一階段是先由子計畫一跟三進行局部整合，還有子計畫二跟四進行局部整合。因此在這次的期中 demo，我們會有兩個子系統的 demo。第一個子系統是由子計畫一跟三合作的立體視訊合成平台。第二個子系統是由子計畫二跟四合作的支援 3D 人機介面的 JAVA 平台。以下就針對這兩個整合的成果進行描述。

整合成果一、立體視訊合成平台

目前總計劃已在 ARM Versatile 平台整合子計畫一與子計畫三，成為可支援立體電視的嵌入式系統設計，其架構如。在系統啟動前已將視訊影像和程式儲存於 Data Flash 和 Flash。首先 ARM 處理器從 SD card 至 SDRAM 搬移視訊影像，並以 Enable 訊號啟動視差估測。視差估測產生 Depth map 存於 SDRAM，並以 Finish 訊號通知 ARM 處理器進行 view synthesis。

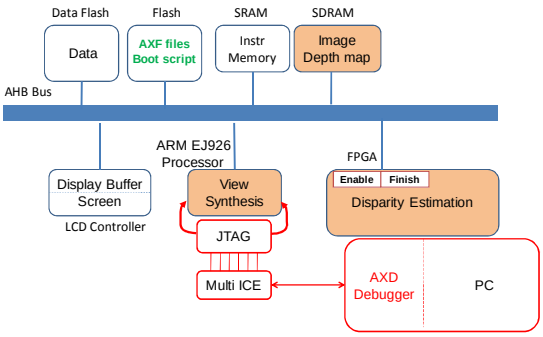


圖1. 立體電視視訊顯示平台架構圖

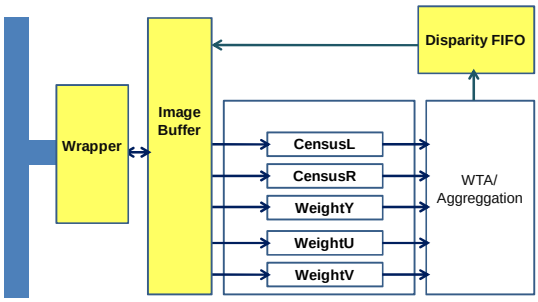


圖2. 子計畫一於 AMBA 的架構

子計畫一整合視差估測於 AMBA AHB 的架構如圖 1。視差估測的核心外以 Disparity FIFO 及 Image Buffer 作為外部讀寫資料的暫存記憶體，並以 Wrapper 作為 AMBA AHB 的介面連接於 bus。 將視差估測演算法實現於 FPGA 上使用資源如下表所示。

FPGA Virtex4-XC4VLX20	
Clock rate	75M Hz
LUT (4-input)	85,843
Slices	45,993
Total Equivalent Gate Count	5,563,039
Dual Port RAM	7,740
16x1 RAMs	144
FPS	7

子計畫三執行流程如圖 3，若 disparity map 有壓縮則以 disparity refinement 處理後再進行 virtual view 的合成，最後再把合成影像轉為立體影像。由讀 Image 至合成影像並輸出螢幕上，其整體的效能如表 1。

表1. Software performance

	Current Version (sec) CIF
Load Stereo Images	~0.04
HW- disparity Calculation(including overhead)	~0.93
Stereo-View Synthesis	~1.22
Write to VGA Buffer	~0.58
Total time	~2.77

圖 4 為子計畫一和子計畫三的成果展示：

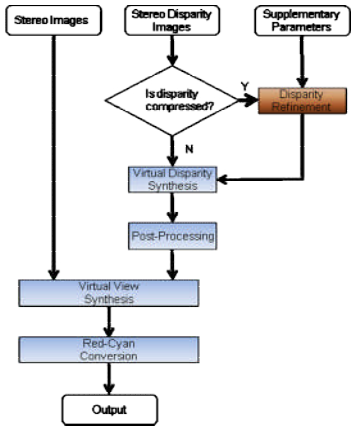


圖3. Software flow chart

圖4. Virtual view synthesis

整合成果二、支援 3D 人機介面的 JAVA 平台

整合子計畫二與子計畫四的目的是希望利用可重組 3D 繪圖加速器(子計畫二)與異質雙核心 Java 執行系統(子計畫四)來建立一個更豐富的嵌入式多媒體平台。執行至今，開發團隊已經建構出全系統的雛形，並且能夠做一些應用程式的展示。目前的整合系統實作在兩個不同的開發平台: Xilinx Spartan 3A DSP 與 Xilinx ML405。兩個平台中間採用 IIC 匯流排做為溝通的介面。完整的整合系統架構圖如下圖所示:

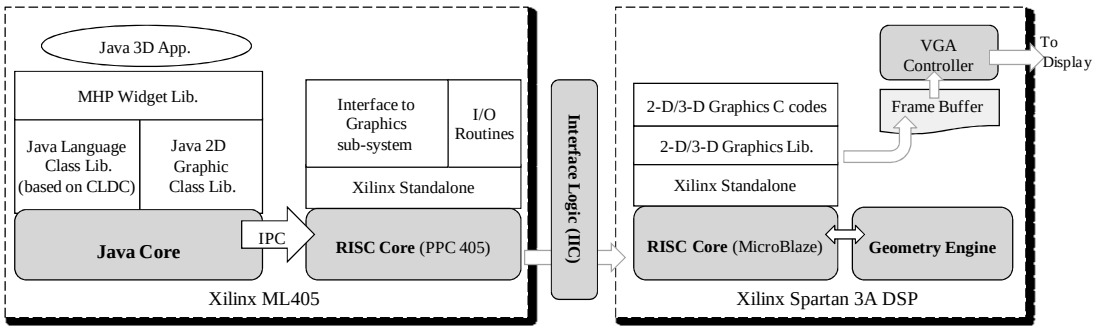


圖5. 支援 3D 人機介面的 JAVA 嵌入式平台架構

在 Java 執行系統方面，雙指令的 Java 處理核心能夠有效地執行 Java 應用程式，並且一些 3D 繪圖加速器的溝通介面會以 RISC 輔助程式的方式增加在執行環境裡。程式執行過程中，如果需要做 3D 繪圖的動作，Java 處理核心會利用快速的內部通訊機制將參數傳送給 RISC 核心。最後，輔助程式與 IIC 匯流排傳送端程式會利用 IIC 匯流排將控制參數傳給 3D 繪圖加速器。

當 Spartan 3A DSP 上的 IIC 匯流排接收端程式收到參數時會將其儲存起來，並且啟動 3D 繪圖加速器。在 3D 繪圖加速器執行過程中，前段的幾何轉換子系統會先讀取頂點和顏色資料作運算，運算完成之後三角形資料將被傳遞到後段的著色子系統做之後的繪圖動作。但是由於 FPGA 容量的問題，因此在目前的開發平台上，3D 繪圖加速器前段的幾何轉換子系統採用硬體的實作，而後段的著色子系統則是用軟體的方式執行。

兩個團隊的硬體架構的合成數據如表 2 與**錯誤! 找不到參照來源。**。圖 6 則是成果的展示。

FPGA Spartan3-XC3SD1800A		
Clock rate		87M Hz
LUT (4-input)	15,751	47%
Slices	11,271	67%
RAM16s	67	79%
DSP48s	40	47%
IOBs	22	4%

表2. 子計畫二硬體架構的合成數據

FPGA Virtex4-XC4VFX20		
Clock rate		101M Hz
LUT (4-input)	8,372	62%
Slices	7,817	91%
RAM16s	48	70%
DSP48s	3	9%
IOBs	93	29%

表3. 子計畫四硬體架構的合成數據

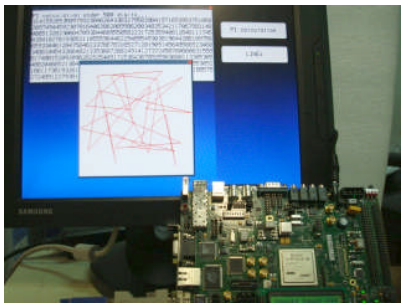


圖6. 子計畫二、四整合成果展示

子計畫一成果：視差估測模組

一、前言

視差資料估測(disparity map estimation)的計算複雜度與記憶體頻寬需求相當龐大，越是高正確度的視差資料估測，需求越高，可達一般用於視訊壓縮的移動估測十倍至百倍以上的複雜度。子計畫一目的在於以硬體實現視差估測演算法，並且輸出高準確度且即時處理速度的視差圖。計畫執行至今已發展 semi-global approach 的演算法與硬體設計，該硬體設計已達到本子計畫預定之效能。此外，論文發表共 7 篇，培育畢業研究生共 3 位。未來，將進一步發展可處理更大影像，且降低成本的硬體設計。

二、研究目的

子計畫一目的在發展適合硬體實現的視差估測演算法與其硬體實作。視差估測模組可讀取已校正的兩視角影像，大小為 CIF(325x288)，輸出高正確度的視差資料，視差估測硬體處理速度達 5 fps 以上。配合總計劃的多媒體開放式平台，視差資料估測運算元將依據 MHP 給予的指令而運作，並且將運算結果交付給 3-D 繪圖合成運算元，進一步合成出 3-D 影像，最後經由 FPGA 平台與晶片設計驗證本計畫成果。

三、結果與討論

子計畫一執行至今已發展出適合硬體實現的半全域視差估測演算法。此演算法基於適應性權重計算(Adaptive Weight)演算法結合微型普查(Mini-Census)的比對方式，以及量子化指數曼哈頓色彩距離(Quantized Manhattan Color Distance)等技巧。演算法如圖 7 所示。

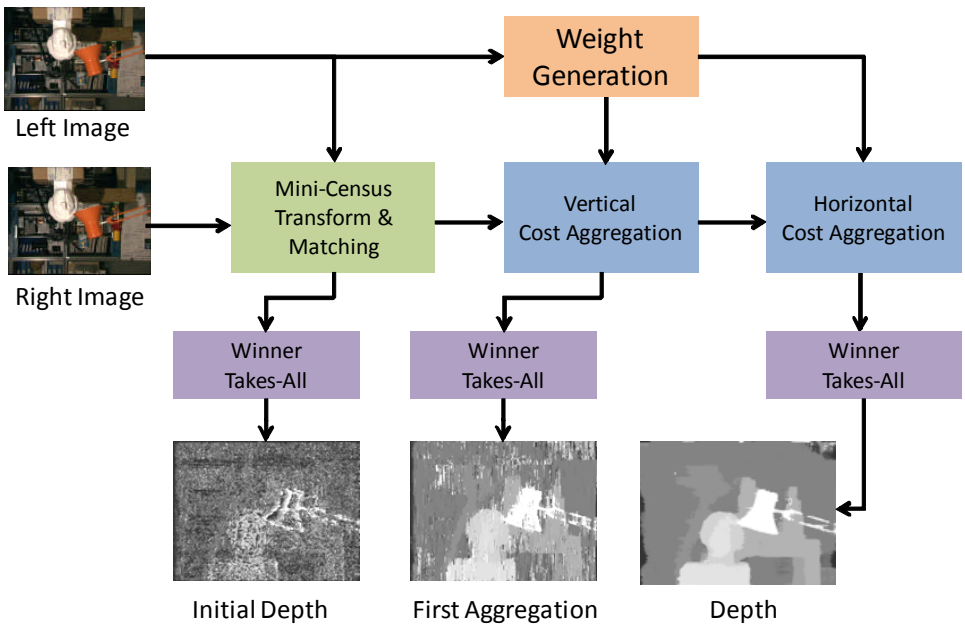


圖7. 視差估測演算法

首先，由 Mini-Census 計算兩像素的比對值(matching cost)可減少運算量，從原始的一個方型視窗的運算量轉換為六個點的運算量。其運算如圖 8 所示。Mini-Census 優點在於較原始演算法更能承受影像受光線影響的問題。

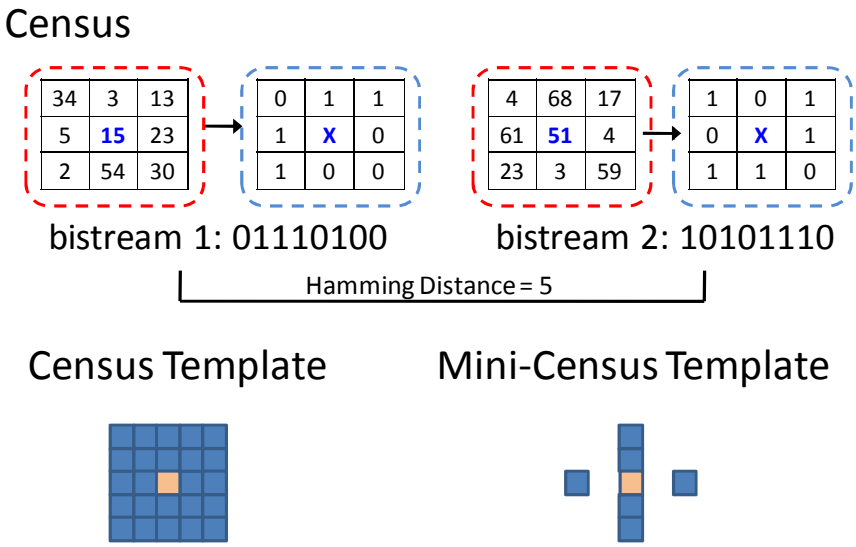


圖8. Mini-Census

接著，以 Weight Generation 以量子化指數曼哈頓色彩距離產生 Weight，此 Weight 用於加權相鄰像素 matching cost 的匯聚。Weight 在影像上的分布如圖 9 所示，Proximity 表示距中心越近的像素點比重越高，Color Similarity 表示與中心顏色越接近得像素點比重越高。此兩項相乘成為 matching cot 匯聚的比重。

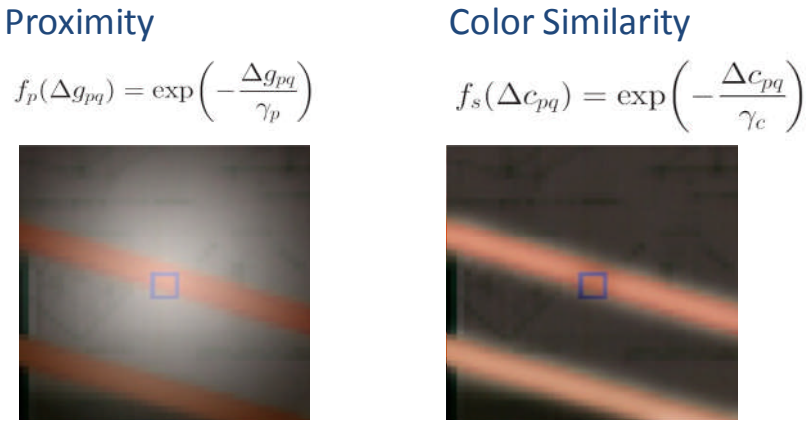


圖9. Cost Aggregation

為減少運算量，我們分析 Weight 有無 Proximity 在視差圖準確度上的變化。圖 10 顯示對於有 Proximity 的視差圖而言，在 Mask size 大於 39 時較無改變。並且，對於 Mask size 在 39 以內，無 Proximity 與有 Proximity 的視差準確度是相同的。因此我們可以省略 Proximity 於 Weight 中，並設定 Mask size 於 39 以內，即可達到相同於包含 Proximity 的效果。

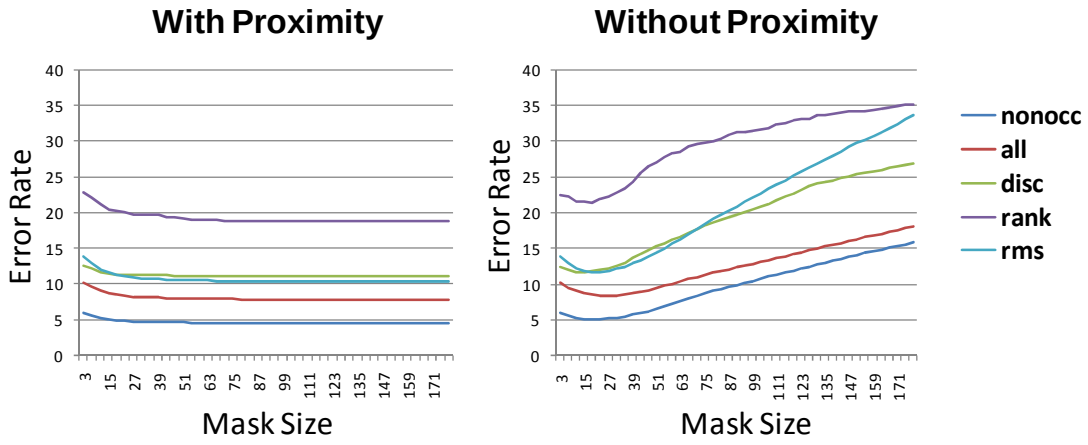


圖10. 比較有 Proximity 與無 Proximity 在視差圖準確度的差別

為提升 Weight Generation 的運算速度，我們在匯聚 matching cost 時分離原始運算成垂直匯聚(Vertical Cost Aggregation)與水平匯聚(Horizontal Cost Aggregation)兩步驟。利用以上所提之方法，可減少原始演算法運算量的 64.2%。另外，我們化簡 Color Similarity 的 exp 運算為 2 冪次的查表運算，各種運算化簡的方式如圖 11 所示。最後我們選擇圖中 x64, Quantize P1-bit 的簡化方式。

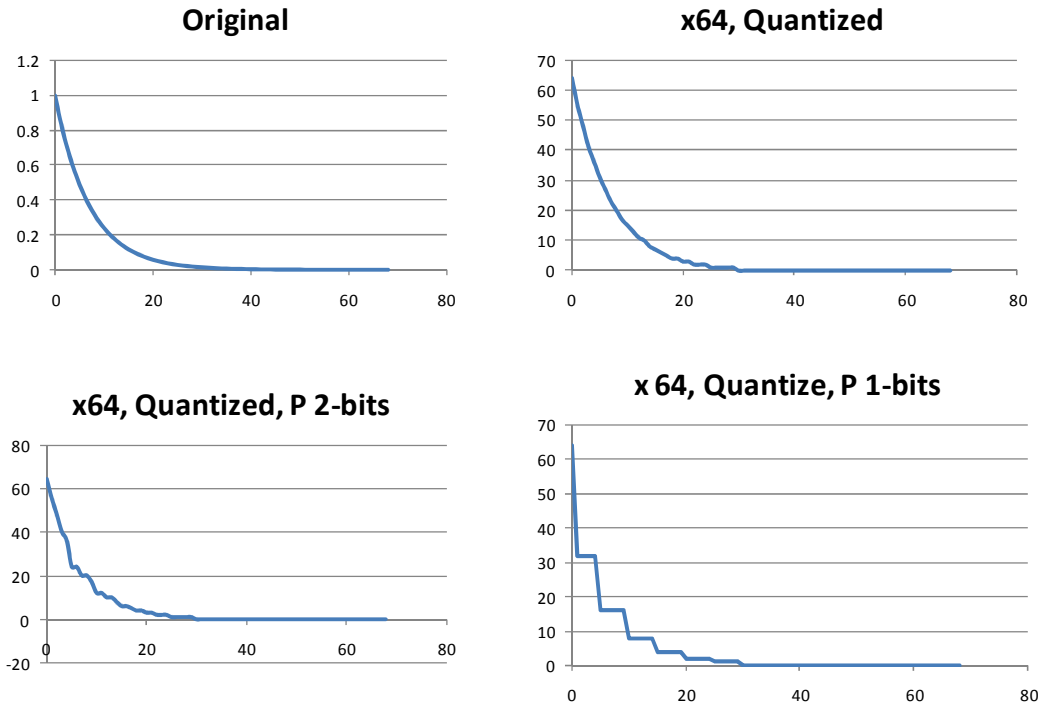


圖11. 比較 Color Similarity 的運算化簡

總和以上改變演算法以利於硬體設計的技巧，在個人電腦運算時間與視差圖準確度如下表所示。對於準確度約降低 1.2%以內，而運算時間大幅降低至原始運算的 1.9%。如此的硬體導向演算法將有利於進一步的硬體架構設計。

Method	Error Rate %				Exec. Time(sec)
	TSUKUBA	VENUS	TEDDY	CONES	
Original	1.85	1.19	13.3	9.79	95.65
+MC+2P	3.47	0.91	14.3	11.2	4.75
+MC+2P+ Manhattan	3.08	0.59	14	10.1	3.12
+MC+2P+ Manhattan +Truc(64,2)	3.03	0.61	14	10.1	2.52
+MC+2P+ Manhattan+Truc(64,1)	**3.06**	**0.66**	**13.9**	**10.1**	**1.84**

在硬體架構設計方面，我們針對 Mini-Census 及 Cost Aggregation 的運算分析各種資料對記憶體存取的方法。在 Mini-Census 有視差優先再利用(Disparity-Order Reuse)以及像素優先再利用(Pixel-Order Reuse)之方法。在 Cost Aggregation 有列局部再利用(Partial Column Reuse)以及行垂直延伸再利用(Vertically Expanded Row Reuse)。考慮運算量、記憶體用量以及頻寬用量，最後採用 Disparity-Order Reuse、Pixel-Order Reuse 與 Vertically Expanded Row Reuse 方法於硬體實現。硬體架構如圖 12 所示。此架構所有運算皆以握手機制(Handshaking)讀寫資料，以達到各運算處理時間的彈性。

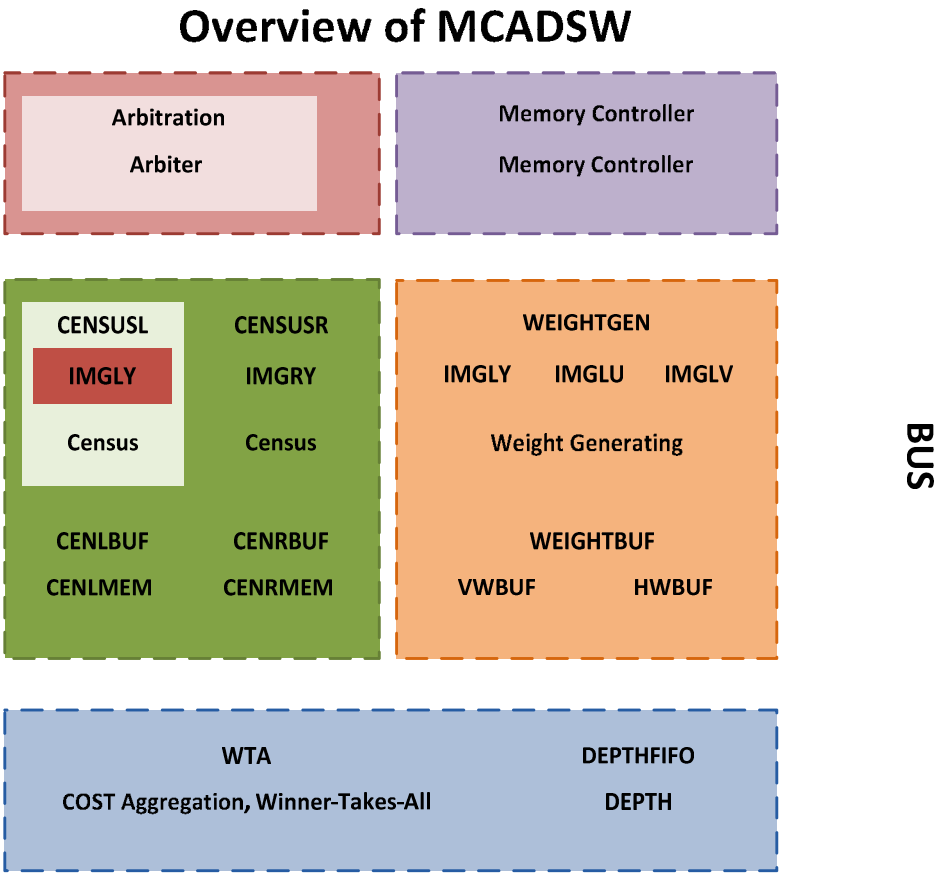


圖12. 視差估測架構圖

本硬體架構在聯華電子 90 奈米製程可工作於 100MHz 且 bus 寬度為 32-bit，可處理 CIF(352x288)影像大小且 64 視差範圍(disparity range)達到 42 fps。下表為合成的記憶體用量與組合電路的邏輯閘數量。總內部記憶體用量約 21k 位元組，總組合電路用量約 562k 邏輯閘。如此硬體效能已達到子計畫一的目標。

Performance under UMC 90nm Technology	
Clock rate	100 MHz
External bus width	32 bit
Image size	352x288
Disparity level	64
Logic	562k equivalent gate counts
Internal memory	21k bytes
FPS	42

對於演算法之效能，以下比較本演算法與其他文獻視差圖的錯誤率以及運算速度。我們的演算法可達高準確度且即時運算速度。

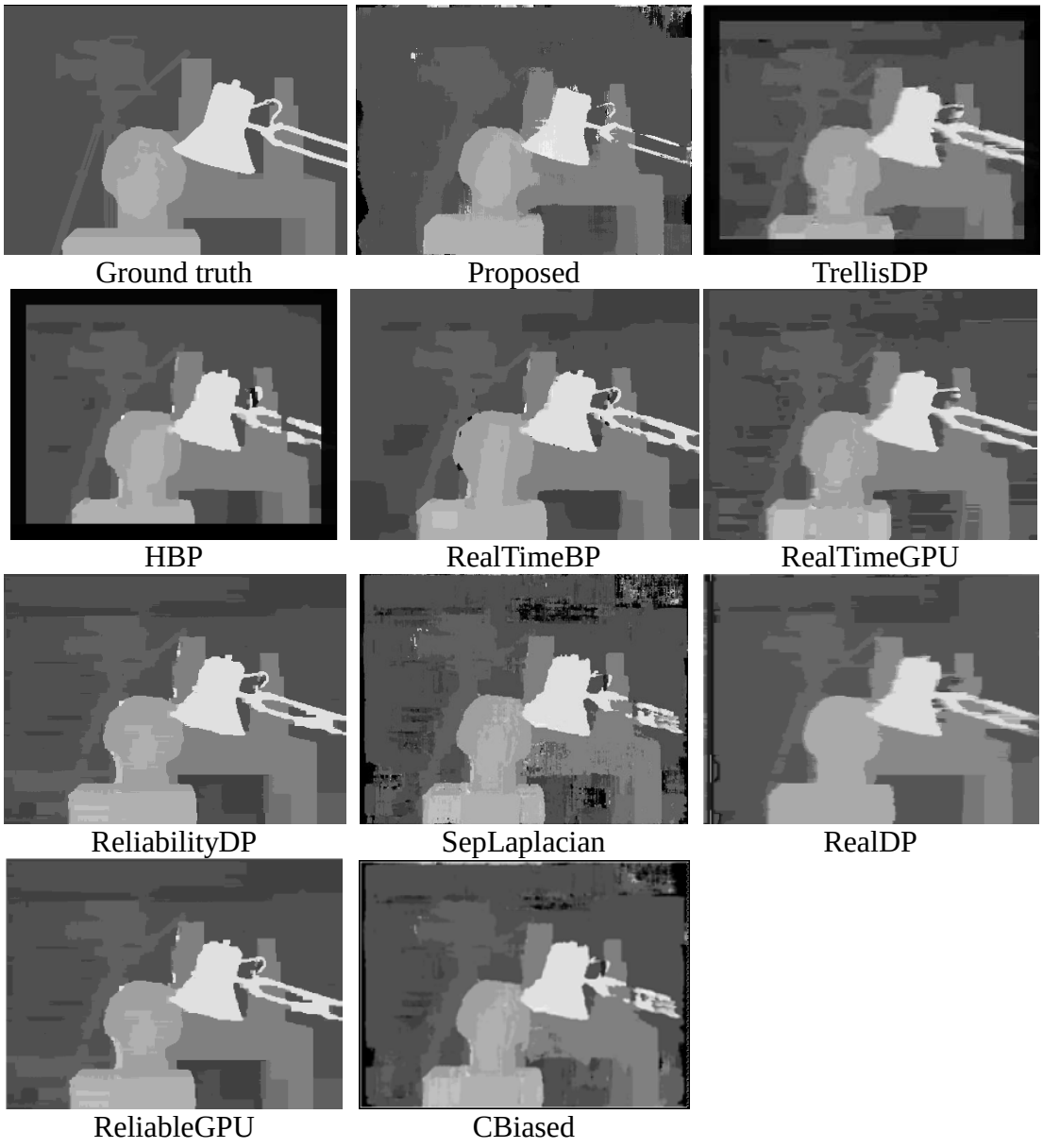


圖13. 視差圖比較

Design	Category	MDE/s	TSU	VEN	TED	CON	SAW	MAP
Proposed	**Hardware**	**272.5**	**2.80**	**0.64**	**13.7**	**10.1**	**2.11**	**3.21**
TrellisDP [1]	Hardware	294	2.63	3.44	-	-	1.88	0.91
HBP[2]	Hardware	73.7	2.85	1.92	-	-	6.25	6.45
EffectAggr [3]	CPU	18.9	2.96	3.53	10.7	4.92	-	-
RealDP [4]	CPU	209	2.85	6.42	-	-	6.25	6.45
Cbiased [5]	CPU+GPU	605	4.77	10.2	-	-	0.82	0.65
SepLaplacian [6]	CPU+GPU	679	13.0	-	-	-	-	-
RealTimeBP [7]	CPU+GPU	19.6	3.40	1.90	13.2	11.6	-	-
RealTimeGPU [8]	CPU+GPU	19.6	4.22	2.98	14.4	13.7	-	-
ReliableGPU [9]	CPU+GPU	-	1.36	1.09	-	-	2.35	0.55
GradientGuided[10]	CPU+GPU	117	2.48	3.91	-	-	1.63	0.73

四、計畫整體成果與自評

總合今年的成果，和原計畫提出的目標大致吻合，僅有部分目標稍有變更。完成與修改之目標如下幾點所列:

1. Local approach 方面，以分析軟體演算法之效果並移植於 DSP 加速運算，可達效能為 50fps@CIF 之速度。但因視差圖的準確度過差，故無將該演算法時作為硬體設計。計畫研究直接轉為發展 semi-global approach。

2. Semi-global approach 方面的演算法發展，初期以 mean-shift 做為 color segmentation 演算法與 semi-global approach 視差估測演算法結合。接著發展 color segmentation 演算法，但因所發展之演算法包含 recursive 運算，運算量過大而不適合硬體設計。最後，我們所提出的 semi-global approach 可使視差圖的準確度接近利用 mean-shift 演算法的效果。因此 color segmentation 演算法的發展中止。

3. Semi-global approach 方面的硬體設計，已完成該演算法的硬體設計，並提前達到子計畫一原定之運算效能。

4. Global approach 方面的演算法發展，已完成 belief propagation 的演算法及硬體架構規劃，但因為該記憶體及匯流排頻寬需求量過大，並且此演算法的視差圖準確度僅與 semi-global approach 相當。故轉而研究並發展提升 semi-global approach 的效能。

5. 已提前達到預定的處理速度且可產生高準確度的視差圖。

6. 人才培育方面，本計畫兩年來共有 1 位博士生及 2 位碩士生畢業。

7. 論文發表方面，本計畫共發表 5 篇國際研討會論文與 2 篇國內研討會論文。

子計畫一兩年來已提前達到預定的處理速度且可產生高準確度的視差圖，但目前開發的視差估測硬體設計仍舊無法整合於總計畫成為單一平台，其原因在於硬體設計面積過大。因此，子計畫一未來將著重於 semi-global approach 演算法的發展，並且降低硬體設計成本，並且期許能夠開發出處裡更大像素的視差估測硬體設計。

子計畫二成果：3D 繪圖加速器模組

一、前言

本子項目二之研究重點為可重組之3D繪圖加速器設計如圖14所示。因此目前我們分別在前端幾何轉換子系統中的打光運算單元以及後端繪圖子系統中的深度壓縮機制提出可重組式的演算法與對應的硬體架構，根據不同的輸入規格重組不同的硬體架構，達到系統的最佳效能。同時也已完成前後兩級的基本的硬體電路設計與FPGA模擬驗證平台。其他可重組/壓縮單元與整個重組加速器系統將持續進行研究與設計。

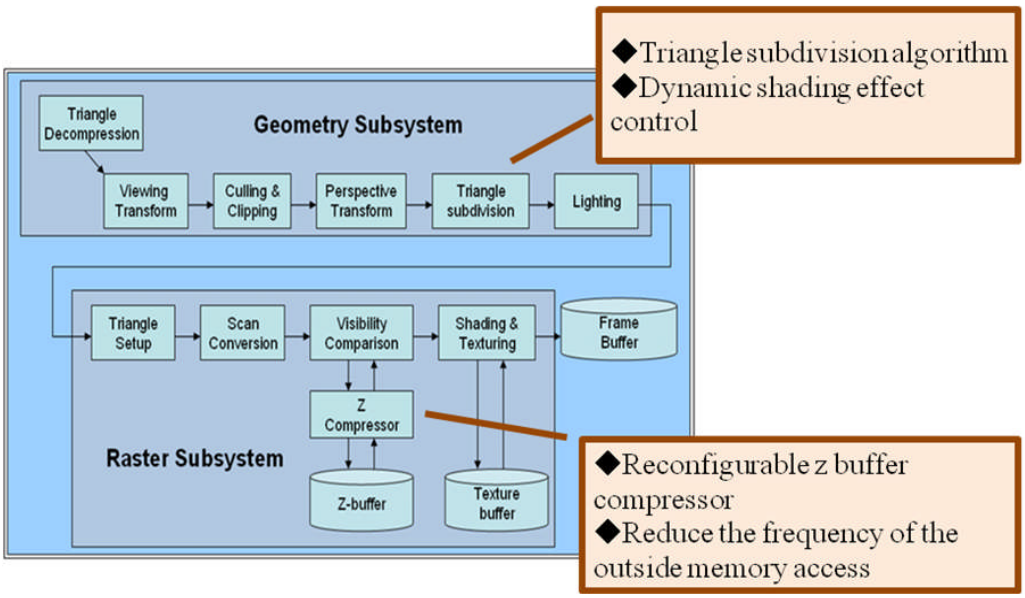


圖14. 重組式 3D 繪圖系統架構圖

二、研究目的

本計畫的 3D 繪圖技術與傳統 3D 視訊的不同點在於由於不同視角影像能使用 3D 繪圖技術即時繪製出來以節省大量頻寬，另一個優點是全虛擬的 3D 模型能夠提供全視角影像而不限於只能由有限的視角觀看，這給予互動式或虛擬實境的應用很大的自由度。在這個過程之中 3D 加速器扮演了相當重要的角色，主要是因為 3D 圖形成像過程中包含了複雜且大量的運算，如座標投影轉換的矩陣乘法，打光的內積、方根、倒數、次方等等，這些運算雖可用軟體來完成但是在高解析度複雜場景的即時顯示的需求下，其獨立的嵌入式 CPU 的架構與記憶體的頻寬並不能有效率地且即時地完成，若採高運算能力嵌入式處理器來達到所需的效能，將導致昂貴的硬體花費與嵌入式處理器使用率不高所造成的浪費。因此在兼顧效能與成本的考量下，需將這些運算使用特定的硬體加速器完成且要大幅減少其頻寬的需求，利用其平行與管線的硬體架構來提高 3D 繪圖的效能，使處理器僅需負責一般程式的執行、介面與硬體控制上，降低處理器需求的規格之餘不但硬體成本得以有效減少，效能也能提升，因此 3D 的相關應用在嵌入式系統比傳統桌上型電腦更需要 3D 繪圖加速器的輔助。

又因 3D 立體視訊的應用面非常廣泛且需求面與功能面非常多元，其硬體設計必須有能力根據不同解析度與不同 3D 圖形品質要求下提供具有不同壓

縮能力(也就是提供不同頻寬的能力)、不同打光能力與滿足即時的需求，所以此設計必須具備可重組性(Reconfgurability)，故子項目二之重點為可重組之 3D 繪圖加速器設計，本子項目二負責進行包括 3D 繪圖軟體平台開發、程式庫與介面開發、具重組暨低頻寬需求之 3D 繪圖加速器 Soft IP/單晶片的設計與其嵌入式平台整合。

因此子項目二目的為研發具有可重組式與低頻寬需求特性之硬體與軟體環境，讓開發者在合理硬體成本下能針對所繪製的場景與模型特型選擇適當的硬體組態來達成繪圖品質來滿足效能的要求。

三、結果與討論

本子計畫我們將各子區塊的實作情形以及目前進度分項列出，主要分為打光器的設計與硬體實現、深度緩衝區壓縮的演算法設計與硬體實現以及前後兩段幾何子系統與著色子系統的設計與硬體實現等，最後總結整個子計畫目前的實作進度與預期目標。

1. Approximating Phong Shading

本子計畫在打光器設計上主要有三項突破點以及實現結果，分別是：三角型切割之演算法與其硬體設計與實現、頂點變數共用機制、邊函數修正機制。以下分別討論之。

甲、 三角形切割演算法

圖 15 和圖 16 分別是兩種主流打光演算法及 Subdivision Based Shading 的結果，由這三張圖得比較後可知 Subdivision Based Shading 與 Phong Shading 的繪圖結果相當接近，而兩者間運算量得比較可參考圖 17，圖中黑色的部分表示須作打光運算的像素，由圖可知在打光效果相近的情形下，切割式渲染演算法大約可以省下近 50%的打光運算。

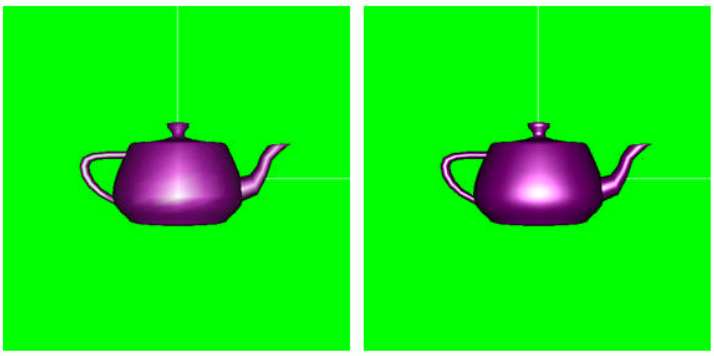


圖15. 左)Gouraud shading 的茶壺打光 右)Phong shading 的茶壺打光

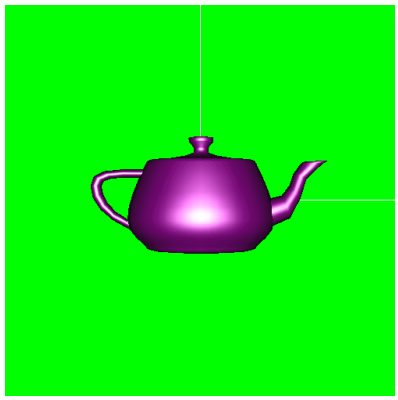


圖16. Subdivision Based Shading 的茶壺打光

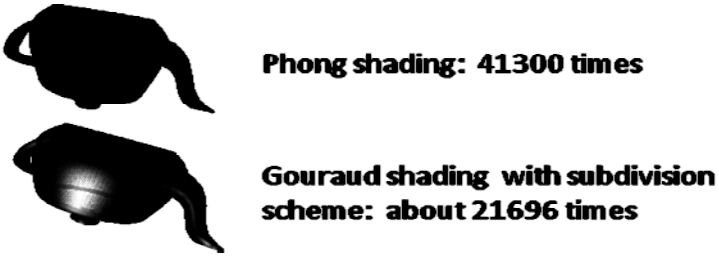


圖17. Phong shading 和 Subdivision Based Shading 運算次數比較

切割器是切割式渲染法的核心，而切割器不適合在軟體中實作的其中一個原因是切割時會產生新的頂點，若切割的動作是在處理器中完成，則切割出來的新頂點需要花費額外的頻寬將它們送到繪圖硬體上；反之若切割的動作是在繪圖硬體上完成則新頂點就不需要在匯流排上傳送，如此就可省下許多的頻寬而使繪圖效能提升。傳統的切割演算法使用遞迴的方式，但遞迴演算法不適合在硬體上實作，主要的原因在於堆疊的支援與切割出頂點的管理。由於遞迴式演算法會在共用邊上切割出相同的頂點，管理這些頂點所需的資料結構在硬體上實現相當複雜。但如果不使用管理頂點的機制而直接將產生的任何頂點直接送到打光單元中，就會對相同的頂點重複打光。打光的運算是相當耗時的，所以我們希望盡量避免重複的打光動作，為了兼顧硬體複雜度與效能我們使用了漸增式(Incremental)切割的方式來實作切割器。圖 18 說明漸進式切割的概念，要將一個三角形切成四個小三角形需要計算出在各邊上中點的頂點，而 dy 與 dx 可以事先計算，透過這種逐一加上一個向量的方式就可以循序地產生所有的頂點。由於頂點是依序被產生因此可以直接送到打光單元進行打光而不需要擔心重覆頂點的問題，因此也不需要額外的頂點管理機制。漸增式切割可以很容易的進行任意數量的切割運算，如圖 19 所示，只要計算出需要的 dx 與 dy 向量並透過漸增的方式即可產生所有需要的頂點。

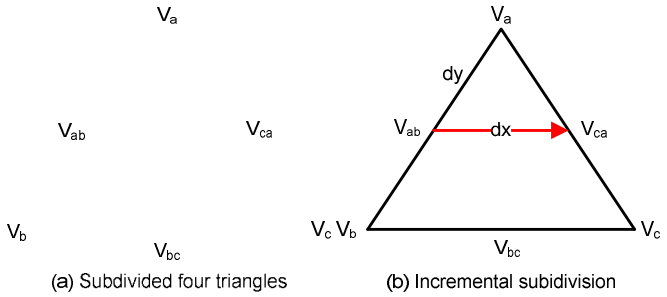


圖18. 漸增式切割演算法

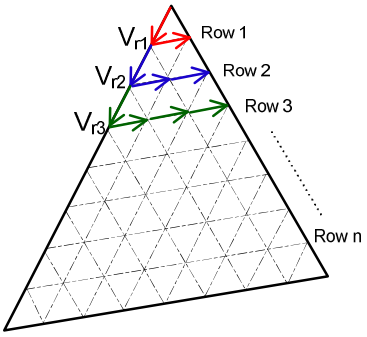


圖19. 任意數量的漸增式切割

乙、 頂點變數共用機制

而經由切割器切割後產生的許多小三角形無疑的會對效能造成衝擊，頂點數量增加意味著需要更多的幾何轉換，三角形的數量增加也代表三角形準備的運算和後端著色子系統所有的效能負擔的增加，因此前後端如何互相配合才能不影響系統效能也成為我們的研究重點。在後端我們透過了係數共用的機制便可以減少一半以上的三角形準備運算，如圖 20 所示，共用的係數一旦被計算出來便可以提供給所有切割出來的三角形使用而不需要重新計算這些係數。表 4 整理了變數共用機制前後的運算量比較，可以看出使用這套方法確實能有效的降低運算複雜度進而減少切割式渲染演算法對系統效能的影響。

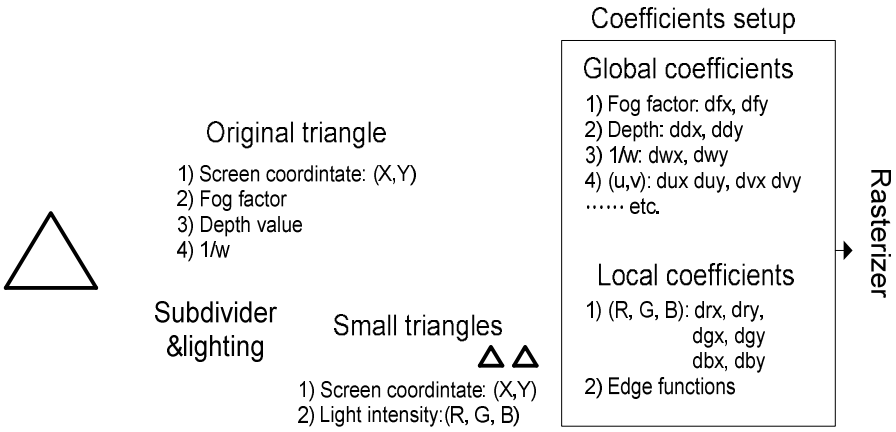


圖20. 係數共用機制圖

表4. 變數共用機制前後運算量之比較表

	Conventional subdivision algorithm	**Proposed subdivision algorithm**	**Complexity reduction in percent**
Number of the lighting operations	24	15	37.5%
Number of the 4x4 matrix multiplications for perspective transformation	24	3	87.5%

Number of the clipping/culling test operations	16	1	93.75%
Number of the 3x3 matrix multiplications for setup operation for rasterization	80	27	66.25%
Rasterization anomalies	Sometimes occur	Completely eliminated	

丙、 邊函數修正機制

第二個問題來自於切割過程中產生的誤差，如圖 21 所示，原本在共用邊上的頂點應該要在相同的位置上，但是切割時的誤差使得最後的位置產生落差，因而造成像素化異常的現象，如圖 22 之左所示，我們在畫面上可以看到物體上因為像素化異常而產生破洞。為了處理這個問題，我們也提出以原本三角型的頂點取代部分切割後的頂點的方法，成功的解決了這個問題，如圖 22 之右所示所有的破洞已經完全消失。

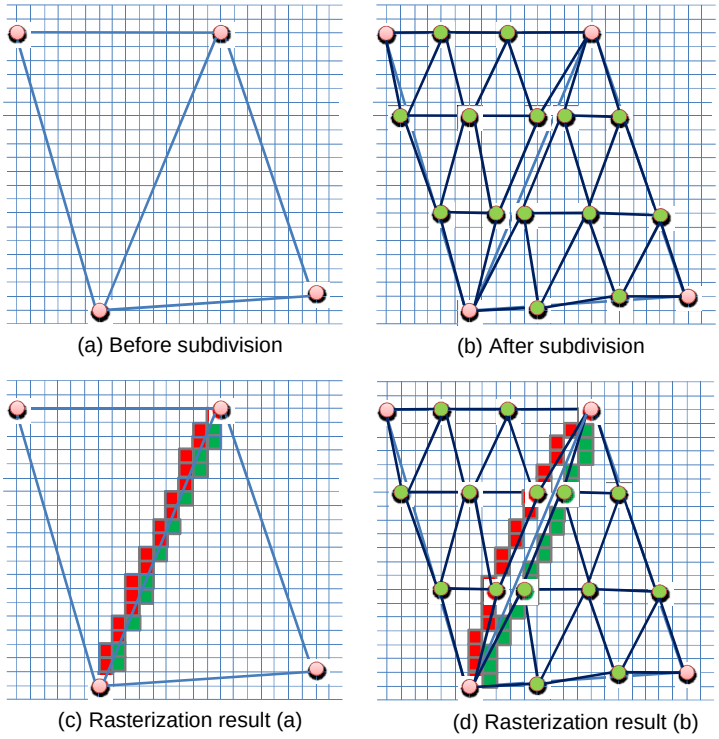


圖21. 破洞圖解說明

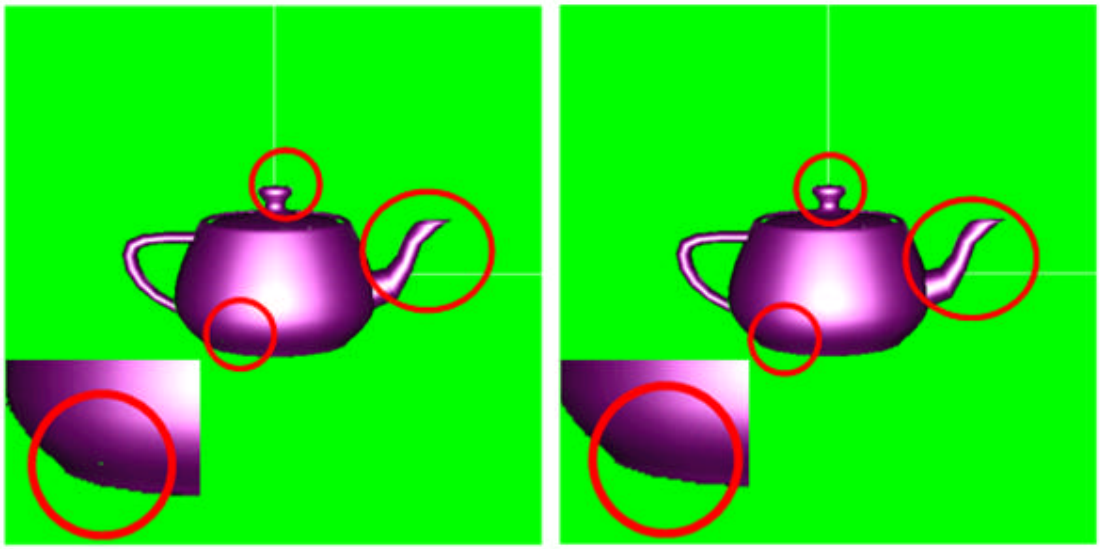


圖22. 消除破洞模擬結果

2. 可重組深度緩衝區壓縮器

目前此子項的硬體設計所提供使用者的壓縮演算法共有 Modified HA, Modified DDPCM, General DDPCM, one/two plane mode…等等數種主流的壓縮法，並可自動判斷輸入的 3D 場景適合何種壓縮法。經由模擬的驗證已經確定我們所提出的演算法能夠有效提升平均壓縮率，並改善約 38.7%至 76.9%，目前已經進入硬體驗證階段。而圖 23a)藍色曲線為適應性壓縮演算法的壓縮比，紅色曲線則為目前被廣泛運用之 DDPCM 深度壓縮演算法之壓縮比；b)同樣場景下適應性壓縮演算法(藍色)和另一種廣泛運用之 HA 深度壓縮演算法之壓縮比比較。三種演算法(proposed：DDPCM：HA)壓縮比比較為 1.91：1.37：1.09。

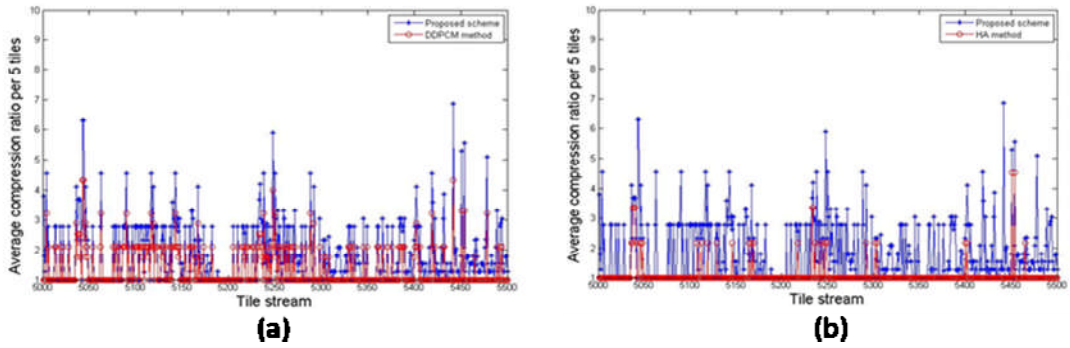


圖23. 常用演算法和 proposed 演算法之壓縮比之比較

3. Rasterizer Subsystem

在後端 Rasterizer Subsystem 部分，其系統流程圖如圖 24，總共可以區分成三塊主要子系統，包含 Triangle Setup Engine 負責將前級系統的 Triangle-level 轉換成後級的 Pixel-level；Depth Testing Unit 負責測試每個像

素(pixel)的深度值，決定該像素是否需要顯示在螢幕上；Texture Mapping Unit 負責替每個像素套上該場景下的色彩。接下來會分別深入探討三個子系統的實際設計理念。

另外為了提高後級著色子系統的效能以配合前級三角型切割所造成的額外負擔，我們在計算材質貼圖的模型精確度(LOD)參數時，採取漸進式的演算法，利用多邊形和其相對應的貼圖材質模型間固定的長寬比關係，在第一次計算 LOD 參數時額外計算一次該模型位置長寬和 LOD 參數間的差值關係，之後即可用簡單的加減法運算取代整套複雜的連續乘加運算，以加快系統速度與節省功率消耗。

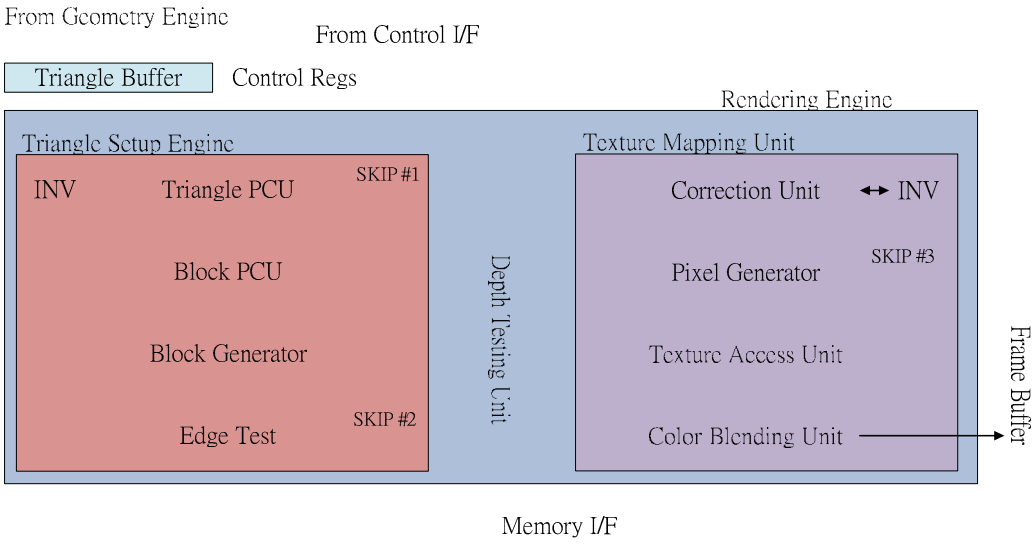


圖24. 著色子系統的細部流程圖

甲、 Triangle Setup Engine (TSE)

主要工作在於將前級的 Triangle-level 轉變成後級的 Pixel-level。首先 TSE 會接收來自前級傳送過來的三角形三個頂點(Vertex)資訊及控制訊號，然後透過 Triangle PCU 和 Block PCU 算出後端各級漸進式演算法的初始值，使之後的各級管線省下一連串複雜的初始乘除法運算，而可以用較容易的加減法運算代替，之後經由 Block Generator 產生一連串該三角形內所包含的 8x8 block，最後再經由 Edge Test 檢測各 block 與該三角形間的幾何關係以剔除 ghost block，以減少後半部元件多餘的資料量，整個流程結束後，會將每個像素資料傳給深度測試元件(Depth Testing Unit)。另外，TSE 中的 SKIP #1 與 SKIP#2 是分別將太小的三角形清除與空的 Block 清除。

乙、 Depth Testing Unit

此元件的主要功能為減少 Texture Mapping Unit 的運算資料量，由於 3D 模型轉換到 2D 座標系統時，許多物件的像素有可能對應到相同座標，但其中只有一個像素需要顯示在座標系統上，於是透過比較深度值的方式，決定哪個像素的 Z 值最小，表示最接近人的眼睛、及需要顯示在螢幕上，並清除其他像素的資料，進一步減少後端貼圖運算單元的

運算量。

丙、 Texture Mapping Unit

Texture Mapping Unit 就是傳統的 Pixel Process，主要在於計算每個像素的色彩資訊，其中包含了透視角校正機制(Correction Unit)、像素產生器(Pixel Generator)、材質讀取元件(Texture Access Unit)和色彩混和器(Color Blending Unit)。透視角校正機制接收來自 TSE 的資料，進行貼圖座標的透視角校正，而像素產生器則接收來自深度測試元件的資料，產生相對應的像素資料，之後透過材質讀取元件從記憶體讀取貼圖資訊，最後再利用色彩混和器將之前的色彩與記憶體讀取的色彩做混和的動作，並把結果傳送到 Frame Buffer。

4. FPGA 驗證平台

下表 5 為 3D 管線前後兩個子系統在 Xilinx Spartan 3A DSP 開發板所使用的 FPGA 資源，由此表看來無論是 FPGA 的 Slice 使用量或是特殊運算單元 DSP48A 的使用量都算是十分吃緊，因此在之後本子計畫與整合平台將會移到 Xilinx ML507 開發板。

表5. 各子區塊所占 FPGA(Spartan 3A DSP 1800)資源統計表

Module name	slices	Slices Flip Flops	4 input LUT	DSP48A
Geometry Engine	4069	3605	6168	35
Raster Engine	4220	4776	7776	44
Total	8289	8381	13944	79
%	24%	50%	42%	94%

目前整合在 Xilinx Spartan 3A DSP 開發平台上的硬體系統架構如圖 25。由 FPGA 中的 MicroBlaze processor 啟動 FPGA 的中幾何轉換子系統，而幾何轉換子系統抓取記憶體中的頂點和顏色資料運算完後，會把處理後的三角形資料傳回 MicroBlaze 做之後的繪圖動作，最後 MicroBlaze 把處理後的像素資料寫入畫面緩衝區。

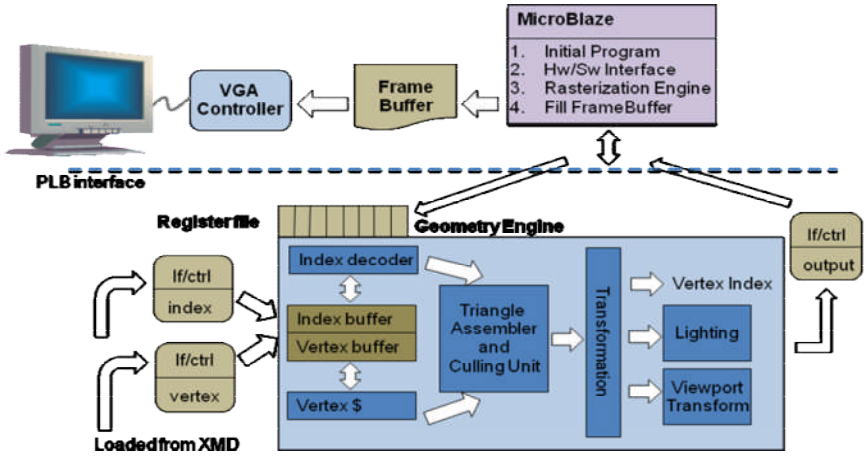


圖25. FPGA 驗證平台之系統架構

本子計畫在 Xilinx ML507 開發板所使用的 FPGA 資源如表 6 所示，目前所使用資源不到百分之五十，故整合平台與其他子計畫能夠一起整合在此開發板中。而由於後端著色子系統中有使用到多週期路徑(multicycle path)，因此系統顯示的最快時脈只有 40MHz，實際上系統可乘載的周期能夠和前端幾何子系統配合，約在 100MHz 左右。

表6. 各子區塊所占 FPGA(ML507)資源統計表

Device : Vertex-5 5vfx70t ff1136 -1			
	Geometry Engine	**Raster Engine**	**Total**
Slice Logic Utilization			
Slices Register	7361(16%)	14074(31%)	47%
Slices LUTs	5823	8844	31%
Number used as Logic	5644	8844	31%
IO Utilization			
Number of IOs	902	1601	-
Number of bonded IOBs	0	0	-
Specific Feature Utilization			
Number of DSP48E	35	28	49%
Maximum Frequency	153.86 MHz	40.912 MHz	

四、計畫整體成果與自評

本子計畫在本年度執行到目前，各項預期進度均依據計畫的執行步驟進行，無論是理論面、實作面或是應用面都有良好的成果展現。設計實現方面，部分研究重點的子方塊也已經進入硬體驗證階段，呈現前端幾何轉換子系統、後端繪圖子系統以及 FPGA 開發板相關流程三大主軸平行進行，本子計

畫除了在演算法方面提出可重組式的架構，另外也進行基本管線的實作與 FPGA 驗證，同時研究成果也陸續發表在 IEEE 的期刊與國際研討會上，例如 IEEE Trans. Computers，2009 年的 ISCAS、2008 年的 ICME、MUE。

目前本子計畫已在 Xilinx Spartan 3A DSP 開發板上與子計畫四部分整合完成，其系統展示圖如圖 26(左)，畫面中央綠色背景的視窗中即為本子計畫 3D 繪圖引擎中前段幾何轉換子系統在 FPGA 中即時運算模擬的結果。另外由於 Spartan 系列 FPGA 容量不足以支援本子計畫完整設計的整合驗證，因此計畫轉移展示平台到 Xilinx ML507 的開發板，其開發中的畫面如圖 26(右)。未來將會繼續以 ML507 為主要驗證平台，完成整體 3D 電腦繪圖引擎的設計與驗證。

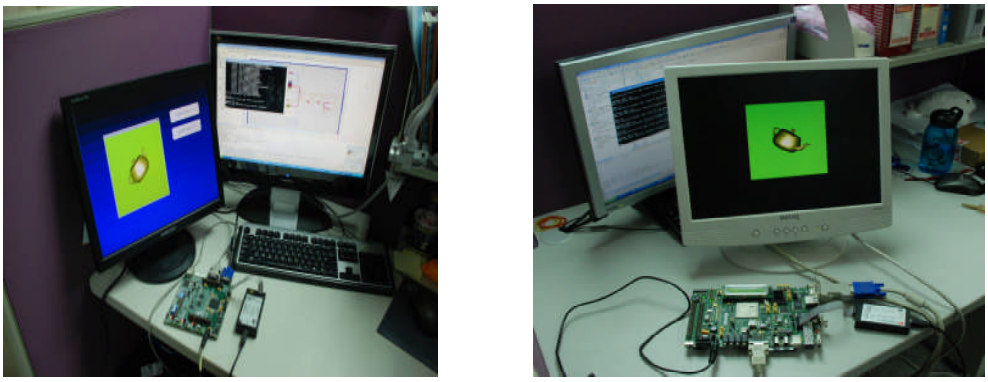


圖26. 左)Spartan 3A DSP Demo 展示平台 右)ML507 Demo 展示平台

過去學界雖不乏有類似本子計畫的 3D 電腦繪圖加速晶片研究或設計，但均以獨立高效能晶片為主軸；而本子計畫採取的是以可以應用在各種不同領域的平台、提供可重組高能源效率的加速電路，以提供更有彈性的加速功能與更優質的 3D 立體畫面及人機介面。

子計畫三成果：立體視訊顯像模組

一、前言

本文簡述子計畫三前兩年的研究成果。本子計畫的研究目標，在於深度圖因視訊壓縮所造成之量化效應(Quantization Effect)下，利用參照影像間隱藏之深度資訊，修補失真之深度圖。本子計畫提出了一個檢測單一像素之合成誤差的模型(Per-pixel Synthesis Distortion Model)，透過此模型，可分析各種造成影像失真的原因；此外，基於此分析模型，發展出合成品質導向之深度圖修補演算法，此演算法可有效偵測出有問題的深度值，並進行修正。此演算法相較於現行的 MPEG FTV 標準架構下，平均多 1.2 dB，且在主觀視覺上也較接近於原始影像。

二、研究目的

深度圖(Depth map)經過現行視訊壓縮標準的編解碼後，將造成解碼重建後深度圖有不同程度的失真。這些失真可能會影響到最終虛擬影像的合成品質，例如在物體邊界的深度值發生錯誤時，可能明顯地造成主、客觀影像品質評量上的降低。因此，本子計畫針對重建後的深度圖對虛擬影像合成的影響，提出一個單一像素之合成誤差的檢測模型。透過此模型之分析，可有效預測出每個像素可能造成的合成誤差。利用不同參照視角(Reference Views)畫面之間對應點關係，找出隱藏的深度資訊，並依此修正不可靠的深度像素(Unreliable Depth Pixel)，使其在虛擬影像合成上可得到較好的品質。採用現行最新的視訊壓縮標準對原始深度圖(圖 27) 進行壓縮後，並再解碼重建深度圖 (圖 28)。主觀視覺比較上，可在圖 27 與圖 28 看見明顯的落差，例如圖 28 中，人背上的深度影像產生明顯的區塊效應(Block Effect)。

目前 MPEG 標準制定小組已出現兩篇於接收端修正深度圖的演算法：第一篇由 Tanimoto 提出，其利用合成相鄰參照視角時的合成誤差，以 Linear Prediction 的方式，推測出虛擬視角可能的合成誤差，最後將此預測出來的合成誤差補回虛擬視角之合成影像；第二篇由 Sung 提出，其利用合成虛擬視角時，來自不同參照畫面的亮度值與深度值的差值，若此差值越過某個門檻值，則判定此深度值需要被修正；其次，此方法為每個 Connected Component 找出一個最佳的深度值偏量，使得合成誤差可以降到最低。然而，上述方法未考慮深度值壓縮的情況，兩者皆無法根據不同的壓縮品質進行調整，造成修改的效能嚴重受到壓縮之品質所影響。

為解決上述問題，子計畫三設計一個單一像素之合成誤差的檢測模型，於理論上分析虛擬影像合成時，所有可能影像合成品質的因素。根據 Depth-Image-Based Rendering(DIBR)描述參照視角和虛擬視角的對應點關係，如公式(1)：

$$\Psi:\begin{bmatrix}\mathbf{p}\\1\end{bmatrix}\mapsto\begin{bmatrix}\mathbf{p}'\\1\end{bmatrix}=\mathbf{A}'\mathbf{R}\mathbf{A}^{-1}\begin{bmatrix}\mathbf{p}\\1\end{bmatrix}+\frac{1}{Z_p}\mathbf{A}'\mathbf{T} \tag{1}$$

其中 $\mathbf{p}$ 和 $\mathbf{p}'$ 分別為參照視角和虛擬視角的對應點座標，$\mathbf{R}$ 和 $\mathbf{T}$ 為虛擬相機的旋轉矩陣與位移向量，$\mathbf{A}$ 和 $\mathbf{A}'$ 分別為參照相機和虛擬相機的相機內部參數，本

文以$\Psi(\mathbf{p};Z_p)$簡化表示公式(1)之運算子。在深度圖沒有失真的情況下，可確保$\mathbf{p}'=\Psi(\mathbf{p};Z_p)$，且亮度值相同$I_T(\mathbf{p}')=I_R(\mathbf{p})$；若深度圖有失真($Z_p$加上雜訊，$\hat{Z}_p=Z_p+n_p$)，以$\mathbf{q}'=\Psi(\mathbf{p};\hat{Z}_p)$表示，則合成誤差$\xi_p$可由泰勒展開式逼近：

$$\begin{aligned}\xi_p &= \left(I_R(\mathbf{p})-I_T(\mathbf{q}')\right)^2 = \left(I_R(\mathbf{p})-I_R(\mathbf{q})\right)^2 \\ &\approx \left(I_R(\mathbf{p})-I_R(\mathbf{p})-\nabla I_R(\mathbf{p})\cdot(\mathbf{q}-\mathbf{p})\right)^2 \\ &= \left(-\nabla I_R(\mathbf{p})\cdot(\mathbf{q}-\mathbf{p})\right)^2\end{aligned} \tag{2}$$

其中$\mathbf{q}=\Psi^{-1}\left(\mathbf{q}';Z_q\right)$，$\nabla I_R(\mathbf{p})$為$\mathbf{p}$的 Gradient。利用$\mathbf{p}$、$\mathbf{q}$和$\mathbf{q}'$三點的關係$\mathbf{q}'=\Psi\left(\mathbf{q};Z_q\right)=\Psi\left(\mathbf{p};Z_p+n_p\right)$，可將$(\mathbf{q}-\mathbf{p})$可表示為公式(3)：

圖27. 原始深度圖.

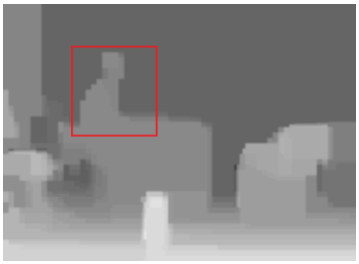


圖28. 解碼後之深度圖.

$$(\mathbf{q}-\mathbf{p})=\frac{-n_p}{Z_q\left(Z_p+n_p\right)}\mathbf{c} \tag{3}$$

其中$\mathbf{c}=\left[\mathbf{I}_2 \;\; \mathbf{0}_{2\times1}\right]\mathbf{A}\mathbf{R}^{-1}\mathbf{T}$。將$(\mathbf{q}-\mathbf{p})$帶入公式(2)，並導出下式：

$$\xi_p \approx \left(\frac{n_p}{Z_q\left(Z_p+n_p\right)}\nabla I_R\left(\mathbf{p}\right)\mathbf{c}\right)^2 \tag{4}$$

在 MPEG FTV 的水平相機設置的限定下，位移矩陣只含水平方向之分量，因此可將公式(4)簡化為下式：

$$\xi_p \approx \left(\frac{n_p}{Z_q\left(Z_p+n_p\right)}\right)^2 \times g_x^2\left(\mathbf{p}\right)\times c^2 \tag{5}$$

其中$g_x^2\left(\mathbf{p}\right)$為$\nabla I_R\left(\mathbf{p}\right)$的水平分量。因此，假設有真實(Ground-truth)深度資訊的情況下，計算出合成影像誤差的條件期望值：

$$
\begin{aligned}
& E\left\{\xi_p \mid Z_p, Z_q\right\} \\
& \approx E\left\{\left(\frac{n_p}{Z_q\left(Z_p+n_p\right)}\right)^2 \mid Z_p, Z_q\right\} \times m_g^{(2)}(\mathbf{p}) \times c^2 \\
& =\frac{1}{Z_q^2} \times\left(\frac{E\left\{n_p^2\right\}}{Z_p^2}-2 \frac{E\left\{n_p^3\right\}}{Z_p^3}+3 \frac{E\left\{n_p^4\right\}}{Z_p^4}-\cdots\right) \times m_g^{(2)}(\mathbf{p}) \times c^2 \qquad (6) \\
& =\frac{1}{Z_q^2} \times\left(\frac{\sigma_n^2(\mathbf{p})}{Z_p^2}+9 \frac{\sigma_n^4(\mathbf{p})}{Z_p^3}+75 \frac{\sigma_n^6(\mathbf{p})}{Z_p^4}-\cdots\right) \times m_g^{(2)}(\mathbf{p}) \times c^2 \\
& \approx \frac{1}{Z_q^2} \times \frac{\sigma_n^2(\mathbf{p})}{Z_p^2} \times m_g^{(2)}(\mathbf{p}) \times c^2
\end{aligned}
$$

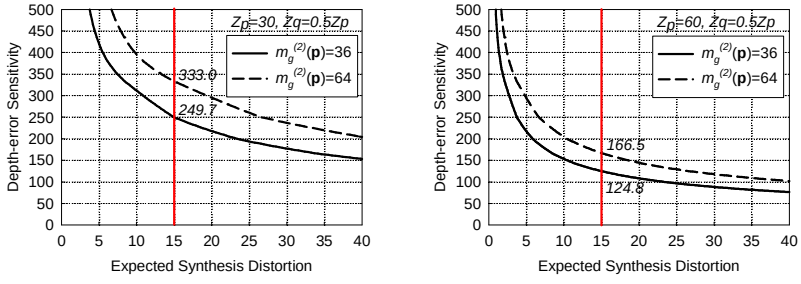
其中 $m_g^{(2)}(\mathbf{p})=E\left\{g_x^2(\mathbf{p})\right\}$，$\sigma_n^2(\mathbf{p})$ 為 n_p 之變異數，並假設 n_p 之數值呈現常態分布 $n_p \sim N\left(0, \sigma_n^2(\mathbf{p})\right)$。透過公式(6)的推導結果得知，單一像素之合成品質係由四個因素共同影響：Depth-error Variance、Intensity Variation、Ground-truth Depth Value、Virtual Camera Location。本文給定公式(6)各種不同的參數設定，並將結果描繪於圖 29 中，並提出四項詳細之觀察：

(1) 當 $m_g^{(2)}(\mathbf{p})$ 越大，則 Depth-error Sensitivity($Z_p / \sigma_n(\mathbf{p})$)越高，可歸納出物體邊界或複雜的圖像內容，這些區域之深度值較不可靠。

(2) 比較圖(a)(c)(e)與(b)(d)(f)，當景深越深，其造成之合成誤差相對越小。

(3) 比較圖(e)與(a)(c)（或是圖(f)與(b)(d)），若 $Z_q > Z_p$，其 Depth-error Sensitivity 會比 $Z_q < Z_p$ 來的小。由圖 30 得知，當 $Z_{q1} \le Z_p \approx Z_p \le Z_{q2}$，$\mathbf{p}$ 相對於 $\mathbf{q_2}$ 為一個背景的點，使用背景的點覆蓋前景的點，將產生較嚴重的合成誤差；反之，$\mathbf{p}$ 相對於 $\mathbf{q_1}$ 為一個前景的點，此覆蓋情形較不會出現明顯的合成誤差。

(4) 由公式(6)的 $\frac{\sigma_n^2(\mathbf{p})}{Z_p^2}$ 和 c^2 可知，若虛擬視角與參照視角的距離越遠，其 Depth-error sensitivity 也會越高。



(a) (b)

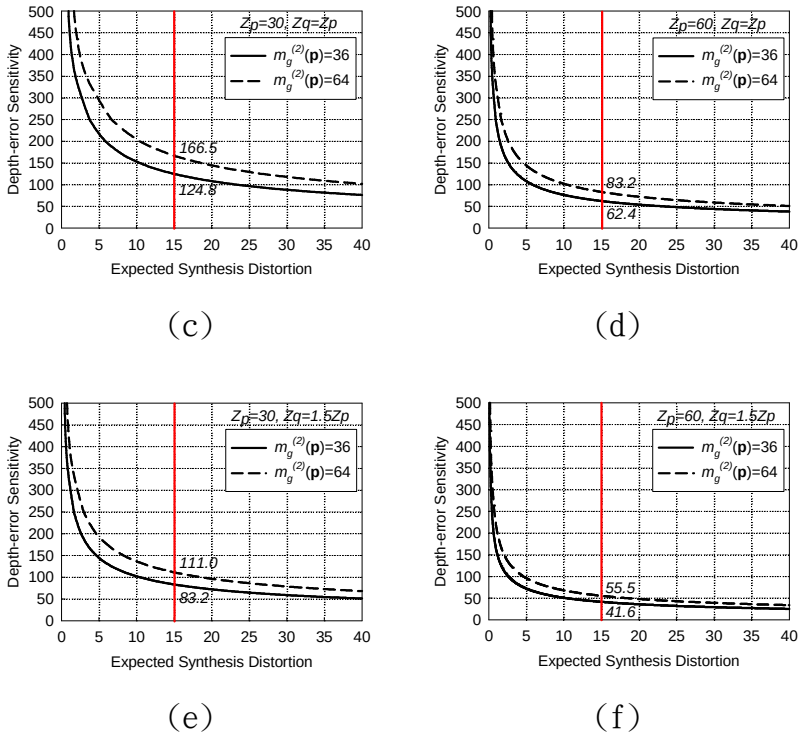


圖29. Measuring the depth-error sensitivity under various settings of Z_p, Z_q and $g_x^2(\mathbf{p})$.

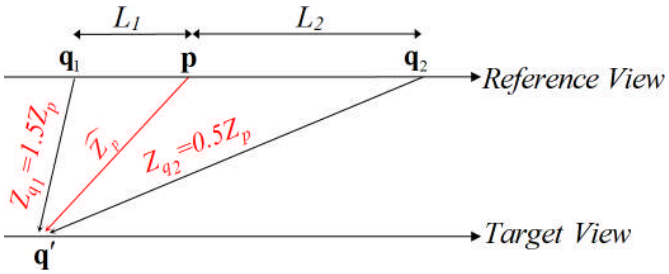


圖30. A geometrical interpretation of the effect of Z_q on depth-error sensitivity.

三、結果與討論

基於 MPEG-FTV 的架構，圖 31 提供子計畫三研究之深度圖修正演算法的系統架構圖，其中圖中虛線方塊處為本計畫提供的模組。影像和深度圖從傳送端被壓縮後，傳至接收端重建影像和深度圖並合成影像。深度圖經過壓縮後會造成許多的幾何誤差，進而可能產生影像合成上的錯誤。因此，本子計畫針對重建後的深度圖對虛擬影像合成的影響，提出一個單一像素之合成誤差的檢測模型(Per-pixel Synthesis Distortion Model)。透過此模型之分析，可有效預測出每個像素可能造成的合成誤差，並在傳送端計算修改深度圖後可能的合成誤差，使其在接收端之虛擬影像合成上，可得到主客觀視覺品質的提升。

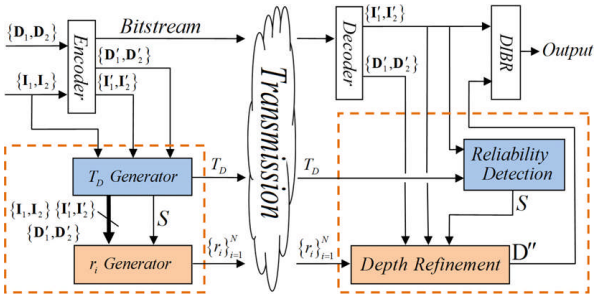


圖31. System Block Diagram.

子計畫三所提出來的深度圖修正演算法有下列特點。首先，在不改變 MPEG-FTV 架構的前提下，對深度圖進行修正。第二，在傳送端作分析，利用不同參照視角之間的對應點關係，若對應點的 Intensity 差異超過門檻值，則此深度值可能有問題，並將 Intensity 差值稱 Synthesis Error。此外，對應點關係隱含了深度資訊在其中，本計畫以此為修正解壓縮後的深度圖之基礎。其三，分析重建後深度圖對於虛擬影像合成的影響，並在傳送端模擬虛擬視訊合成後所產生的誤差，此誤差包括原始與重建後的深度圖之間的差異、分析並產生參數供接收端使用，我們需判斷哪些深度值可能有問題與深度值搜尋範圍，接收端就可以根據這些參數修正解壓縮後的深度圖。

由圖 32 可以得知子計畫三提出的演算法和 Tanimoto、Sung 的演算法相較於 MPEG-FTV 的 PSNR 還高，其原因是此三者演算法都有對深度圖和合成影像進行修補；不過當深度圖失真越大時，子計畫三的演算法會比 Tanimoto 和 Sung 來的好，原因是這兩者演算法並沒有考慮到深度圖壓縮的問題，因為壓縮後的深度圖已經遭到嚴重的破壞，若再使用它的話，合成品質會非常糟糕。從主觀視覺圖 33)來看，也可以發現子計畫三的演算法相較於其他三者，也有更好的觀看品質。

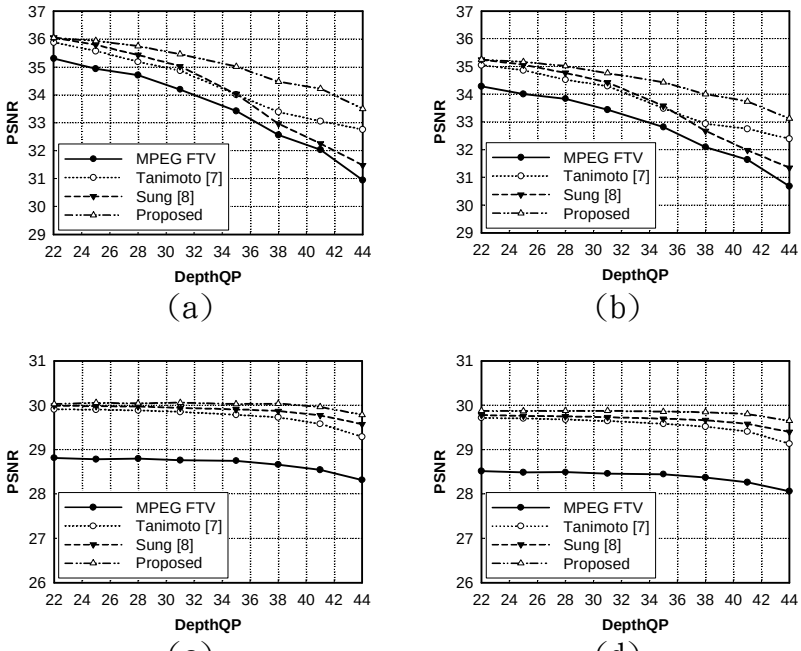


圖32. PSNR of synthesized images as a function of the depth and reference QP. The reference view images are coded with QP=22 (a)(b) and QP=31 (c)(d).

圖33. Subjective quality comparison of synthesized images: (a) MPEG FTV(without depth refinement), (b) Tanimoto, (c) Sung, (d) the proposed scheme. The depth QP is set to 44.

四、計畫整體結果與自評

子計畫三進度和原本的研究目標不盡相同，我們做了以下的解釋：

(1) MVC (Multi-view Video Coding) 標準已經在今年 (2009) 三月結束制定的工作。在所有 MVC 的參照軟體 JMVC 所接受的多視角視訊壓縮工具中，尤以視角間預估 (Inter-View Prediction)、亮度補償 (Illumination Compensation) 與運動向量省略模式 (Motion_Skip Mode) 等技術引起最多與會者的討論。對比於 H.264/AVC 壓縮標準，JMVC 可以帶來平均約 20% 的位元率 (Bit-Rate) 的節省，但仍舊未達當初 MVC 在 Coding Efficiency 上制定的目標。在考慮不同工具在壓縮上所需要付出額外的複雜度，與其所能帶來於 Coding Efficiency 的改善，因此 MVC 僅採用在 Anchor View 上進行視角間預估。因 MVC 標準不接受 Macro-block Level 以下之 Syntax 的改變，只接受視角間預估為標準當中的壓縮工具，有鑑於此，我們把研究發展發向改為與 MVC 並行的 FTV 標準。

(2) MPEG-FTV 計畫主要目標在標準化以下三個子項：深度圖估測演算法、虛擬視訊合成演算法與多視角影像與深度圖壓縮標準。其中壓縮標準目前考慮使用 MVC，壓縮對象包含多視角視訊與其對應的深度圖。FTV 與 MVC 分別為 MPEG 與 JVT 並行的計畫，MVC 已於今年結束制定的工作；而 FTV 目前專注在深度圖估測知正確性與虛擬視訊合成品質上的提升。FTV 為基於多視角 DIBR (Depth-Image-Based Rendering) 的虛擬視訊合成技術，由子計畫的研究發現，現行壓縮標準在壓縮深度圖上，對合成視訊的品質造成許多影響，特別是影像上高對比的區域與深度值較小的區域，合成品質較容易受

到深度圖壓縮的影響。深度圖壓縮是下個階段 FTV 所要專注的重點目標，同時也是許多國內外學者提供解決方案的重點項目，因此本子項轉向提出深度圖對合成影響之模型，藉此模型幫助深度圖壓縮的設計。目前本子計畫已建立壓縮深度圖對合成影像影響之模型，藉由此模型偵測壓縮後受影響的深度像素，進而修正此像素後縮小合成之誤差。初步研究上，平均合成效能於 PSNR 表現上，相較於 FTV 標準已經產生 0.56dB 的提升，在低位元率的深度圖表現上，更有 0.45-3.2dB（平均 1.2dB）的改善，主觀視覺更可以有效消除影像高頻區段的合成誤差，提升視覺上的品質。

(3) 基於 MPEG-FTV 的架構下，本子項已完成視角合成演算法的開發，目前在順利移植到 ARM 嵌入式平台。於此平台上，在 CIF 與 QCIF 的解析度上已經可以達到 0.54 與 1.56 fps，即時運算則是本子項下一步發展的目標。

因此，總合今年的成果，子計畫三完成下列幾點：

(1) 完成深度圖壓縮品質對合成影像影響的文獻研究。

(2) 完成檢測合成影像失真的模型。

(3) 完成修補壓縮後深度圖的演算法。

(4) 在 FPGA 上完成立體影像視訊軟體。

(5) 在人才培育方面，本計畫兩年來共有 1 個博士生、2 個碩士生參與。

(6) 在論文方面，本計畫完成了一篇期刊論文(審察中)。

子計畫四成果：嵌入式 JAVA 平台設計

五、前言

本報告將簡述本計畫前兩年之研究成果。子計畫四的研究目的是在開發一套雙核心 Java 處理器以及最佳化的系統軟體，以提供一個高效能的 Java 執行環境。在過去兩年，子計畫四除了完成異質雙核心的 Java 應用處理器的開發，並完成了支援圖形介面應用程式的系統軟體開發，以用來在 FPGA 平台上驗證我們所開發的 Java 處理器。目前我們所得到的系統執行效能大約是 Google Android 參考平台的四倍，Sun CVM 參考平台的五～八倍。而我們也把研究的成果撰寫了一篇期刊論文（審查中），以及發表了三篇研討會論文。

六、研究目的

隨著嵌入式系統的發展，嵌入式 Java 執行環境已經變得越來越重要了。經過一些調查，我們發現新一代的嵌入式多媒體平台有漸漸地往 Java 執行環境收斂的趨勢。例如在 3GPP 手機上的 CLDC/MIDP 平台，Google 手機的 Android 平台，或是歐規數位電視標準的 DVB-MHP 平台都是一些實際的例子。通常一個能夠支援複雜的使用者圖形介面的 Java 執行環境往往需要一個全功能的作業系統。在嵌入系統的環境中,這通常是 Embedded Linux。以往在 Java 系統的設計上，大部份的研究都集中在 Java bytecode interpreter (或稱作 Bytecode Execution Engine)的效能改進。很少有研究是針對整個 Java 執行環境的最佳化設計進行討論。

通常在設計 Java 執行環境時，為了能讓 Java 執行環境有較高的效能，會把許多 Java 執行環境所需要的功能直接對應到低層的作業系統和硬體平台所提供的功能。不過這樣設計的缺點有下面兩大缺點。首先，如果 Java 執行環境是儘量利用作業系統的功能來提昇效能，那麼這個設計的移植性就不高。現今嵌入式系統的作業系統並沒有統一的趨勢，因此在設計 Java 平台時，如果只針對某一套作業系統最佳化，它的應用就會受到限制。第二個問題是，Java 執行環境本身就等同一套虛擬的作業系統，如果所有的應用程式都是用 Java 開發，那麼，整個系統就只會用到底下的實體作業系統的一小部份。從系設計的角度來看，這是一個相當浪費資源的設計。一個典型的例子就是 Android。雖然 Android 的 Java 執行環境是架構在 Linux Kernel 2.6.25 之上，但是它只利用 Linux Kernel 提供以下幾項主要功能：

- Hardware abstraction layer
- Process management
- Shared library dynamic loading management

事實上，一個簡化的 OS Kernel 就能提供以上三項功能。所以單從 Java 執行環境的支援來看，並沒有理由採用 Linux。不夠 Android 並不是一個傳統的 Java 執行環境。整個 Android 系統有相當大的系功能是透過 native code 完成的，所以 Google 選擇了在 Android 系統內嵌一個 Linux 作業系統核心。

子計畫四的研究是在開發一套雙核心 Java 處理器以及最佳化的系統軟體，以提供一個高效能的 Java 執行環境。一般而言，為了要提升嵌入式 Java

執行環境的效能，目前系統在實作上有兩種主要作法。一個方法是採用 Just-In-Time (JIT)編譯器，另一個方法是利用 Java 協同處理器 (co-processor) 來加速 Java 執行環境。透過一些相關的研究，我們發現上述兩種方法在嵌入式系統的應用上各有一些問題，比方法說：

a. 當使用者啟動了一個 Java 應用程式之後，JIT 編譯器會即時把部份 Java 程式碼編譯成實體處理器可執行的機器碼，因此在一開時會造成應用程式啟動的延遲。為了要降低延遲的時間，系統需要採用一個效能較好的處理器。但這往往會提高成本，或是增加功耗。

b. JIT 編譯器在執行時會使用到大量額外的記憶體，而且也會大幅增加應用程式碼的大小。例如在 Intel x86 處理器上，執行 JIT 編譯之後所產生的程式碼與原來的程式碼相比之下，程式碼的大小增加了四倍; 而在 Sun SPARC 處理器上，則是大約增加了八倍。這樣的結果對資源有限的嵌入式系統而言，會造成相當大的負擔。

c. 如果只是單純根據 Java stack machine 的架構設計處理器（例如 JOP 計畫），那麼在處理通訊及多媒體的工作時，效率又往往不好。

d. Java 協同處理器雖然可以解決上述各種問題，但又會受限於特定的架構。例如 ARM Jazelle 是最有名的 Java 協同處理器，但只能限於採用 ARM 架構的系統，這對於一些想要支援 Java 環境的其它處理器開發團隊而言並不適用。

就嵌入式系統的應用而言，利用 Java 處理器來加速應用程式的執行仍然是比較好的做法。因此，子計畫四目的是開發適合用在嵌入式多媒體平台上的異質雙核心 Java 應用處理器，以及支援圖型介面的 Java 執行環境。

七、結果與討論

為了能設計出一個高效能，有彈性，能跟不同的 general-purpose 處理器和作業系統做整合的 Java 執行環境，在子計畫四，我們所開發的關鍵技術有以下幾項：

- Java 處理器核心
- 把 Java 處理器核心和任意 general-purpose 處理器整合成一個異質雙核心的 Java 應用處理器所需的介面電路設計
- Java 執行環境的系統軟體設計
- 符合 CDC/PBP 及 DVB-MHP 規格的系統中介軟體(包含 Java classes library 及 native code library)

我們會採用異質雙核心架構的理由是，一方面可以透過 Java 核心有效率地執行 Java 程式，而且也能夠利用 RISC 核心來處理 I/O 的動作。以下，我們就針對上述的關鍵技術的開發成果做一個大概的介紹：

- 在 Java 處理器核心的設計方面，我們採用了 double-issue stack

machine 的架構，每一個 clock cycle 會執行兩個 bytecode。Java 核心是設計成一個獨立的 IP，可以和任何 general-purpose 處理器整合，形成一個異質多核心系統。跟目前業界常用的 ARM Jazella 的 Java 核心不一樣的是，ARM 的 Jazella 是透過 ARM 的 co-processor interface 和主處理器溝通，所以一定要和 ARM 處理器合用。而我們的 Java 處理器核心則沒有這個限制。目前我們已經成功地分別和 PowerPC 405 處理器核心以及 Microblaze 處理器核心進行整合測試。

- 在異質雙核心的介面電路設計方面，因為我們的 Java 處理核心是設計成一個獨立有 master interface 的 IP，所以可以透過標準的 shared memory 和 interrupt-driven 機制和主處理器進行 IPC。另外，我們也針對 Java 語言的特性設計了特別的 IPC 機制，可以快速的進行 Java-to-C 的呼叫。
- 在 Java 執行環境的系統軟體設計方面，我們的系統不需要依賴一個功能完整的作業系統(e.g. Linux)就可以執行圖形介面的應用程式。並且整個系統軟體的部分可以輕易的移植到任何能做中斷程序管理的系統內核(kernel)之中。因此不管將來目標平台是採用何種作業系統，都可以輕易和我們的 Java 環境進行整合。目前我們已經驗證測試過和 Linux 的整合，以及無作業系統的獨立運作驗證。
- 在系統中介軟體的設計方面，最關鍵的核心中介軟體如 Java 的基本 class library (如 CDC Foundation classes、Java AWT classes、Java TV classes) 以及 native code 的軟體（如 2D graphics library、JMF media accelerator interface）我們會重新開發出能支援 DVB-MHP UI 介面的一個子集合，而不是單純地把 Sun 的 reference implementation 移植到我們的目標平台。而上層的 DVB-MHP 圖形介面中介軟體則是會由公開程式碼的 OpenMHP 計畫修改而來。目前已完成 DVB-MHP 的中介軟體層，至於其它部份則仍在開發中。

執行至今，子計畫四已經發展出第一代的 Java 軟硬體系統，也利用 Xilinx Vertex-5 的 FPGA 平台完成驗證。雖然目前在系統軟體方面尚不符合標準 Java 執行環境的規範，但是從初步的全系效驗證可以看出我們所開發的異質雙核心 Java 處理器效能相當地好。我們所設計出來的雙核心 Java 執行環境架構如下圖 34 所示，而圖 35 則是雙指令 Java 處理器的硬體架構設計。

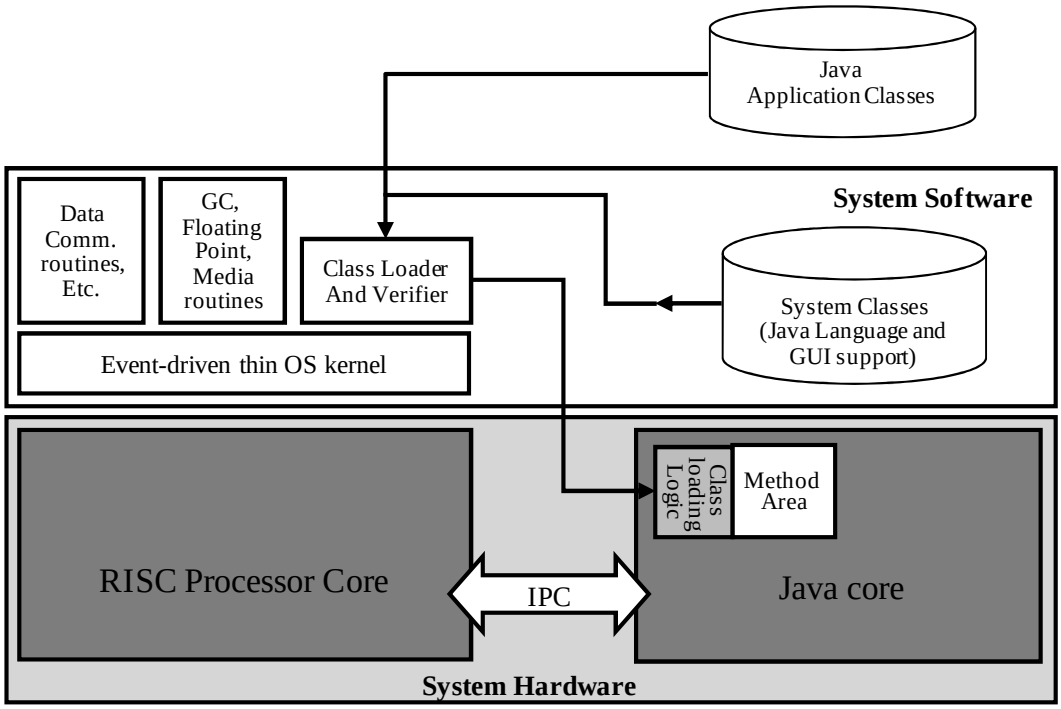


圖34. 子計畫四所提出的雙核心 Java 執行環境

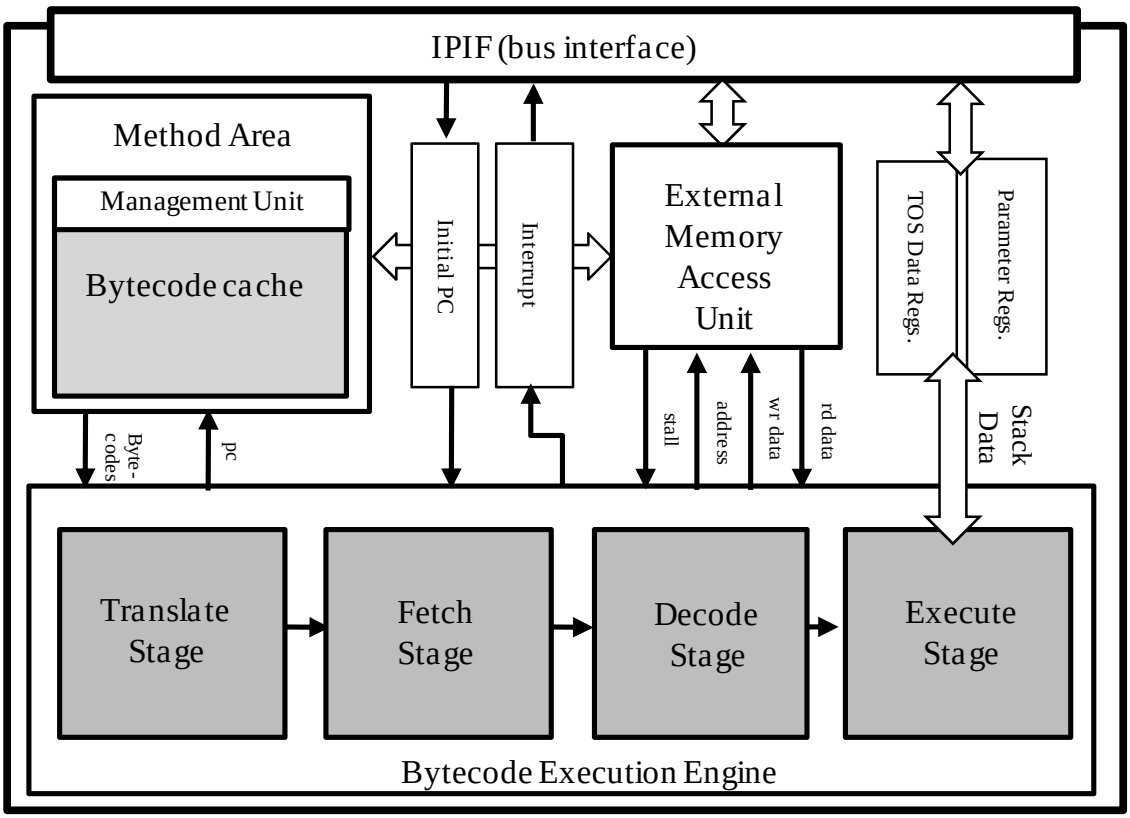


圖35. 子計畫四的 Java 核心微架構設計

在實作的方面，目前則我們已經完成在 FPGA 開發板上的系統驗證。

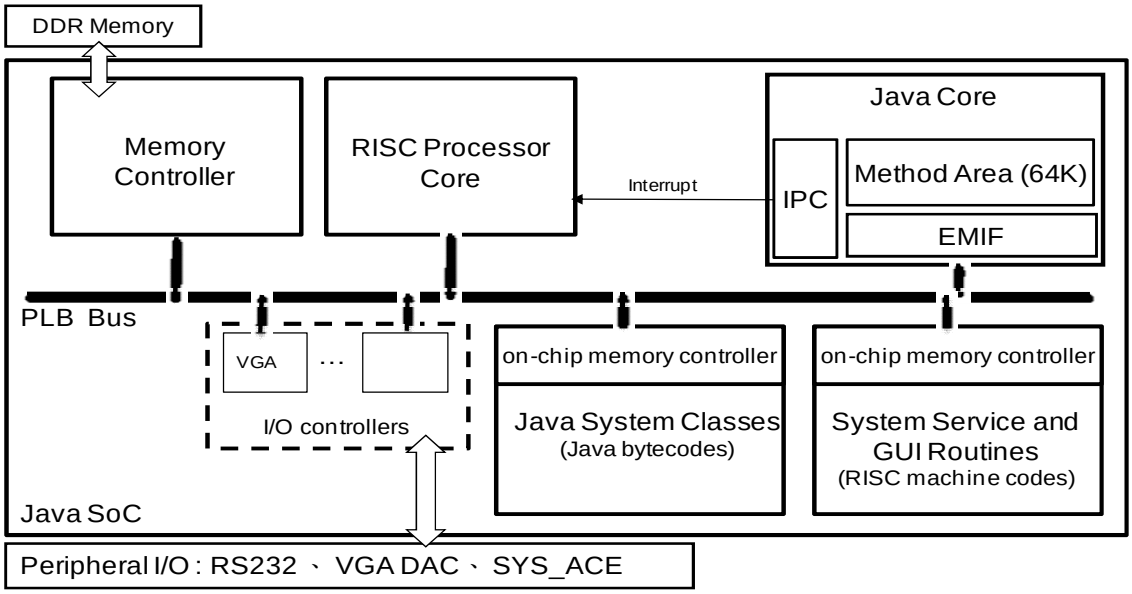


圖36. 子計畫四建立的 Java SoC 基本架構

在整體程式執行效能方面，我們採用三個測試程式與標準 Sun CVM 環境以及 Google Android 平台進行比較。下表中的每個數值都是代表程式執行的時間，因此愈小的數值代表著較佳的執行效能。根據下表的實驗數據，子計畫四所提出的 Java 系統在整體執行效能上有著 4~8 倍的提升。

表7. 基準程式執行效能比較表

	Proposed Java Sys. @100MHz		Sun CVM @100MHz		Android DVM @210MHz	
	[#K cycles]	[ms]	[#K cycles]	[ms]	[#K cycles]	[ms]
PI	19101	191	102000	1020	72059	343
SIEVE	321398	3214	2625200	26252	1313760	6256
LOOP	365272	3653	2369700	23697	1080240	5144

最後，圖 37 是子計畫四開發的 Java 執行環境執行一個 Java 圖形介面應用程式的狀況。在這個程式中，Java 程式不斷呼叫畫線的系統 API 來隨機畫線。畫線的系統 API 是用 C 寫成的，由 RISC 處理器核心來執行，把輸出直接填到 video frame buffer。整個系統（包含視窗環境）並沒有用到任何的作業系統，而是由我們自行開發的 thin kernel 及 Java runtime 環境所執行的。

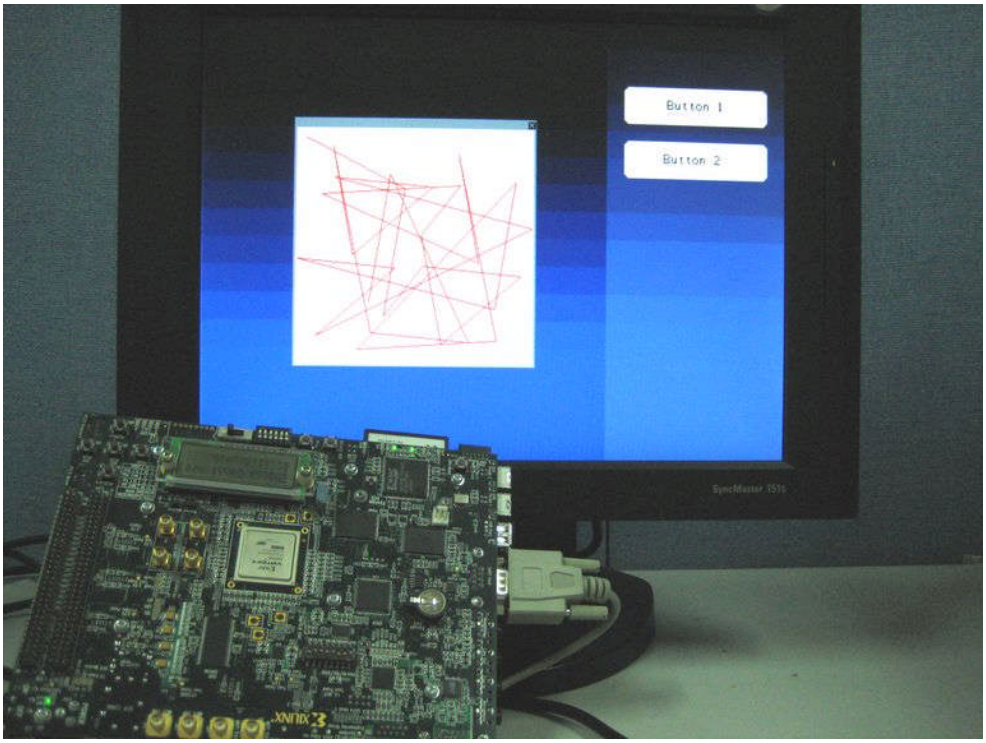


圖37. 子計畫四開發的系統執行 Java 圖形介面應用程式

八、計畫整體成果與自評

總合今年的成果，和原計畫提出的目標相當吻合。在達成預期目標情況方面有以下數點：

1. 我們已經如預期完成 Java 處理器的文獻研究。
2. 我們已經如預期完成 double-issue Java 處理器核心設計。
3. 我們已經如預期完成專為 Java 語言設計的高效能異質雙合心 IPC 呼叫介面電路。
4. 我們已經部份完成 Java dynamic class loading 的機制設計。
5. 原先預計完成的 garbage collection 機制，在評估後發現在嵌入式系統中，通常 memory 是在應用程式執完成後才執行回收，所以 garbage collection 機制相對不是很重要，因此暫時不設計。
6. 為了能有效執行較大的 Java 應用程式，我們多增加了 Java bytecode caching 的機制設計。
7. 我們完成了 MHP UI middleware 及支援基本圖形介面的 graphics library，至於符合 CDC/PBP 規格的 Java class library 則仍在開發中。
8. 在人才培育方面本計畫兩年來共有 1 個博士生，4 個碩士生參與。
9. 在論文方面，本計畫完成了一篇期刊（審查中）及三篇研討會論文。

II、 已發表之論文

子計畫一

[1] J.-C. Chan, N. T.-C. Chang, and T.-S. Chang, “ISID: In-order scan and indexed diffusion segmentation algorithm for Stereo Vision,” in Proc. IEEE Int. Symposium on Circuits and Systems, 2008.

[2] T.-H. Tsai, N. Y.-C. Chang, and T.-S. Chang, “Data reuse analysis of local stereo matching,” in IEEE Int. Symposium on Circuits and Systems, 2008.

[3] Y.-C. Tseng, N. Y.-C. Chang, and T.-S. Chang, “Block-based belief propagation with in-place message updating for stereo video,” in Proc. IEEE Asia Pacific Conf. on Circuits and Systems, Macao, 2008.

[4] N. Y.-C. Chang, Y.-C. Tseng, and T.-S. Chang, “Analysis of color space and similarity measure impact on stereo block matching,” in Proc. IEEE Asia Pacific Conf. on Circuits and Systems, Macao, 2008.

[5] Y.-C. Tseng, N. Y.-C. Chang, and T.-S. Chang, “Low-memory cost belief propagation architecture for disparity estimation,” in Proc. IEEE Int. Symposium on Circuits and Systems, Taiwan, 2009.

[6] L.-Y. Ku, S.-H. Wen, N. Y.-C. Chang, and T.-S. Chang, “A low-Cost real-time command control system based on stereo-vision and hand motion,” in Proc. Conf. on Computer Vision, Graphics, and Image Processing, 2008.

[7] J.-C. Chan, N. Y.-C. Chang, Y.-C. Tseng, and T.-S. Chang, “Local belief aggregation for MRF-based color image segmentation,” in Proc. Conf. on Computer Vision, Graphics, and Image Processing, 2008.

子計畫二

[1] T. Y. Sheu, L. D. Van, T. R. Jung, C. W. Lin, and T. W. Chang, "Low complexity subdivision algorithm to approximate Phong shading using forward difference," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2009, pp. 2373-2376, Taipei, Taiwan.

[2] P. Y. Chen, L. D. Van, and H. C. Reddy and C. T. Lin, "A new VLSI 2-D fourfold-rotational-symmetry filter architecture design," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2009, pp. 93-96, Taipei, Taiwan.

[3] P. Y. Chen, L. D. Van, and H. C. Reddy and C. T. Lin, "A new VLSI 2-D diagonal-symmetry filter architecture design," in *Proc. IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), Nov.* 2008, pp. 320-323, Macao, China.

[4] T. R. Jung, L. D. Van, T. Y. Sheu, C. W. Lin, W. C. Fang, "Design of multi-mode depth buffer compression for 3D graphics system," in *Proc. IEEE*

Int. Conf. Multimedia and Expo. (ICME), Jun. 2008, accepted, Hannover, Germany.

[5] T. R. Jung, L. D. Van, W. C. Fang, T. Y. Sheu, "Reconfigurable depth buffer compression design for 3D graphics system," in *Proc. Int. Conf. MUE, Apr.* 2008, pp. 470-474, Busan, Korea.

子計畫三

[1] C. C. Chen, Y. W. Chen, W. H. Peng, and F. Y. Yang, "A Synthesis-Quality-Oriented Depth Refinement Scheme for MPEG Free Viewpoint Television (FTV)," *IEEE Int'l Symposium on Multimedia*, 2009.

[2] Y. W. Chen, T. W. Wang, W. H. Peng, and S. Y. Lee, "A Parametric Window Design for Overlapped Block Motion Compensation with Variable Block-size Motion Estimates," *IEEE Int'l Workshop on Multimedia Signal Processing*, 2009.

[3] H.-C. Lin, H.-M. Hang, and W. H. Peng, "Fast Temporal Prediction Selection for H.264/AVC Scalable Video Coding," *IEEE Int'l Conf. on Image Processing*, 2009.

[4] H.-C. Lin, W. H. Peng, and H.-M. Hang, "Fast Context-adaptive Mode Decision Algorithm for Scalable Video Coding with Combined Coarse-grain Quality Scalability (CGS) and Temporal Scalability," *IEEE Trans. on CSVT*, 2009. (Submission)

[5] W. H. Peng, J. K. Zao, H. T. Huang, T. W. Wang, and L. S. Huang, "A Rate Distortion Optimization Model for SVC Inter-layer Encoding and Bitstream Extraction," *Journal of Visual Communication and Image Representation*, vol.19, no.8, pp.543-557, Dec. 2008.

[6] W. H. Peng, J. K. Zao, T. W. Wang, H. T. Huang, "Multidimensional SVC Bitstream Adaptation and Extraction For Rate-Distortion Optimized Heterogeneous Multicasting and Playback," *IEEE Int'l Conf. on Image Processing*, 2008.

[7] H.-C. Lin, W.-H. Peng, and H.-M. Hang, "A Fast Mode Decision Algorithm with Macroblock-Adaptive Rate-Distortion Estimation for Intra-Only Scalable Video Coding," *IEEE Int'l Conf. on Multimedia and Expo*, 2008.

[8] C.-H. Li, W.-H. Peng, and T. Chiang, "Design space exploration of an H.264/AVC-based video embedding transcoder using transaction level modeling," *IEEE Int'l Conf. on Multimedia and Expo*, 2008.

[9] C.-H. Li, W.-H. Peng, and T. Chiang, "A Reconfigurable Video Embedding Transcoder Based on H.264/AVC: Design Tradeoffs and Analysis," *IEEE Int'l Symposium on Circuits and Systems*, 2008.

子計畫四

[1] K.-N. Su, H.-J. Ko, and C.-J. Tsai, “Java Runtime Environment Design for Embedded Multimedia Services,” in Proc. of 19th VLSI Design/CAD Symp., Keng Ting, Taiwan, Aug. 2008.

[2] K.-N. Su and C.-J. Tsai, "Fast Host Service Interface Design for Embedded Java Application Processor," Proc. of 2009 IEEE Int. Symp. on Circuit and Systems (ISCAS`09), Taipei, Taiwan, May 2009.

[3] H.-J. Ko, K.-N. Su and C.-J. Tsai, "Design of a Dual-Core Java Application Processor for Multimedia Embedded System," Submitted to IEEE Transaction on Parallel and Distributed Systems.

[4] Chien-Feng Hwang, Chia-Yun Bai, Kuan-Nian Su, and Chun-Jen Tsai, "Dual-Core Java RE SoC with Embedded GUI Middleware" VLSI/CAD, Taiwan, 2009.

III、 參考文獻

[1] TS 102 812, “DVB Multimedia Home Platform (MHP) Specification 1.1”, Nov. 2001.

[2] MPEG video group,“Description of Exploration Experiments in 3D Video Coding”, JTC1/SC29/WG11 N9991, Hannover, Germany, July 2008.

[3] MPEG video group, “Applications & Requirements of FTV,” ISO/IEC JTC1/SC29/WG11 N9466, Shenzhen, China, Oct 2007.

[4] M. Z. Brown, D. Burschka, and G. D. Hager, “Advances in Computational Stereo,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp.993-1008, August, 2003.

[5] D. Scharistein and R. Szeliski, “A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms,” *International Journal of Computer Vision,* vol. 47(1/2/3), pp. 7-42, 2002.

[6] J. Diaz, E. Ros, R. Carrillo, and A. Prieto, “Real-Time System for High-Image Resolution Disparity Estimation,” *IEEE Trans. on Image Processing*, vol. 16, no. 1, pp.280-285, Jan. 2007.

[7] O. Veksler, “Stereo Correspondence by Dynamic Programming on a Tree,” *in Proc. IEEE Computer Vision and Pattern Recognition(CVPR),* 2005.

[8] Y. Deng and X. Lin, “A Fast Line Segment Based Dense Stereo Algorithm Using Tree Dynamic Programming,” *in Proc. European Conference on Computer Vision (ECCV),* 2006.

[9] C. Lei, J. Selzer, and Y. H. Yang, “Region-Tree based Stereo Using Dynamic Programming Optimization,” *in Proc. IEEE Computer Vision and Pattern Recognition (CVPR),* 2006.

[10] S. Park and H. Jeong, “VLSI Architecture for MRF Based Stereo Matching,” in Proc. International Symposium on Systems, Architectures, Modeling and Simulation (SAMOS), 2007.

[11] P. F. Felzenszwalb and D. P. Huttenlocher, “Efficient Belief Propagation for Early Vision,” *in Proc. IEEE Computer Vision and Pattern Recognition (CVPR),* 2004.

[12] S. Shimizu, H. Kimata,“Experimental Results on Depth Estimation and View Synthesis with subpixel-precision”, JTC1/SC29/WG11 M15584, Hannover,

Germany, July 2008.

[13] M. Tanimoto, T. Fujii and K. Suzuki,“Improvement of Depth Map Estimation and View Synthesis”, ISO/IEC JTC1/SC29/WG11, M15090, Jan. 2008 Antalya, Turkey.

[14] Bertalmio, M., Sapiro, G., Caselles, V. and Ballester, “Image inpainting.” *Proceedings of ACM SIGGRAPH 2000*, ACM Press, 417–424.

[15] MPEG video group,“Improving AVC compression performance by template matching with decoder-side”, JTC1/SC29/WG11 M15375, Archamps, France, Apr. 2008.

[16] Y. Suzuki, C. S. Boon, T. K. Tan, “Inter Frame Coding with Template Matching Averaging,” *ICIP* 2007.

[17] Bor-Sung Liang et al., “Index Rendering: Hardware-Efficient Architecture for 3-D Graphics in Multimedia System,” IEEE Trans. on Multimedia, vol.4, no.2, Jun. 2002.

[18] D. Kim et al., “An SoC With 1.3 Gtexels/s 3-D Graphics Full Pipeline for Consumer Applications”, IEEE JSSC, vol.41, no.1, Jan. 2006.

[19] M.Deering et al., “The triangle processor and normal vector shader: A VLSI system for high-performance graphics,” in Proc. SIGGRAPH, pp21-30, 1998

[20] David Blythe, Microsoft Corporation, “The Direct3D 10 System,” 2006.

[21] H. Gouraud, “Continuous Shading of Curved Surfaces,” Comm. ACM, vol. 18, 1971 .

[22] Bui Tuong Phong, “Illumination for computer generated pictures,” Comm. ACM, vol.18, Jun. 1975.

[23] J.N. Mitchell Jr., “Computer Multiplication and Division Using Binary Logarithms,” IRE Trans. Electronic Computers, vol.11, pp.512-517, Aug. 1962.

[24] M. Combet, H. Zonneveld, and L. Verbeek, “Computation of the Base Two Logarithm of Binary Numbers,” IEEE trans. Electronic Computers, vol. 14, Dec. 1965.

[25] K.H. Abed and R.E. Siferd, “CMOS VLSI Implementation of a Low-Power Logarithmic Converter,” IEEE Trans. on Computers, vol.52, no.11, Sep. 2003.

[26] K.H. Abed and R.E. Siferd, “CMOS VLSI Implementation of a Low-Power Antilogarithmic Converter,” IEEE Trans. on Computers, vol.52, no.11, Sep. 2003.

[27] Thomas A. Brubaker and John C. Becker, “Multiplication Using Logarithms

Implemented with Read-Only Memory," IEEE Trans. on Computers, vol.24 no.8, Aug. 1975.

[28] F. Bensaali, A. Amira and A. Bouridane, "Accelerating matrix product on reconfigurable hardware for image processing applications," IEE Proc.-Circuits Devices Syst., vol. 152, no. 3, pp. 236-246, Jun. 2005.

[29] J.-H. Sohn, J.-H. Woo, M.-W. Lee, H.-J. Kim, R. Woo, and H.-J. Yoo, "A 155-mW 50-Mvertices/s Graphics Processor With Fixed-Point Programmable Vertex Shader for Mobile Applications," IEEE Journal of Solid-State Circuits, vol. 41, no. 5, pp. 1081-1091, May 2006.

[30] DEROO J., MOREIN S., FAVELA B., WRIGHT M., "Method and Apparatus for Compressing Parameter Values for Pixels in a Display Frame," In US Patent 6,476,811.

[31] Jon Hasselgren and Tomas AkenJu-ine-Möller, "Efficient Depth Buffer Compression," In Graphics Hardware, pp. 103-110, 2006.

[32] MOREIN S., "Method and Apparatus for Efficient Clearing of Memory," In US Patent 6,421,764.

[33] Orenstein et al., "Z-Compression Mechanism," In US Patent 6580427.

[34] Van Dyke et al., "Method and Apparatus for Managing and Accessing Depth Data in a Computer Graphics System," In US Patent 6961057.

[35] Van Hook, "Method and Apparatus for Compression and Decompression of Z Data," In US Patent 6630933.

[36] Jonathan Corbet et al, "Linux Device Drivers, 3rd,".

[37] Keronos Group, "OpenGL ES 1.1.10 Specification," 2007.

[38] OpenSceneGraph, http://www.openscenegraph.org/projects/osg, 2007.

[39] Sun Microsystems, "The Java 3D API Specification Version 1.2," Apr. 2000.

[40] Q. H. Mahmoud, J2ME for Home Appliances and Consumer Electronics Devices, Sun Microsystems White Paper, Jan. 2003.

[41] Digital Video Broadcasting (DVB), Multimedia Home Platform (MHP) Specification 1.0.2, ETSI TS 101 812, June, 2002.

[42] T. Lindholm and F. Yelling, The Java Virtual Machine Specification, Addison-Wesley, 1996.

[43] A. Krall, K. Ertl, and M. Gschwind, Java VM Implementation: Compilers versus Hardware, *John Morris (ed.), Computer Architecture (ACAC '98)*, Perth, pp. 101-110, 1998.

[44] Martin Schoberl, JOP: A Java Optimized Processor for Embedded Real-Time Systems, Ph.D. Thesis, Tech. Universitaet Wien, Jan 2005.

[45] A. Kim and M. Chang, “Designing a Java Microprocessor Core Using FPGA Technology,” *Computing & Control Engineering Journal*, June 2000, pp.135-141.

[46] Tan, Y.Y. Yau, C.H. Lo, K.M. Yu, W.S. Mok, P.L. Fong, A.S. “Design and implementation of a Java processor,” *IEE Proceedings*, Volume: 153 , On page(s): 20 – 30, 2006 ·

[47] Ramesh Radhakrishnan, Ravi Bhargava, Lizy K. John. Improving Java performance using ardware translation. *ACM Press*. June 2001 Pages: 427 – 439.

[48] ARM. Jazelle – ARM Architecture Extensions for Java Applications. White paper.

[49] Andreas Krall. Efficient JavaVM Just-in-Time Compilation. In Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques (PACT ’98), pages 205–212, Paris, October 12–18, 1998. IEEE Computer Society Press.

[50] Georg Acher. JIFFY — Ein FPGA-basierter Java Just-in-Time Compiler für eingebettete Anwendungen. PhD thesis, Technische Universität at München, 2003.

[51] Sun. *picoJava-II Microarchitecture Guide*. Sun Microsystems, March 1999.

[52] D.S. Hardin. Real-Time Objects on the Bare Metal: An Efficient Hardware Realization of the JavaTM Virtual Machine. In *Proceedings of the Fourth International Symposium on Object-Oriented Real-Time Distributed Computing*, page 53. IEEE Computer Society, 2001.

[53] R. Zulauf. Entwurf eines Java-Mikrocontrollers und prototypische Implementierung auf einem FPGA. Master’s thesis, University of Karlsruhe, 2000.

[54] S.A. Ito, L. Carro, and R.P. Jacobi. Making Java Work for Microcontroller Applications. *IEEE Design & Test of Computers*, 18(5):100–110, 2001.

[55] Ramesh Radhakrishnan, Deependra Talla and Lizy Kurian John, “Allowing for ILP in an Embedded Java Processor,” *ACM SIGARCH Computer Architecture News*, pp. 294-305, 2000.

[56] Zhilei Chai, Wenke Zhao, Wenbo Xu. System On Chip Design And Software Supports (SODSS): Real-Time Java Processor Optimized for RTSJ, *ACM*

Press. March 2007

[57] R. Radhakrishnan, *Microarchitectural Techniques to Enable Efficient Java Execution*, PhD thesis, University of Texas at Austin, 2000.

[58] C. J. Glossner. The DEFLT-JAVA Engine, PhD thesis, Delft University of Technology, 2001.

[59] MPEG video group,“Results of 3D Video Expert Viewing”, JTC1/SC29/WG11 N9992, Hannover, Germany, July 2008.

附件一出席國際學術會議報告

與會人姓名	陳漪紋 Yi-Wen Chen	就讀校院（科系所）	國立交通大學資訊科學與工程所 博士班研究生
時間 會議地點	October 5-7, 2009 Rio de Janeiro, Brazil.		
會議名稱	2009 IEEE International Workshop on Multimedia Signal Processing (MMSP2009)		
發表論文題目	(中文)適用於變動方塊大小的跨區塊動量補償(OBMC) (英文) A Parametric Window Design for OBMC with Variable Block Size Motion Estimates		

一、 參加會議經過

由於本會議於今年 10 月舉行，且容許之後再補上與會心得。

二、 與會心得

三、 考察參觀活動(無是項活動者省略)

四、 建議

五、 攜回資料名稱及內容

附件二重要發表論文

以下附錄為四篇計畫相關的已發表論文全文，每項子計畫列出一篇：

- T.-H. Tsai, N. Y.-C. Chang, and T.-S. Chang, “Data reuse analysis of local stereo matching,” in *Proc. IEEE Int. Symposium on Circuits and Systems*, 2008.
- T. Y. Sheu, L. D. Van, T. R. Jung, C. W. Lin, and T. W. Chang, "Low complexity subdivision algorithm to approximate Phong shading using forward difference," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS),* May 2009, pp. 2373-2376, Taipei, Taiwan.
- C. C. Chen, Y. W. Chen, W. H. Peng, and F. Y. Yang, “A Synthesis-Quality-Oriented Depth Refinement Scheme for MPEG Free Viewpoint Television (FTV),” in *Proc. IEEE Int’l Symposium on Multimedia*, 2009.
- K.-N. Su and C.-J. Tsai, “Fast Host Service Interface Design for Embedded Java Application Processor,” Proc. of 2009 IEEE Int. Symp. on Circuit and Systems (ISCAS`09), Taipei, Taiwan, May 2009.

Data Reuse Analysis of Local Stereo Matching

Tsung-Hsien Tsai, Nelson Yen-Chung Chang and Tian-Sheuan Chang
Department of Electronics Engineering, National Chiao-Tung University
1001 Ta-Hsueh Rd., Hsinchu 300, Taiwan, R.O.C.
{thtsai, ycchang, tschang}@twins.ee.nctu.edu.tw

***Abstract*—External memory bandwidth and internal memory size have been major bottlenecks in designing VLSI architecture for real-time stereo matching hardware because of large amount of pixel data and disparity range. To address these bottlenecks, this work explores the impact of data reuse on disparity-order and pixel-order along with the partial column reuse (PCR) and vertically expanded row reuse (VERR) techniques we proposed. The analysis suggest that a disparity-order reuse with both PCR and VERR techniques is suitable for low memory cost and low external bandwidth design, whereas the pixel-order reuse with both techniques is more suitable for low computation resource requirement.**

I. INTRODUCTION

Computational binocular stereo is one of the major topics in computer vision, which is useful in many fields, such as intelligent robots, autonomous vehicles, 3D scene reconstruction, and multiview video coding [1]. The goal of computational binocular stereo is to estimate the displacement between each corresponding pair of pixels in the left and right views. The displacement is referred as disparity and the process is referred as disparity estimation. The most efficient way to estimate disparity is referred as local stereo matching [2] which finds the stereo correspondence by means of local matching windows.

Local stereo matching is the most prominent approach in implementing a real-time stereo matcher. For more complicated local stereo matching algorithms having better stereo matching performance, VLSI hardware implementation is inevitable. However, these powerful stereo matching algorithms often demands enormous data storage and memory bandwidth. To resolve such storage and bandwidth issues, data reuse techniques must be employed. Previous work [3] [4] have studied the data reuse in simpler local stereo matching algorithms based on block matching. However, the data reuse of more complex and robust local stereo matching [5] [6] algorithms with additional cost aggregation step has not been considered.

Unlike the simple local stereo matching algorithms, the more complex aggregation-based stereo matching algorithm poses a challenge in data reuse. The data reuse order and direction within different steps in the aggregation-based stereo matching algorithms may sometimes interfere with each other and result in even larger internal memory size or more bandwidth. To ensure the success of a hardware architecture for real-time aggregation-based stereo matcher, it is necessary to clarify the how different data reuse order and direction would affect the overall performance.

Motivated by the aforementioned reason, this work investigated disparity-order and pixel order data reuse for the initial matching cost computation. In addition, we proposed a partial column reuse (PCR) and a vertically expanded row reuse (VERR) techniques for the cost aggregation. The impacts of different combinations of data reuse orders and techniques on internal memory size and external memory bandwidth are analyzed. The analysis and comparison result suggest that if the internal memory size is the most important requirement, disparity-order reuse with PCR technique is the best candidate. On the other hand, if external memory bandwidth is deemed more important, disparity-order reuse with both PCR and VERR techniques should be more suitable.

The rest of the paper is organized as follows. Section II explains the flow of aggregation based method. Section III analysis the data reuse problem and explaining how the data reuse methods function. Section IV will compare the performance of different solutions. Then, we conclude in Section V.

II. REVIEW OF AGGREGATION BASED ALGORITHMS

According to Scharstein and Szeliski [7], the flow of aggregation based algorithms for stereo matching consist of three steps: cost computation, cost aggregation, and disparity computation. The detail of each step is explained in the following subsections.

A. Matching Cost Computation

Matching cost computation computes the initial matching cost of each pixel at different disparities. The matching cost is usually computed from luminance. There are many different similarity measures that can be used as the matching cost. The simplest similarity measure only requires one reference and target pixel to compute, such as the commonly used absolute-difference (AD). Other similarity measures requires a support window to take information of neighboring pixels into consideration, examples include sum-of-absolute-difference (SAD), adaptive support weight SAD, zero-mean SAD (ZSAD), census hamming distance [8], and mutual information [9]. The size of the window is arbitrarily selected depending on the performance requirement. Larger window often result in more accurate disparity map with blurry

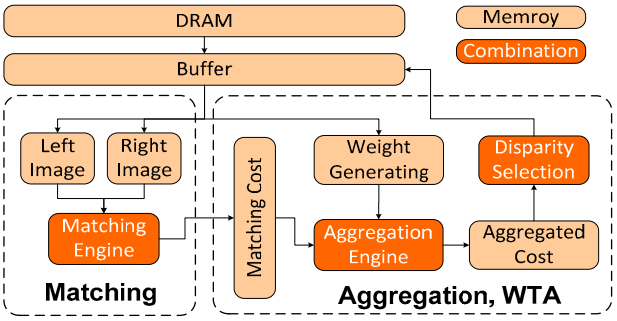


Figure 1. The overview of hardware architecture

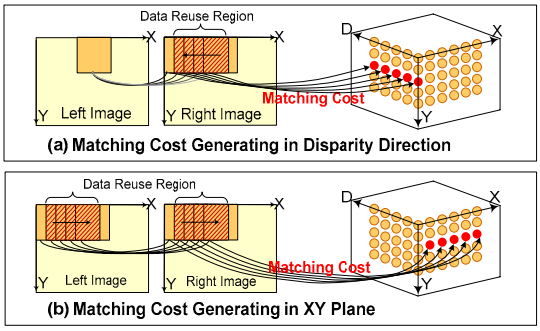


Figure 2. The two data reuse directions

boundary, whereas smaller window result in opposite way. Here we define the matching cost of a pixel at (x,y) in the left reference image with disparity *d* be

$$C_{(x,y,d)} = Matcher\big(L(x,y), R(x-d,y)\big), \quad (1)$$

where *L(x,y)* and *R(x-d,y)* represents the center pixels of the support window in left and right images respectively, and *Matcher()* is the similarity measure function. For each pixel at (x,y), if the disparity range is 32, there would be 32 matching costs for (x,y) with each associated to a disparity value.

B. Cost Aggregation

In this step, the initial cost of a pixel and its neighboring pixels within an aggregation window are summed together. This aggregated cost includes the influence of the neighboring pixels to enforce the smoothness constraint [7]. The cost aggregation can be iteratively performed to include the influences from farther pixels. The aggregation of iteration t+1 can be defined as

$$C^{t+1}_{(x,y,d)} = \frac{1}{R_{(x,y)}} \sum_{i=-\frac{r-1}{2}}^{\frac{r-1}{2}} \sum_{j=-\frac{r-1}{2}}^{\frac{r-1}{2}} C^{t}_{(x+i,y+j,d)} \cdot \lambda_{(x,y,i,j)}, \quad (2)$$

$$R_{(x,y)} = \sum_{i=-\frac{r-1}{2}}^{\frac{r-1}{2}} \sum_{j=-\frac{r-1}{2}}^{\frac{r-1}{2}} \lambda_{(x,y,i,j)}, \quad (3)$$

where λ(x,y,i,j) is the weighting of each pixel within the window of size *r*x*r* pixels, and R(x,y) is the weighting normalization term which is the summation of the weightings. The weighting gives different degree of influence to different neighboring pixels to achieve better disparity result. Examples of algorithm using weighted aggregation includes non-linear diffusion [10], outlier-rejection [6], and adaptive support weight [5].

C. Disparity computation

The disparity of a pixel is determined in a winner-takes-all (WTA) manner. The disparity with the minimal aggregated cost is selected as the disparity for this pixel.

III. Analysis of Data Reuse Methods

On implementing aggregation based method under real-time constraint, there are many solutions to the data reuse issue. We will use the hardware architecture shown in Fig. 1 to explain different solutions.

A. Data Reuse Problem

In the matching cost computation, if data reused along the disparity axis is preferred, the computation of all the matching costs for a pixel is computed before jumping to the next pixel. This allows the data within the matching cost support window to be reused. However, the cost aggregation sums the initial matching costs of the same disparity together, which would prefer initial costs to be output along the spatial (X-Y) plane than the disparity axis. As a result, to compute the aggregated cost within an aggregation window, all the matching costs at each disparity must be stored before the aggregation can be performed. These initial matching costs that need to be stored before the aggregation of an aggregation window can be represented with a cuboid in the disparity-spatial (D-X-Y) space. The volume of this cube represents the memory size needed to store the initial costs. One way to reduce the storage requirement is to avoid the conflict in data reuse direction. For instance, change the reuse direction in matching cost computation to the X-Y plane so that it meets the direction in cost aggregation. Although doing so removes the conflict between the matching cost computation and the cost aggregation, the conflict between the cost aggregation and the disparity computation exists. To determine the disparity of a pixel, the disparity computation needs to have all the matching costs at each disparity for that pixel. However, the aggregated costs are generated in the X-Y plane first, which is different from the preferred direction of the disparity computation. Consequently, additional storage would be required to store the aggregated costs. These conflicts in data generation and reuse directions play a key role in determining the storage requirement. Therefore, it is important to derive the best data reuse strategy which resolves these conflicts so that the storage requirement can be minimized.

B. Matching Cost Computation Data Reuse

The data reuse in the matching cost computation can be categorized into two types according to the reuse order. The details of these data reuse method are explained below.

1) Dipsarity-Order Reuse

The disparity-order reuse reuses the data in the matching window of different disparities. Fig. 2(a) illustrates how disparity-order reuse works. When we compute the disparity of a pixel in the left image, the matching window in the right image would slide leftward within the disparity range. In other words, the matching cost of different disparities for a pixel in

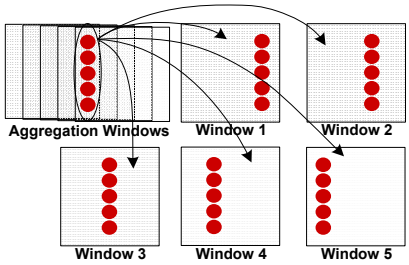


Figure 3. The partial column reuse(PCR) in 5x5 aggregation window

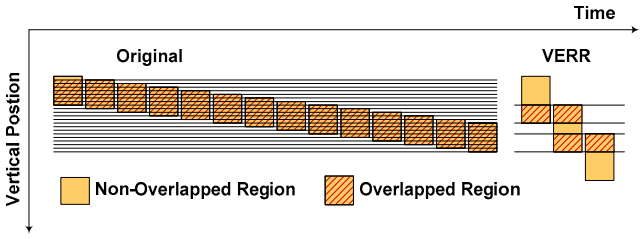


Figure 4. Vertically Expanded row reuse(VERR)

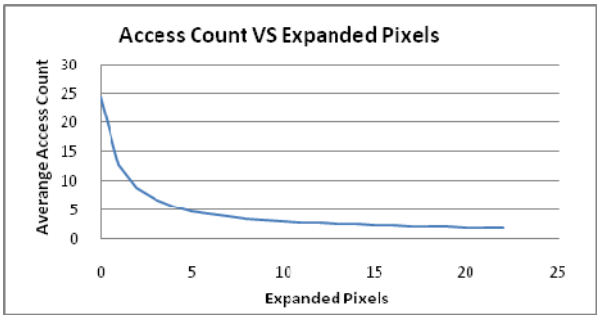


Figure 5. The average access count versus the number of expanded pixel

the left image is first computed. Then the matching cost computation of the next pixel in the left image is performed. With the disparity-order reuse, the overlapped data within the matching window in the right image shown in Fig. 2(a) can be reused to compute the matching cost at different disparities. As a result, if the pixel data are stored in external memory, there is no need for repeating accesses of the overlapped pixels. Hence, the bandwidth requirement to external memory can be reduced. However, the order of matching cost generation is different from the order of the matching cost consumption in the following cost aggregation step. This would result in additional memory storage requirement.

2) Pixel-Order Reuse

Similar to the disparity-order reuse, the pixel-order reuse reuses the data overlapped by the neighboring matching window in both left and right images.

Fig. 2(b) illustrates the detail of pixel-order reuse. The matching cost of the same disparity for each pixel is first computed. Then the cost of the next disparity for each pixel is computed. As a result, the matching window in the left and the right images both slides synchronously with the same disparity offset. With the pixel-order reuse, the overlapped data within the matching windows shown in Fig. 2(b) can be reused. Therefore, the pixel-order reuse can also reduce the external memory bandwidth requirement. In contrast to the disparity-order reuse, the order of matching cost generation is the same as the order of the cost consumed by the following cost aggregation step. Hence, the buffer size between the two steps can be reduced. However, the data reuse can only be exploited during the cost computation of one single disparity. There is no data reuse between the computations of different disparities. Once all the computation of the previous disparity has been completed for all the pixels in the whole image, pixel data have to be read from the external memory again. Unless all the previously read pixel data could be stored within the internal memory, otherwise repeating external memory accesses are inevitable.

C. Cost Aggregationt Data Reuse

In addition to the data reuse in the matching cost computation, there are two data reuse methods in the cost aggregation. The details of these two data reuse methods are explained as follows.

1) Partial Column Reuse

The partial column reuse method reduces the local memory size in the cost aggregation by distributing the computation of aggregated cost to each column. Instead of computing the aggregated cost after all the initial costs in an aggregation window are available, the PCR computes the partial sum of a column after the initial costs of this column are available. As a result, the size of the local memory can be reduced from a window to only one column. Moreover, the partial sum of each column can contribute to the aggregated cost of multiple overlapped windows. Storing partial column cost requires less local memory size than storing all the initial matching costs in a column.

Fig. 3 illustrates an example of the PCR with a 5x5 aggregation window size. An aggregated cost requires the partial sum of five initial cost columns. With PCR, the current partial column sum in Fig. 3(a) can be reused to contribute to the aggregated cost of windows 1 to 5.

2) Vertically Expanded Row Reuse

The vertically expanded row reuse reduces the bandwidth requirement to the initial cost memory by deliberately access additional rows of initial costs. When the aggregation finishes processing the current row and jumps to the next row, the overlapped data between the windows at the previous row and the current row have to be read from the initial cost memory again. Fig. 4 shows an example of the situation that the data are overlapped. To avoid accessing the already accessed costs, the VERR vertically expand the rows of initial costs to be read so that they can be reused to compute multiple rows of aggregated cost at once.

Fig. 4 shows how VERR can reduce the overlapped data. Without the VERR, most of the data in the windows are overlapped for many times. These overlapped data are read repeatedly multiple times. However, with the VERR, the portion of overlapped data becomes much smaller than the case without the VERR. Moreover, the overlapped data in the VERR case only overlap once. This implies that with the

TABLE I. ANALYSIS OF DISPARITY-ORDER AND PIXEL-ORDER DATA REUSE WITH PCR AND VERR METHODS

Section	Property	Disparity-Order				Pixel-Order			
		Original	+PCR	+VERR	+PCR +VERR	Original	+PCR	+VERR	+PCR +VERR
Step 1	***Internal Memory Size*** *(KBytes)*	2.4	2.4	2.6	2.6	2.2	2.2	2.4	2.4
	Bandwidth Requirement from External DRAM *(MBytes/sec)*	3.3	3.2	0.9	0.9	207.9	207.9	10.1	10.1
Step 2	***Internal Memory Size*** *(KBytes)*	563.2	1.6	44.8	`1.8	0.6	X	1.8	0.1
	Bandwidth Requirement from Cost Computation Engine *(MBytes/sec)*	**6.5**	**158.7**	**44.3**	**44.3**	**6.5**	**X**	**4.7**	**4.7**
Step 3	***Internal Memory Size*** *(KBytes)*	0.1	0.1	0.1	0.1	228.1	X	228.1	228.1
Total	***Internal Memory Size*** *(Bytes)*	**565.7**	**4.1**	**47.6**	**4.5**	**230.9**	**X**	**232.3**	**230.5**
	Bandwidth Requirement from External DRAM *(MBytes/sec)*	**3.3**	**3.2**	**0.9**	**0.9**	**207.9**	**207.9**	**10.1**	**10.1**
	Real-time Constraint (30 fps)	**Meet**	**Meet**	**Meet**	**Meet**	**Fail**	**Fail**	**Meet**	**Meet**

VERR, the repeating accesses of the overlapped data would be fewer than the case without the VERR.

Fig. 5 plots the relationship between the average access count of an initial matching cost and the value k given an aggregation window size of 25x25. The value k represents the number of expanded rows. It can be observed that the average access count decreases as k increases. This suggests that with more rows expanded, less bandwidth is needed. However, increasing the value of k will also increase the local memory size and computing resource requirement.

IV. COMPARISON

TABLE I compares the estimated memory size and bandwidth requirement of the disparity-order and pixel-order reuse methods. The target disparity image is 352x288 pixels large with 64 disparities. The real-time constraint is 30 fps. The architecture is assumed to operate at 100MHz clock with a 32-bit data port to the external memory. The size of support window in the matching cost computation and cost aggregation are 9x9 and 25x25 pixels respectively.

From TABLE I, only original pixel-order and pixel-order reuse with PCR technique fail the real-time constraint because of enormous external memory bandwidth requirement. This is because applying PCR with the pixel-order reuse alone would limit the cost aggregation throughput due to column-based computation. For reuse combinations that meet the real-time constraint, if minimal internal memory size is required, the disparity-order reuse with PCR is the best candidate. On the other hand, if external memory bandwidth is deemed more important, the disparity-order with both PCR and VERR is more suitable. However, the disparity-order has large bandwidth requirement from the matching cost computation engine, which implies large computation resource requirement. Hence, if the computation resource is the concerning issue, the pixel-order reuse with both PCR and VERR should be adopted.

V. CONCLUSION

This paper explores the impact of disparity-order and pixel-order data reuse in the matching cost computation and proposed the partial column reuse (PCR) and vertically expanded row reuse (VERR) techniques for the cost aggregation for local stereo matching architectures. The analysis and comparison conclude that the architecture using the disparity-order reuse with both the PCR and VERR techniques is suitable for the design of low memory cost with high computation resource. On the other hand, the architecture using pixel-order reuse with VERR technique requires less computation resource, but needs large internal memory in storing the aggregated cost. It is up to the designers to adopt the reuse combination that meets best to their constraints and requirements. Further study on impact of various reuse parameter settings shall be conducted in the future.

REFERENCES

[1] ISO/IEC JTC1/SC29/WG11 N6501, "Requirements on Multi-view Video Coding," Redmond, USA, July 2004.

[2] MZ Brown, D. Burschka, and G. Hager, "Advances in Computational Stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 25, No.8, pp. 993-1008, August 2003.

[3] M. Hariyama, T. Takeuchi, M. Kameyama. "VLSI processor for reliable stereo matching based on adaptive window-size selection," *Proceedings of IEEE International Conference on Robotics and Automation*, pp.1168- 1173, May 2001.

[4] J Díaz, E Ros, R Carrillo, A Prieto, "Real-Time System for High-Image Resolution Disparity Estimation," *IEEE Transactions ON Image Processing*, 2007.

[5] K.J. Yoon and I.S. Kweon. "Adaptive support-weight approach for correspondence search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2006.

[6] M. Gerrits and P. Bekaert, "Local stereo matching with segmentation-based outlier rejection," *in Proc. 3rd Canadian Conf. on Computer and Robot Vision (CRV'06)*, pp. 66-66, 2006.

[7] D. Scharstein and R. Szeliski, "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms," *Int'l J. Computer Vision*, Vol. 47, No. 1, pp. 7–42, 2002.

[8] R. Zabih and J. Woodfill, "Non-parametric Local Transforms for Computing Visual Correspondence," *In ECCV*, Vol. 2, pp. 151–158, 1994.

[9] G. Egnal, "Mutual Information As a Stereo Correspondence Measure," *Technical Report MS-CIS-00-20*, Computer and Information Science, University of Pennsylvania, Philadelphia, USA, 2000.

[10] D. Scharstein and R. Szeliski, "Stereo Matching with Nonlinear Diffusion," *IJCV*, Vol. 28, No 2, pp. 155-174, 1998.

Low Complexity Subdivision Algorithm to Approximate Phong Shading Using Forward Difference

Teng-Yao Sheu, Lan-Da Van, Tzung-Rung Jung, Cheng-Wei Lin, Ting-Wei Chang
Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
E-mail: ldvan@cs.nctu.edu.tw

***Abstract*—In this paper, we proposed a low complexity subdivision algorithm to approximate Phong shading. It is a combination of a subdivision scheme using forward difference and a recovery scheme to prevent rasterization anomaly. Dual space subdivision, triangle filtering and variable sharing schemes are also proposed to reduce the computation. Compared with the conventional recursive subdivision shading algorithm, the proposed algorithm reduces 46.6% memory access, 80% projective transformations and perspective division, 93.75% clipping/culling tests and 60% 3x3 matrix multiplications for setup operation.**

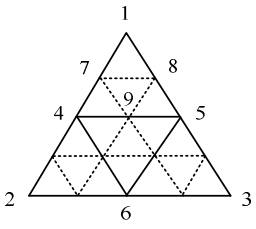
I. INTRODUCTION

Gouraud shading [2], per-vertex lighting, is commonly used in real-time application because it only applies reflection model [1] on vertices of polygons and linearly interpolates the intensities for internal pixels. Phong shading [3], per-pixel lighting, linearly interpolates face normal across a polygon for internal pixels and applies reflection model on each pixels. Phong shading can render smooth and realistic highlight but demand huge computation to light every pixel. Therefore, it's not suitable for real-time application without hardware acceleration.

In order to reduce complexity of Phong shading, many researchers concentrate on approximating scheme with reasonable tradeoff between cost and quality. The Taylor expansion [4] is used to approximate Phong reflection model. The average cost is high in the scenes with small polygons or multi-light sources. Spherical interpolation algorithms [6][7] aim to avoid renormalization. But, the setup cost is expensive for small polygons. The mix shading [11] combines two shading methods. When the hightlight covers the polygon, it is rendered using Phong shading. Otherwise, Gouraud shading is employed. The problem is that per-pixel lighting is involved in the highlight region. To completely eliminate pre-pixel lighting, quadratic interpolation [8][9] uses quadratic function to interpolate light intensities between six points in a triangle. The quadratic scheme may introduce notable gap on the edge if the triangle is too large. Subdivision scheme [5][12] is another way to approximate Phong shading. Subdivision scheme subdivides a triangle into smaller ones and renders these small triangles individually with Gouraud shading. Compared to per-pixel lighting, only vertices are lit. It is manifest that the computational complexity is greatly reduced.

The attractive feature of subdivision scheme is that it can control rendering quality dynamically. If high quality shading is demanded, more small triangles are generated. Otherwise, fewer triangles are generated to reduce the processing time and power consumption. Previous works use recursive subdivision algorithm to subdivide triangles in software. Subdivision in software has unnecessary bus traffic from CPU to graphic hardware because the generated vertices must be transferred to graphic hardware for rendering. To save bandwidth, the subdivision algorithm should be integrated into graphic hardware. Recursive subdivision algorithm is not efficient for hardware implementation because it uses stack to buffer the intermediate vertices. As illustrated in Fig. 1, the memory accesses consume large amount of power and the required memory size depends on the recursive depth. Another problem of previous algorithms is the overhead introduced by the generated triangles. They decrease performance greatly if they were not correctly handled. Little attention has been given to these points above.

In this paper, we proposed a low complexity subdivision algorithm to approximate Phong shading using forward difference technique. With least memory access and constant memory size requirement, it's suitable for hardware implementation. A recovery scheme is proposed to prevent rasterization anomaly. We also present three different schemes to reduce the overhead of the subdivision schemes.



Push vertices 1, 2, 3 (3 write)
Pop vertices 1, 2, 3 (3 read)
Gen vertices 4, 5, 6
Push vertices 1, 4, 5 (3 write)
Push vertices 4, 2, 6 (3 write)
Push vertices 4, 6, 5 (3 write)
Push vertices 5, 6, 3 (3 write)
Pop vertices 1, 4, 5 (3 read)
Gen vertices 7, 9, 8
Output triangle 1, 7, 8
Output triangle 7, 4, 9
Output triangle 7, 9, 8
Output triangle 8, 9, 5
……

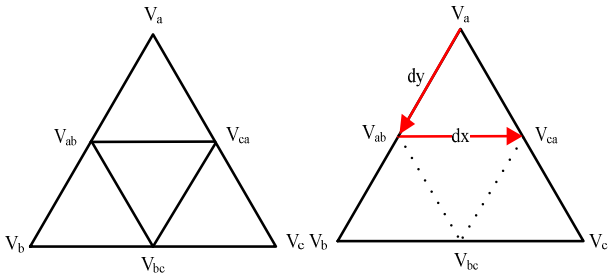
Fig. 1. Illustration of recursive subdivsion scheme.

This work was supported in part by the National Science Council (NSC) Grant NSC-96-2220-E-009-038, NSC-97-2220-E-009-055

This paper is organized as follows: the proposed algorithm is described in Section II. In Section III, we present three schemes to improve the subdivision algorithms. Complexity analysis is given in Section IV. At last, the brief statements conclude the presentation.

II. Proposed Subdivision Algorithm

In this section, we present proposed subdivision algorithm to approximate Phong shading. Our algorithm is the combination of a subdivision scheme using forward difference and a recovery scheme used to prevent rasterization anomaly. Forward difference is a technique used to refine mesh [10]. Herein, we use it to subdivide triangles. An example is illustrated in Fig. 2. First, we compute the differentials vectors, dx and dy in horizontal and vertical direction, respectively:



(a) Subdivided four triangle (b) Incremental subidivision

Fig. 2: Illustration of the proposed subdivision scheme.

$$dx = (V_c - V_b) / level$$
$$dy = (V_b - V_a) / level$$
$$\text{level} = \log_2(\text{number of generated triangles})$$

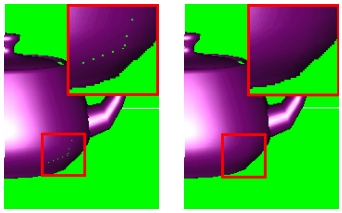
After obtaining the differentials, the middle vertices, V_{ab}, V_{bc} and V_{ca}, are computed by adding the difference vectors:

$$V_{ab} = V_a + d_y$$
$$V_{ca} = V_{ab} + d_x$$
$$V_b = V_{ab} + d_x$$
$$V_{bc} = V_b + d_x$$
$$V_c = V_{bc} + d_x$$

Finally, all vertices: V_a, V_b, V_c, V_{ab}, V_{bc}, and V_{ca} are obtained. They are packed into four triangles and output. This scheme is extended easily to subdivide arbitrary level.

The proposed subdivision scheme is more efficient than recursive ones because generating one vertex only access memory one time. Since only two difference vectors and two accumulation vertices for different direction to be buffered, the required memory size is constant. Therefore, the proposed scheme is very suitable for hardware implementation.

Subdivision schemes using forward difference may result in rasterization anomaly where pixels are lost in the rendered object. As shown in Fig. 3(a), the green pixels on the teapot are the lost pixels. The forward difference technique is numerical instable with small step width [10].

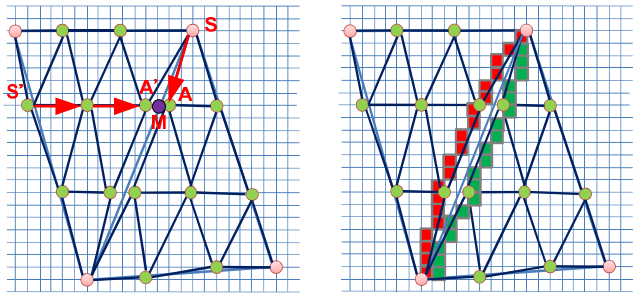


(a) Anomalies occurs (b) No Anomalies

Fig. 3: Example of teapot.

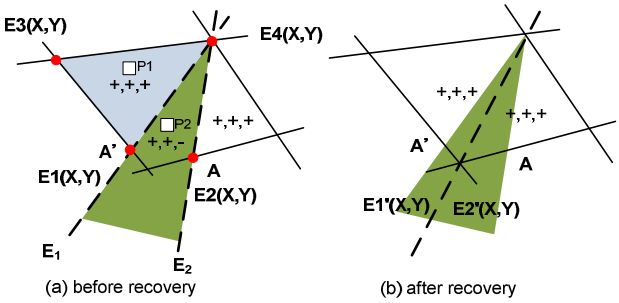
$$E_0(x,y) = a_0x + b_0y + c_0$$
$$= -(y_2 - y_1)(x - x_1) + (x_2 - x_1)(y - y_1)$$
$$= -(y_2 - y_1)*x + (x_2 - x_1)*y + (y_2*x_1 - y_1*x_2)$$
$$a_0 = -(y_2 - y_1)$$
$$b_0 = -(x_2 - x_1)$$
$$c_0 = y_2*x_1 - y_1*x_2$$

Fig. 4: Edge function definition.



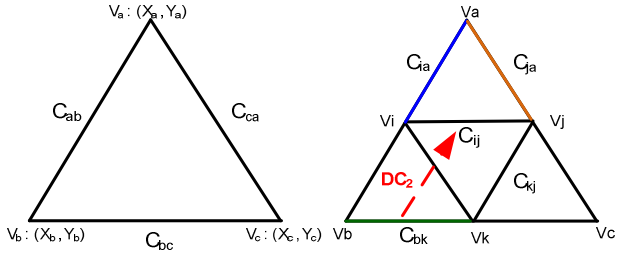
(a) After subdivision (b) Rasterization result

Fig. 5: Example of rasterization anomaly.



(a) before recovery (b) after recovery

Fig. 6: Illustration of edge function.



$$C_{ab} = Y_b*X_a - Y_a*X_b$$
$$C_{bc} = Y_c*X_b - Y_b*X_c$$
$$C_{ca} = Y_a*X_c - Y_c*X_a$$

$$C_{ia} = C_{ab}/2$$
$$C_{bk} = C_{bc}/2$$
$$C_{ja} = C_{ca}/2$$
$$DC_1 = ((X_c - X_b)*(Y_b - Y_a) + (Y_c - Y_b)*(X_a - X_b))/2$$
$$DC_2 = ((X_a - X_b)*(Y_c - Y_b) + (Y_a - Y_b)*(X_b - X_c))/2$$
$$DC_3 = ((X_b - X_a)*(Y_a - Y_c) + (Y_b - Y_a)*(X_c - X_a))/2$$
$$C_{kj} = -C_{ia} - DC_1$$
$$C_{ji} = -C_{bk} - DC_2$$
$$C_{ik} = -C_{ia} - DC_3$$

Fig. 7: Illustration of the recovery scheme.

Moreover, the vertices on the sharing edge are calculated with different starting vertex and differential vectors. Therefore, the subdivision leads to considerable numerical errors. As illustrated in Fig. 5(a), M is a vertex on the

sharing edge. Since the accumulated numerical errors, the A and A' have different coordinates. As a result, small triangles defined by A and A' are not adjacent to each other. Consequently, the rasterization anomaly occurs. Fig. 5(b) shows the result after rasterization.

To eliminate the anomalies, a recovery scheme is proposed. The proposed scheme takes the advantage of the edge function [14]. The edge function is a line equation thought two vertices as shown in Fig. 4. Pixels are regarded as being in the triangle if they are located in the region where the values of the edge functions of the triangle are all positive. For example, in Fig. 6(a), the P1 is in the triangle because it is located the blue region where the values of edge functions E1, E3 and E4 are all positive. Since A and A' have different coordinates, they define different edge functions E1 and E2 on edge E_1 and E_2 in Fig. 6(a), respectively.. The pixels, for example P2, in the green region have negative values of both E1 and E2 and are not regarded as pixels in any triangle. Thus, these pixels will not be rendered which result in rasterization anomaly. With the proposed recovery scheme, the problem can be eliminated.

In Fig. 7, C_{ab}, C_{bc}, C_{ca} are edge function constants of the original triangle. C_{ai}, C_{ij} and C_{ja} are edge function constants of the upper small triangles and can be derived from the following steps. First, by the linearity, C_{ai}, C_{ja}, and C_{bk} are calculated with dividing C_{ab}, C_{ca}, and C_{bc} by two, respectively. Secondly, the forward difference, DC_2, is derived using the formula listed in Fig. 7. Finaly, C_{ij} is derived by adding DC_2 to C_{bk}. Edge function constants of other small triangles can be derived in the similar way. After all constants are obtained, these small triangles with the derived constants in the previous steps are rendered.

Fig. 6(b) depicts an illustration of the recovery scheme. The dash line represents the edge functions E1' and E2' derived by our scheme. Edge function constants of E1' and E2' have the same magnitude and opposite sign. The pixels in the green region must have positive value of one of the two edge functions and are regarded as being in one of the two triangles. Therefore, they are rendered correctly. In Fig. 3(b), the teapot is rendered without anomalies using the recovery scheme. The proposed recovery scheme only involves one addition for evaluating one edge function constant.

III. PROPOSED SCHEMES TO IMPROVE SUBDIVSION ALGORITHM

In this section, we present three schemes to improve the proposed subdivision algorithm.

A. Dual space subdivision

Many graphic libraries support three kinds of light sources. Point and spot lights require coordinates of vertices in the eye space for light directions. Therefore, subdivision in the eye space is instinctive. After lighting, coordinates in the eye space are projected to the screen space for the rasterization. Since the projection involves the projection transform and perspective division which are expensive operations, we introduce a dual space subdivision scheme depicted in Fig 8.

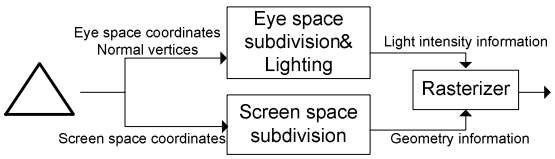


Fig. 8: Data flow of dual space subdivision.

First, the original triangle is transformed to the screen space but the eye space coordinate are maintained. Secondly, the triangle is subdivided in both spaces. Finally, for each small triangle, the eye space coordinates and normal vectors are used for lighting. The screen space coordinates are provided as the geometry information for rasterization. With this scheme, the proposed subdivision scheme is more efficient. The light intensity of the rendered image, of course, is not perspective correct. This problem is neglectable because human eye is not sensitive to the light intensity difference and perspective correctness is more important to texture coordinates, not light intensity.

B. Triangle filtering:Pre-Culling/ Clipping/Specular test

The conventional subdivision based approximating Phong shading algorithms process small triangle individually. They are not efficient because each small triangle is processed with individual complicated culling and clipping operations. Thus, a pre-culling scheme is proposed to reduce the computation. This scheme is based on the fact that the small triangles subdivided from the original triangle are lying on the same plane in the space. Therefore, they have the same face normal and the same culling test result. If the original triangle is culled, subdivision is unnecessary. Otherwise, all small triangles are rendered without culling test. The proposed pre-clipping scheme uses the same property. Once a big triangle is clipped, all small triangles from the original clipped triangle are guaranteed to be inside the viewing frustum. Therefore, no additional clipping operations were needed. The H test [11] is also adopted in the triangle filtering scheme to eliminate unnecessary subdivision operations. As illustrated in Fig. 9(a), the red region indicates the triangles do not pass the H test and will not be subdivided. Fig. 9(b) depicts the data flow of the proposed filtering mechanism. Moreover, we deliver the $H \cdot V$ terms to the Gouraud shader to eliminate the recomputation for these terms.

C. Vertex attributes sharing

The conventional subdivision based approximating Phong shading algorithms subdivide all vertex attributes including depth, fog factor, and texture coordinates. These attributes are interpolated linearly in rasterization stage using planar equation [13]. Computing the planar equation coefficients for one attribute involves a 3x3 matrix

multiplication. A sharing scheme is proposed to reduce the computation. Since the original triangle and the small triangles are lying on the same plane, they define the same plan equation. Therefore, the coefficients of the planar equation derived from the small triangles are the same.

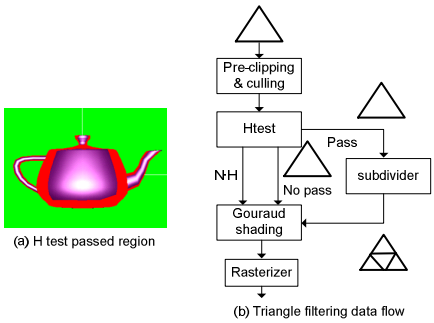


Fig. 9: Illustration of triangle filtering scheme.

Reusing these coefficients eliminates the subdividing and the setting up vertex attributes for the small triangles. However, these coefficients have to re-setup for small triangles for rasterization. The cost depends on the rastization algorithm. It takes about three multiplications to re-setup for each small triangle with tiled traversal scheme [14].

IV. COMPLEXITY ANALYSIS

In this section, we present a brief analysis and comparison for the proposed algorithm and the conventional subdivision algorithm. To simplify the comparison, we assume that one triangle is subdivided into 16 smaller ones. Each vertex has five attributes. In this case, 15 vertices where generated and they define 16 small triangles. The conventional recursive scheme requires five stack push and five stack pop operations. Each stack operation involves three writes or reads, in Fig. 1, the memory were accessed 30 times. In proposed scheme, generating one vertex only accesses memory once and the total number of memory accesses is 16.

The conventional scheme takes 15 projection transforms and perspective divisions to project the generated vertices to the screen space. With dual space subdivision scheme, it only needs three of them. The triangle filtering scheme also reduces the culling and clipping computation from 16 times to once. Setting up five attributes, in this case, requires 80 3x3 matrix multiplications. With variable sharing scheme, the expensive 3x3 matrix multiplication can be replaced with inexpensive 1x3 matrix multiplication for re-setting up each small triangle. Thus, the setup cost of the subdivided triangles is equivalent to $80*(1/3) + 5 \cong 32$ 3x3 matrix multiplications where 5 is the initial setup cost. The summary of the complexity analysis is listed in Table. 1.

Table. 1: Complexity Comparison

	Conventional subdivision algorithm	**Proposed subdivision algorithm**	**Complexity reduction in percent**
Number of memory accesses	30	16	46.6%
Number of the projective transforms and perspective divisions	15	3	80%
Number of the clipping/culling test operations	16	1	93.75%
Number of the 3x3 matrix multiplications for setup	80	32	60%

V. CONCLUSION

In this work, a subdivision algorithm for approximating Phong shading is presented. The proposed algorithm is a combination of a subdivision scheme using forward difference and a recovery scheme to prevent rasterization anomaly. Three schemes are proposed to lower the computation of subdivision-based approximating Phong shading. Compared with the conventional recursive subdivision, the proposed algorithm is more efficient and suitable for hardware implementation.

References

[1] A. Watt, "3D computer graphics," 3rd Edition, *Addison Wesley*, 2000.

[2] H. Gouraud, "Continuous shading of curved surfaces," *IEEE Trans. Comp.*, pp.623-628, June, 1971.

[3] A.T. Phong, "Illumination for computer generated pictures," *Communications of the ACM*, Vol. 18, No. 6, June, pp.311-317, 1975.

[4] G. Bishop, and D. M. Weimer., "Fast Phong Shading," *Proc. Computer Graphics and interactive Technique*, 1986, pp.103-106.

[5] J. Pöpsel, and Ch. Homung, "Highlight shading lighting and shading in a PHIGS+PEX environment," *EUROGRAPHICS*, 1989, pp.317-332.

[6] A. A. Mohamed, L. S. Kalos, and T. Horváth., "Hardware implementation of Phong shading using spherical interpolation," *Periodica Polytechnica* Vol. 44, Nos 3-4, 2000.

[7] T. Barrera, A. Hast, and E. Bengtsson., "Faster shading by equal angle interpolation of vectors," *IEEE Trans. Visualization and Computer Graphics*, pp.217-223, March, 2004.

[8] A. A. Mohamed, L. S. Kalos, G. Szijártó, T. Horváth, and T. Fóris., "Quadratic interpolation in hardware Phong shading and texture mapping," *SCCG'01*, April, 2001, pp.181-188.

[9] T. Barrera, A. Hast, and E. Bengtsson., "Fast near Phong-quality software shading," *WSCG'06*, January, 2006, pp.109-115.

[10] S. Bischoff, L.P. Kobbelt, and H.P. Seidel, "Toward hardware implementation of Loop subdivision". *Proc. SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, 2000, pp.41-50.

[11] K. Harrison, D. A. P. Mitchell, and A. H. Watt., "The H-test: a method of high speed interpolative shading," *Proc. New Trends in CG.*, 1988, pp.106-166.

[12] Y. Cho, U. Neumann, and J. Woo., "Improved specular highlights with adaptive shading," *Proc. of CG. International,* June, 1996, pp.38-46.

[13] M. Olano, and T. Greer., "Triangle scan conversion using 2D homogeneous coordinates," *Proc. SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, August, 1997, pp.89-95.

[14] J. McCormack and R. McNamara., "Tiled polygon traversal using half-plane edge functions," *Proc. SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware,* 2000, pp.15-21

A Synthesis-Quality-Oriented Depth Refinement Scheme for MPEG Free Viewpoint Television (FTV)

Chun-Chi Chen[1], Yi-Wen Chen[2], Wen-Hsiao Peng[3] and Fu-Yao Yang
Department of Computer Science, National Chiao Tung University
1001 Ta-Hsueh Rd., HsinChu 30010, Taiwan
E-mail:{[1]cheerchen.cs96g@g2.nctu.edu.tw, [2]ewchen@csie.nctu.edu.tw, [3]w.peng@ieee.org}

Abstract

This paper addresses the problem of refining depth information from the received reference and depth images within the MPEG FTV framework. An analytical model is first developed to approximate the per-pixel synthesis distortion (caused by depth-image compression) as a function of depth-error variances, intensity variations, ground-truth depth and virtual camera locations. We then follow the model to detect unreliable depth pixels by inspecting intensity gradients and to refine their values with a candidate-based block disparity search. Additional side information is transmitted to make both operations robust against compression effects. Experimental results show that our scheme offers an average PSNR improvement of 1.2 dB over MPEG FTV and consistently outperforms the state-of-the-art methods. Moreover, it can remove synthesis artifacts to a great extent, producing a result that is very close in appearance to the ground-truth view image.

1 Introduction

Technology evolution in the capture and display of 3D videos will soon extend visual sensation from 2D to 3D while allowing unrestricted spatiotemporal scene navigation. In general, offering a 3D depth impression of a real-world scene requires two separate images captured from properly arranged viewing positions. To enable scene navigation, a multi-view video may have to be acquired through a dense camera set-up. However, due to the complexity involved in acquisition, storage and transmission, it is unlikely to have a large number of camera inputs. An efficient 3D data format is thus needed to allow the generation of intermediate views from a sparse sampling of the observed scene.

For this, the MPEG committee has recently defined a "multi-view video plus depth" data format [1], which specifies a way of multiplexing the coded representations of a multi-view video and its associated per-pixel depth information. With explicit scene geometry, an arbitrary virtual view can be generated at the receiver side by means of the so-called depth-image-based rendering (DIBR) [2][3][4][5], requiring only a small number of view images for scene navigation. Since depth images must be conveyed together with the corresponding view images, both types of scene representations are compressed, based mostly on H.264/AVC, for an efficient use of network bandwidth.

Although block-based hybrid coding is equally applicable to depth-image compression, it causes undesirable synthesis artifacts. This is because depth images represent scene geometry information, the characteristics of which are very different from those of intensity data. It was shown in [6] that visually imperceptible depth errors can still have a profound effect on synthesis quality.

A few approaches have been proposed to alleviate synthesis artifacts caused by depth-image compression. In [7] Tanimoto et al. found that the magnitude of synthesis errors is linearly proportional to the distance between the virtual and reference cameras. They proposed to compensate the synthesis errors in a virtual view by estimating their magnitudes from the errors observed in a nearby reference view. Sung et al. [8], on the other hand, made use of the Lambertian condition to refine depth images. The process involves using the similarity between the depth (and intensity) values of corresponding pixels to detect unreliable depth pixels and then refining their values through a group-based disparity search. Because both schemes rely entirely on the decoded information for intensity correction or depth refinement, their performance is greatly influenced by compression effects.

In this paper, we propose a synthesis-quality-oriented depth refinement scheme. Rather than trying

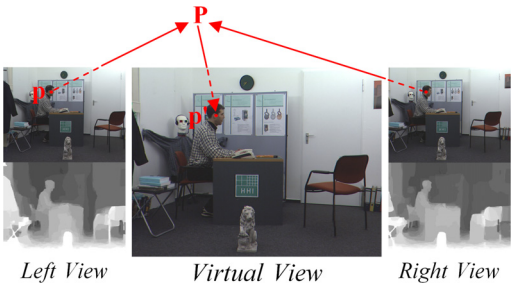


Figure 1. View synthesis based on multi-view video plus depth.

to minimize depth errors, our scheme, as implied by its name, intends to detect and refine only those depth pixels that are highly sensitive to errors. An analytical model was derived to measure how sensitive a depth pixel is to its error in terms of synthesis distortions. The model was also used as a guide for detection and refinement. In order for the two operations to be able to adapt to statistical changes due to compression effects, the settings of their control parameters are first determined at the sender side by evaluating the performance as perceived by the receiver over the range of all possible choices, and then sent to the receiver as the side information. Although extra bits are required for signaling, the overhead is negligible and justified by the significant improvement in synthesis quality. Experimental results show that the proposed scheme has an average PSNR gain of 1.2dB over MPEG FTV and consistently outperforms the state-of-the-art methods.

This paper is organized as follows: Section 2 contains a brief review of DIBR. Section 3 introduces an analytical model for characterizing synthesis distortions caused by depth-image compression. Section 4 describes our proposed synthesis-quality-oriented depth refinement scheme. Section 5 compares the proposed scheme with the state-of-the-art approaches in terms of synthesis quality. The paper is concluded with a summary of our observations.

2 Depth-Image-based Rendering

Depth-image-based rendering (DIBR) is a view generation method that renders virtual views of a scene from a known reference image and its associated per-pixel depth information. Often referred to as 3D image warping, the process involves first reprojecting the reference image into the 3D space utilizing its depth information, followed by the projection of the reconstructed scene onto the image plane of a virtual view camera. The warping defines a vector-valued function Ψ that takes pixel coordinates $\mathbf{p} = [x, y]^T$ in the reference view as input and returns the corresponding coordinates $\mathbf{p}' = [x', y']^T$ in the virtual view as output:

$$\Psi : \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} \mathbf{p}' \\ 1 \end{bmatrix} = \mathbf{A}'\mathbf{R}\mathbf{A}^{-1} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} + \frac{1}{Z_p}\mathbf{A}'\mathbf{T}, \quad (1)$$

where the rotation and translation matrices, $\mathbf{R}$ and $\mathbf{T}$, specify the relative position of the virtual camera; $\mathbf{A}'$ and $\mathbf{A}$ indicate respectively the intrinsic parameters of the virtual and reference cameras; and Z_p is the depth value associated with $\mathbf{p}$. In the above, we have tacitly assumed parallel camera configuration. The reader is referred to [2] for details. For brevity we use $\Psi(\mathbf{p}; Z_p)$ to denote the warping of the pixel $\mathbf{p}$.

Eq. (1) establishes a depth dependent relation between the pixel coordinates of corresponding points in an image pair. According to the equation, an arbitrary virtual view can in principle be generated, provided that the depth value Z_p is known for every pixel $\mathbf{p}$ in the reference image and that camera parameters are available. In practice, however, the viewpoint navigation is constrained by disocclusion problems: "holes" appear in synthesized images if areas occluded in the reference view become visible in a virtual view. Such artifacts become most obvious when the virtual view is very far away from its reference.

To reduce the effects of disocclusion, the MPEG committee has recently proposed a "multi-view video plus depth" data format that enables the generation of a virtual view to make use of more than one reference view. Figure 1 shows a classic illustration of the view synthesis based on such data format. In the example, each pixel in the virtual view is formed by a weighted sum of its corresponding points in the two reference views, and depending on the disocclusion level, the weight vector can vary from one pixel to another. To find the corresponding points, depth images must be transmitted along with their video signals. Due to the enormous amount of data involved, both view and depth images must be compressed prior to transmission. The influence of depth-image compression on synthesis quality is the subject of the next section.

3 Per-Pixel Synthesis Distortion Model

In this section, we introduce an analytical model for characterizing synthesis distortions caused by depth-image compression. The model is explained with reference to Figure 2, which illustrates an example of disparity-compensated interpolation based on an impaired depth representation. In the figure, I_T denotes

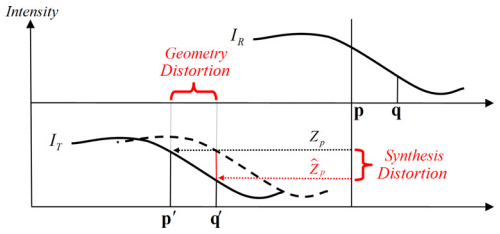


Figure 2. Disparity-compensated interpolation using an impaired depth representation.

a virtual view image generated from the reference image I_R utilizing its ground-truth, per-pixel depth information. As mentioned previously, the warping Ψ establishes a relation between the intensity values of the reference and virtual images: $\mathbf{p}' = \Psi(\mathbf{p};Z_p)$ and $I_T(\mathbf{p}') = I_R(\mathbf{p})$. To simplify our discussion, we assume that the reference image I_R is without coding errors. The more general case can be analyzed along the same lines of derivations that follow.

To examine the influence of depth-image compression on synthesis quality, we approximate the coding effects of depth images by an additive noise model, i.e., $\widehat{Z}_p = Z_p + n_p$. Through the warping function Ψ, the depth error n_p causes the projection of the pixel $\mathbf{p}$ to move from $\mathbf{p}' = \Psi(\mathbf{p};Z_p)$ to $\mathbf{q}' = \Psi(\mathbf{p};\widehat{Z}_p)$**;** the effect is known as *geometry distortion*. It then follows that $I_R(\mathbf{p})$ is substituted for $I_T(\mathbf{q}')$ as the intensity value of the pixel $\mathbf{q}'$; the squared difference indicates the synthesis distortion contributed by n_p:

$$\begin{aligned}\xi_p &\triangleq \left(I_R(\mathbf{p}) - I_T(\mathbf{q}')\right)^2 = \left(I_R(\mathbf{p}) - I_R(\mathbf{q})\right)^2 \\ &\approx \left(I_R(\mathbf{p}) - I_R(\mathbf{p}) - \nabla I_R(\mathbf{p})\cdot(\mathbf{q}-\mathbf{p})\right)^2 \\ &= \left(-\nabla I_R(\mathbf{p})\cdot(\mathbf{q}-\mathbf{p})\right)^2, \qquad (2)\end{aligned}$$

where $\mathbf{q}'$ is inversely projected to I_R by the inverse mapping function $\Psi^{-1}(\mathbf{q}';Z_q)$ and a Taylor's series expansion is used to approximate $I_R(\mathbf{q})$. Recognizing that $\mathbf{q}' = \Psi(\mathbf{q};Z_q) = \Psi(\mathbf{p};Z_p + n_p)$, we solve for the vector difference $(\mathbf{q}-\mathbf{p})$ as

$$\mathbf{q}-\mathbf{p} = \frac{-n_p}{Z_q(Z_p+n_p)}\mathbf{c},$$

where $\mathbf{c} = \begin{bmatrix}\mathbf{I}_2 & \mathbf{0}_{2\times 1}\end{bmatrix}\mathbf{A}\mathbf{R}^{-1}\mathbf{T}$ is a vector depending solely on camera parameters. Substituting this result into Eq. (2) then yields

$$\xi_p \approx \left(\frac{n_p}{Z_q(Z_p+n_p)}\nabla I_R(\mathbf{p})\cdot\mathbf{c}\right)^2. \qquad (3)$$

Now let us consider parallel camera configuration, with which the vector $\mathbf{c}$ has the form of $[c,0]^T$ where $|c|$ is proportional to the distance between the reference and virtual cameras. Then it is obvious that

$$\xi_p \approx \left(\frac{n_p}{Z_q(Z_p+n_p)}\right)^2 \times g_x^2(\mathbf{p}) \times c^2, \qquad (4)$$

where $g_x(\mathbf{p})$ denotes the x component of the gradient $\nabla I_R(\mathbf{p}) = [g_x(\mathbf{p}), g_y(\mathbf{p})]^T$ computed at $\mathbf{p}$. To obtain the expected per-pixel synthesis distortion conditioned on ground-truth depth values, we take conditional expectations of both sides and expand $\left(n_p/Z_q(Z_p+n_p)\right)^2$ into its Taylor series in n_p:

$$\begin{aligned}&E\{\xi_p|Z_p,Z_q\} \\ \approx\ & E\left\{\left(\frac{n_p}{Z_q(Z_p+n_p)}\right)^2|Z_p,Z_q\right\}\times m_g^{(2)}(\mathbf{p})\times c^2 \\ =\ & \frac{1}{Z_q^2}\times\left(\frac{E\{n_p^2\}}{Z_p^2} - 2\frac{E\{n_p^3\}}{Z_p^3} + 3\frac{E\{n_p^4\}}{Z_p^4} - \ldots\right) \\ & \times m_g^{(2)}(\mathbf{p})\times c^2 \\ =\ & \frac{1}{Z_q^2}\times\left(\frac{\sigma_n^2(\mathbf{p})}{Z_p^2} + 9\frac{\sigma_n^4(\mathbf{p})}{Z_p^4} + 75\frac{\sigma_n^6(\mathbf{p})}{Z_p^6} + \ldots\right) \\ & \times m_g^{(2)}(\mathbf{p})\times c^2 \\ \approx\ & \frac{1}{Z_q^2}\times\frac{\sigma_n^2(\mathbf{p})}{Z_p^2}\times m_g^{(2)}(\mathbf{p})\times c^2, \qquad (5)\end{aligned}$$

where $m_g^{(2)}(\mathbf{p}) = E\{g_x^2(\mathbf{p})\}$ can be viewed as a measure of how rapidly the intensity changes along the horizontal direction at $\mathbf{p}$, and $\sigma_n^2(\mathbf{p})$ indicates the corresponding depth-error variance. In the above, n_p is assumed to be independent of $g_x(\mathbf{p})$ and to obey the normal distribution, i.e., $n_p \sim N(0,\sigma_n^2(\mathbf{p}))$. The last approximation in Eq. (5) is justified because Z_p is usually much greater than $\sigma_n(\mathbf{p})$.

Eq. (5) provides a non-stationary model for the expected per-pixel synthesis distortion, which suggests that the depth error for different pixels should have different contributions to the overall synthesis distortions. From the equation, the distortion caused by n_p is determined by several factors measured at $\mathbf{p}$: the depth-error variance, the intensity variation, the (ground-truth) depth value**,** as well as the position of the virtual camera. Further insight into the combined effects of these factors is gained by looking at Figure 3, which displays the ratio of Z_p to $\sigma_n(\mathbf{p})$ as a function of $E\{\xi_p|Z_p,Z_q\}$, under various settings of Z_p, Z_q, and $m_g^{(2)}(\mathbf{p})$ simulating smoothly- or rapidly-changing depth/intensity fields. In the experiment, $\sigma_n(\mathbf{p})$ was varied to identify the highest level of error variance at which the specified distortion is achieved. The result is then used to compute $Z_p/\sigma_n(\mathbf{p})$. Intuitively, the ratio, which we call *depth-error sensitivity*, characterizes

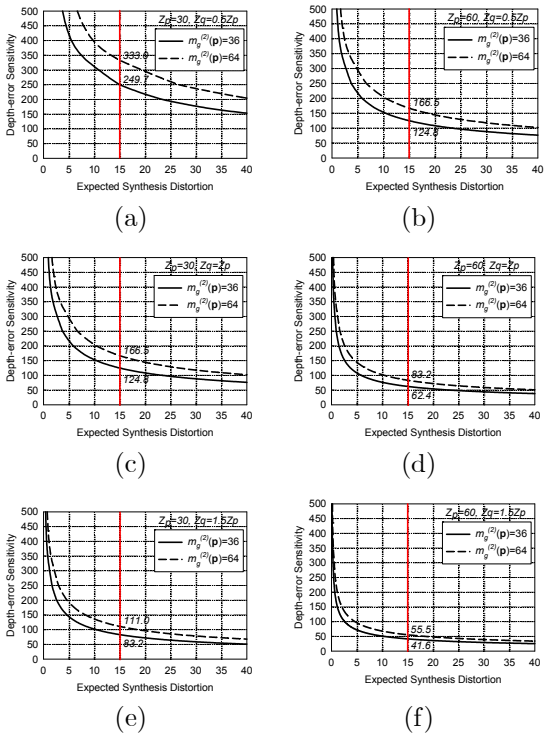


Figure 3. Measuring the depth-error sensitivity under various settings of Z_p, Z_q and $m_g^{(2)}(\mathbf{p})$.

how sensitive a pixel is to its depth error in terms of the extent of synthesis distortions. A higher ratio (sensitivity) implies that a small error in depth can lead to a considerable distortion.

From the figure, several important observations can be made:

1. Compare the curves produced with different settings of $m_g^{(2)}(\mathbf{p})$. The larger the value of $m_g^{(2)}(\mathbf{p})$, the more sensitive the pixel $\mathbf{p}$ is to its depth error; namely, when depth errors happen in areas with vertical edges or fine texture details, their effects on synthesis quality are more apparent. This observation is also corroborated by [7].

2. Compare parts (a)(c)(e) with parts (b)(d)(f). When a pixel corresponds to a farther clipping plane, it exhibits a lower depth-error sensitivity. In this case, the pixel has a larger depth value Z_p and according to Eq. (1), the resulting geometry distortion is less significant.

3. Compare part (e) with parts (a)(c) (or (f) with (b)(d)). When a pixel $\mathbf{p}$ is ill-warped to $\mathbf{q}'$, the resulting synthesis error is less observable if Z_q is much greater than Z_p (and hence $\widehat{Z}_p$). The result can be explained using the example shown in Figure 4, where $\mathbf{q}_1$ and $\mathbf{q}_2$ denote respectively the inverse projections of $\mathbf{q}'$ for the two extreme cases: $Z_{q1} \gg \widehat{Z}_p$ and $Z_{q2} \ll \widehat{Z}_p$. Since $Z_{q1} \gg \widehat{Z}_p \approx Z_p \gg Z_{q2}$, the pixel $\mathbf{p}$ is much closer to $\mathbf{q}_1$ than to $\mathbf{q}_2$. In general, pixels that are spatially closer to each other exhibit higher intensity correlation, which explains the less significant change in intensity when $Z_q \gg Z_p$.

4. Observe the reciprocal relation between $\sigma_n^2(\mathbf{p})/Z_p^2$ and c^2 in Eq. (5). It suggests that when a pixel $\mathbf{p}$ is warped to a virtual view that is farther away from the reference view, it is more sensitive to depth errors.

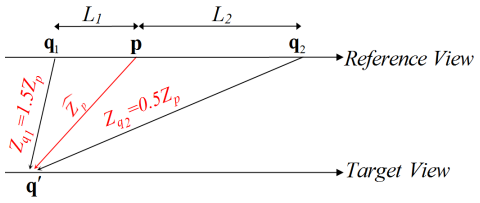


Figure 4. A geometrical interpretation of the effect of Z_q on depth-error sensitivity.

These observations remain valid for other camera configurations, except that the effects of the intensity variation and camera arrangement must jointly be considered by evaluating $E\{(\nabla I_R(\mathbf{p}) \cdot \mathbf{c})^2\}$.

4 Algorithm Details

The framework of MPEG FTV [9] views the transmitted depth images as deterministically specifying the depth information for the reference images. The compression effects of depth images were neglected during the rendering of virtual views. As seen from the analysis in §3, depth errors can cause disturbing synthesis artifacts, especially at areas with sharp edges or fine texture details. To tackle the problem, we propose to regard both the received view and depth images as sources of information about the ground-truth depth of the scene, and provide ways to detect and refine unreliable depth values.

4.1 System Architecture

To allow for an easier understanding of our algorithm, Figure 5 depicts the system block diagram with

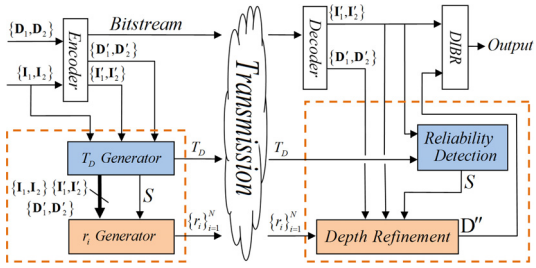


Figure 5. System Block Diagram.

a highlight on the data communicated between functional blocks. As shown, for an economic use of network bandwidth, both reference images $\{I_1, I_2\}$ and their respective per-pixel depth information $\{D_1, D_2\}$ are compressed prior to transmission. These data are decoded and reconstructed at the receiver side before they are used for the creation of virtual views. The "prime" symbols in the figure differentiate the coded view and depth images from their original sources.

Recognizing that depth-image compression may give rise to depth errors, we introduce a depth refinement mechanism at the receiver side. The objective is to improve synthesis quality by refining the depth values for those pixels (which we call *unreliable pixels*) being highly sensitive to depth errors. The process consists of two sequentially operated steps: (1) the detection of unreliable pixels and (2) the refinement of their depth values, both need to access the coded view and depth images. To make their performance robust against compression effects, additional control parameters are transmitted to the receiver as the side information, with their settings being determined at the sender side by evaluating the detection and refinement quality as perceived by the receiver over the range of all possible choices. The details are elaborated in the subsequent sections.

4.2 Reliability Detection

The detection process at the receiver side aims to discover unreliable pixels–i.e., those that are highly sensitive to depth errors and hence require higher fidelity for their depth values in order to minimize rendering errors. From the theoretical analysis in §3, a pixel is likely to be unreliable if it locates in a region with large intensity variation, or if it represents a pixel in a near clipping plane. Although both facts can jointly be utilized to form detection criteria, we consider only the use of intensity variation because view images are generally better compressed than their depth representations, making the intensity information more reliable for decision-making.

To quantify intensity variation, we adopt the Gaussian derivative operator to compute gradient for all the pixels in view images. A pixel $\mathbf{p}$ is considered to be unreliable and its depth value deserves refining if the magnitude $\|\nabla I'_R(\mathbf{p})\|$ of its gradient exceeds a given level T_D[1]. According to Observation #1 in §3, such a pixel is highly sensitive to depth errors, hence requiring higher precision for its depth value. Apparently, the value of T_D plays a pivotal role in determining the detection accuracy. With non-stationary signal statistics, we propose to adapt T_D on a frame-by-frame basis. This is realized by transmitting its value as the frame-level side information.

In determining the value of T_D for a particular frame, we wish to strike a good balance between the hit and false-alarm rates. The best setting of T_D, denoted by T_D^*, should have the subset of pixels $\mathcal{S}(T_D^*) = \{\mathbf{p} : \|\nabla I'_R(\mathbf{p})\| > T_D^*\}$ contain as many unreliable pixels as possible while keeping the number of reliable ones to be minimal. To find T_D^*, we first associate each plausible choice of T_D and the corresponding set of pixels $\mathcal{S}(T_D)$ with a matching score that weights the hit rate against the false-alarm rate:

$$J(T_D) = \sum_{\mathbf{p}\in\mathcal{S}(T_D)} \left(\mathbf{1}_{\mathcal{S}}(\mathbf{p})\xi_p - (1 - \mathbf{1}_{\mathcal{S}}(\mathbf{p}))\pi\right),$$

where $\mathbf{1}_{\mathcal{S}} : \mathbf{p} \in \mathcal{S} \rightarrow \{0, 1\}$ is an indicator function defined as

$$\mathbf{1}_{\mathcal{S}}(\mathbf{p}) = \begin{cases} 1 & \text{if } \xi_p \geq \delta \\ 0 & \text{if } \xi_p < \delta \end{cases}.$$

Then we choose, among all possible choices, the one that yields the highest matching score, i.e., $T_D^* = \arg\max_{T_D} J(T_D)$. The approach can be interpreted as to evaluate, at the sender side, the detection quality as perceived by the receiver.

In the course of computing the matching score, it is necessary to decide whether a hit or false alarm occurs. This is accomplished by evaluating the per-pixel synthesis distortion ξ_p at the sender side with I_1 and I_2 (or in the reverse order) being used in place of I_R and I_T, respectively (cf. Eq. (2)). Specifically, if ξ_p is greater than or equal to a threshold δ, indicating that the depth associated with the pixel $\mathbf{p}$ may be unreliable, a hit is identified; otherwise, a false alarm is signaled. Ideally, the δ should be set to zero according to the Lambertian condition; however, in practice a non-zero value was used to compensate camera noises and illumination difference between view images. The

[1] With parallel camera configuration, only the x component of the gradient is computed and compared with T_D (cf. Eq. (4)). Also, I'_R represents a coded reference image.

settings of δ and π that yield the best synthesis quality (in terms of PSNR) are searched exhaustively at the sender side. Note that they need not be transmitted to the receiver.

4.3 Depth Refinement

After we discover all the unreliable pixels, our next step is to refine their depth values. Because depth refinement is performed by the receiver, its operation must be made computationally simple and efficient. For this reason, we adopt a candidate-based disparity estimation scheme to derive depth from the received view images. As in most block-based algorithms, a constant disparity is searched for each block of pixels (of size 7×7), centered on an unreliable pixel $\mathbf{p}$, by minimizing the error between the two view images after disparity compensation. However, unlike their techniques, which usually require examining a large number of disparities, ours restricts the search to only those disparities that correspond to an integer depth value in the interval of $[\widehat{Z}_p - R_p, \widehat{Z}_p + R_p]$. On one hand, this constraint is an expediency out of complexity considerations, and on the other hand, it prevents the simple block-based search from getting an improper disparity.

Although reducing the number of search candidates helps to simplify the disparity search, the issues are how to determine a proper value of R_p for each unreliable pixel and how to signal the information efficiently. As described previously, the value of R_p determines the maximum modification of $\widehat{Z}_p$ that can be caused by depth refinement–i.e., it controls the strength of refinement. It was found in our analysis that the depth error sensitivity of a pixel is related to its ground-truth depth value, implying that the adaptation of R_p should refer to the value of $\widehat{Z}_p$ (which is an approximation of Z_p). For a trade-off between quality and overhead, we divide the set $\mathcal{S}(T_D^*)$ into N disjoint subsets $s_i(T_D^*), 1 \leq i \leq N$, each of which is assigned a refinement search range r_i. A uniform quantizer that operates on the received depth $\widehat{Z}_p$ is used to categorize the unreliable pixels in $\mathcal{S}(T_D^*)$ into one of the N subsets. After that, the best settings of $\{r_i\}_{i=1}^N$ are searched exhaustively at the sender side and transmitted to the receiver as the side information.

Figure 6 shows a sample result of our refinement process. Observe that depth compression introduces blocking artifacts on the decoded depth image (see Figure 6 (b)(e)). With depth refinement, we can remove the artifacts largely (see parts (c) and (f) of Figure 6); note the clarity of object boundaries that simply are not visible in the decoded depth image. Interestingly, the refinement can even recover some details that are removed by the enforcement of depth smoothing (compare parts (a)(d) and (c)(f) of Figure 6).

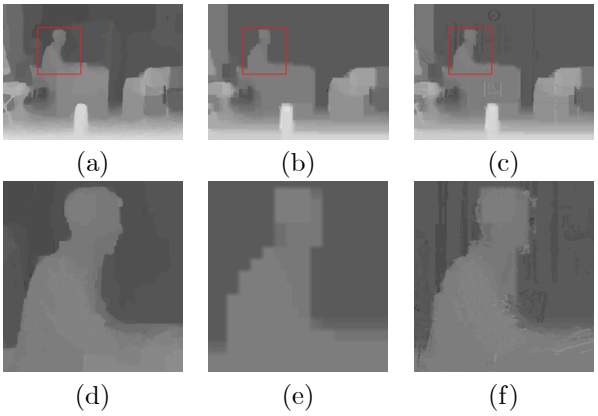


Figure 6. A sample result of the proposed depth refinement algorithm: (a)(d) the original depth image, (b)(e) the decoded depth image, and (c)(f) the refined depth image.

5 Experiments

Simulation was carried out to demonstrate the performance of the proposed scheme, and the results were compared with that of [7] and [8]. All the refinement schemes were implemented with the MPEG committee software VSRS 2.1. All experiments used DERS 2.0 to generate depth images and JMVC 3.0.1 to encode multi-view videos and their depth. The average PSNR of synthesized images was computed based on the first 100 frames of each test sequence. Particularly, in implementing the method described in [7], we employed the magnitude of synthesis errors rather than manually generated edge maps to distinguish pixels of different categories. For a fair comparison, all the threshold values used in [7] and [8] were determined by optimizing the quality of synthesized images.

Figure 7 compares the PSNR of various schemes when the depth QP is varied from 22 to 44. The curves associated with MPEG FTV were produced without depth refinement. To see the effects of reference quality, parts (a) and (b) show the results generated utilizing high-quality references (QP=22), whereas parts (c) and (d) are their low-quality counterparts (QP=31). It can be seen that all three schemes outperform MPEG FTV in all test sequences, and as expected, the improvement is the greatest when depth images are coarsely quantized. Moreover, ours has the highest gain of all the schemes–an average PSNR improvement of 1.2dB over MPEG-FTV. The results are

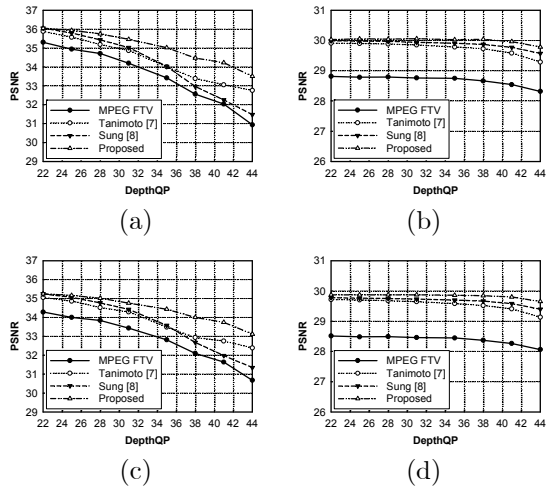


Figure 7. PSNR of synthesized images as a function of the depth and reference QP. The reference view images are coded with QP=22 (a)(b) and QP=31 (c)(d).

consistent with different test conditions.

Figure 8 further compares the subjective quality of synthesized images. Part (a) illustrates what can happen if incorrect depth information is used for view synthesis. Parts (b) through (d) show the results obtained by correcting depth with one of the three schemes just described (i.e., [7], [8], and ours). As can be seen, "ghost effects" appear around object boundaries if the depth is not refined; in comparison, the visual results with depth refinement are considerably improved. Our scheme even produces a result that is very close in appearance to the ground-truth view image. The reason behind the superior performance can be explained with Figure 9, which makes visible the unreliable pixels detected by the three schemes. As expected, our scheme tends to correct more depth pixels locating in areas with fine texture details or vertical edges–namely, those that will crucially affect synthesis quality.

6 Conclusion

To alleviate the coding effects of depth images, we proposed in this paper a synthesis-quality-oriented depth refinement scheme. The approach is characterized by the unique consideration of attempting to refine only those depth pixels that are likely to cause noticeable synthesis artifacts. In the course, we developed an analytical model to establish criteria for reliability detection and to form guidelines for depth refinement. Since both operate on the decoded information, additional side information is transmitted to make them robust against compression effects. Experimental results show that our scheme has the highest PSNR gain of all the state-of-the-art methods. It also produces a result that is visually similar to the ground-truth image. Better performance is expected with the incorporation of more sophisticated disparity search. Besides, the analytical model can find its application in developing depth compression algorithms.

References

[1] C. Fehn, R. Barre, and R. S. Pastoor, "Interactive 3-DTV: Concepts and Key Technologies," *Proceedings of the IEEE*, vol. 94, pp. 524–538, March 2006.

[2] C. Fehn, "A 3D-TV Approach Using Depth-Image-Based Rendering (DIBR)," *Proceedings of Visualization, Imaging, and Image Processing*, September 2003.

[3] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski, "High-Quality Video View Interpolation Using a Layered Representation," *ACM Transactions on Graphics*, vol. 23, pp. 600–608, August 2004.

[4] A. Smolic, K. Muller, K. Dix, P. Merkle, P. Kauff, and T. Wiegand, "Intermediate View Interpolation based on Multiview Video plus Depth for Advanced 3D Video Systems," *IEEE Int'l Conf. on Image Processing*, October 2008.

[5] E. Cooke, P. Kauff, and T. Sikora, "Multi-view Synthesis: A Novel View Creation Approach for Free Viewpoint Video," *Signal Processing: Image Communication*, vol. 21, pp. 476–492, July 2006.

[6] P. Merkle, A. Smolic, K. Muller, and T. Wiegand, "Multi-view Video plus Depth Representation and Coding," *IEEE Int'l Conf. on Image Processing*, October 2007.

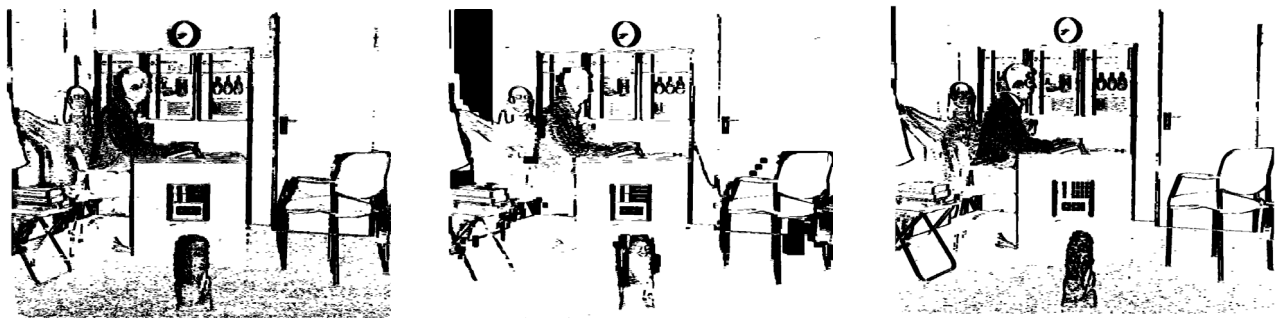
[7] M. Tanimoto, T. Fujii, M. P. Tehrani, M. Wildeboer, and H. Furihata, "Error Cancellation in Free-viewpoint Image Generation for FTV," *ISO/IEC JTC1/SC29/WG11, MPEG09/M16607*, April 2009.

[8] J. Sung, Y. J. Jeon, J. H. Lim, and B. M. Jeon, "Improving View Synthesis Results based on Depth Quality Measure," *ISO/IEC JTC1/SC29/WG11, MPEG09/M16417*, April 2009.

[9] "Applications and Requirements on 3D Video Coding," *ISO/IEC JTC1/SC29/WG11, MPEG09/N10570*, April 2009.

(a) (b) (c) (d)

Figure 8. Subjective quality comparison of synthesized images: (a) MPEG FTV (without depth refinement), (b) Tanimoto [7], (c) Sung [8] and (d) the proposed scheme. The depth QP is set to 44.



(a) (b) (c)

Figure 9. Pixels whose depth values are judged unreliable: (a) Tanimoto [7] (category 2), (b) Sung [8] and (c) the proposed scheme.

Fast Host Service Interface Design for Embedded Java Application Processor

Kuan-Nian Su
Department of Computer Science
National Chiao Tung University
Hsinchu, Taiwan

Chun-Jen Tsai
Department of Computer Science
National Chiao Tung University
Hsinchu, Taiwan
cjtsai@cs.nctu.edu.tw

***Abstract*—In this paper, we have proposed a fast inter-processor communication interface (IPC) for a dual-core Java application processor. The dual-core Java application processor is a SoC composed of a RISC core and a double-issued Java bytecode execution core. The proposed fast IPC mechanism provides Java system software a high-level way to invoke a host processor service routine from Java source code. The proposed IPC has much lower overhead than that of the standard Java Native Interface (JNI). Unlike other fast native call interface designed for VM interpreter, the proposed IPC mechanism is exclusively designed for the communication between two physical hardwired processor cores. Based on the experimental results, the proposed mechanism is very promising for embedded Java runtime environment.**

I. INTRODUCTION

Java runtime environment is becoming very important for embedded applications. Due to its machine code-level portability across different operating systems and processors, it has been selected by many standard organizations (such as 3GPP and DVB) as the standard application environment for multimedia-capable mobiles and set-top boxes [6][7]. A dual-core Java application processor was proposed in [1]. The Java application processor is composed of a host processor core (PowerPC 405 in [1]) and a double-issue Java bytecode execution core. The later is referred to as the Bytecode Execution Engine (BEE) core.

In order to support the full Java runtime environment (JRE), the proposed joint software-hardware architecture is shown in Fig. 1. Upon the execution of a Java application, the class loader running on the host processor core will load and parse the main class file into the runtime method/class data structures and store the runtime information in the method area. The BEE core will then be initialized to fetch-and-execute the byte codes of the application class files from method area. In order to reduce the implementation cost of the Java bytecode execution core logic, some complex instructions such as the floating point operations, system resource access operations(e.g. for memory allocation, media accelerators, and I/O service), etc., will be implemented on the host processor core as service routines. The communication between the BEE core and the host processor core must be achieved through some efficient inter-processor communication interface.

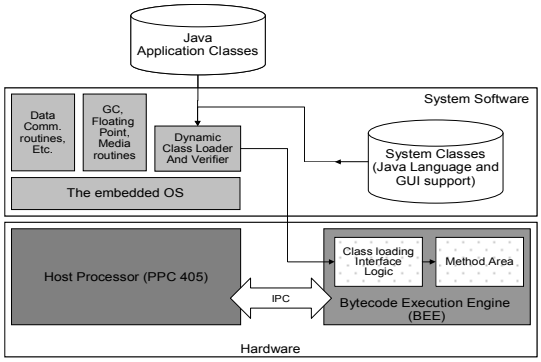


Fig. 1 Proposed dual-core JRE architecture.

Ideally, the IPC should allow the Java source code to invoke a host processor service routine. One possible approach is to implement the Java Native Interface (JNI) [2]. JNI is designed to handle situations where Java applications need to call a library implemented in native binary form (usually a dynamic loading library). As a two-way interface, the JNI can support two types of function calls: down-calls and up-calls. Down-calls are cases when a Java application invokes native functions, while up-calls are cases when native functions invoke JNI interface function to access the Java application resource such as field and data. Due to its general and flexible nature, JNI overhead are pretty high. It involves call stack conversion and dynamic native function loading and calling (automatically handled by the OS). Therefore it is not efficient enough for the one-way host system service invocation we need in the proposed dual-core JRE architecture.

Most VM implementation leaves a back-door interface for fast native system function calls. However, the designs are in general for software-based VM interpreters. Therefore, calling native operating system services from a native VM interpreter application are quite straightforward. In this paper, the design of a Fast Host Service Interface for invoking host system

routines from the Java processor is proposed. A special Java class, *mmes.native*, as a one way BEE-to-host calling interface is implemented to make services calls to host processor transparent to Java source programs. Any references to the methods in this special class from a Java application will be intercepted at the byte code level by the BEE core and turned into native calls to host service routines. Neither stack conversion nor data structure conversion is necessary for such native calls since all the parameters will be passed into host processor directly by exporting the internal Java stacks to the host processor via memory-mapped I/O mechanism. FHSI aims to provide an extendable and efficient design for inter-processor communications between the host processor and the Java bytecode processor.

The paper is organized as follows. Section II describes the details of FHSI, including runtime method resolution and the parameter passing mechanism. Section III describes the implementation platform and shows some experimental results. Some concluding remarks are given in section IV.

II. FAST HOST SERVICE INTERFACE

Proposed Fast Host Service Interface has two major steps. At first, Java application invocates the method defined in *mmes.HostService* through fast dynamic method resolution. Then, interrupt will be enabled to pass arguments to host system service routine.

A. Fast Dynamic Method Resolution

In our proposed dual-core JRE architecture, the dynamic class loader is a routine running on the host processor core. It locates and loads Java classes upon the request of the BEE core (triggered by, for example, a "new" instruction). This dynamic class loader shall not be mistaken as the Java class loader. Note that there can be more than one class loaders in a Java application, but the dynamic class loader running on the host processor is unique. This loader converts class files into Java runtime information images and incorporate them into a large runtime information structure. Each class file is stored in a structure shown in Fig. 2, which is composed of four parts, including class table of Constant Pool TOC (TOC), constant pool, field, and method information.

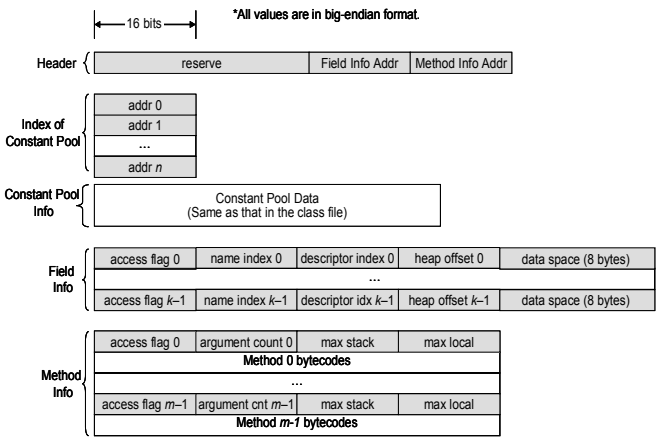


Fig. 2 Definition of Java runtime information.

The last three parts are copied directly from the original java class file and the offset address of field information and method information can be indexed by "Filed info Addr" and "Method info Addr". Each entry in the Constant Pool TOC is the address (relative to the base address) to the TAGs in the constant pool extracted directly from the class file [8]. Some indirect references will be resolved by the class loader in advance so that dynamic resolution during runtime will be faster and simpler. An example is shown in Fig. 3. A byte code instruction, invokestatic 1D, refers to the constant pool entry 1D and "Methodref_info" represents a symbolic reference to the method declared in a class. A typical JavaVM resolves this symbolic reference at runtime. Our class loader will resolve some references if possible during class loading to speed up runtime operations.

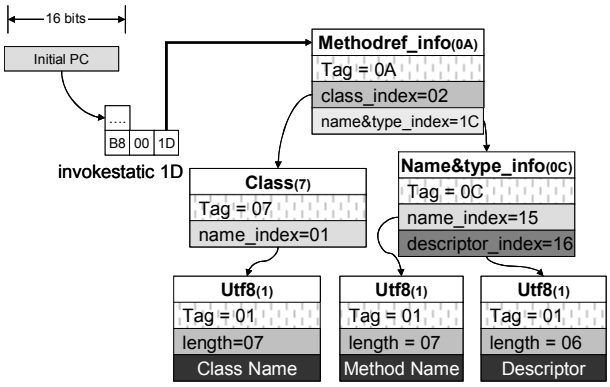


Fig. 3 Dynamic method resolution of Java.

Fig. 4 shows our mechanism for fast resolution. The class loader uses the memory space following a "Methodref_info" entry to store the target address of the method reference. During runtime, the instruction, invokestatic 1D, refers to the constant pool entry 1D of the constant pool TOC, and read the data of that entry. Then, the java BEE core will retrieve the target address points to the method entry directly. With this mechanism, dynamic resolution will be faster at runtime. Symbolic references to other information, such as interfaces and fields, are implemented in the same way.

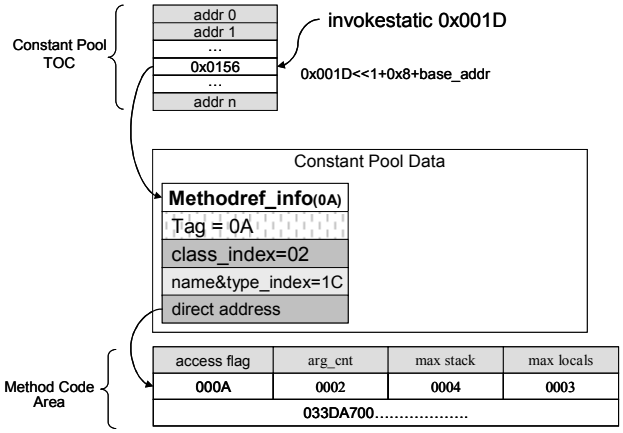


Fig. 4 Fast dynamic method resolution.

B. Java Stack Operation for Method Invocation

According to the Java VM specification [8], when an instance method is invoked, a reference to its instance is passed in through local variable 0 in the stack frame. In the Java programming language the instance is accessible via the keyword, this. Class (static) methods do not have an instance, so a class method starts using local variables at index zero. Therefore our method invocation is set up by pushing arguments onto top of stack. When the frame for the new method is created, the arguments passed to the method become the initial values of the new frame's local variables.

Note that only a pair of VP (variable pointer) and SP (stack pointer) exists in stack. VP stands for the first local variable of the current method, while SP means a next top of entry in stack. In each frame, some information besides local variable are stored against the program execution. Pervious JPC (java program counter) record the return point and Pervious VP points to the original VP at the previous frame.

After the information of the invoker is recorded, the new method is invocated. When it returns, its return value is pushed onto the operand stack of the frame of the invoker. The VP and SP are then reset. Fig. 5 and Fig. 6 show the transformation of stack for method invocation and return. They key idea for the proposed Fast Host Service Interface is to let the java BEE core identify and intercept all method invocations to a special Java class (*mmes.HostService*) and then signal an interrupt to the host service handler routine running on the host processor. The caller stack will also be exposed to the host processor service routine.

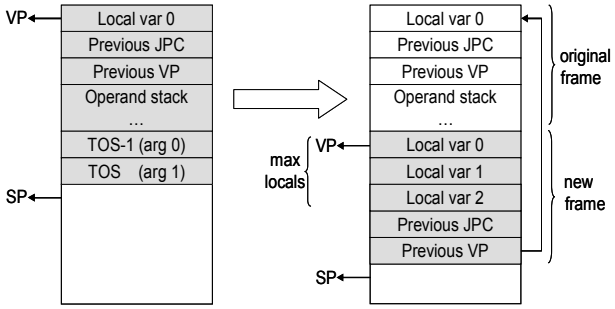


Fig. 5 Java stack operation for method invocations.

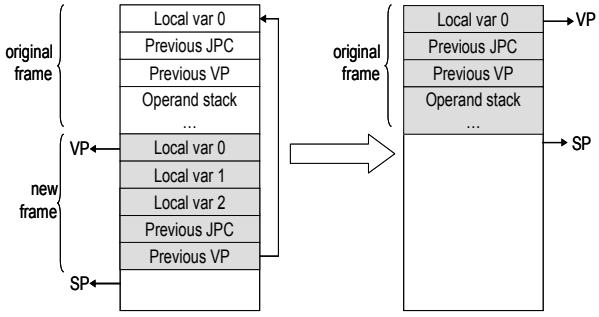


Fig. 6 Stack transformation for method return.

C. Execution Flow of Fast Host Service Interface

Fig. 7 shows a detail Fast Host Service Interface example when a Java application calls an I/O service, *mmes.HostService.print()*, on the host side.

1. The Java code that calls *mmes.HostService.print()* is compiled by Java compiler into an indirect reference which is composed of instruction and a unique ID referred to the constant pool.
2. The proposed fast method resolution mechanism resolves this indirect reference and start executing the bytecode sequence of mmes.hostservice.print(). Each method in the class mmes.hostservice has some inline user-defined bytecode that assign the unique service ID to the interface register, ServiceID, and signals an interrupt to the host processor.
3. Upon reception of the interrupt, the host processor executes the host service routine that corresponds to the ServiceID register. Note that there are still some custom data registers are cached for parameter passing even though the proposed BEE core has three register for three top stack elements. These custom data registers are designed for host services only. Therefore, the service routine on the host processor can accesses parameters directly through the custom data registers which are consistent with operand stack.

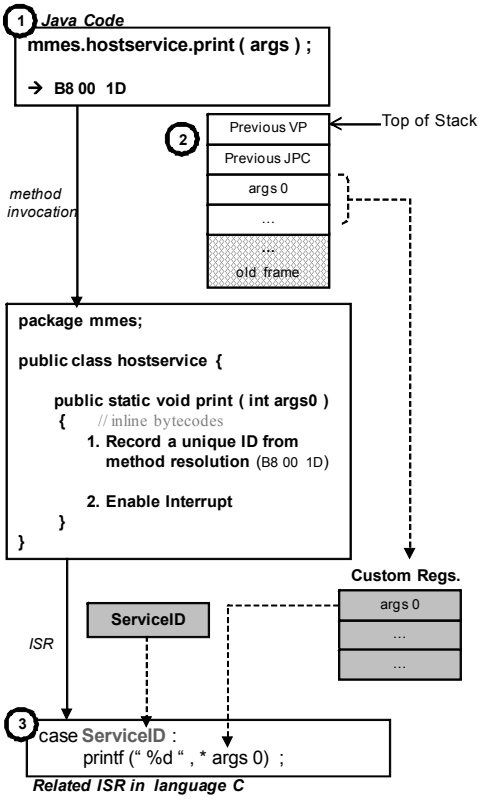


Fig. 7 Execution flow of FHSI.

III. EXPERIMENTAL RESULT

The proposed dual-core JRE is implementation on an SoC emulation platform, the Xilinx ML405. The platform is based on a Virtex IV FPGA with a PowerPC hard IP core. Both the processor frequency and the bus frequency are 100 MHz. On the RISC side, a thin OS kernel is used for the experiments. Some host services (to support the behavior including new object, new array, and print out) are encapsulated in an ISR. The RTL model of the java BEE core is written in VHDL and the synthesis report using SynplifyPro for the Virtex IV device is shown in TABLE II.

TABLE I. SYNTHESIS REPORT OF THE BEE CORE

Device : vertex-4 xc4vfx20 ff672-10				
Number of Slices	3390	out of	8544	39%
Number of Slice Flip Flops	3394	out of	17088	19%
Number of 4 input LUTs	4405	out of	17088	25%
Number of IOs	524			
Number of bonded IOBs	0	out of	320	0%
Number of FIFO16/RAM16s	12	out of	68	17%
Number of as RAM16s	12			
Number of DSP48s	3	out of	32	9%
Minimum period	9.600 ns			
Maximum Frequency	104.170 MHz			

A. *Interrupt Overhead of the Target Platform*

The communication efficiency between the java BEE core and the host processor core is crucial for such heterogeneous dual-core model. In general, interrupt-driven and polling are two common ways of the inter-processor communication. In TABLE II. , the communication latency of each method is shown. The unit of cycle means that the clock cycles are required, and milliseconds is the multiplication of cycles and period (10 nanoseconds). Although polling has smaller latency, we choose to use interrupt mechanism for its flexibility.

TABLE II. THE LATENCY OF INTER-PROCESSOR COMMUNICATION

	#Cycle	Milliseconds
Interrupt-driven	474	0.05
Polling	135	0.01

B. *Efficiency of the Proposed Host Service Invocation*

TABLE III. is the experimental result comparing the proposed dual-core JRE to the standard CVM running on the same emulation platform. The value in TABLE III. stands for execution time in milliseconds for some Java operations (method invocation, native method invocation, and the new object) executed for 10,000 iterations. The less value means the higher performance.

Method invocation is the common function call and it represents a symbolic reference to the method declared in a class. This experiment presents the capability of dynamic resolution. We get about 10 times improvement of this operation due to the design of fast dynamic method resolution.

Native method invocation means a Java application invokes the native C dummy function for 10,000 times. Although proposed JRE has extra interrupt overhead per native call, the performance is still slightly better than that of CVM. According to the experiment, the overhead of interrupt is 98.2% (15,979,980 of 16,269,995 cycles). Only 1.8% of the execution time is for the Java bytecode. Note that IPC overhead is unavoidable for a dual-core processor, but the advantage is that the overall system performance is higher. For example, we have use two benchmarks, PI and SIEVE, to show the full system performance. For PI, calculation of π to 500 decimal digits, the execution time of the proposed dual-core JRE is 176 milliseconds while the execution of Sun's CVM is 1086 milliseconds. For SIEVE benchmark, the execution times of the proposed dual-core JRE and Sun's CVM are 2,887 milliseconds and 26,199 milliseconds.

The last experiment tested the overhead of "new object," which instantiates an object through the "new" bytecode instruction. It is the worst case of the proposed system due to memory management overhead under the host processor side. This overhead can be reduced if the memory management functions are optimized for the proposed system.

TABLE III. EXPERIMENTAL RESULT

	Dual-core JRE	Sun's CVM
method invocation (java-call-java)	3 ms	42 ms
native invocation (java-call-c)	163 ms	171 ms
new object	375 ms	42 ms

IV. CONCLUSIONS

A dual-core Java application processor and a fast IPC for the Java core to invoke system service routines running on the host core is presented in this paper. The proposed IPC mechanism is very extensible and the experimental results show that it is also very efficient.

V. ACKNOWLEDGEMENT

This research is partly funded by National Science Council, Taiwan, R.O.C., under grant number NSC 97-2220-E-009-024.

REFERENCES

[1] H. J. Ko and C. J. Tsai, "A Double-issue Java Processor Design For Embedded Application," *Proc. of IEEE Int. Symp. on Circuits and Systems*, May. 2007.

[2] S. Liang. *The Java Native Interface: Programmer's Guide and Specification*. Addison-Wesley 1999.

[3] M. Schoebel, "Evalution of a Java Processor," *Tagungsband Austrochip* 2005, pp. 127-134, Oct. 2005.

[4] S. Nino, T. Mori, Y. Ko, Y. Shibata, and K. Oguri, "FPGA Implementation of a Statically Reconfigurable Java Environment for Embedded Systems," *IEEE Int. Symp. on Field-Programmable Technology*, 2007.

[5] D. Kurzyniec and V. Sunderam, "Efficient Cooperation between Java and Native Codes – JNI Performance Benchmark," http://janet-project.sourceforge.net/papers/jnibench.pdf

[6] Sun Microsystems, *J2ME Building Blocks for Mobile Devices*, Sun Microsystems White Paper, May. 2000.

[7] Sun Microsystems, *Connected, Limited Device Configuration Specification Version 1.0a*, Sun Microsystems White Paper, May. 2000.

[8] T. Lindholm and F. Yelling, *The Java Virtual Machine Specification*, Addison-Wesley, 1996.