USER DIRECTED VIEW SYNTHESIS ON OMAP PROCESSORS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MÜRSEL YILDIZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

JUNE 2009

Approval of the thesis:

USER DIRECTED VIEW SYNTHESIS ON OMAP PROCESSORS

submitted by **MÜRSEL YILDIZ** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İsmet Erkmen
Head of Department, **Electrical and Electronics Engineering** _____

Prof. Dr. Gözde Bozdağı Akar
Supervisor, **Electrical and Electronics Engineering Dept., METU** ____

Examining Committee Members

Prof. Dr. Hasan Cengiz Güran
Electrical and Electronics Engineering Dept., METU _____

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering Dept., METU _____

Assist. Prof. İlkay Ulusoy
Electrical and Electronics Engineering Dept., METU _____

Assist. Prof. Çağatay Candan
Electrical and Electronics Engineering Dept., METU _____

Dr.-Ing. Klaus Schmidt
Universitaet Erlangen-Nürnberg _____

Date: 29.06.2009

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Mürsel YILDIZ
Signature :

ABSTRACT

USER DIRECTED VIEW SYNTHESIS
ON OMAP PROCESSORS

Yıldız, Mürsel
M.S., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Gözde Bozdağı Akar

June 2009, 117 pages

In this thesis, real time image rendering for handheld devices is studied according to user's view point choice and using image frames with corresponding depth maps obtained from 2 different cameras, of which positions on coordinate system is known. User's view point choice is restricted to the area between right, and left cameras. Occlusion handling methods for image rendering systems is explored and discussed together with frame enhancement techniques. Median filtering is studied for multicolor image frames and post processing methods are discussed for image enhancement at the end of rendering algorithm. In this thesis, OMAP3530 microprocessor is used as the main processor which processes suggested rendering algorithm with occlusion handling and frame enhancement. Proposed algorithms are implemented on DSP core and ARM cores of OMAP3530 separately and their performances are evaluated through experiments. Embedded Linux (Kernel-2.6.22) is run as the operating system for applications. Driver usage together with devices for Linux embedded operating system is explored and studied. 3 boards are used for the realization of proposed system. OMAP35x EVM board from Mistral

Solutions Company is used for processor utilization, high resolution LCD utilization, system monitoring, user interface and communication purposes. Two daughter cards are designed for user view point determination. First daughter card handles communication process with EVM board and calculates view point according to input from second daughter card with single axis response GYRO sensor (ADIS16060). Spartan®-3A DSP FPGA family is utilized in this system for view point determination. DSP slices that are hardly present inside gate arrays of this FPGA family are utilized and their performance is studied. Asynchronous memory interface, i2c bus interface, SPI interface are studied and implemented on FPGA.

Keywords: Real time view synthesis, intermediate view reconstruction

ÖZ

OMAP İŞLEMCİLERİ ÜZERİNDE KULLANICI KONTROLLÜ GÖRÜNTÜ SENTEZLEME

Yıldız, Mürsel
Yüksek Lisans, Elektrik-Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Gözde Bozdağı Akar

Haziran 2009, 117 sayfa

Bu tezde, koordinat düzleminde yerleri belirlenmiş 2 ayrı kameradan alınan görüntü dizisi ve derinlik haritaları kullanılarak, kullanıcının seçtiği izleme noktasına göre, taşınır cihazlar için görüntü üretimi incelenmiştir. Kullanıcının izleme noktası seçimi sağ ve sol kameraların aralarında kalan bölgeyle sınırlıdır. Görüntü üretim sistemleri için oklüzyon bölgeleriyle başa çıkma metotları, görüntü karelerinin iyileştirilmesi için önerilen tekniklerle birlikte araştırılmıştır ve tartışılmıştır. Çok renkli görüntüler için Median filtreleme araştırılmıştır ve görüntü üretim algoritmasından sonra görüntünün iyileştirilmesi için algoritma sonrası işleme yöntemleri tartışılmıştır. Bu tezde, OMAP3530 mikro işlemcisi önerilen görüntü üretim algoritmasıyla birlikte oklüzyon bölgeleriyle başa çıkma ve görüntü karelerinin iyileştirilmesi işlemlerini yapan ana işlemci olarak kullanılmıştır. Önerilen algoritmalar OMAP3530 işlemcisinin DSP koru ve ARM korları üzerinde ayrı ayrı uygulanmıştır ve bu korların performansları deneyler sonucunda değerlendirilmiştir. Uygulama için gömülü Linux (Kernel-2.6.22) işletim sistemi olarak koşturulmuştur. Linux gömülü işletim sistemi için aygıt ve sürücü kullanımı araştırılmıştır ve çalışılmıştır. Önerilen sistemin gerçeklenmesi adına 3 kart kullanılmıştır. İşlemci kullanımı, yüksek

çözünürlüklü LCD kullanımı, sistem gözlenmesi, kullanıcı ara yüzü ve iletişim amaçlarına yönelik Mistral Solutions şirketinden OMAP35x EVM devre kartı kullanılmıştır. Kullanıcının izleme noktası belirlemesi adına iki kız kart tasarımı yapılmıştır. Birinci kız kart EVM devre kartıyla iletişim işlemlerini gerçekleştirir ve tek eksene duyarlı GYRO (ADIS16060) sensorlu ikinci kız kartından gelen girdiye göre izleme noktası hesabını yapar. Sistemde Spartan®-3A DSP FPGA ailesinden görüntü izleme noktası belirlenmesi için yararlanılmıştır. Bu FPGA ailesinin kapı dizileri içinde gömülü bulunan DSP blokları kullanılmıştır ve performansları üzerine çalışılmıştır. Asenkron bellek ara yüzü, i2c veri hattı, SPI ara yüzü üzerine çalışılmıştır ve FPGA üzerine uygulanmıştır.

Anahtar kelimeler: Gerçek zamanlı görüntü sentezleme, ara görüntü oluşturulması

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor Prof. Dr. Gözde Bozdağı Akar, for her understanding, guidance and sincerity. I regard that it would be a great chance and opportunity for a student to work with her. I am very grateful for her motivation and broad vision, which leaded progress of my research immensely.

I would like to thank my family, especially to my father Mustafa Yıldız and my mother Vahide Yıldız for their love and support. I believe that hard research progress goes on fifty percent morale strength and fifty percent technical skills. I got exactly fifty percent morale support from my parents. Without their support, it would never be possible for me to write an acknowledgement page for a thesis.

Special thanks to Mehmet Uçak, Volkan Serter, Salih Alper Engin and Recep Ali Yıldırım for their helpful tips on engineering issues, friendship and assistance. It is unforgettable for anyone who needs a helpful hand when he is stuck on a problem.

TABLE OF CONTENTS

ABSTRACT iv

ÖZ vi

ACKNOWLEDGEMENTS viii

TABLE OF CONTENTS ix

TABLE OF FIGURES xi

CHAPTERS

1. INTRODUCTION 1

1.1 Literature Review 2
1.2 Scope of the Thesis 4
1.3 Outline of the Thesis 5

2. OMAP3530 PROCESSOR 7

OMAP3530 Processor General Overview 7
2.1 ARM Core 9
2.2 IVA2.2 Subsystem 12
2.2.1 IVA2.2 Subsystem Integration 14
2.3 Inter-processors (DSP – ARM Cores) Communication 17

3. INTERMEDIATE VIEW RECONSTRUCTION 19

3.1 Intermediate View Reconstruction 19
3.2 Previous Work 20
3.3 Pinhole Camera Model 20
3.3.1 Pinhole Camera Geometry 22
3.4 Intermediate View Reconstruction Algorithm 25
3.4.1 Algorithm Outline 28

4. OCCLUSION HANDLING 35

4.1 Literature Review 35
4.2 Fast Occlusion Handling Algorithm 39
4.2.1 Algorithm Outline 45

5. FRAME ENHANCEMENT 55

5.1 Median Filtering 55
5.1.1 Literature Review 57
5.1.2 Hilbert's Space-Filling Curves 63

5.1.3 Proposed Median Filter 65
5.2 Post Processing 71
5.2.1 Proposed Algorithm Outline 75

6. SYSTEM DESCRIPTION 81

6.1 System Architecture 81
6.1 Hardware Architecture 83
6.1.1 EVM Board 83
6.1.2 Daughter Cards (EVM Expansion Board) 85
6.2 SOFTWARE ARCHITECTURE 90
6.2.1 Image Formation Algorithm 90
6.2.2 Intermediate View Reconstruction 92
6.2.3 Occlusion Handling 93
6.2.4 Frame Enhancement 94
6.2.5 View Point Determination 95
6.3 PERFORMANCE of ALGORITHMS on OMAP Cores 101

7. CONCLUSION 103

7.1 Discussion and Future Work 104

REFERENCES 107

APPENDIX.A 111

TABLE OF FIGURES

FIGURES

Figure 2.1: Generalized block scheme of OMAP3530 9
Figure 2.2: 13 Stage pipeline of ARM architecture 10
Figure 2.3: Co-processor pipeline structure 11
Figure 2.4: Pipeline based algorithm instruction representation 12
Figure 2.5: IVA2.2 Subsystem block diagram 13
Figure 2.6: Pipeline based DSP core algorithm instruction structure 14
Figure 2.7: ARM controlled IVA subsystem boot 16
Figure 2.8: Mailbox – interrupt mechanism block diagram 17
Figure 2.9: General architecture of codec engine operation 18
Figure 3.1: Image formation through pinhole camera 21
Figure 3.2: Pinhole camera geometry 22
Figure 3.3: 3D point projection on image plane 23
Figure 3.4: Camera positions and virtual camera orbit 26
Figure 3.5: Two frames with corresponding depth maps from camera 6 27
Figure 3.6: Two frames with corresponding depth maps from camera 4 28
Figure 3.7: 3D point determination using camera coordinate system 30
Figure 3.8: Mapping 3D scene point on second image plane 32
Figure 3.9: Original input frame from camera 5 33
Figure 3.10: 5^{th} camera frame reconstruction output from camera 4 33
Figure 3.11: 5^{th} camera frame reconstruction output from camera 6 34
Figure 4.1: Occlusion region corresponding to different camera position 36
Figure 4.2: Efficient view-dependent image-based rendering with projective texture-mapping example 38
Figure 4.3: Occlusion problem after 3D image warping [21] 40
Figure 4.4: Occlusion region filling process [21] 40
Figure 4.5: Virtual camera position between real cameras 41

Figure 4.6: Basic flowchart for two processor system occlusion filling algorithm 42
Figure 4.7: Basic flowchart for multi-processor system, efficient occlusion filling algorithm 44
Figure 4.8: Occlusion map corresponding to camera 6 frame with respect to camera 5 virtual position 46
Figure 4.9: Back projection of occlusion region pixels 47
Figure 4.10: Reconstructed camera 5 from camera 6 and 4 49
Figure 4.11: Arbitrarily drawn real scene top view 51
Figure 4.12: Edge image of camera 4 depth map in vertical direction 52
Figure 4.13: Vertical difference of edge images for camera 6 depth map and camera 4 depth map in +x dirrection 53
Figure 5.1: Median filter process 56
Figure 5.2: Representation of two vectors in 2D 60
Figure 5.3: Parallel processing Median filter structure in FPGA 61
Figure 5.4: Transformation of RGB into space filling curves 62
Figure 5.5: Filtering based on transformation from Z^N to Z 63
Figure 5.6: First, second and third order 2D SFC 64
Figure 5.7: Impulse noise removal from input sequence 67
Figure 5.8: Modified window shape examples 68
Figure 5.9: 3x3 window shape median filter output 69
Figure 5.10: Modified window shape median filter output 69
Figure 5.11: Modified median filter output 70
Figure 5.12: Pixel count in original frame from camera 4 72
Figure 5.13: Pixel count in original frame from camera 6 73
Figure 5.14: Background pixel observation through dehiscence region 74
Figure 5.15: Edge image in +x direction of camera 6 depth map 76
Figure 5.16: Abruptly drawn top view of ballet scene 77
Figure 5.17: Background purification with the suggested algorithm 79
Figure 5.18: Closer look to purified background pixels on object 80

Figure 6.1: System hardware setup 82
Figure 6.2: EVM functional block diagram 84
Figure 6.3: EVM board top view 85
Figure 6.4: Functional block diagram of daughter cards 86
Figure 6.5: Front view of main daughter card 86
Figure 6.6: Back view of main daughter card 87
Figure 6.7: Front view of gyro card 87
Figure 6.8: Rate sensitive axis of gyro sensor 89
Figure 6.9: Mapping true color to RGB565 from RGB888 92
Figure 6.10: Object dehiscence representation 94
Figure 6.11: Timing diagram for SPI data read 96
Figure 6.12: Timing diagram for SPI device control 96
Figure 6.13: Gyro controlled virtual camera orbit 98
Figure 6.14: I2C port communication timing diagram 99
Figure 6.15: Touch-screen controlled virtual camera orbit 101
Figure A.1: Gyro-read process flow chart 112
Figure A.2: Virtual point detection flow chart based on look-up table 113
Figure A.3: Virtual point detection flow chart based on LED positions 114
Figure A.4: I2C communication protocol part 1 115
Figure A.5: I2C communication protocol part 2 116
Figure A.6: I2C communication protocol part 3 117

CHAPTER 1

INTRODUCTION

Utilization of 3D display systems with extraction, transmission or display techniques have been a popular research area for recent years. Due to wide variety of 3D system applications ranging from entertainment to cartography, requirement of robust solutions for such systems is becoming more important.

In recent years, there has been growing interest in applications for hand-held devices, especially for the ones with graphical accelerators and video display capabilities. Among these entertainment devices, 3D display system inclusion and free-view point video application capability are very new to the industry. It is foreseen that such entertainment devices will be gaining more interest day by day. This fact stems from two reasons. Firstly, there is a magnificent opportunity for algorithm developers in the market, because general processor units of hand-held devices are developed considerably, which in turn gives rise to development of applications. Secondly, researches on image processing techniques on view synthesis and rendering are extended.

Intermediate view reconstruction depends on 3D geometry of filmed scenes and is required for 3D video applications as a basic step. Second step of 3D display systems is directly related with screen characteristic or necessity of

stereoscopic eyeglasses. For example, auto-stereoscopic displays are categorized into two main groups, namely two-view and multi-view auto-stereoscopic displays, which presents more than one image from different angle at a time for both eyes [1]. Intermediate view reconstruction algorithms provide these images for display system so that users can perceive 3D sense, which is the reason why these algorithms are important for 3D display systems.

Although processors for hand-held devices are very powerful for many operations on signaling techniques, it is inevitable for algorithm builder to research and test many suggested algorithms in intermediate image reconstruction, in order to build a real time operating video application. Parallel processing units are very powerful and preferable for image reconstruction. Nowadays, many leading companies in FPGA technology, provide parallel processing units with additional DSP slices for signal processing purposes, however, due to high cost of these units; it is not always possible to design hardware with an FPGA peripheral for hand-held devices.

In this thesis, intermediate view reconstruction algorithm for hand-held devices with gyroscope sensor and touch-screen capabilities is studied. Our aim is to provide a free view point video application for which view point changes with the rotation of the device from left to right or vice versa, by user. Furthermore, frame enhancement techniques against deformation due to rendering algorithms are discussed in details.

1.1 Literature Review

Nowadays, free-view point video rendering systems are becoming very popular to the industry. Many researchers have been developing 3D based

intermediate view reconstruction or image-based rendering techniques for changing view points.

Ince et al. [2] proposed an image-based view interpolation algorithm based on usage of 4 input images taken from different view points. Proposed algorithm consists of three steps. Firstly, all pixels in the input image are classified in terms of their visibility for the new view point. Using different image inputs, disparity for each pixel is calculated in the second phase, depending on the first phase output. Finally, each pixel color is calculated adaptively at the final step from image pairs that are selected by their visibility labels. Experimental results for this algorithm shows improvement on occlusion region handling. However, this algorithm should be implemented on systems with parallel processing capabilities because many middle step images are constructed for algorithm not only from a single image but from 4 images. Moreover, high speed memory usage is required for this system.

Debevec et al. [3] suggested an image-based intermediate view reconstruction technique with occlusion handling. They basically filled occlusion hole-filling problem using polygon view maps, which are basically depending on connectivity between each pixels at vertex positions. Firstly, they set up a linked list showing connectivity between vertexes, secondly they computed average color according to linked list and finally they assigned remaining invisible pixels with the closest visible pixel. Their method gives successful results for images having almost smooth depth maps, i.e. images in which objects are far away from camera. However, occlusion regions reveal themselves clearly for scenes in which objects are closer to cameras.

In [27], a fast view synthesis method that generates multiple intermediate views in real time for a 3D display system, using predetermined camera geometry and depth map of each image frame which is very similar to theoretical approach for intermediate view reconstruction method discussed in this thesis. Very fast view synthesis method is suggested by processing each frame in blocks entirely in parallel by many multiprocessors simultaneously, under the control of a CPU. They suggested specialized processor architecture of which experimental environment consists of a dual core CPU with 1.86 GHz speed, 2048 MB RAM, 1.62 GHz GPU of 16 multiprocessors having 128 cores in total. All of these processor units are devoted to realization of the algorithm, which in fact gives rise to the need of another processor unit having an operating system for the realization of any other task for the device. Moreover, the system consists of many high speed processors, which in turn brings about high costs for the device, too much PCB space for production and very high power requirement not only for handheld devices but also for any other display system.

In [1], view point is detected by tracking the position of single observer and virtual camera view is rendered according to eye position of observer. The system is implemented on a home PC and tracking operation is done by a webcam connected to the PC. However, this system is proposed for a single user, and system is suggested with an additional webcam for eye tracking. This system would not show high performance for handheld devices because user's head and eyes are almost stable while watching video from handheld devices.

1.2 Scope of the Thesis

This thesis deals with the application of real time view synthesis system for hand-held devices. This system aims to provide free-view point video for

user, who determines view point by gyro sensor, touch-screen or keypad. Application is examined on OMAP3530 microprocessor, both using ARM processor and DSP processor separately. Linux (Kernel-2.6.22) is used as the operating system for application.

OMAP35x EVM (evaluation module) is used with designed two daughter cards. First daughter card consists of FPGA, DSP and expansion connectors for future work. Second daughter card is devoted for gyroscope and directly connected to first daughter card. Gyro sensor applications are researched and implemented on FPGA.

Main blocks of system are defined as intermediate view reconstruction block, frame enhancement block and view point detection block by 3 methods namely, touch-screen, keypad and gyro sensor. Each of these main blocks was examined by different approaches for real time operation.

1.3 Outline of the Thesis

This thesis is composed of 7 main chapters. In chapter 2, main processor of system, i.e., double core processor OMAP3530 is introduced briefly. Each core of processor is described with general aspects, and interconnection of processors and boot process of DSP core are summarized.

Chapter 3 of thesis is devoted to intermediate view reconstruction algorithm and optimization tips for algorithm in order to achieve a real time operating system. Background information about camera model used is described in details.

Detailed information on occlusion handling problem is given in chapter 4. Previous works on occlusion handling is discussed, compared and

contrasted. A new fast algorithm for this problem is suggested and explained in details.

In chapter 5, frame enhancement techniques for image deformation after occlusion handling and intermediate view reconstruction algorithm is described in details. Background information on Median filters is given. Previous works on median filters are compared and contrasted. Furthermore, a new algorithm is suggested for dehiscence of objects after intermediate view reconstruction.

In chapter 6, proposed system is described in general aspects. Hardware design and main blocks of software are described. Two daughter cards designed for this thesis are introduced with main blocks.

Finally, chapter 7 summarizes the thesis and gives concluding remarks. Moreover, future work for this thesis is described in detail.

CHAPTER 2

OMAP3530 PROCESSOR

In this chapter, very brief information on the OMAP3530 processor is given. This chapter is devoted to the main core and co-processors included in this microprocessor. The peripherals and processor architectures are related with the algorithms used in this thesis. In order for the reader to understand only the general aspects of the architecture, superficial information is provided. This chapter consists of two main sections of which the first section describes the ARM core while the second part describes the DSP core.

OMAP3530 Processor General Overview

OMAP3530 is a high-performance, applications processor based on OMAP™ 3 architecture provided by Texas Instruments for 2008, which is designed to provide video, image and graphics processing [11]. This processor is offered for streaming video, 2D/3D mobile gaming, video conferencing, high-resolution still image, Video capture in 2.5G wireless terminals, 3G wireless terminals, and rich multimedia-featured handsets, and high-performance personal digital assistants (PDAs) [11].

In [11], general subsystems of the device are summarized as:

- Microprocessor unit (MPU) subsystem based on the ARM Cortex™-A8 microprocessor.

- IVA2.2 subsystem with a C64x+ digital signal processor (DSP) core
- SGX subsystem for 2D and 3D graphics acceleration to support display and gaming effects (3530only)
- Camera image signal processor (ISP) that supports multiple formats and interfacing options connected to a wide variety of image sensors
- Display subsystem with a wide variety of features for multiple concurrent image manipulation, and a programmable interface supporting a wide variety of displays. The display subsystem also supports NTSC/PAL video out.
- Level 3 (L3) and level 4 (L4) interconnects that provide high-bandwidth data transfers for multiple initiators to the internal and external memory controllers and to on-chip peripherals

In this thesis, OMAP3530 processor is explored not only by the application of image warping algorithm but also by using its peripherals widely. This processor is a double core processor, namely 600-MHz ARM Cortex™-A8 core with NEON™ SIMD coprocessor as the general purpose main processor unit of the system and advanced very-long-instruction-word (VLIW) – TMS320C64x+™ DSP core [11]. This processor is low power IC that is suggested by TI (Texas Instruments) especially for hand-held devices, gaming consoles, video and image processing, and communication devices. In figure – 2.1, generalized block scheme of processor is given [11].

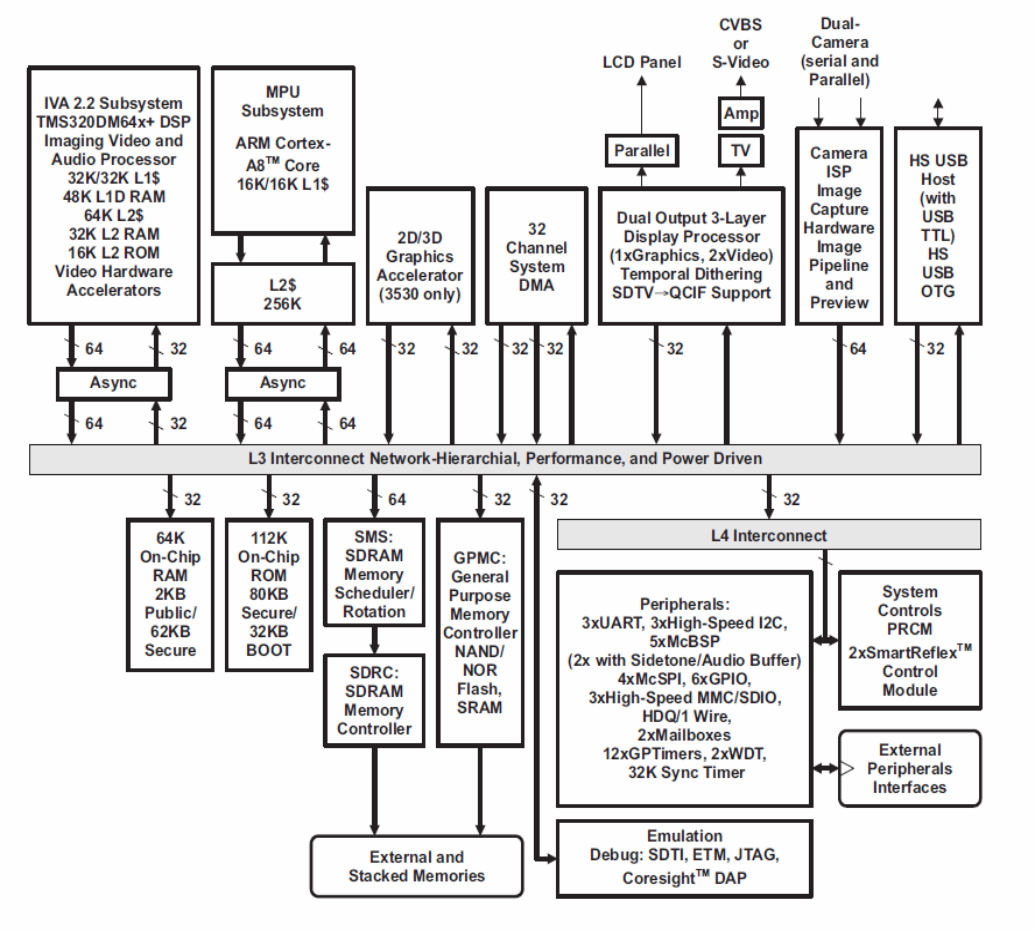

Figure 2.1: Generalized block scheme of OMAP3530

This processor supports high level operating systems such as: WINDOWS CE, SYMBIAN OS, LINUX and PALM OS [11]. In this thesis, Linux (Kernel-2.6.22) is used as the embedded operating system.

2.1 ARM Core

ARM Cortex8 core is claimed to be the pioneer core in including new technologies that are available in ARMv7 architecture [12]. "New technologies seen for the first time include NEON™ for media and signal

processing and Jazelle® RCT for acceleration of runtime compilers, such as just-in-time, dynamic or ahead-of-time compilers" [12]. Moreover TrustZone® technology stands for security and Thumb®-2 technologies for code density while the VFPv3 floating point architecture [12].

NEON media and signal processing technology is targeted for audio, video processing and 3D graphics, which can handle both integer and single precision floating-point values, and includes support for unaligned data accesses and easy loading of interleaved data stored in structure form [12].

This core architecture includes 13-stage pipeline. In figure – 2.2, this pipeline stages are illustrated [12].

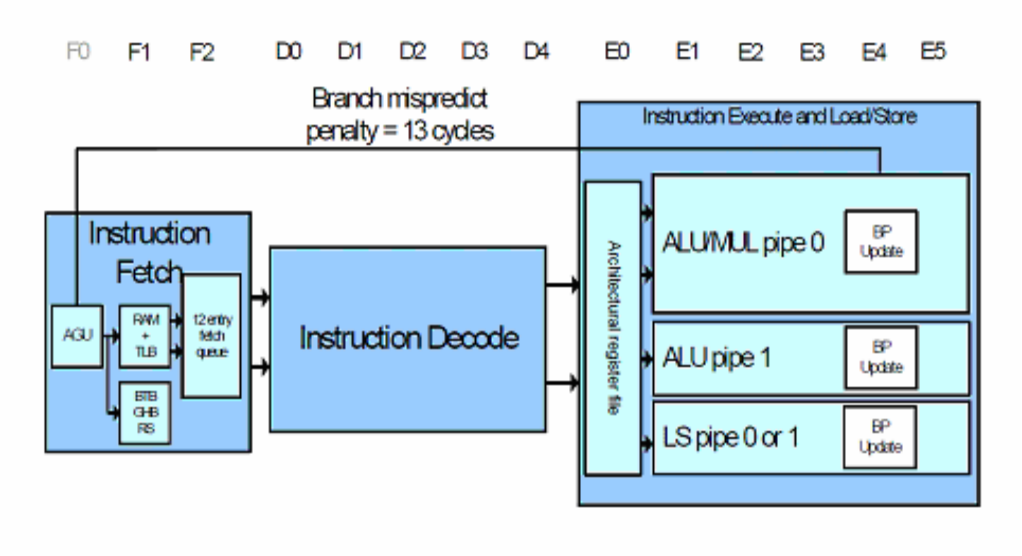

Figure 2.2: 13 Stage pipeline of ARM architecture

In this so called 13 stage pipeline, F0 is devoted for address prediction stage in order to minimize branch miss-prediction. At the end of this pipeline

order, Neon coprocessor pipeline starts. The NEON media engine has its own 10 stage pipeline. All miss-predicts and exceptions are resolved in the ARM integer unit, once an instruction has been issued to the NEON media engine it is completed as it cannot generate exceptions [12]. This type of architecture makes algorithms implemented to be much faster without software developer consideration. In figure – 2.3, coprocessor pipeline is illustrated [12].

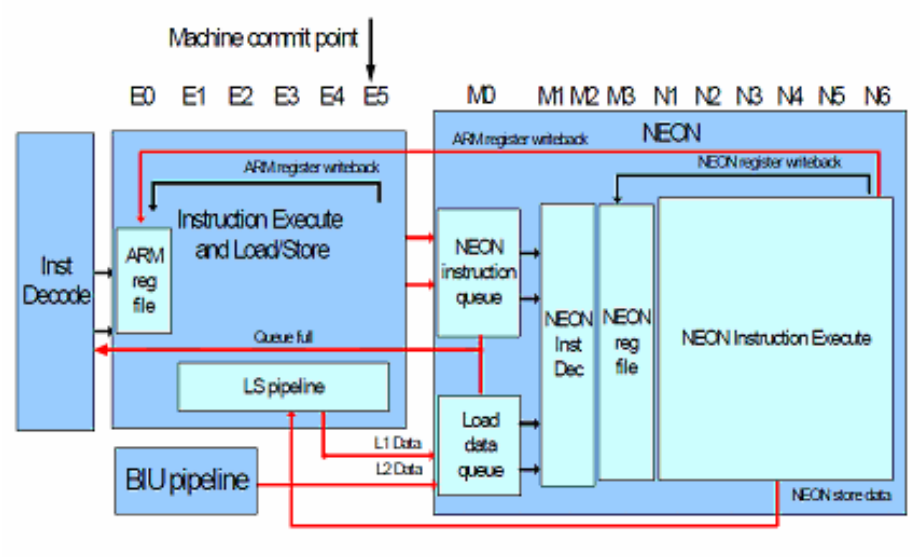

Figure 2.3: Co-processor pipeline structure

While considering algorithm software design, this pipeline architecture was taken into consideration in order not to make instructions wait for another. Warping algorithms were implemented in a for loop, which in turn bring about change of instructions possibility inside for loops without considerations. In figure – 2.4, this concept is summarized. In chapter 3, detailed information is given for algorithms that are implemented.

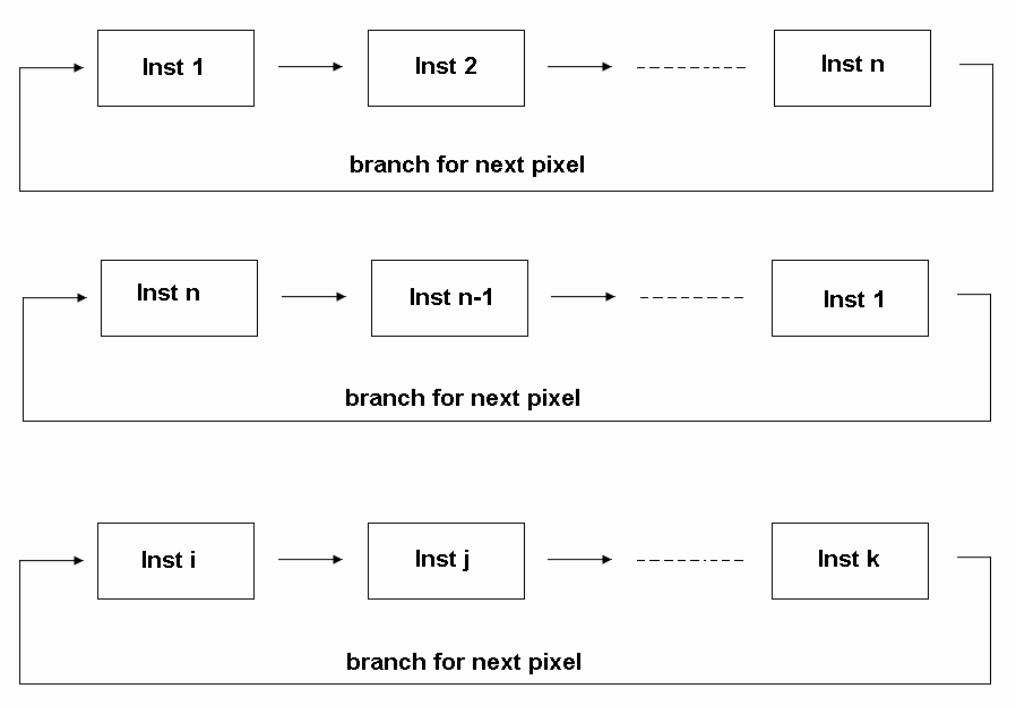

Figure 2.4: Pipeline based algorithm instruction representation

Considering proper initialization of critical variables at the start point of for loop, change of instruction queue does not change output. Therefore these instruction sets are aligned in such a way that all instructions are independent of each other in each 13 stage pipeline.

2.2 IVA2.2 Subsystem

This subsystem consists of TMS320C64x+™ DSP core, enhanced direct memory access controllers (EDMA) and video hardware accelerators [11]. In figure – 2.5, block diagram of this subsystem is given [13].

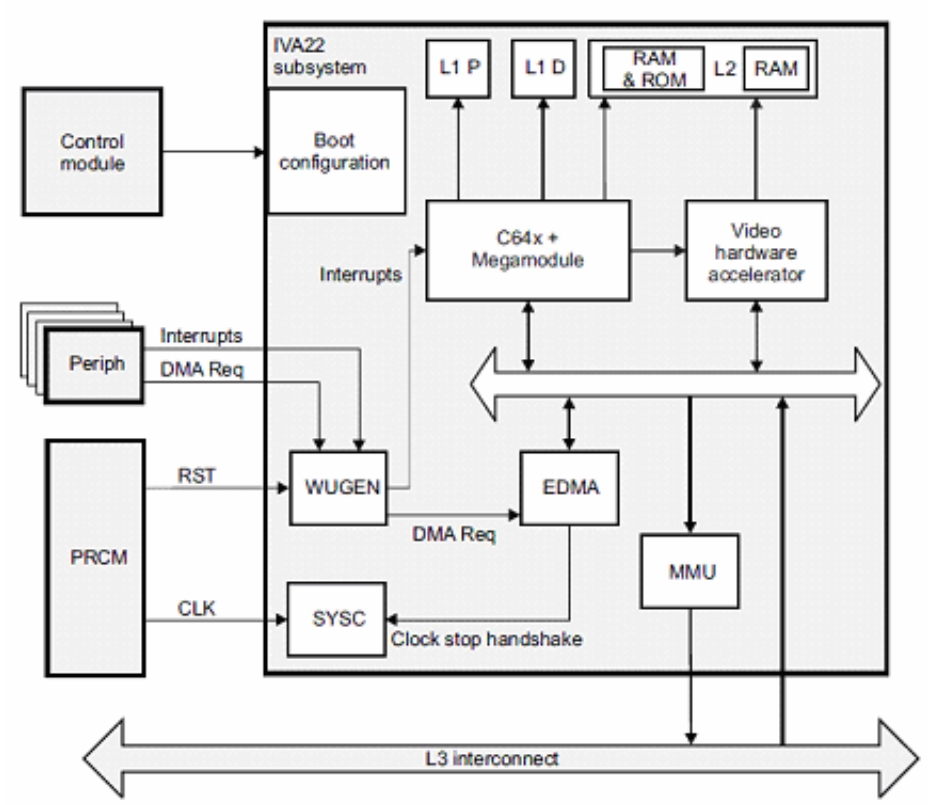

Figure 2.5: IVA2.2 Subsystem block diagram

Considering figure – 2.1, in which block diagram of OMAP3530 is shown, this subsystem can reach other subsystems through L3 interconnect.

This DSP core is based on 32 bit fixed point algorithms. This information is critical for image warping algorithm especially for occlusion handling problems in which not only fixed point but also floating point operations are implemented. Once fixed point signal processing algorithms are implemented, this subsystem is very powerful and operates faster than ARM core. This is based on capability of 8 instructions/cycle and 8 execution units of eight 8 x 8 or 16 x 16 multiply and accumulate (MAC) per cycle, eight slave asynchronous die (SAD) per cycle, eight interpolations (a + b + 1) >>1

per cycle and two (32-bit x 32-bit > 64-bit) multiply operations per cycle [13].

Similar to ARM core, DSP core architecture is based on pipeline operation. While exploring DSP performance for considered algorithms, because of capability of 8 instructions per cycle and pipeline architecture, DSP core algorithms were implemented in such a way that each instructions are independent from each other (each pipeline stage is independent) and variables are chosen for 8 successive pixels (8 instructions per cycle).

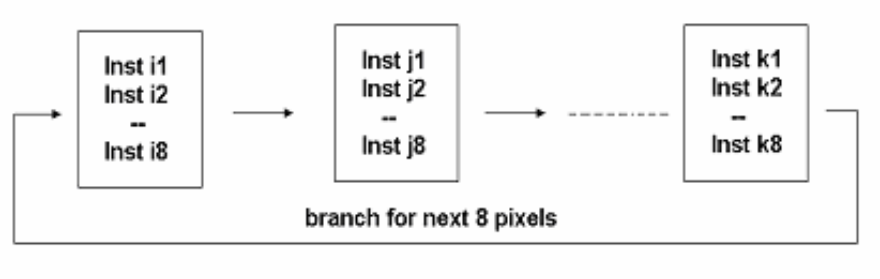

Figure 2.6: Pipeline based DSP core algorithm instruction structure

2.2.1 IVA2.2 Subsystem Integration

Once an application program is written, it is not possible to reach directly DSP core as if it is a driver for the ARM core, i.e, user can not reach DSP core anytime he desires before subsystem integration. IVA2.2 subsystem should be booted before the application program reaches to DSP core. Once DSP core boots its application code, it waits for a reset signal from ARM core in order to start execution.

In this thesis, two methods for booting and reaching DSP core were explored. The first one is the boot model under control of ARM core and second one is autonomous boot model [13].

2.2.2 IVA2.2 Boot

2.2.2.1 ARM Controlled Boot Operation

In this methodology, ARM core can initialize DSP at any instant. Once boot code of DSP is ready on any part of SDRAM memory, which may also be installed at anytime user desires by Ethernet connection, ARM core can reset DSP core and by controlling boot registers and their controller registers, DSP starts to boot its application code from the predefined and controlled memory space. In figure – 2.7, this process is depicted by algorithmic state machine graph [13].

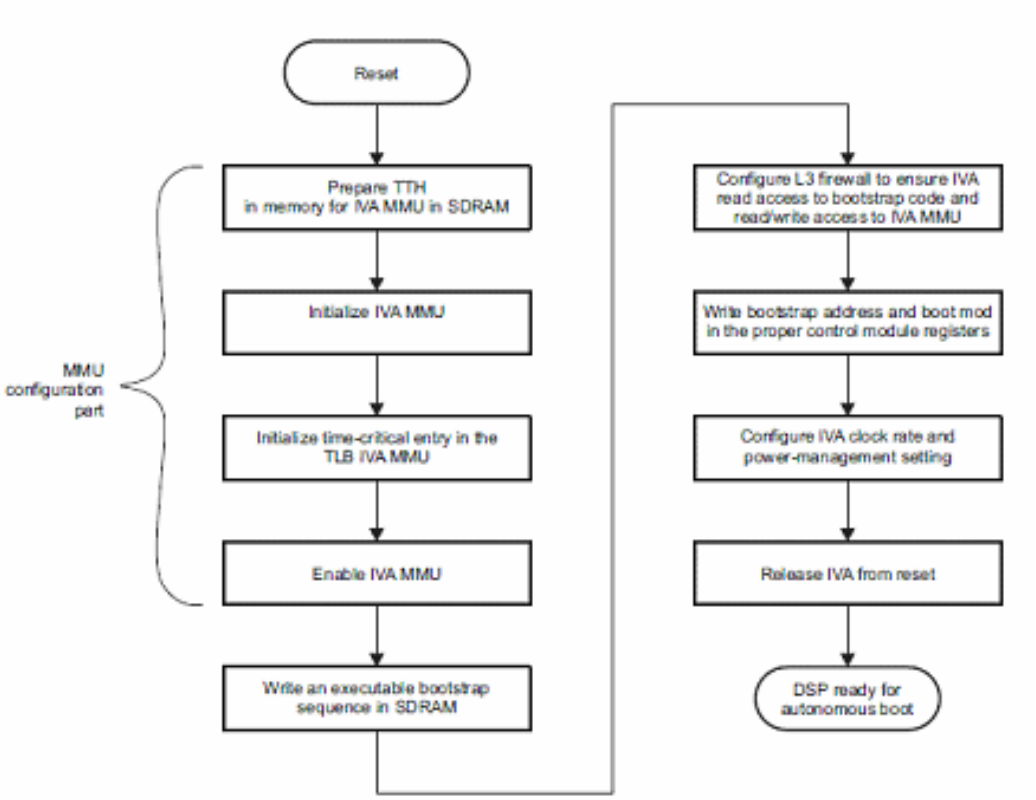

Figure 2.7: ARM controlled IVA subsystem boot

2.2.2.2 Autonomous Boot Model

At the instant of an OFF-to-ACTIVE transition (under MPU control or upon interrupt wakeup), the IVA2.2 subsystem is released from reset [13]. "After hardware configuration of values for C64x + Mega module generic parameters, the IVA2.2 starts fetching from the address 0x00000000 in ROM. The IVA2.2 must follow the (non-exhaustive) boot process"[13]. This method can be used for direct DSP core performance exploration.

2.3 Inter-processors (DSP – ARM Cores) Communication

On-chip cores of the processor communicate through a queued mailbox – interrupt mechanism [14]. In figure – 2.8, block diagram of this mechanism is given [14].

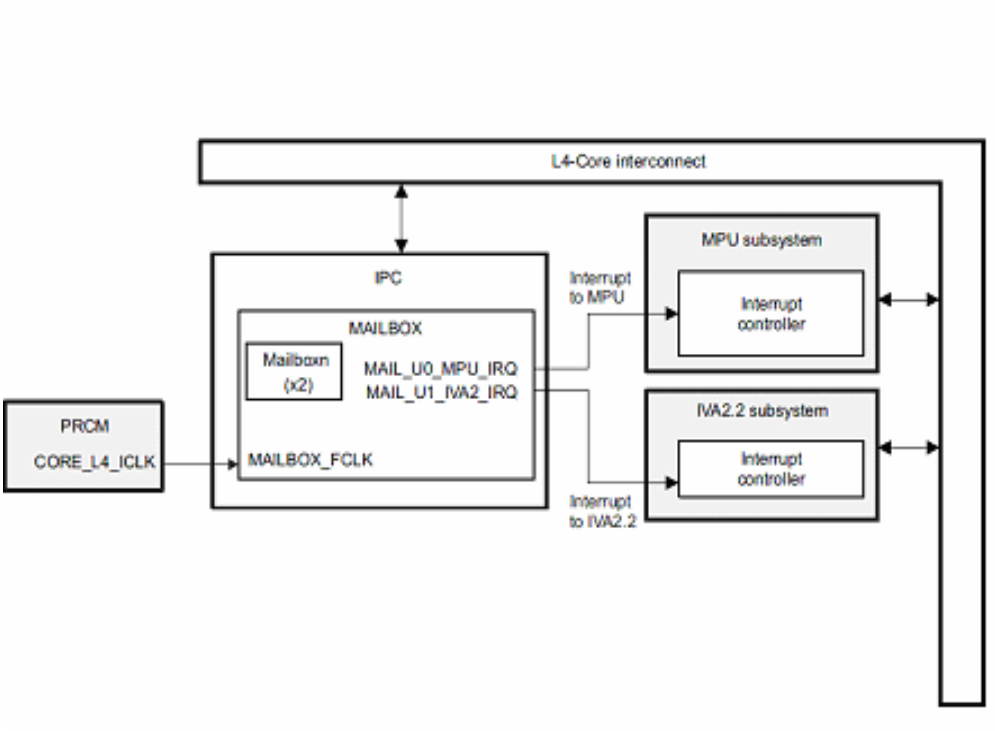

Figure 2.8: Mailbox – interrupt mechanism block diagram

There are two mailbox message queues for microprocessor unit (MPU) and imaging video and audio accelerator (IVA2.2) communications in which 32 bit message width is used and consists of message FIFO depths [14].

In this thesis, codec base communication between DSP core and ARM core is performed, which indeed bases on message box communication that is handled by codec engine. The application code calls the Codec Engine APIs

that are provided by DVSDK code support tool chain [15]. Inside Codec Engine, the video, image, speech or audio (VISA) applications use stubs and skeletons while accessing the core engine and the actual codecs, and this operation can be local or remote [15]. In figure – 2.9, general architecture of engine operation together with application is depicted [15].

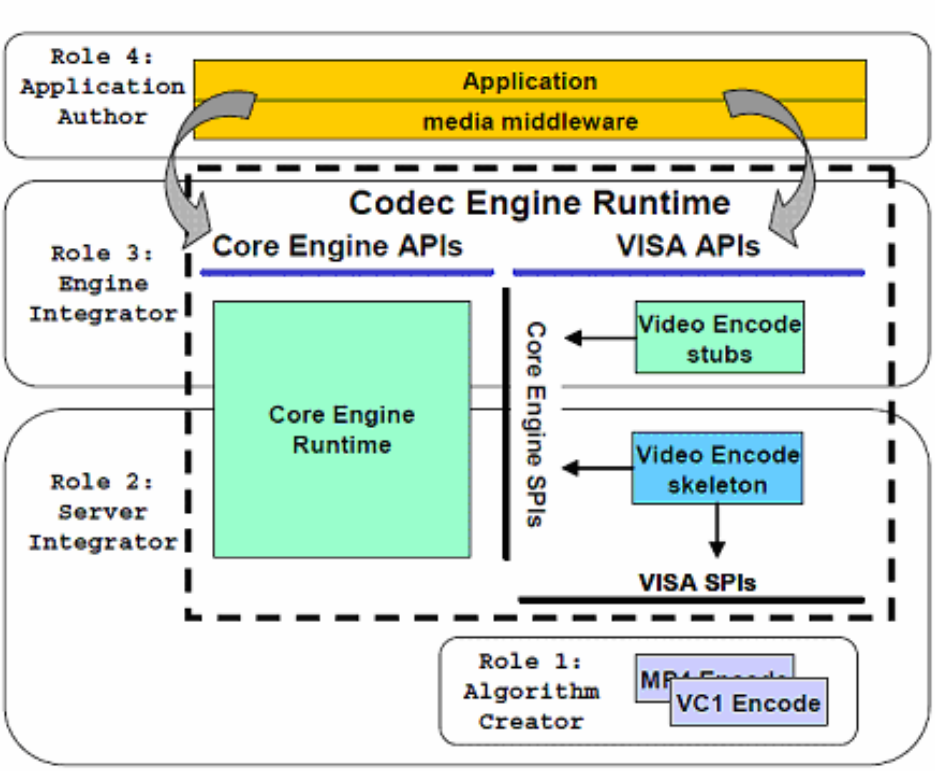

Figure 2.9: General architecture of codec engine operation

Performing applications with provided codec's provide user with reaching DSP anytime needed. However, user should decide on inter-processor communication algorithms more once he tries to operate DSP and ARM core together in parallel. At instants when DSP and ARM cores together try to reach some critical memory locations or use peripherals, drivers can result in unpredictable problems.

CHAPTER 3

INTERMEDIATE VIEW RECONSTRUCTION

In this chapter, basic intermediate view reconstruction techniques for multiple calibrated camera inputs with depth maps is provided, moreover, basic tips for intermediate view reconstruction technique are suggested for further optimization of rendering algorithms in order to achieve real time operation.

This chapter mainly consists of three main parts. Firstly, previous work on intermediate view reconstruction is discussed, secondly camera model used in rendering algorithm is cleared and finally detailed information on suggested algorithm is presented.

3.1 Intermediate View Reconstruction

In the presence of stereoscopic views, construction of virtual camera views using calibrated camera inputs is called intermediate view reconstruction [1]. Calibrated camera inputs are used together with predetermined depth maps for input frames in order to construct intermediate virtual views. Calibrated camera inputs are defined to be images for which camera calibration parameters, namely interior and exterior parameters are known [1].

3.2 Previous Work

Image-based rendering methods and techniques are widely used for construction of virtual scenes in three dimensional applications by using depth information for each input scene or warping more than one two-dimensional image [16]. There exist many different approaches to image-based rendering methods which are mainly discriminated according to usage of depth image for scenes.

Mainly two problems arises when using images for image-based modeling and rendering primitives, namely simplification of modeling techniques for complex scenes and improvement of rendering speed [16]. These methods require extra detailed and complex pre-processing and extra time for scene structure for rendering, which in turn brings about loss of time and extra processing power. Such methods are preferable for parallel processing units such as FPGAs or CPLDs.

In this thesis, an altered version of intermediate view reconstruction techniques based on 3D geometric model of scene represented by depth image, is proposed. Pin-hole camera model is used for realization of the algorithm.

3.3 Pinhole Camera Model

Camera models are used in order to relate 3D world coordinate system with 2D image coordinate system [1]. This relation is represented by:

$$\mathbf{x} = \mathbf{P}\,\mathbf{X}$$

where **x** represents 2D pixel position, **P** represents camera matrix and **X** represents 3D coordinate of sampled point in space.

In this thesis, finite projective camera model which is also known to be pinhole camera model is used. It is assumed that light enters through an infinitesimal hole which results in a formation of inverted image on the image plane as depicted in figure-3.1 [17].

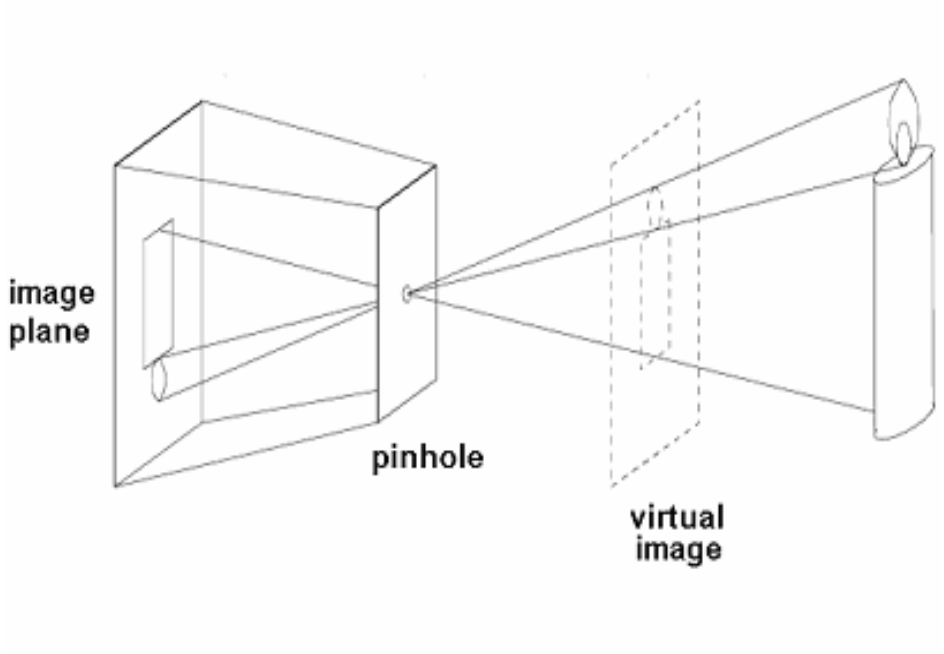

Figure 3.1: Image formation through pinhole camera

As can be noticed from figure-3.1, size of the object projected on image plane depends on actual size of the object and its distance from camera center. This is the reason why pinhole camera is known as perspective projection camera model.

3.3.1 Pinhole Camera Geometry

Images are projected on image plane with an inverted shape for pinhole camera as mentioned before. In order to simplify calculations, pinhole camera geometry is modeled as shown in figure-3.2.

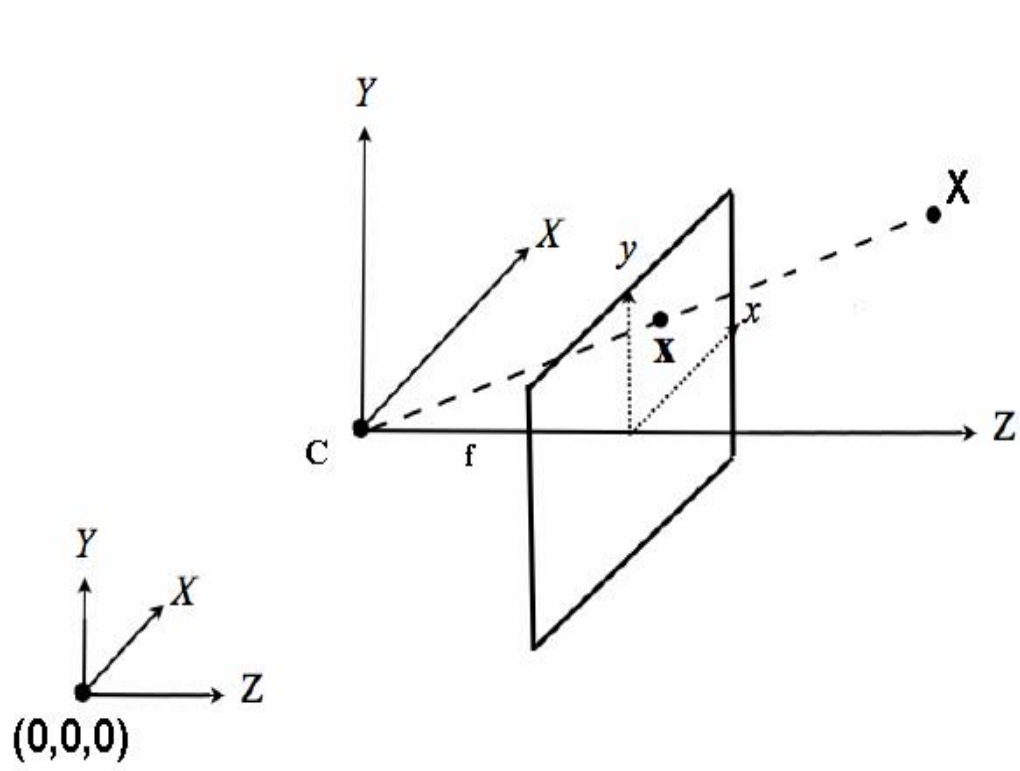

Figure 3.2: Pinhole camera geometry

C is denoted to be camera center with respect to world coordinate system. X is known point on real 3D world whereas x is pixel position on image plane. The line passing through camera center point C in the direction perpendicular to image plane is called principal axis and intersection of principal axis with image plane is called principal point [18]. For the simplicity of calculations, it is assumed that camera center sits at origin at

start point. Later, rotation and translation of points with respect to world's coordinate system is considered.

In [1], intrinsic parameters of the pinhole camera which constitute camera calibration matrix are given to be:

- Focal length f,
- Principal point, (u_0, v_0)
- Aspect ratio, a
- Skew, s

Considering figure-3.3, it is shown that a 3D point in (X, Y, Z) is projected on image plane using similarities of triangles [18].

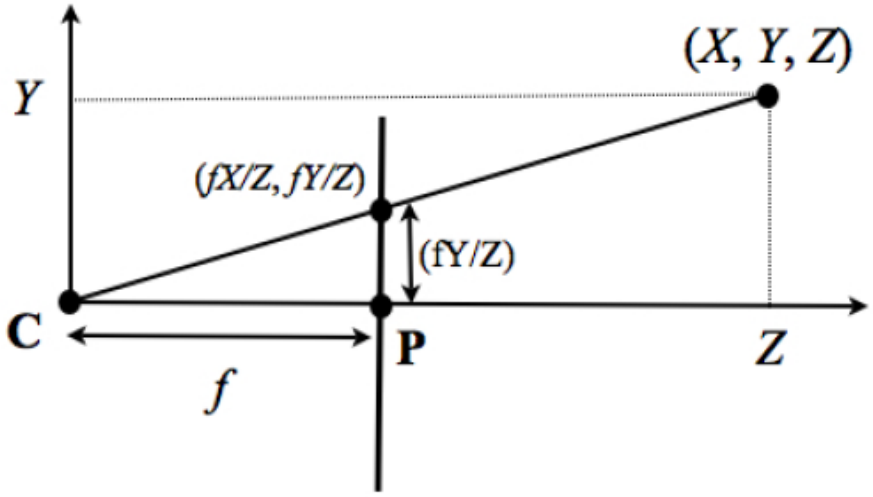

Figure 3.3: 3D point projection on image plane

Therefore first formation on point $x = [\ f.X/Z \quad f.X/Z]^T$. Considering principal point affect on translation of point x by u_0 and v_0, one can formulate new point x as:

$$x = [(f.X/Z + u_0) \quad (f.X/Z + v_0)]^T$$

When angle (β) between optical axes on image plane is not a right angle, skew parameter is added to the formulated x point as [1]:

$$x = [\ (\ f.(X/Z) - f.\cot(\beta).Y/Z + u_0\ ) \quad (f/\sin(\beta).Y/Z + v_0)]^T$$

Aspect ratio gives information on relation between width and height of camera pixels. Aspect ratio is the ratio between these two quantities. If aspect ratio α_x / α_y is different from unity, then as a scaling factor between pixel dimensions and camera focal length, one can express $\acute{\alpha}_x = f / \alpha_x$ and $\acute{\alpha}_y = f / \alpha_y$ [1]. Finally, the transformation can be modeled as [1]:

$$x = \begin{bmatrix} \acute{\alpha}_x & -\acute{\alpha}_x \cot(\beta) & u_0 \\ 0 & \dfrac{\acute{\alpha}_y}{\sin(\beta)} & v_0 \\ 0 & 0 & 1 \end{bmatrix} [\ I \mid 0\ ]\ X$$

Extrinsic parameters are changeable and depend on coordinate system adjustment between camera center and real world coordinate system, which are operations of translation and rotation between two coordinate systems. Considering that X denotes 3D real point coordinates according to world's coordinate system, pixel position x is reformulated as:

$$x = K\ [R \mid t\ ]\ X$$

where K matrix denotes intrinsic parameters of the camera and in this algorithm, R matrix and t vector are given as rotation matrix and translation vector respectively.

3.4 Intermediate View Reconstruction Algorithm

In this thesis, intermediate view reconstruction based on 3D warping algorithms is implemented. For the realization of algorithm, 3D video colored frames and their corresponding depth map images are used. An input frame sequences are obtained from Microsoft research group Sing Bing Kang, which provides 100 frames and their corresponding depth maps for a ballet sequence [19].

Intermediate view reconstruction algorithm is applied on two camera frame sequence namely camera 6 and camera 4. Any point inside region between camera 6 and camera 4 is considered to be a potential virtual camera position. A smooth transition within this region is provided when user changes virtual camera position. Figure-3.4 resembles corresponding region and camera positions together with virtual camera orbit.

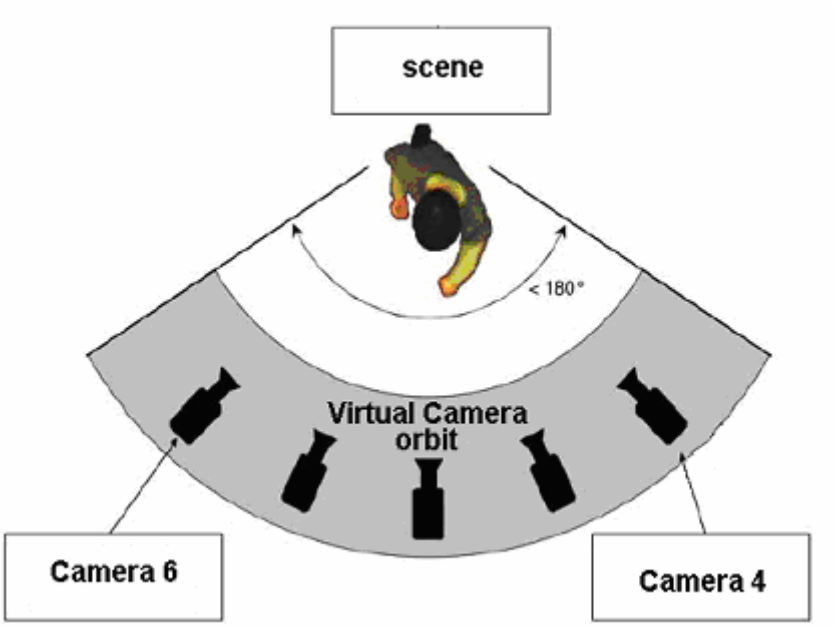

Figure 3.4: Camera positions and virtual camera orbit

Figure-3.5 and figure-3.6 shows two simultaneously taken frames for two different cameras and their corresponding depth maps as an example.

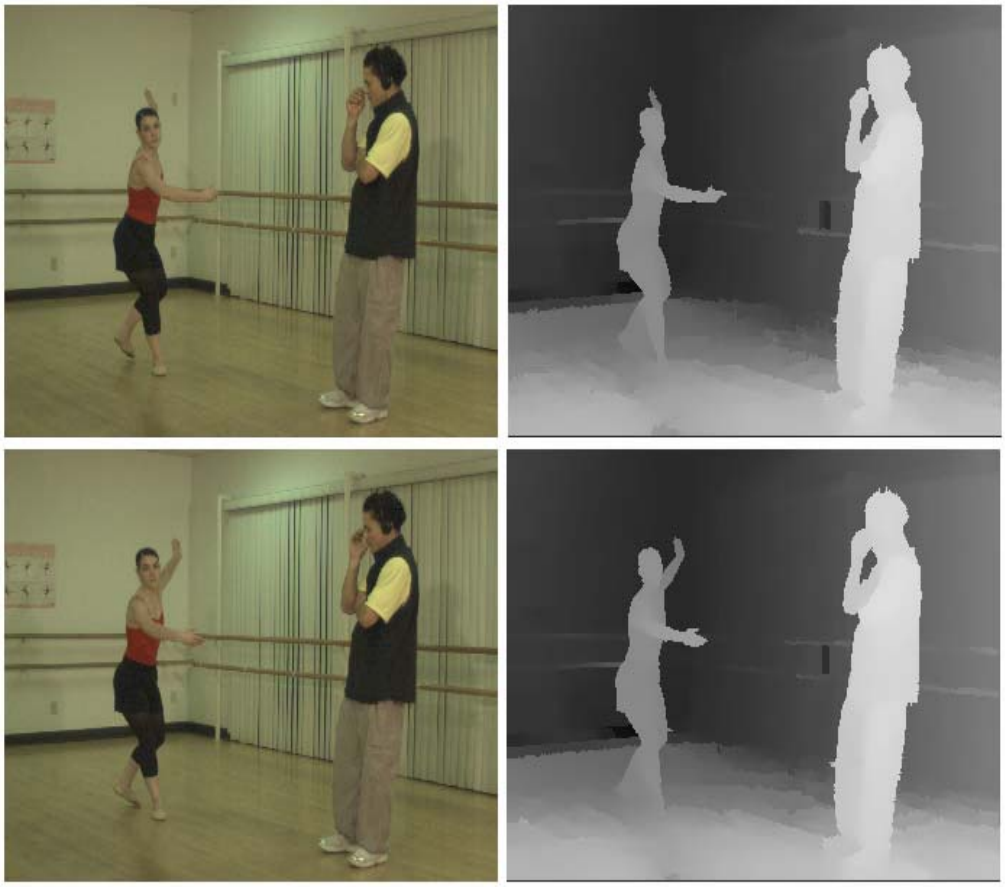

Figure 3.5: Two frames with corresponding depth maps from camera 6

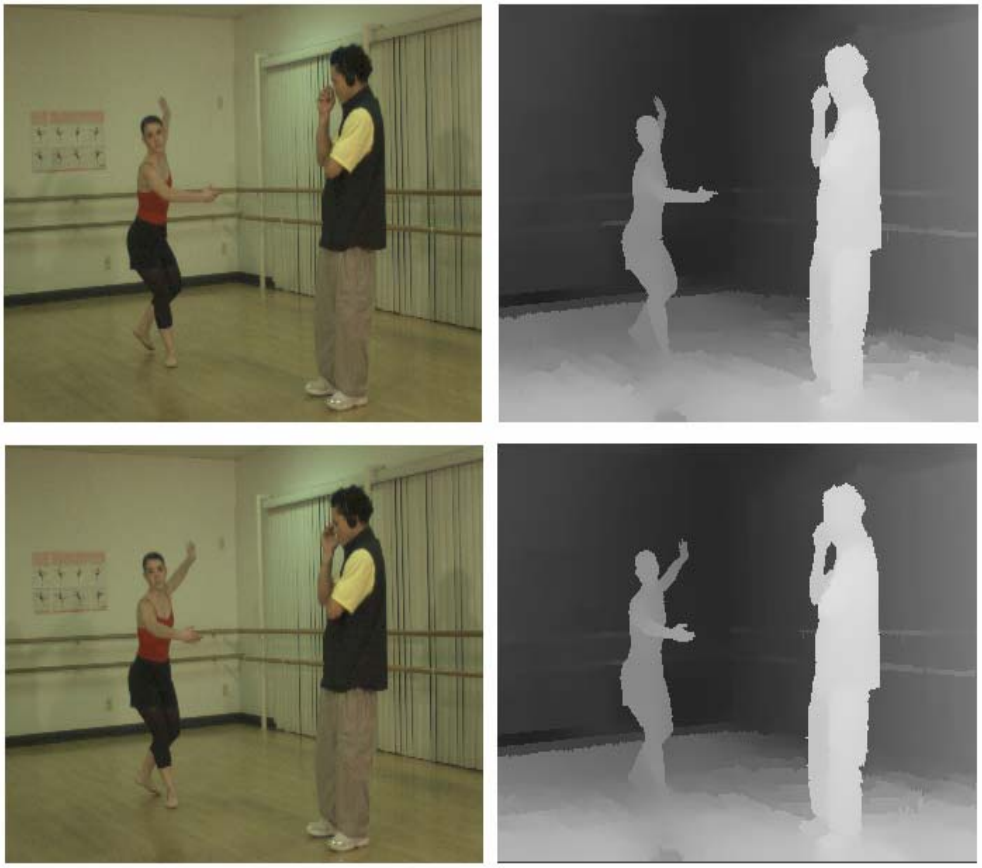

Figure 3.6: Two frames with corresponding depth maps from camera 4

3.4.1 Algorithm Outline

Considering that desired virtual camera position is determined by user in advance, intermediate view reconstruction algorithm is consists of 3 main steps:

1. Decode depth map pixel value in order to obtain real distance of corresponding point in space.
2. Using pixel position on image plane, distance from camera center and real camera position in 3D space determine 3D points.

3. Using camera intrinsic parameters and virtual camera position, project 3D points on to virtual camera image plane.

It can be observed from figure-3.5 and figure-3.6 depth maps, that nearby points are shown brighter with respect to the points that are far away from real camera center. This gives primary information for algorithm builder, about point positions and their depth value. This monochromatic image contains pixels ranging from 0 to 255 which do not directly give distance information from camera center. Realistic distances normalized for camera intrinsic parameters are calculated by [19]:

$$Distance = 1 / ((Pixel / 255)*0.0161 + 0.0077)$$

This formula contains division terms. It was observed that divisions and floating point multiplications inside given formula slow down processing speed considerably. Therefore as an optimization tip, rather than using this formula inside algorithm loop, a look up table is initially constructed and used during algorithm loop.

After determination of realistic depth value, using base camera position information from translation matrix provided, it is possible to obtain 3D point corresponding to each pixel.

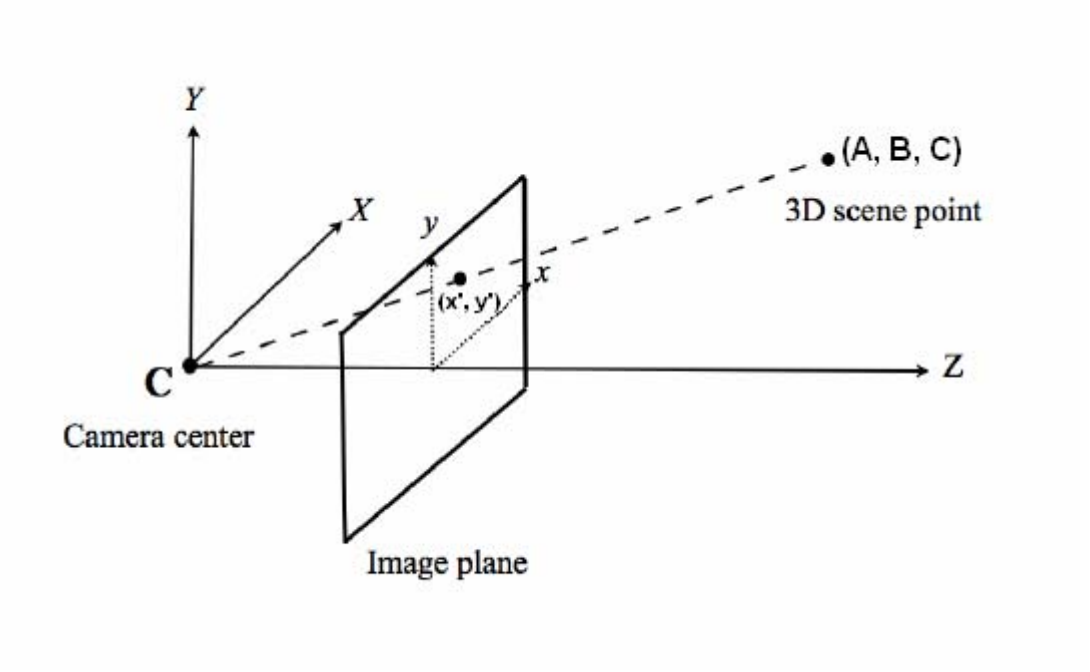

Figure 3.7: 3D point determination using camera coordinate system

Using line equation;

$$\frac{A - Xc}{k_x} = \frac{B - Yc}{k_y} = \frac{C - Zc}{k_z}$$

$$\frac{x' - Xc}{k_x} = \frac{y' - Yc}{k_y} = \frac{f - Zc}{k_z}$$

Moreover, obviously (x', y', f) and (A, B, C) points are on the same line. (A, B, C) is unknown while (x', y', f) is known. Therefore it is sensible to consider for the first equation that:

$$k_x = x' - Xc$$
$$k_y = y' - Yc$$
$$k_z = f - Zc$$

$$M(k_x^2 + k_y^2 + k_z^2) = (A^2 + B^2 + C^2) = D^2$$

where M is a constant depending on real depth value that is previously determined by initially constructed depth look-up table, which speeds up the process inevitably. Obviously, M is calculated by known D, k_x, k_y and k_z, which in turn gives the possibility for algorithm builder to calculate x, y and z coordinates of considered point namely A, B, C.

Given k_x, k_y and kz equations, it is clear that these constants depend on rendering sequence for each pixel on the image, in other words, once rendering sequence is fixed, it is clear that square sum of the given constants are known in advance, which gives opportunity to algorithm builder to use another look-up table.

Such an algorithm speeds up intermediate view reconstruction considerably. Moreover, rather than dealing with square operations, it is sensible to construct look-up tables after taking square roots.

Finally, as depicted on figure-3.8, 3D scene point is mapped of second image plane after rotating and translating point according to virtual camera center and by using intrinsic parameters of virtual camera.

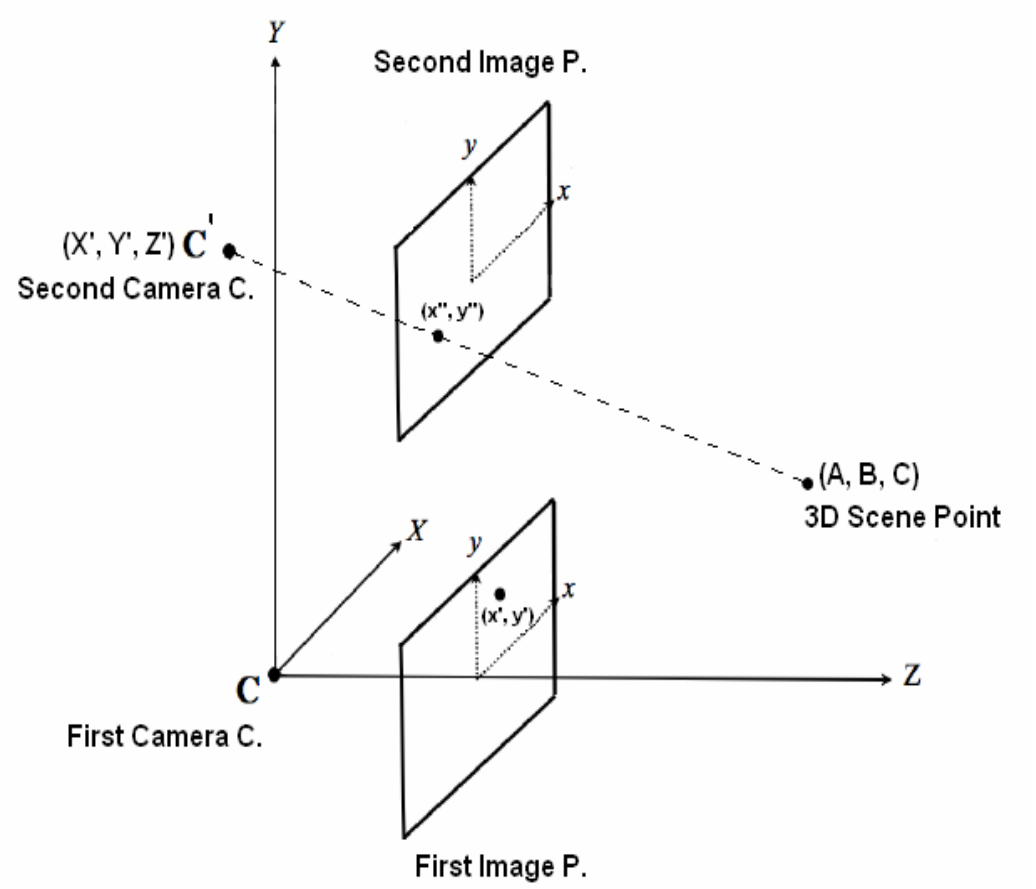

Figure 3.8: Mapping 3D scene point on second image plane

Without optimization in algorithm, an arbitrary intermediate view is constructed in 2 seconds without dealing with occlusion regions by ARM core, in 3 seconds by DSP core. After optimization, 18 frame/sec rate is achieved by ARM core while 12 frame/sec rate is achieved by DSP core.

In figure-3.9 original frame for camera 5 is given. In figure-3.10, constructed intermediate views from camera 4 and in figure-3.11 constructed intermediate view for camera 5 from camera 6 are given. These outputs are constructed without occlusion region handling which is explained in chapter-4 in details.

Figure 3.9: Original input frame from camera 5

Figure 3.10: 5th camera frame reconstruction output from camera 4

Figure 3.11: 5th camera frame reconstruction output from camera 6

CHAPTER 4

OCCLUSION HANDLING

In this chapter, basic occlusion region filling methods are explained and a new faster algorithm for occlusion handling is documented for real time video systems.

In this chapter, performance of suggested algorithm is explored using Microsoft research group ballet sequence of camera 6 frame 0, camera 4 frame 0 and camera 5 frame 0 [19].

This chapter is mainly composed of two main sections. Firstly, previous work on occlusion region filling algorithms is discussed with general aspects. In the second section, newly suggested fast algorithm on occlusion region handling is discussed.

4.1 Literature Review

Occlusion area is defined as the area appearing in one of input images while disappearing from the other camera inputs due to depth image intensity differences resulted by scene structure [2]. This problem is illustrated in figure-4.1 [2].

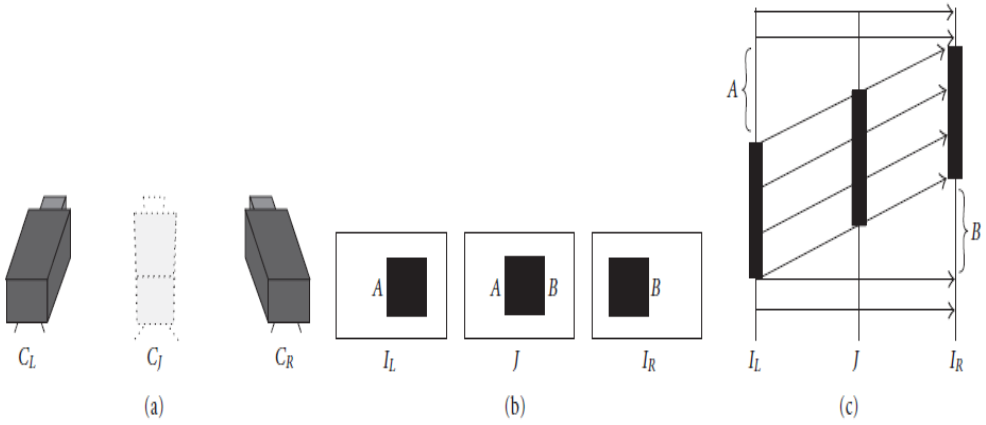

Figure 4.1: Occlusion region corresponding to different camera position

In figure-4.1.a, two real camera positions and one virtual camera position is illustrated. In this example virtual image generation for camera position Cj is examined. Area B is an occlusion region for camera L and area A is an occlusion region for camera R. For virtual camera J however, both of the given areas are visible partially, due to depth differences resembled in figure-4.1.c.

3D models are mostly generated manually by 3D artists using 3D modeling programs in gaming and animation technology [20]. However this is not always possible for 3D real scenes. For example, live broadcasting would not give possibility for 3D artists to operate such modeling programs.

If few input images are available for free-view point videos, occlusions pose an important challenging phase for image warping [2]. Therefore, algorithm builder should pay significant attention on disparity estimation and view interpolation, when occlusions exist. Many algorithms are suggested for this significant issue in image-based rendering systems. According to a newly

proposed algorithm in [2], construction of a new view point image consists of three main steps for which 4 input images are used for construction [2]:

- All pixels in the input image are classified in terms of their visibility for the new view point.
- Using different image inputs, disparity for each pixel is calculated in the second phase, depending on the first phase output.
- Each pixel color is calculated adaptively at the final step from image pairs that are selected by their visibility labels.

This algorithm is useful for parallel processing systems and for systems where memory space is not a restricted issue. That is because for an image warping operation with occlusion regions are filled, there exists many steps and four synchronized input frames are required.

In [3], three basic steps are offered for hole-filling problem using polygon view maps, which basically depend on connectivity between each pixels at vertex positions. These steps are summarized as follows:

1. Set up a linked list representing connectivity of pixel to each other which in turn gives possibility to reach all neighboring pixels for each one.
2. Using so-called linked list, compute average color of neighboring pixels and assign it to the ones which are invisible for virtual image.
3. If there still exist invisible pixels, assign each of their vertices the colors of closest polygon which are visible for virtual image.

This method gives successful results in filling holes for images having smooth depth maps and for scenes in which objects are far away from camera position. In figure-4.2, an example for this algorithm is given [3]. Occlusion regions are filled with the suggested algorithm. Occlusion regions

are difficult to be noticed for the given image in which averaging is possible for polygons. These polygons are obviously very close to each other which in turn results in smooth transitions between hole edges.

Figure 4.2: Efficient view-dependent image-based rendering with projective texture-mapping example

For hand-held devices, users would like to see regions which are invisible for the present free-view point. It would be attractive for the user to see

change in view-point angle; moreover it would also be an exciting experience for the user to have knowledge of invisible regions by changing virtual camera position and its angle. Therefore, occlusion regions should be filled by exact scene view using second camera input rather than averaging nearby pixels.

Although, there exist many virtual image rendering techniques up to now, few of them handle occlusions accurately [2]. In this thesis a new algorithm is proposed for occlusion region handling. This algorithm is suggested to be performed for real time free-view point and multi-view 3D systems.

4.2 Fast Occlusion Handling Algorithm

It would be convenient to consider filling occluded regions by the help of second camera for those systems, where parallel processing is available. Regions that are invisible to one of input images are usually visible for the other input image, depending on objects of scene and their distance from camera positions. In other words, depending on 3D scene, some of the regions may be invisible for both of real camera positions.

3D image warping operation can expose specific areas to be shaded where according to reference frame; there exists no information [21]. This problem is illustrated in figure-4.3.

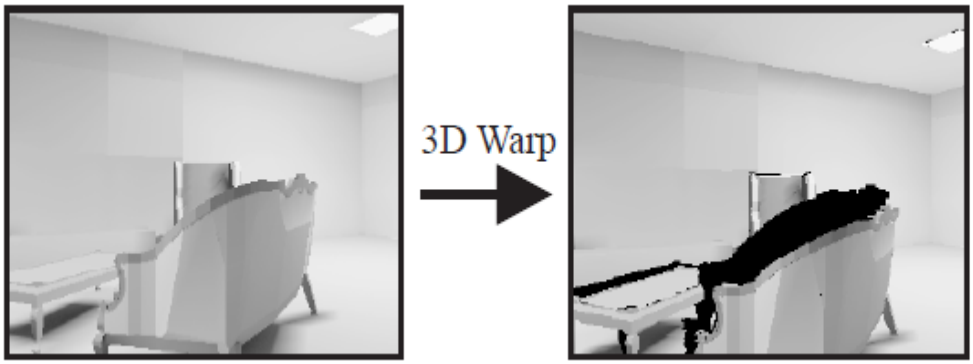

Figure 4.3: Occlusion problem after 3D image warping [21]

It is possible to fill considered occlusion areas by the help of additional synchronized frames from different view points. This process is represented in figure-4.4.

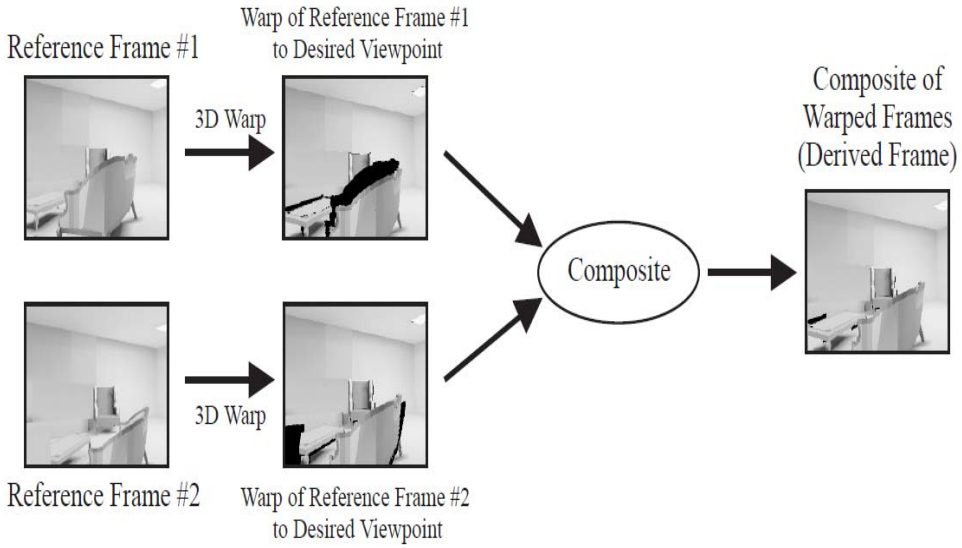

Figure 4.4: Occlusion region filling process [21]

In this example, it is possible to realize that some of occlusion regions are invisible for both of camera positions which resulted in hole regions in the output image. In this thesis, in order to decrease probability of such cases, restricted area for Virtual camera position is used between two real camera positions as depicted in figure-4.5.

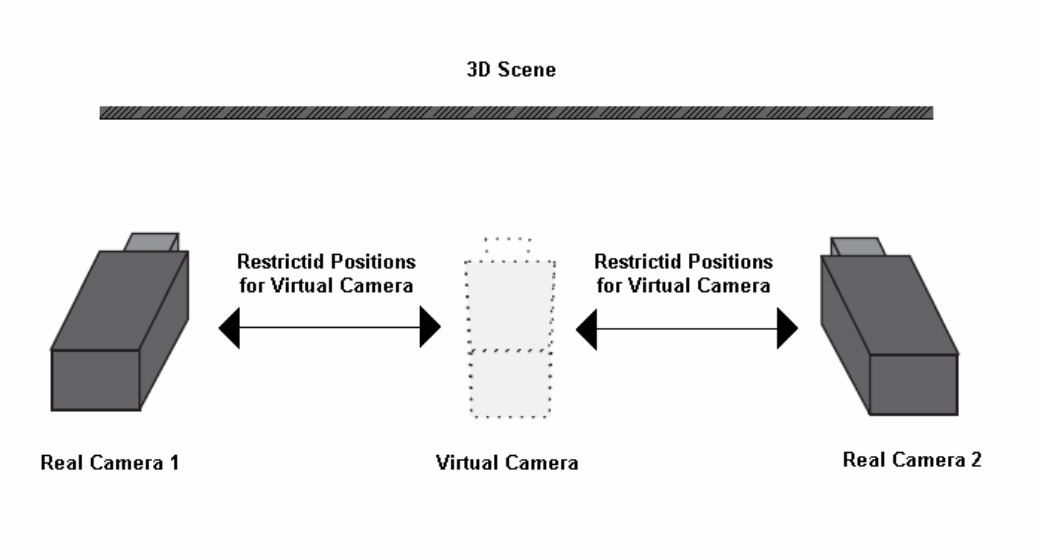

Figure 4.5: Virtual camera position between real cameras

Considering a hardware system with two processors having independent memory access simultaneously, while rendering an image frame from base camera by the general purpose processor, co-processor may render synchronized image frame from second camera, for occluded regions. This basic algorithm can be best implemented on FPGAs, where parallel processing can be best realized.

After detecting occlusion regions during warping algorithm, all pixels lying on occluded regions would be filled one by one according to co-processor synchronized image output. This basic occlusion region detection algorithm is realized by assigning a specific character for each pixel position on the output image, depending on whether or not, there exists a hole on the pixel

position. In figure-4.6, a very basic flowchart of this simple algorithm is given.

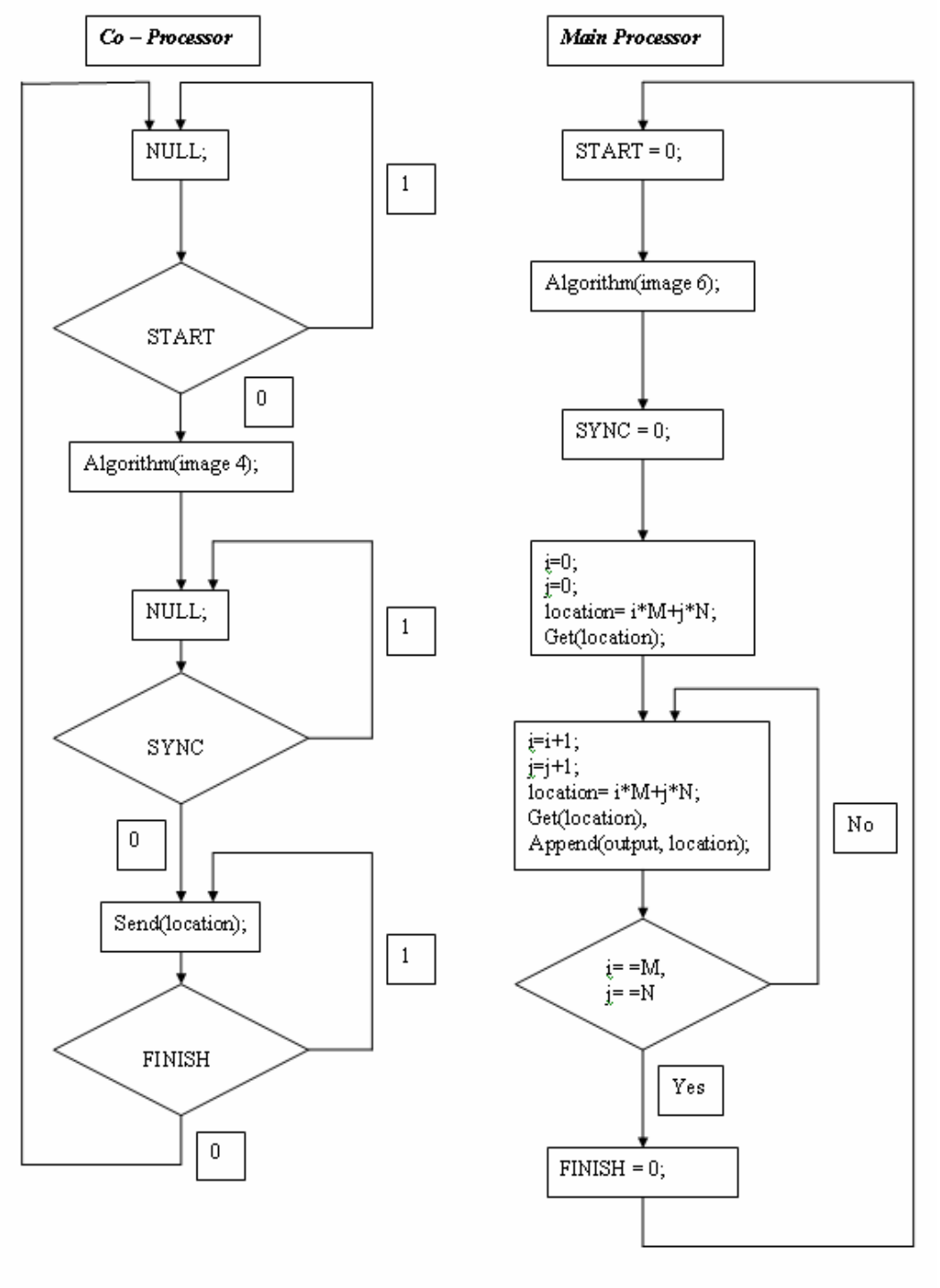

Figure 4.6: Basic flowchart for two processor system occlusion filling algorithm

This algorithm is interpreted to be inefficient even for considered multi-processor or FPGA based signaling systems. Power consumption would be doubled while running more than one processor at a time doing same amount of multiplications and additions for each frame. Moreover, second processor may not be devoted to any other task during warping algorithms. Rather than applying rendering algorithm for all pixels one by one by the co-processor, considering only pixels that are candidates to be used while filling occluded regions would definitely decrease amount of multiplications, divisions and additions for co-processor, reducing power consumption and giving the opportunity for software developer to devote co-processor to any other task until main processor finishes its rendering algorithm.

This re-configured algorithm would also be efficient for single processor systems without a doubt. In figure-4.7, basic representation of this algorithm is given for multiprocessor systems.

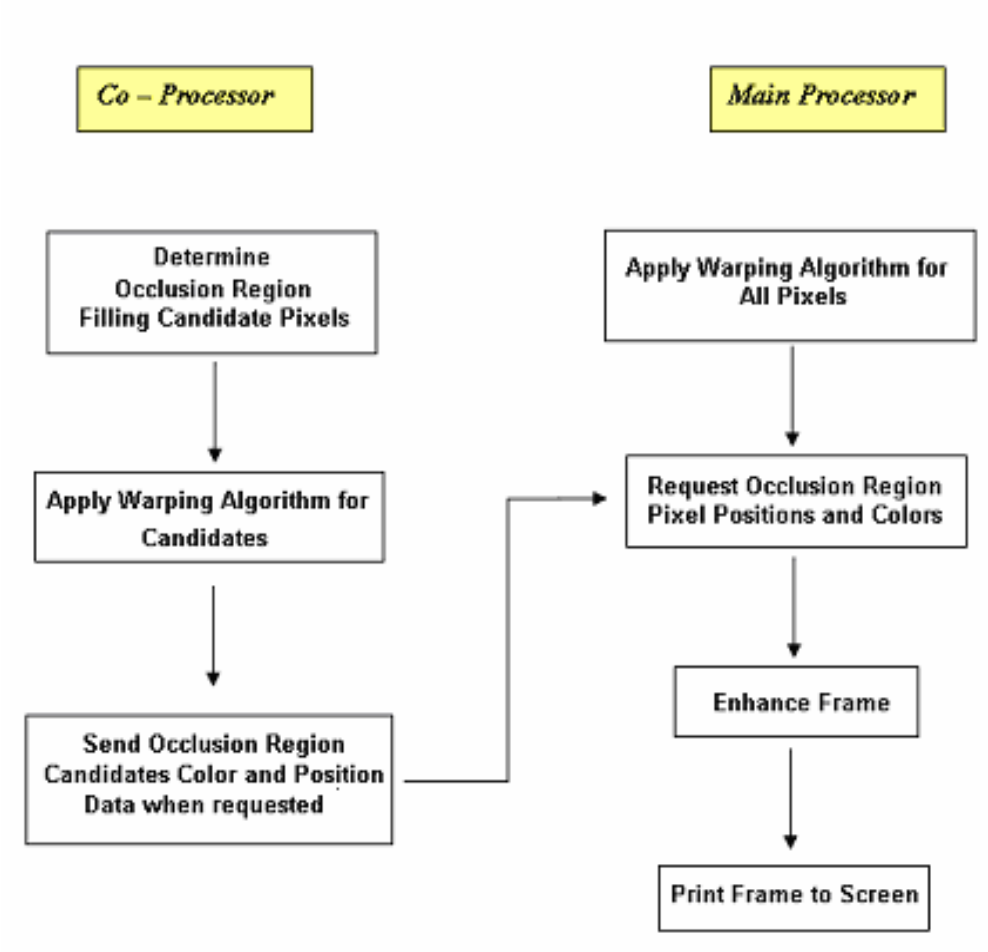

Figure 4.7: Basic flowchart for multi-processor system, efficient occlusion filling algorithm

Disadvantage of this new re-configured basic algorithm is the detection of pixels which would be used while filling occluded regions and which are not. This would in turn bring about an additional decision process for each pixel, whether the pixel would be one of those that fill occluded regions, or not.

4.2.1 Algorithm Outline

For a given virtual camera position that is determined by user, considering camera-6 input frame as the base camera input and camera-4 input frame to be auxiliary frame for hole filling, this algorithm can be outlined in 4 steps for single-core system as follows:

1. Apply image-based warping algorithm for input frame of camera-6 as described in chapter-3, while saving a character array of warped positions with '1' indicating non-occlusion pixel, and '0' indicating occlusion region pixel.
2. Determine pixel positions that are candidates for filling occlusion regions, using camera-4 input frame depth map.
3. Apply image-based algorithm taking additional rotation matrix into consideration for camera-6 position care, for these candidate pixels.
4. Check each element of warped position holding array and fill occlusion regions with the corresponding to specific positions of camera-4 occlusion filling pixels.

4.2.1.1 Image-Based Warping Algorithm

Detailed explanation for this algorithm is given in chapter-3.

4.2.1.2 Occlusion Region Determination

Considering N x M image frames to be rendered for a single processor system, while applying rendering algorithm for each frame of base camera, an array of N * M character elements is assigned holding '1' for non-occluded pixels and '0' for occluded regions, which at the end of algorithm, basically shows occlusion regions. In figure-4.8, occluded and non-occluded

map for camera-6 frame 0 is given to be an example determined occluded regions for generated camera-5 frame 0. Note that occlusions are resembled by black regions.

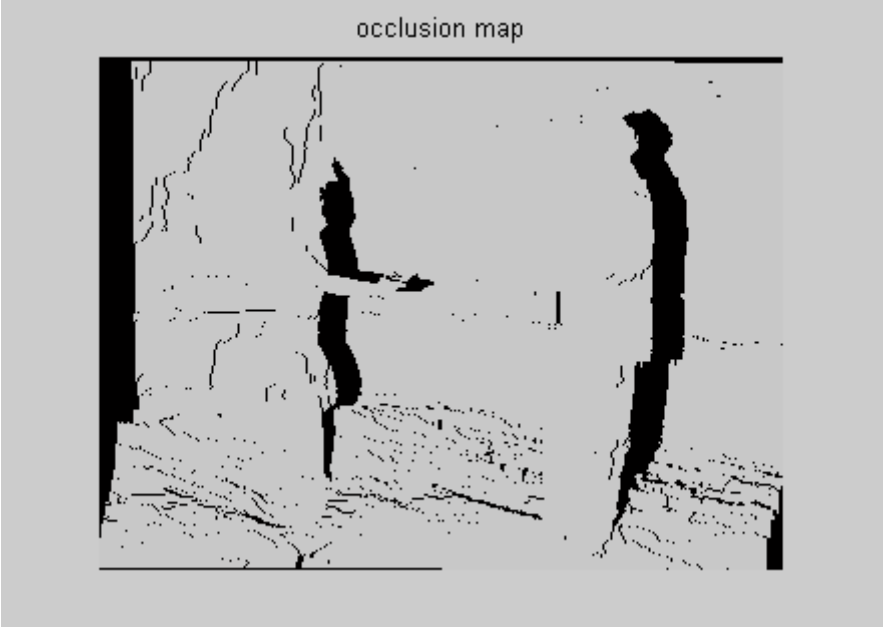

Figure 4.8: Occlusion map corresponding to camera 6 frame with respect to camera 5 virtual position

Having determined occluded pixel positions, it is not possible to find corresponding pixel values on camera 4 synchronized frame using reverse action. This is because there is only the knowledge of f5 (fifth camera focal length), f4 (fourth camera focal length) and **x** (pixel position that is determined from camera 6 occluded region array). It is possible to determine equation of the line that connects camera 5 center with the point **x** on image plane 5. However, without depth information of possible projecting point P, it is not possible to determine exact location of pixel from camera 4 that would map to point **x**. There are infinitely many points on 3D coordinate

system which would be projected on image plane 4. As can be seen from figure-4.9, **P1** and **P2** are two of these points.

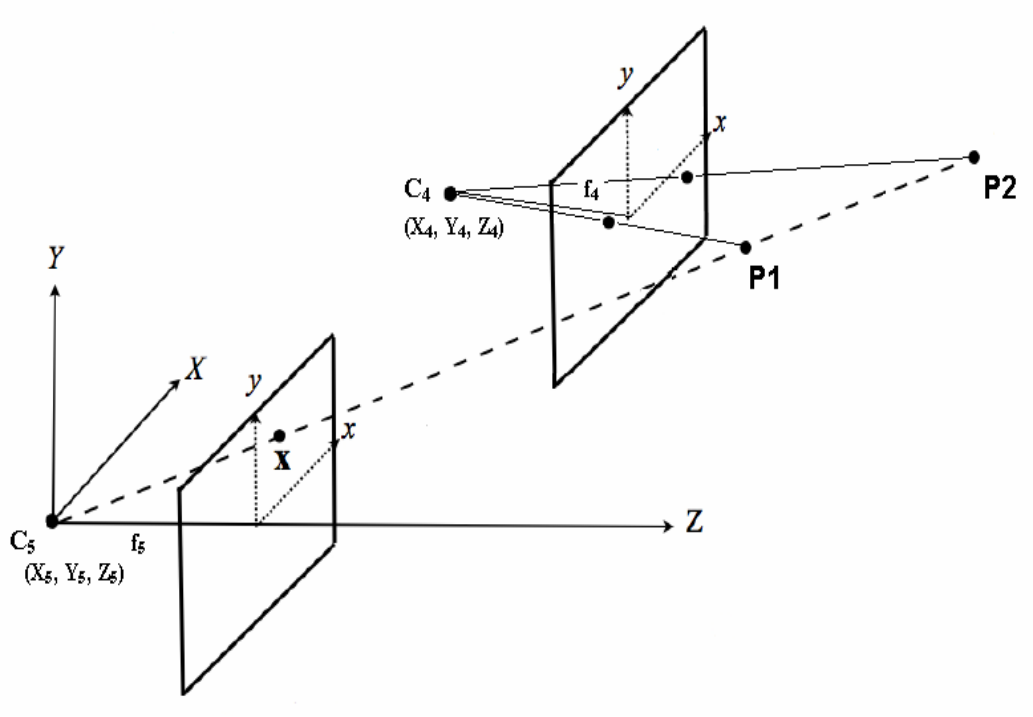

Figure 4.9: Back projection of occlusion region pixels

Considering **x** point of coordinates (x, y, f_5), line equation is:

$$\frac{(x - X_5)}{k_x} = \frac{(y - Y_5)}{k_y} = \frac{(z - Z_5)}{k_z}$$

Where,

$$k_x = (x - X_5);$$

$$k_y = (y - Y_5);$$

$$k_z = (f_5 - Z_5);$$

Rather than trying to determine occlusion region filling pixels using the map array that is created while applying warping algorithm on camera 6 frame, it would be sensible and efficient to determine these pixels and their corresponding projected positions from camera 4 input frame. In this newly proposed method, occlusions of warped image from camera 6 and corresponding filling pixels of camera 4 are determined using camera 4 depth map alone. This will bring about additional, however, fast process, in which software developer can determine these pixels.

4.2.1.3 Occlusion Region Filling Pixel Determination Method

In figure-4.10, generated camera five output from camera four and camera six is shown.

Figure 4.10: Reconstructed camera 5 from camera 6 and 4

Occlusion regions for the generated output of camera 6 are filled by loading corresponding pixels of generated output of camera 4 directly.

Considering camera 4 frame 0 depth map, scanning column by column from left to right, it is possible to understand starting and finishing pixel positions

for occlusion region. This algorithm can be realized using camera 4 depth map alone. Moreover it is easier and faster to determine starting pixel location by camera 4 depth map and finishing pixel location by camera 6 depth map.

Considering about abruptly drawn top view of 3D scene in figure-4.11, it is obvious that R1 and R2 regions can not be viewed by camera 6. Although R1 and R2 regions are seen by camera 4, it will be unnecessary to determine where pixels in R1 regions are projected according to camera 5, because R1 region is not in the sight of camera 5 and these pixels will be unseen at the end of rendering algorithm.

R2 resembles the occlusion regions according to camera 6. In this algorithm, Ps (starting pixel of occlusion region filling candidates) and Pf (finishing pixel of occlusion region filling candidates) are determined for each row on image frame from camera 4 and corresponding pixels are rendered using camera 4 input in order to fill holes that will occur at the end of rendering algorithm for camera 6.

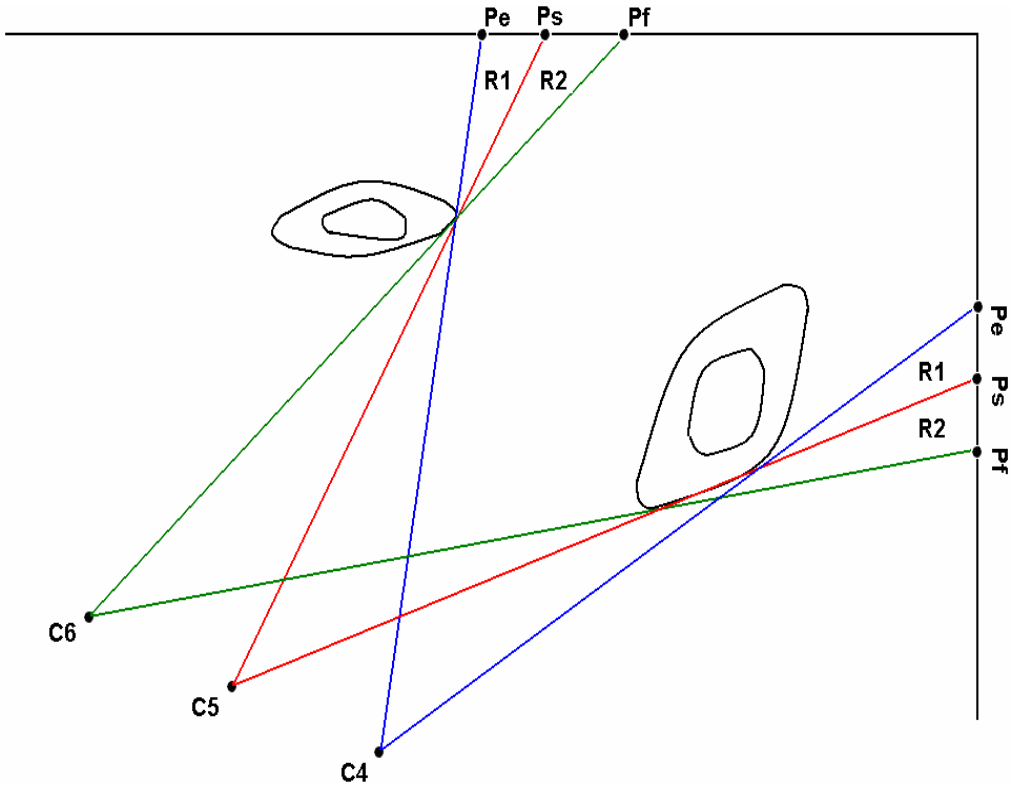

Figure 4.11: Arbitrarily drawn real scene top view

4.2.1.3.1 Determination of Ps

After applying simple edge detection algorithms in horizontal +x direction to depth map of camera 4 frame, it is possible to determine exact location of Pe as shown in figure-4.12. For each row, using a threshold value, after determining Pe pixel location, it is possible to determine Ps location using rendering algorithm which is explained in chapter-3 in details. Defining $Warp_{4\text{-}5}$ transformation as the mapping of pixel position from camera 4 to camera 5;

$$Ps_5 = Warp_{4\text{-}5}\{ Pe_4 \}$$

It is unnecessary to map pixels lying on R1 region that are visible to camera 4 but invisible to camera 5. This is due to make process faster for real-time video systems. Similarly it is necessary to determine Pf_4 pixel position in order to give stop order for rendering process from camera 4.

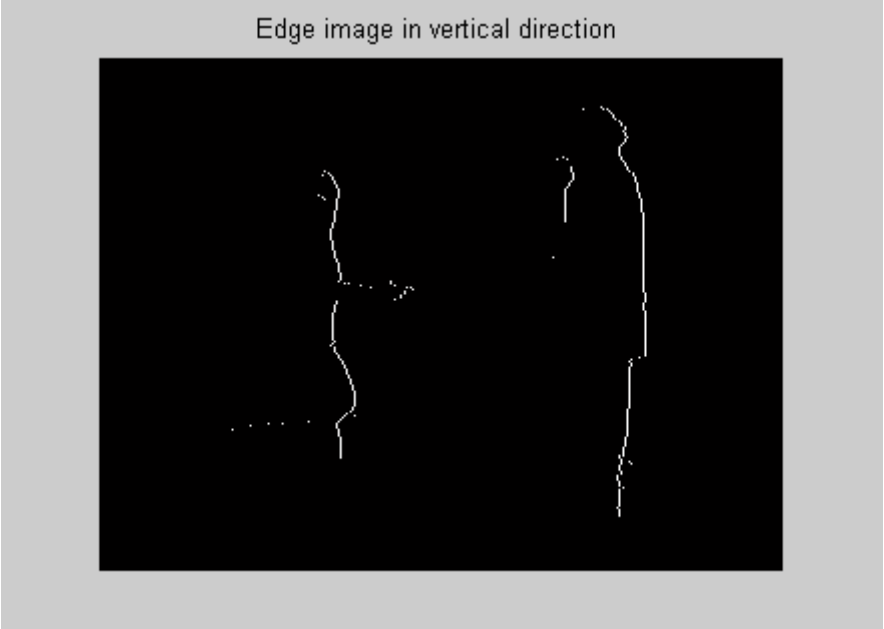

Figure 4.12: Edge image of camera 4 depth map in vertical direction

4.2.1.3.2 Determination of Pf

This time, after applying simple edge detection algorithms in horizontal +x direction to depth map of camera 6 frame, it is straightforward to determine exact location of Pf for each row.

In this algorithm, it is not possible to directly pass location of Pf_6 as the finishing pixel of camera 4 occlusion filling candidate pixels. This is due to rotation matrix consideration. Pf does not directly map to its corresponding pixel for camera 4 frame. This time it is necessary to map corresponding Pf_6

pixel to camera 4 frame. Considering rotation matrices of camera 6 and camera 4, outline of this algorithm patch is:

1. Determine Pf_6 using edge image of camera 6 depth map
2. Calculate relative inverse rotation matrix for camera 6 and camera 4 by,

   Reverse_Rotation = inv(Rotation_6) * Rotation_4
3. Find corresponding pixel location for Pf_4 using Reverse_Rotation matrix;

   $Pf_4 = Warp_{6\text{-}4}\{ Pf_6,$ Reverse_Rotation matrix, Camera Coordinates $\}$

In figure-4.13, simple edge image of camera 6 depth map is shown. Edge image of camera 6 is drawn over edge image of camera 4 in order to depict coordinate system differences of both camera depth maps.

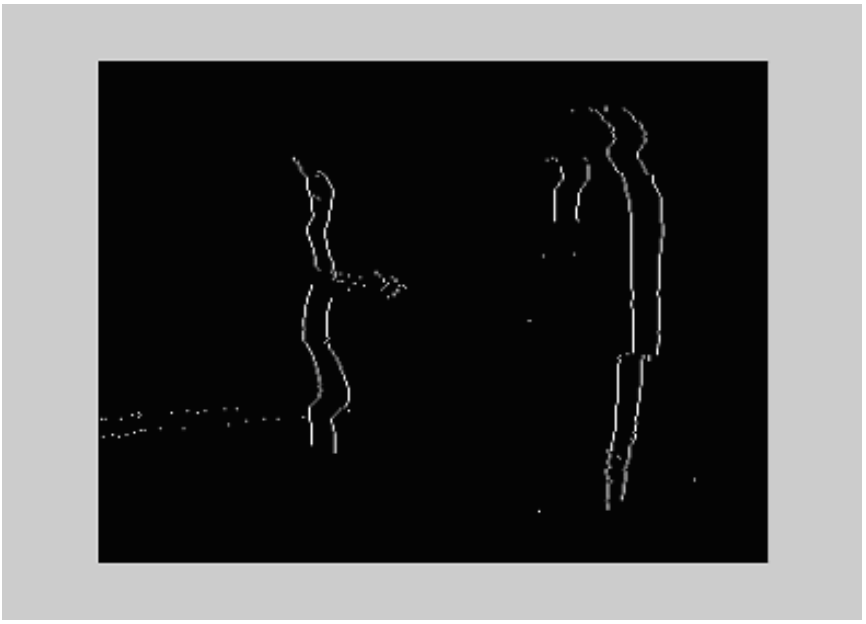

Figure 4.13: Vertical difference of edge images for camera 6 depth map and camera 4 depth map in +x dirrection

It is possible for algorithm developer to consider using look-up tables for this algorithm. An array of finishing pixel values that are depending on depth difference between background and object ahead and camera position value can be pre-stored at the initialization phase in order to make algorithm much faster by not dealing with rotation matrix multiplications and translation matrix summation.

CHAPTER 5

FRAME ENHANCEMENT

In this chapter, enhancement of algorithm output frames using basic median filtering techniques and a newly proposed method is discussed.

In this chapter, performance of suggested algorithm is explored using Microsoft research group ballet sequence of camera 6 frame 0, camera 4 frame 0 and camera 5 frame 0 [19].

This chapter is mainly composed of two sections. In the first part, different median filtering techniques are discussed. In the second part, a new solution is suggested for separation and spread of pixels belonging to same object.

5.1 Median Filtering

Nonlinear image processing techniques have been developed in the last decades, having the advantage of minimizing distortions of informative characteristics [22]. Median filtering technique is one of these non-linear image processing techniques that are widely used for salt and pepper noise removal.

Median filtering techniques are based on the assumption of presence of salt and peeper type of impulse noise [23]. Considering median filtering process through input signal, each time a sample patch of input signal is examined for each pixel position and its neighboring pixels in order to determine whether the pixel belongs to input signal. Main idea lying behind median filtering is the sort operations of pixels inside considered sample and choosing the median pixel as output for that sample. The aim is to remove a pixel if it is not a representative of signal, i.e., representing noisy part.

In figure-5.1, an example of median filtering process is depicted [23].

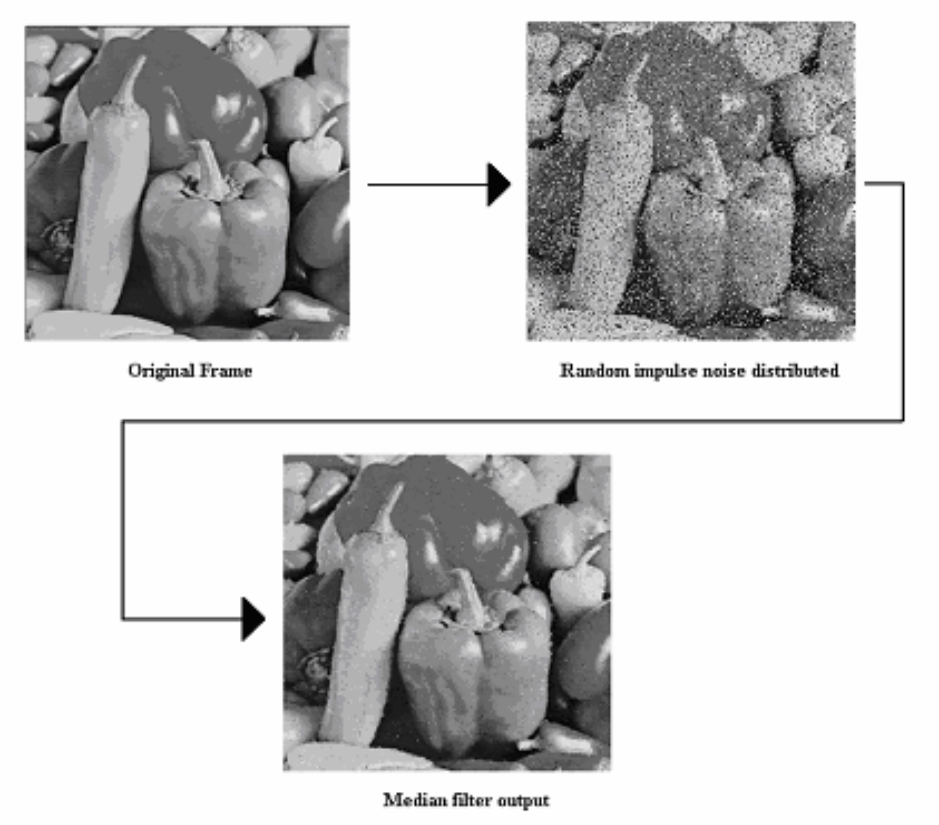

Figure 5.1: Median filter process

In this example original image is corrupted with 25% impulse noise and then filtered by switching median filtering technique [23].

In this thesis, basic median filtering techniques are examined and an improved fast median filtering algorithm is proposed.

5.1.1 Literature Review

Although morphological filters are very successful for filtering monochromatic images, their applications for vector-valued images impose difficulties and problems because of necessity for value-ordering step while implementing them [22]. Each pixel has only one chromatic value generally lying between 0 and 255 for monochromatic images, which gives the ease of sorting operations within pre-defined range. Unfortunately, vector-valued pixels are not easy to be sorted. This is because algorithm developer can not directly sort images over sets of Mathematical Morphology operators for multi-variate images because of angle and module information for each pixel and there exist requirement of high computational techniques involving correlation consideration between each vector components [22].

The main challenging problem in median filtering for each pixel having RGB information is to find a way of fast sorting algorithm for real time image processing systems. Moreover, it is not very surprising to wait for unsorted input data because there exists no ”natural” and unambiguous order in the data before processing [24]. There exist mainly three popular median filtering methods, depending on sorting algorithms [24]. Considering given $\{x_1, x_2, x_3, \ldots x_N\}$ be a set of $N=s^2$ vectors within the sliding window W, the considered median filter types are:

1. Vector Median Filter: This algorithm basically depends on distance between each vector. Output of filter is the vector that has smallest total distance to any other vector inside window.

$$D_i = \sum_{j=1}^{N} ||x_i - x_j||_L$$

$$VMF(W) = x_M$$

$$D_M = min_{i=1..N} D_i$$

2. Basic Directional Filter: This algorithm depends basically on the angle measurement between each vector in the considered window W. Output of filter is the vector that has smallest total angle between any other vector inside window.

$$\alpha_i = \sum_{j=1}^{N} A(x_i, x_j)$$

$$BVDF(W) = x_M$$

$$\alpha_M = min_{i=1..N} \alpha_i$$

3. Directional Distance Filter: This algorithm depends basically on combination of both directional filter and vector median filter techniques, i.e., angle between vectors and distance together. Output of filter is the vector that has smallest total angle and distance to any other vector inside window.

$$\Omega_i = {D_i}^{1-\omega}.\alpha_i^{\omega}$$

$$\Omega_i = \left(\sum_{j=1}^{N} ||x_i - x_j||_L\right)^{1-\omega} . \left(\sum_{j=1}^{N} A(x_i, x_j)\right)^{\omega}$$

$$DDF(W) = x_M$$

$$\Omega_M = min_{i=1..N}\Omega_i$$

Any of these methods can easily guarantee correct and robust sorting operation. Unfortunately, it is not possible to apply any of these algorithms for real time imaging systems unless parallel processing is available. This is because of calculation complexity especially for angle detection, which involves arccosine function, square root operations, multiplications and additions within each window. It is suggested to use a look up table for arccosine function; however square root operations should be performed, because it is not always possible for a system to waste a memory space in ranges of mega bytes.

In figure-5.2, V_1 and V_2 are presented in 2D.

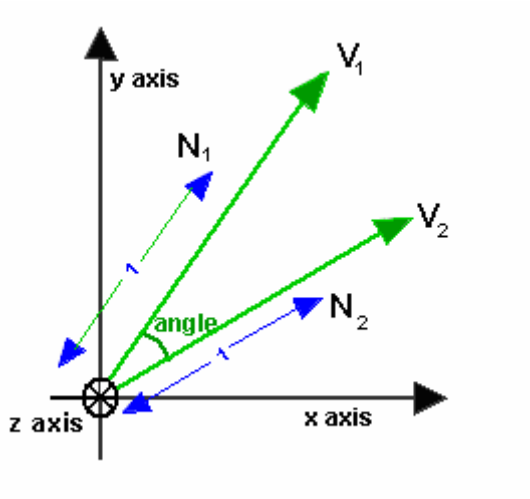

Figure 5.2: Representation of two vectors in 2D

$$\text{angle} = \arccos\{ (V_1 \bullet V_2) / (\| V_1 \| * \| V_2 \|) \}$$

Gerasimos et al. proposed an adaptive circuitry that detects the existence of impulse noise in an window and applies the median filter to that potion of samples when necessary [25]. This algorithm prevents blurring of the image while processing and provides the algorithm developer with integrity of edge and detail information preservation [25]. In order to minimize computational time, proposed digital hardware structure is performed with fully pipeline-architecture with parallel processing which was implemented in FPGA [25]. Structure of the proposed filter is shown in figure-5.3 [25].

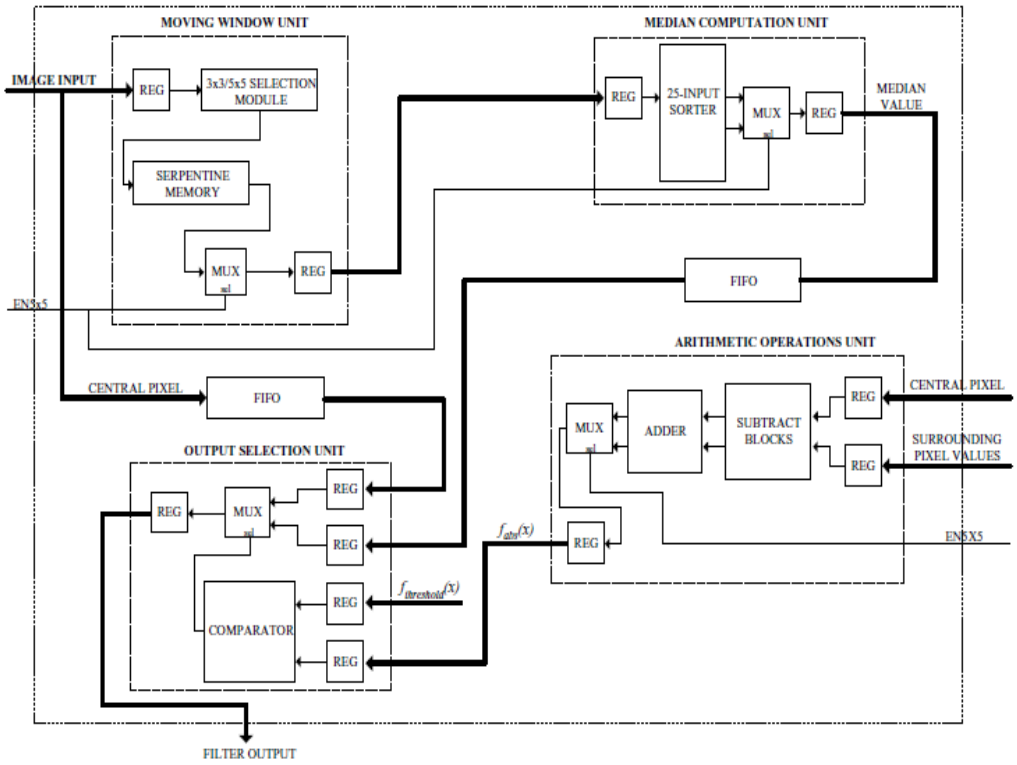

Figure 5.3: Parallel processing Median filter structure in FPGA

Obviously such a system requires definitely parallel processing in order not to grab CPU for a long time while firstly doing region detection where impulse noise appears and secondly applying median filter. The proposed method also requires care about synchronization for real time processing. This is due to reason that amount of impulse noise is not fixed for all image frames, which would result in different time periods for each frame.

Another efficient method suggested by E. Stringa, A.Teschioni, C.S.Regazzoni is to perform a transformation before sort operation, based on the concept of space filling curves from multi-valued sets into scalar sets [22]. A pre-processed transformation from vector pixels into scalars give opportunity to algorithm developer in order to preserve module information,

which is represented by an appropriate norm, which additionally take into consideration of angle information [22]. The transformation is based on space filling curves an example of which is represented in figure-5.4 [22].

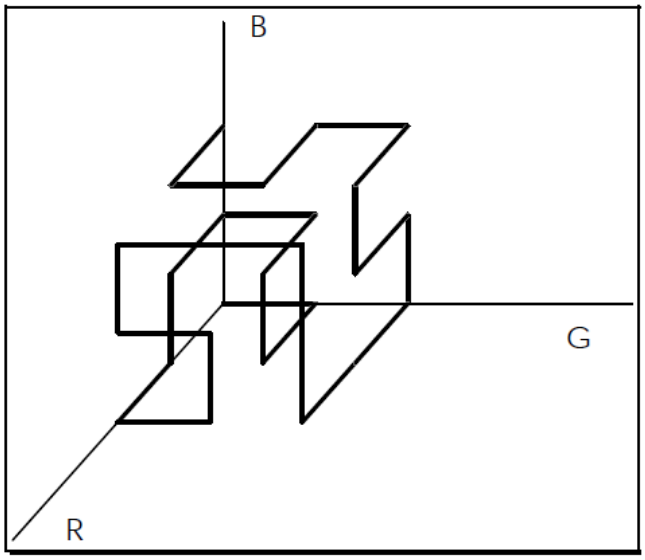

Figure 5.4: Transformation of RGB into space filling curves

This method consists of two steps. In the first step, a vector to scalar transformation is performed. After this operation, morphological operation such as erosion, dilation or median filtering can be performed as if the transformed image is a single valued gray-level image. The problem is to define a function that performs one-to-one map from Z to Z^N after morphological operation is performed, in order to return vectorial domain. This method is summarized in figure-5.5

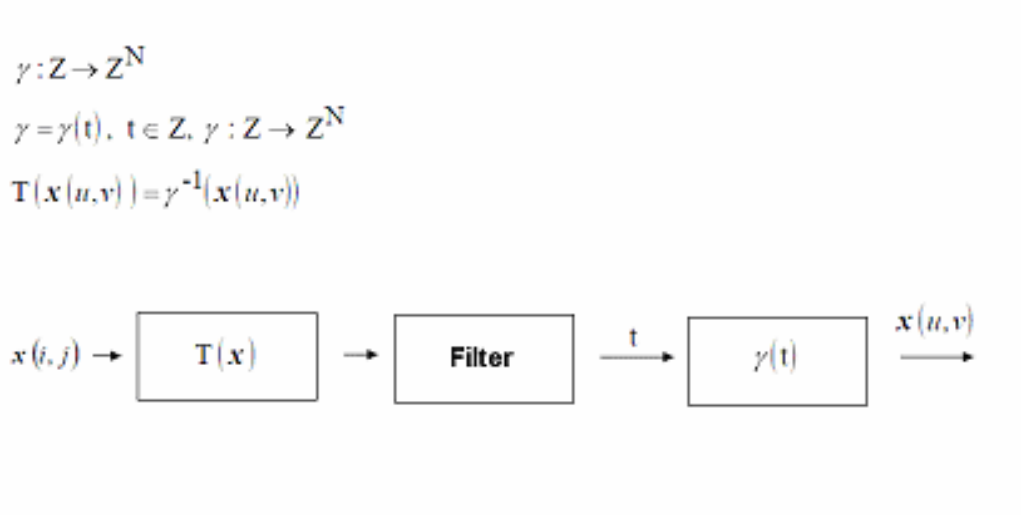

Figure 5.5: Filtering based on transformation from Z^N to Z

Note that given pre-transformation, filter and inverse transformation can be considered as nonlinear extra operations, causing additional sorting operations, and controlling nearby pixels. Moreover additional memory space is required in order to form a look-up table for forward and backward transformation. Although this algorithm has negligible drawbacks, it can still be applied because better solution can be implemented with scalar sort operations rather than vectorial computations.

5.1.2 Hilbert's Space-Filling Curves

In this thesis, median filtering is applied for frame enhancement after a transformation from vectorial-space to scalar numbers. Modified version of Hilbert space filling curves is applied before sort operations.

Hilbert's space-filling curves start from first order filling curves; the order increases as vector dimensions and vector scale increases. Figure-5.6 shows first, second and third order 2D SFC (Space-filling curvers), respectively [26].

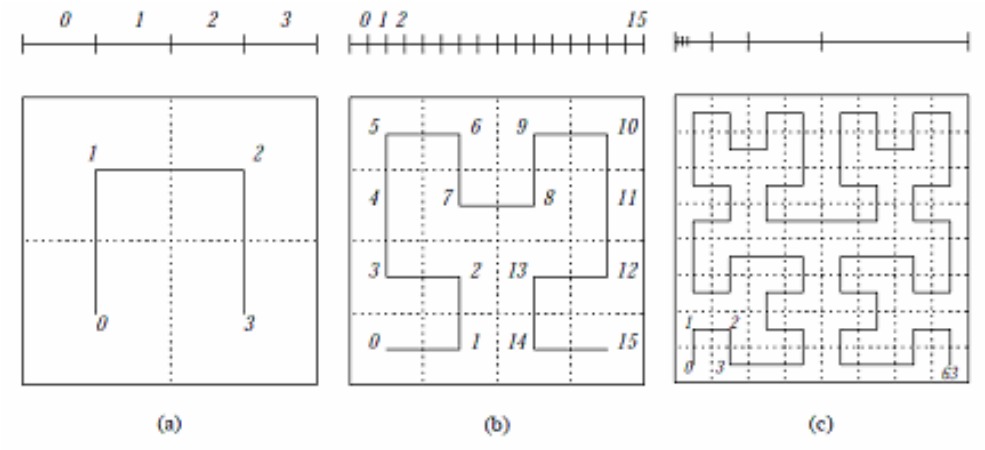

Figure 5.6: First, second and third order 2D SFC

Figure-5.6.a shows a transformation of <0,0> → 0, <0,1> → 1, <1,0> → 2, <1,1> → 3. Similarly, figure-5.6.b shows a transformation of <0,0,0> → 0, <0,0,1> → 1 …….. <1,1,1> → 15.

This process can be continued to infinity, which guarantees a unique infinite sequence of nested squares which for which there exist unique representation [26].

In practice for many applications, algorithm developers need higher order curves which imposes requirement of some algorithmic procedures for the mapping which, given coordinates, produces the ordinal number and vice versa [26].

In chapter-6, transformation from vectorial space to scalar space is explained for image formation algorithm. This algorithm is required not only for display utilities of OMAP3530 processor but also for utilization of a transformation from 3D vector-space to scalar-space. After the image

formation for display units of processor, modified median filtering algorithm is implemented.

5.1.3 Proposed Median Filter

In this thesis, considering frame rate requirements for real-time video system, basic median filtering techniques are reformed in the sense of windowing shape and sorting algorithm.

i) Sorting Algorithm

Sorting algorithms require basic additions and subtractions in N^2 levels for N numbers. This in turn brings about loss of time and slower frame rates. Rather than using a sort operation for each pixel inside a window, checking difference between each pixel and the pixel sitting in the middle of window is considered to be more sufficient which requires subtractions in N-1 levels for N numbers.

A new concept on median filtering is proposed in this thesis. This new algorithm will basically depend on signing pixels which are considered to belong to noisy part rather than input signal. In this basic algorithm, similar to median filtering; a non-linear operation on image is performed, a 3 sample window is used.

Initially a threshold value is chosen. If the center pixel inside the window is greater or less than other pixels by threshold value, it is replaced by the left pixel. An example is given below:

x = [2 80 6 5 3 5 90 6]
y[1] = improved_med [2 2 80] = 2 // no signed pixel

y[2] = improved_med [2 80 6] = 2 // 80 is signed and replaced by left pixel

y[3] = improved_med [2 6 3] = 6 // no signed pixel

y[4] = improved_med [6 5 3] = 5 // no signed pixel

y[5] = improved_med [5 3 5] = 3 // no signed pixel

y[6] = improved_med [3 5 90] = 5 // no signed pixel

y[7] = improved_med [5 90 6] = 5 // 90 is signed and replaced by left pixel

y[6] = improved_med [5 6 6] = 6 // no signed pixel

Finally, output of this non-linear filter is given to be:

y = [2 2 6 5 3 5 5 6]

Obviously elements 80 and 90 are far away from the sequence that are signed as impulse noise and these elements are removed with the element sitting on the left of noise element. For smoothness, averaging left and right pixel can be considered which would result in an output of:

y = [2, 4, 6, 5, 3, 5, 5.5, 6]

In figure-5.7, removal of impulse noise from input sequence is represented.

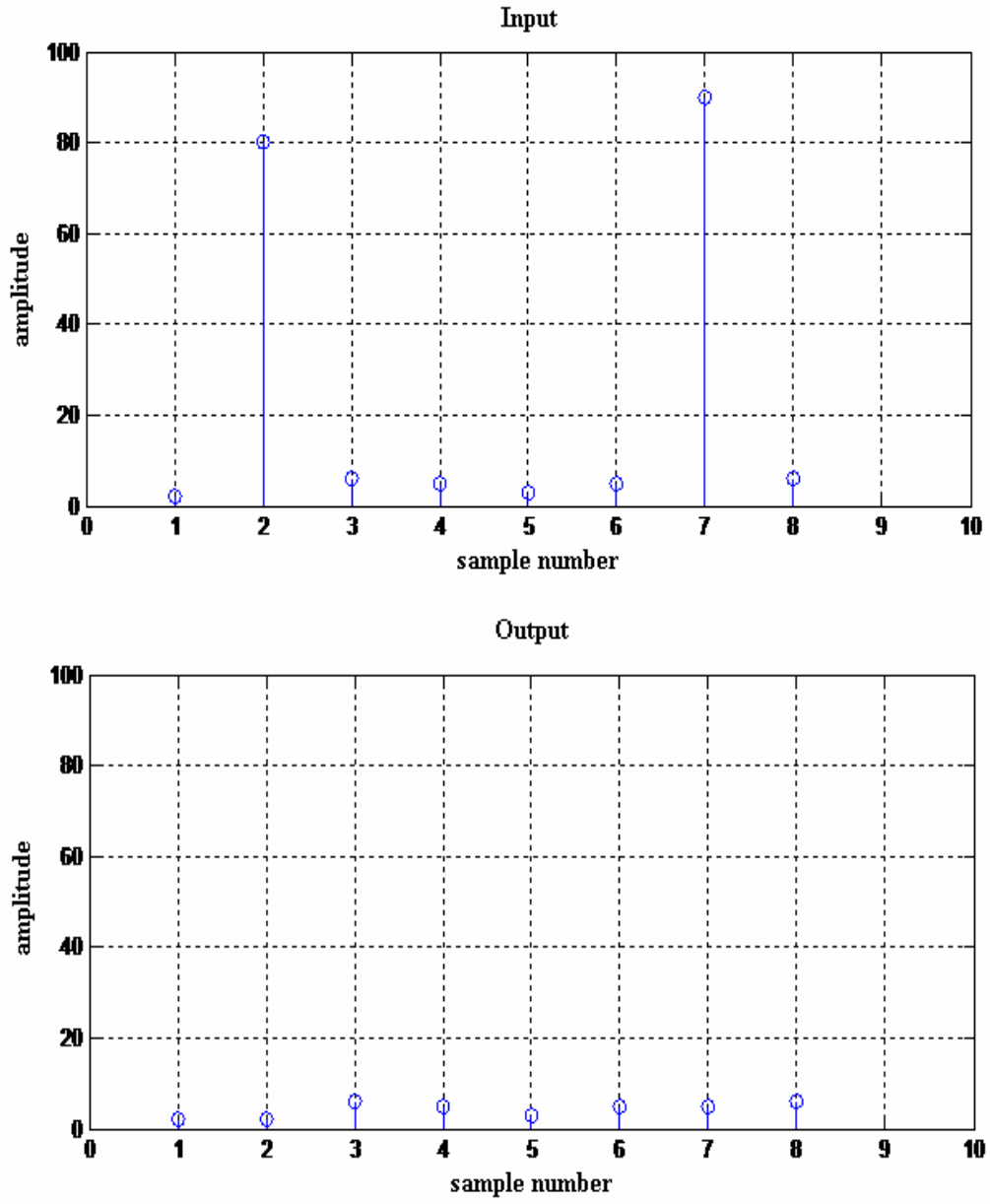

Figure 5.7: Impulse noise removal from input sequence

ii) Window Shape

Rather than using an NxN window, in order to accelerate algorithm basic 3x3 window shape is modified as shown in figure-5.8.

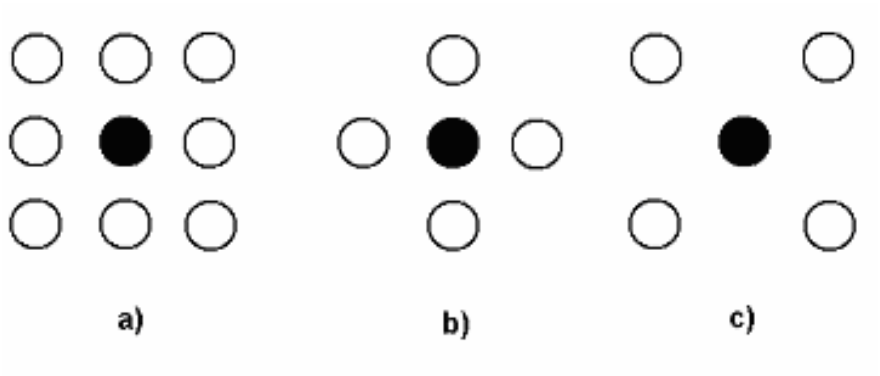

Figure 5.8: Modified window shape examples

In figure-5.8.a, basic 4x3 median filter window is depicted. This window is firstly modified by removing pixels on the corners so that speed of modified sorting algorithm is doubled in the sense of noise check, as shown in figure-5.8.b. This modified window type is not good for images that involve cross-lines.

Figure-5.8.c depicts second modified median window which is not suggested for the images that involve straight-lines.

In figure-5.9, using median filter with window size of 3x3, impulse noise is eliminated. Edges are smoothed and details of image are not clear after applying given median filter.

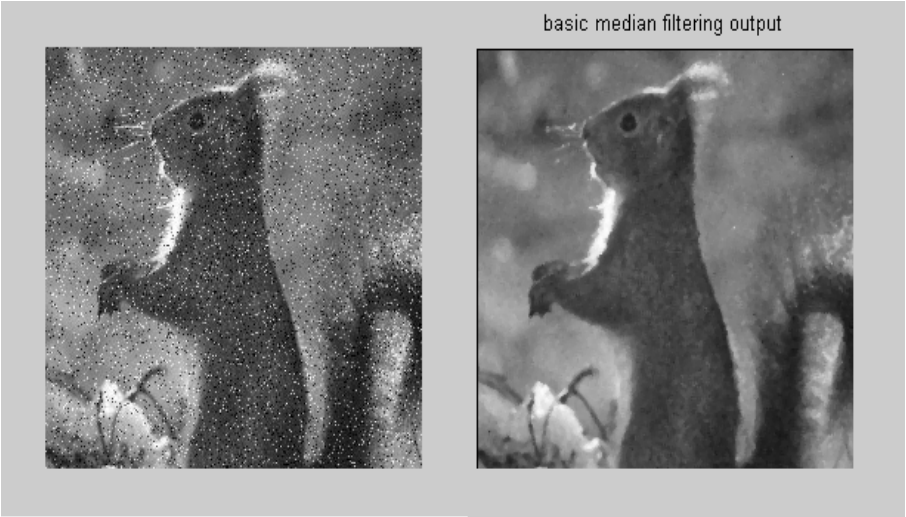

Figure 5.9: 3x3 window shape median filter output

In figure-5.10, using suggested window shape, with window size of 3, impulse noise is eliminated by classical median filtering technique. Edges are not smoothed and details of image are much clearer after applying given median filter.

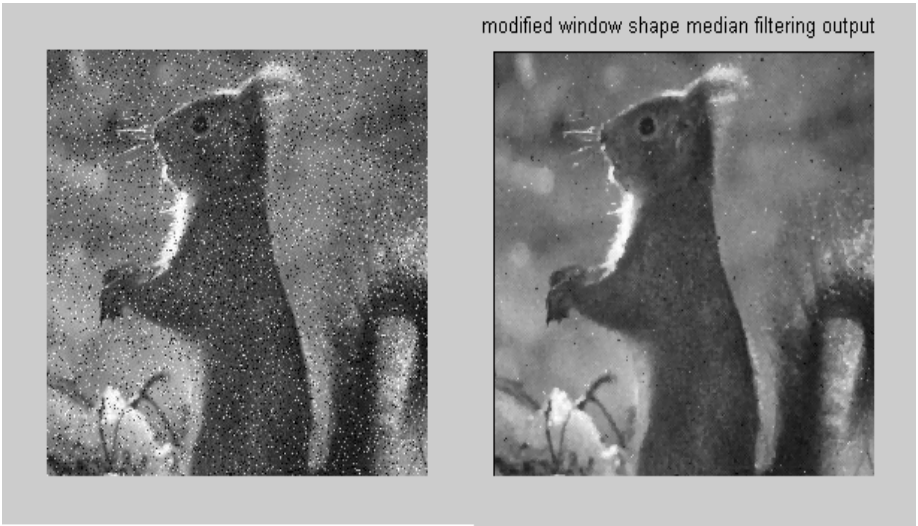

Figure 5.10: Modified window shape median filter output

In figure-5.11, using suggested window shape and modified sorting algorithm, with window size of 3, impulse noise is eliminate. Edges are not smoothed.

Figure 5.11: Modified median filter output

As inferred from figure-5.9, figure-5.10 and figure-5.11, best solution in impulse noise removal is to use basic median filtering. However, considering processing time for both of suggested modified median filtering techniques, it was observed that these two methods are much faster than 3x3 window size median filter. Moreover, edges are not smoothed.

SNR (signal-to-noise ratio) is a measure for quality of reconstructed image at the end of algorithm, which depends on a basic idea of computing a single number that reflects output signal quality [30]. Although this traditional concept tells more on output quality, it is not directly equal to human eye subjective perception [30]. It is better to use PSNR (peak signal-to-noise ratio) in db for measuring quality of constructed images.

Given A(i,j) as the NxN source image and a(i,j) as the NxN output image, firstly MSE (mean squared error) is calculated as:

$$MSE = \frac{\sum \left[a(i,j) - A(i,j) \right]^2}{NxN}$$

Finally, calculating RMSE as square root of MSE, PSNR is given to be:

$$PSNR = 20 \log \left(\frac{255}{RMSE} \right)$$

PSNR for basic median filter is 72,377 db, for modified window shaped median filter is 66,850 db and for modified sorting algorithm is 53,007 db.

5.2 Post Processing

After application of rendering algorithm for a fixed virtual camera position, it is observed that there occur region dehiscence on the objects that are near to base camera. Such opening on the objects gives rise to visibility of background through the object that is near to base camera. Moreover, once virtual camera position gets far away from base camera, such openings and as a result deformations on images increase.

This problem occurs due to insufficient pixel amount that represent parts of objects on image relative to camera position and angle, in other words, points, which are overlapped by neighboring points in real world according to base camera position, are the main reason of pixel deficiency. Considering

objects on a scene, according to camera position and relative orientation of objects with real camera, it is possible to observe that amount of pixels are increased as expected, which is depicted in figure-5.12 and figure-5.13.

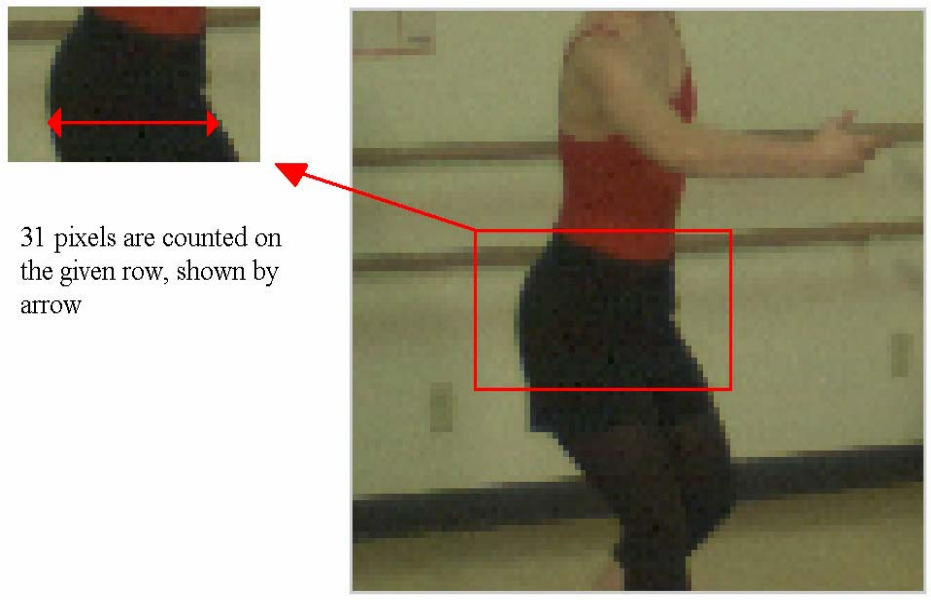

Figure 5.12: Pixel count in original frame from camera 4

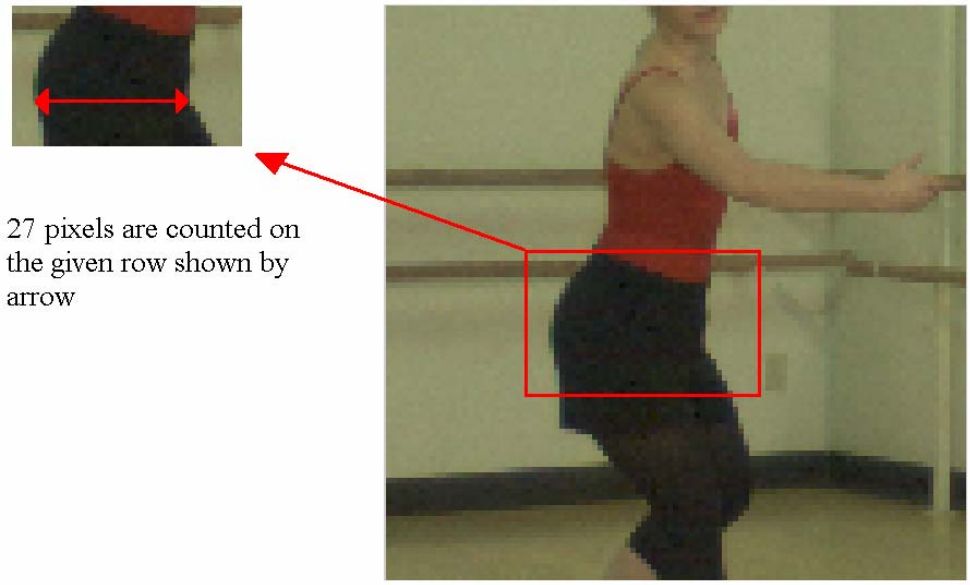

Figure 5.13: Pixel count in original frame from camera 6

As seen in figure-5.12 and figure-5.13, camera 6 object pixel number is less than the one for camera 4 for which object view angle is closer to right angle. When these pixels are rendered with image-based rendering algorithm, there occur holes between neighboring pixels which are filled by background pixels as expected.

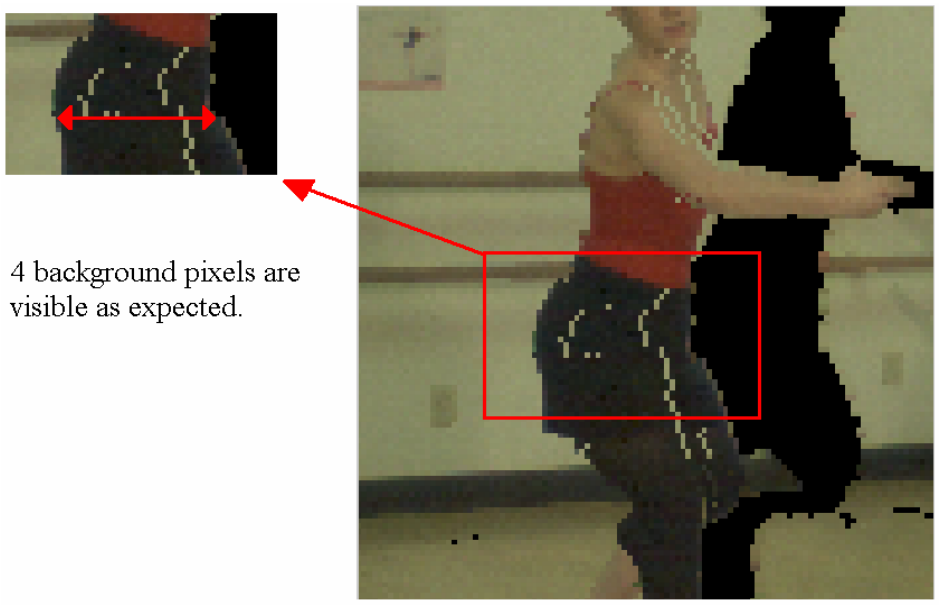

Figure 5.14: Background pixel observation through dehiscence region

As observed from figure-5.14, there exists dehiscence on ballet object in the scene. Through those regions, it is possible to see spread of background pixels which become more disturbing on video sequences once those frames are played on a screen.

Object dehiscence problem after warping algorithm results in spread of background pixels among nearby object image because of change in camera view angle with respect to object. This problem is observed more evidently for virtual view points far away from base camera. In this section of frame enhancement chapter, a new robust algorithm for object dehiscence problem is proposed.

5.2.1 Proposed Algorithm Outline

In this algorithm, the aim is detecting regions on background that are possibly overlapped with object image patches after image-based rendering algorithm. After detecting these regions, they will either be ignored in order not to give rise for spread of background pixels on object image patches, or their rendered regions will be deleted and re-rendering is done only for nearby object.

For a given virtual camera position, that is on the right of base camera and determined by user, considering camera-6 input frame as the base camera input and camera-4 input frame to be auxiliary frame for occlusion region filling, this algorithm can be outlined in 4 steps as follows:

1. Apply image-based warping algorithm for input frame of camera-6 as described in chapter-3.
2. Determine start point of regions on the background that will be overlapped by objects near to the base camera, using camera-6 input frame and edge image of its depth map.
3. If pixel scan is from left to right, delete all warped pixels from overlapped background pixels, else if pixel scan is from right to left, jump to the starting pixel of overlapping region.
4. Continue image-based warping algorithm.

5.2.1.1 Overlapped Background Region Determination

Overlapped background region is mainly determined using edge image of depth map for base camera frame in +x direction and input frame.

Considering camera 6 frame 0 depth map, scanning column by column from left to right, it is possible to understand starting and finishing pixel positions for background regions that are possibly going to be overlapped with an object image that is near to the base camera.

Edge image for camera input frame 0 depth map is given in figure-5.15. This edge image is obtained simply by detecting positions of discontinuities from left to right.

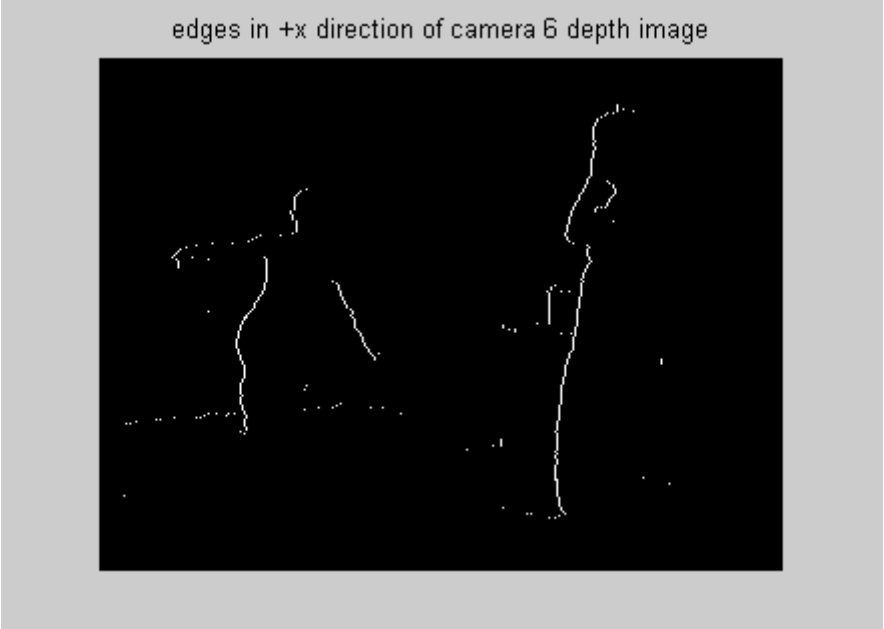

Figure 5.15: Edge image in +x direction of camera 6 depth map

Considering about abruptly drawn top view of 3D scene in figure-5.16, it is clear that R1 region is visible for real base camera center C6 and virtual camera center C5. Left of the object is visible for base camera until Pf. Although R2 region is visible to camera 6, it is going to be overlapped by the

object in front of virtual camera center C5 after application of warping algorithm.

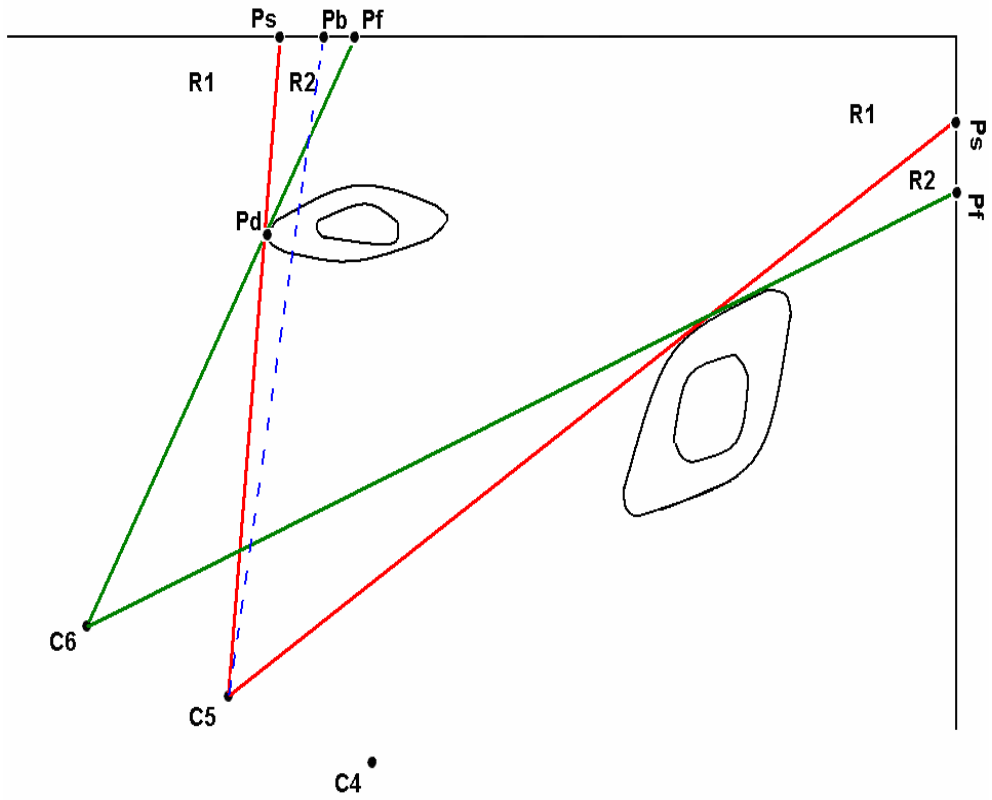

Figure 5.16: Abruptly drawn top view of ballet scene

Because the number of pixels belonging to object is limited with respect to base camera in comparison with virtual camera position, holes between neighboring pixels are going to be filled by background pixel, e.g. Pb is seen by C5 although it should be overlapped by object. This is the reason why algorithm developer should mark background pixels that should not be rendered in order to prevent object dehiscence.

i) Pixel Scan from Left to Right

If rendering algorithm is applied by scanning rows from left to right, it is not possible to determine exact position of Ps in advance. Rendering algorithm is applied and discontinuities on depth map with an initially determined threshold value are checked simultaneously. At the instant of discontinuity detection, which refers to Pd position, rendered pixel for Pd is determined on the output which maps exactly to the same point of Ps. Specific output position belonging to Ps is marked. Similarly pixel before Pd which is exactly Pf is rechecked and its rendered position on output image is marked. Finally, output row between two marked positions is deleted.

ii) Pixel Scan from Right to Left

If rendering algorithm is applied by scanning rows from right to left, it is possible to determine exact position of Ps in advance. Rendering algorithm is applied and discontinuities on depth map with an initially determined threshold value are checked simultaneously. At the instant of discontinuity detection, which refers to Pd position, rendered pixel for Pd is determined on the output. Starting from this point, algorithm differs with the one for pixel scan from left to right.

Continuing rendering algorithm with pixel Pf, it is clear that rendered pixel position will have greater +x value with respect to Pd, therefore pixels outputs are ignored until rendered output position has smaller +x value with respect to Pd, which is the next pixel on the left of Ps.

In this thesis, pixel scan form left to right algorithm is applied. In figure-5.17, ignored background pixels are clearly shown at the output of rendering algorithm.

Figure 5.17: Background purification with the suggested algorithm

A closer look for given image is shown in figure-5.18.

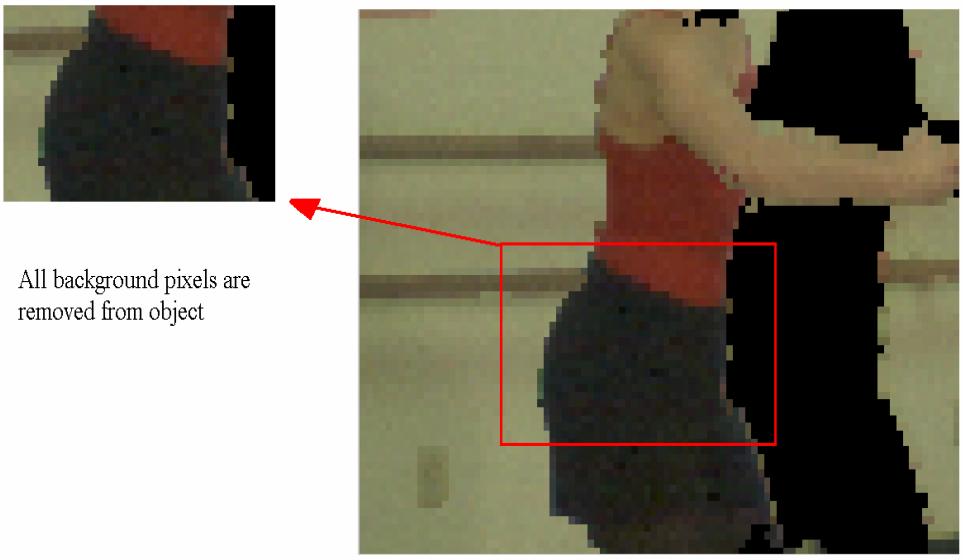

Figure 5.18: Closer look to purified background pixels on object

CHAPTER 6

SYSTEM DESCRIPTION

In this chapter, detailed information on system, its components and hardware/software architecture is given. Main board, its algorithmic flow charts, daughter board and algorithmic flow charts are introduced. Finally performance analysis of suggested algorithms on OMAP cores is given.

6.1 System Architecture

In this thesis, OMAP3530 Evaluation Module Board (EVM) is used as the main processor board for realization of algorithm and exploring capabilities of double core processor OMAP3530. An additional daughter card is designed for control of view angle by GYRO sensor. Data flow between main board and daughter card is done through board-to-board connectors provided by EVM board. In figure-6.1, system hardware setup is shown.

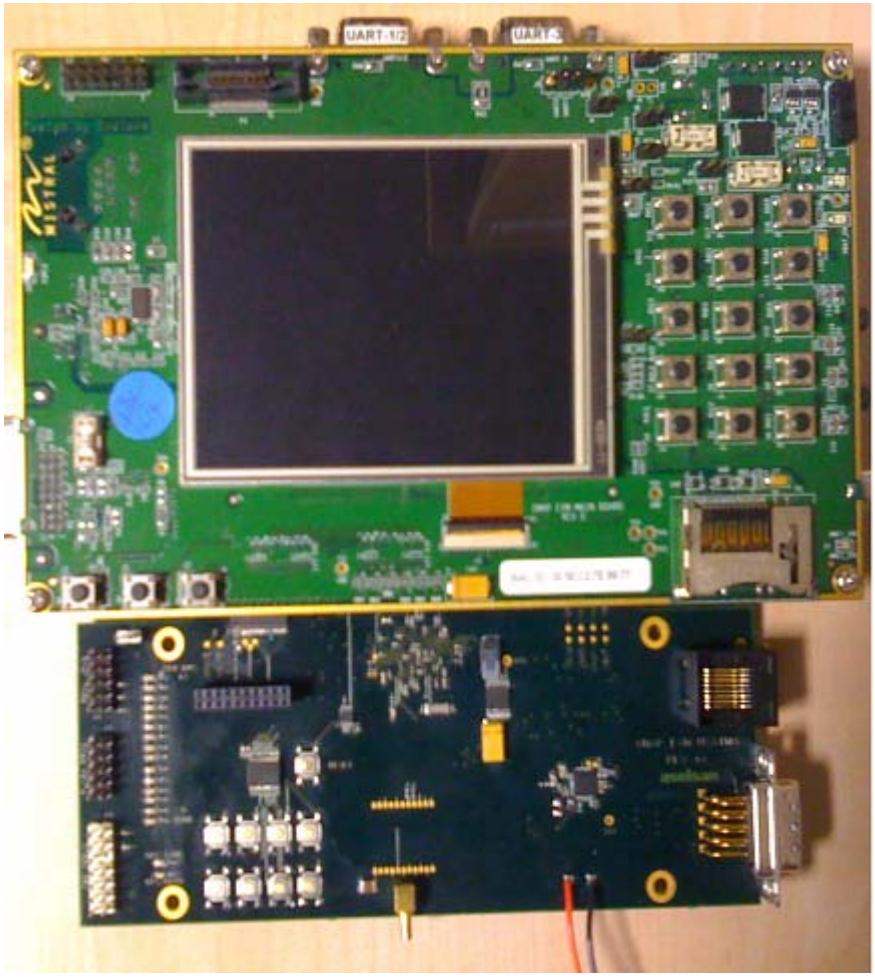

Figure 6.1: System hardware setup

A computer is used for monitoring, controlling embedded operating system and starting applications. Linux (Kernel-2.6.22) is used as the embedded operating system and applications are developed by OMAP35x_SDK_1.0.2 (arm tool-chain) together with DVSDK_3_00_00_29 provided by Texas Instruments. Operating system is Linux UBUNTU.

6.1 Hardware Architecture

In this thesis, one main board of OMAP35x Evaluation Module and two designed daughter cards are used. EVM board is used for realization of algorithm and used as main processing unit. First daughter card is designed for view point calculation in order to interrupt main board and transmit view point once user changes it either by touch-screen, keyboard or gyro sensor. Second daughter card consists of gyro sensor independent from first daughter card.

In this thesis, single Z-axis (yaw rate) response gyro sensor is used. In order to sense angular movement of system in correct direction, this card is designed independently from first daughter card and kept perpendicular to EVM board and first daughter card. If 3-axis response gyro sensor was used, this perpendicular position requirement would no longer be necessary.

In this chapter, detailed information on system boards is given.

6.1.1 EVM Board

This board is developed by Mistral Solutions for exploring and developing both DSP and ARM applications on OMAP35x processors. Processor on the board is monitored and controlled by Hyper Terminal. In figure-6.2, EVM functional block diagram is given [4].

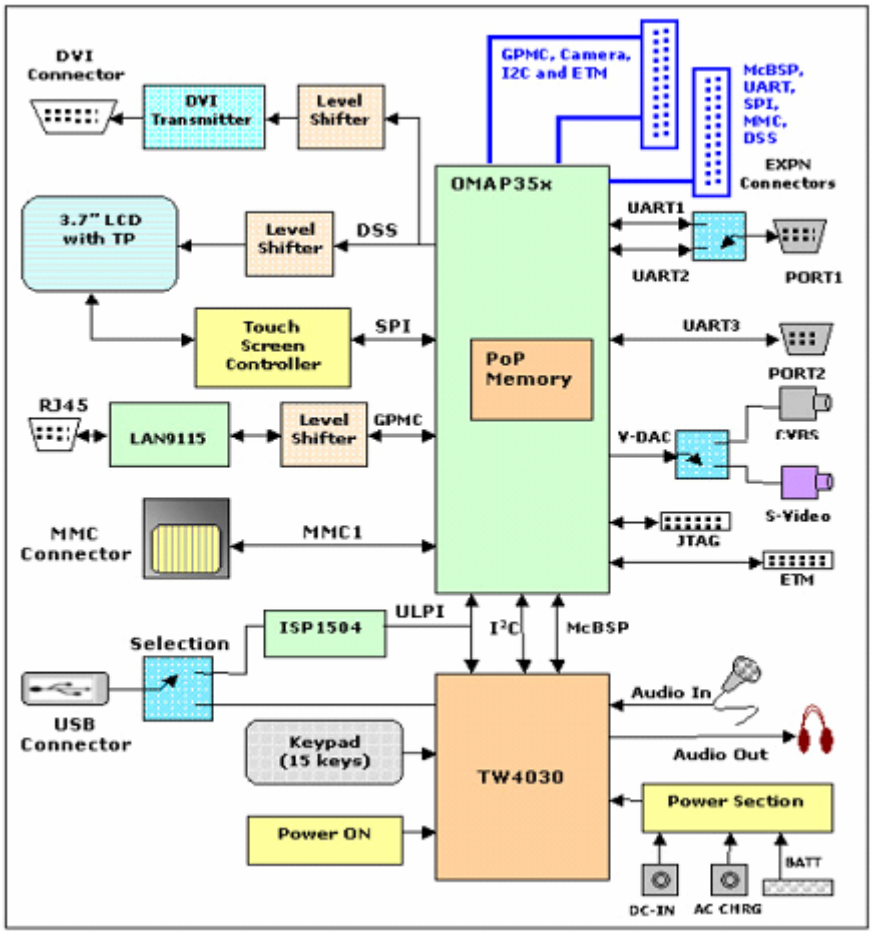

Figure 6.2: EVM functional block diagram

In figure-6.3, EVM board top view is shown [4].

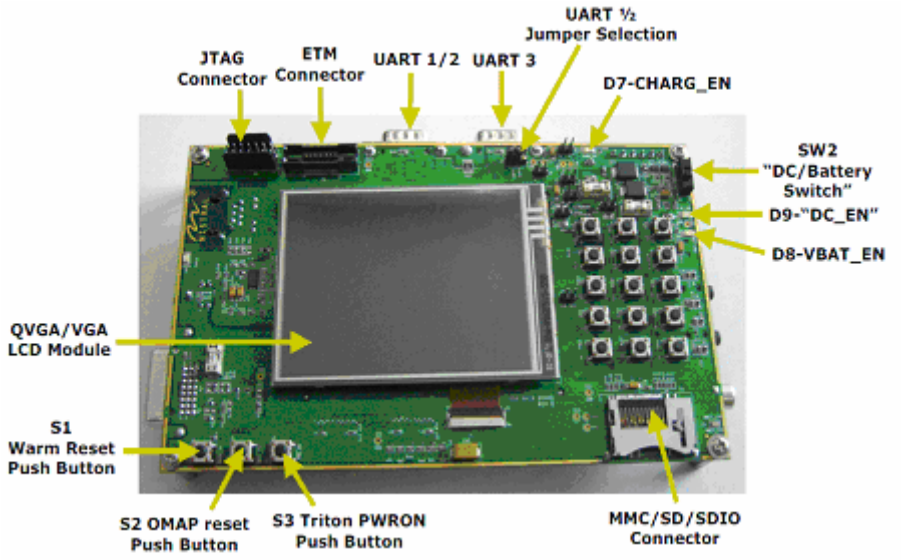

Figure 6.3: EVM board top view

6.1.2 Daughter Cards (EVM Expansion Board)

In this chapter two daughter cards are explained in details. For simplicity, both of daughter cards are interpreted together in total.

This board is developed and designed for exploring GYRO sensors, FPGA (SPARTAN family with DSP slices) and developing applications together with OMAP3530 processor. In figure-6.4, functional block diagram of this card is given.

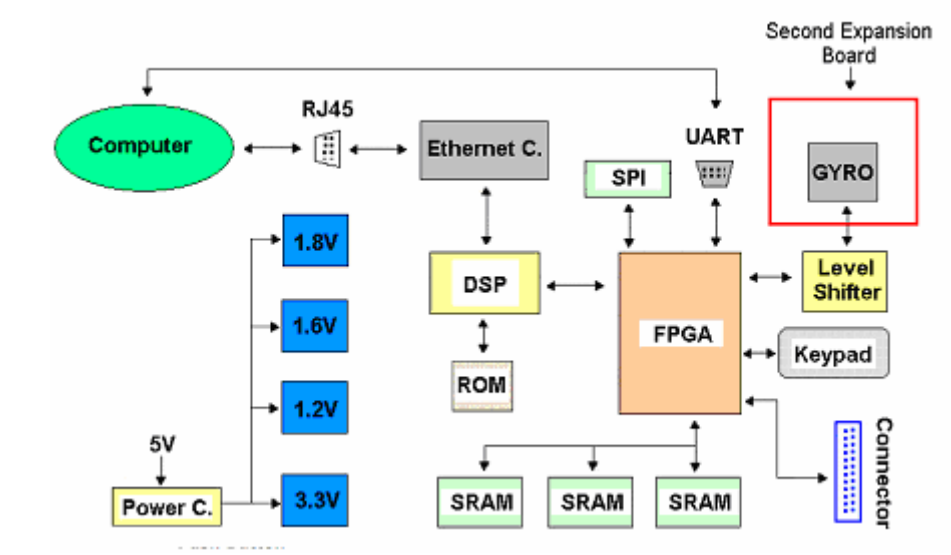

Figure 6.4: Functional block diagram of daughter cards

Figure-6.5, shows front view of main daughter card.

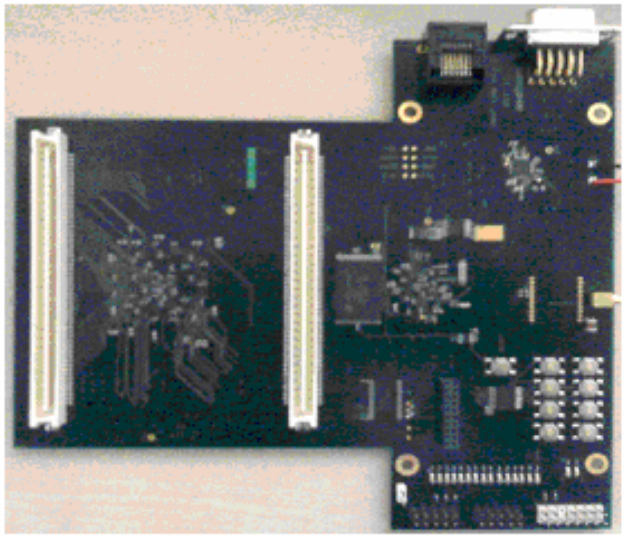

Figure 6.5: Front view of main daughter card

Figure-6.6, shows back view of main daughter card.

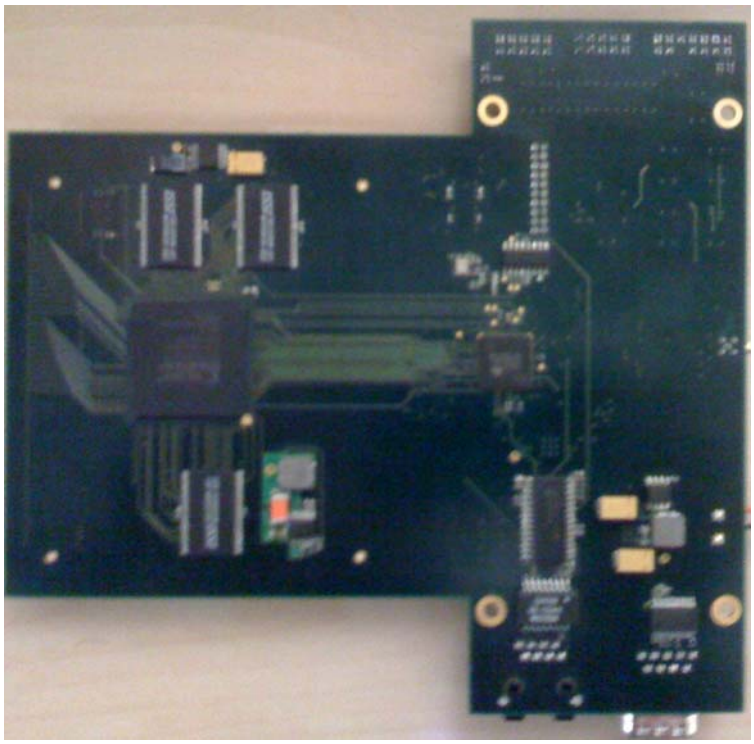

Figure 6.6: Back view of main daughter card

Figure-6.7, shows front view of gyro daughter card.

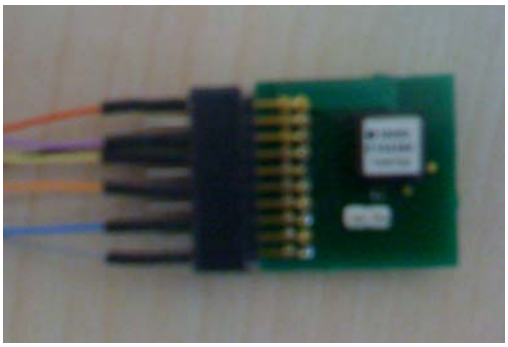

Figure 6.7: Front view of gyro card

6.1.2.1 FPGA

SPARTAN-3A family FPGA having 84 Extreme DSP slices is used. These DSP slices perform multiply and accumulate (MAC) operations for camera position detection algorithm according to GYRO sensor input. This IC communicates through 120 pin main board-to-board connector with OMAP3530 once camera position changes and provides new position information for main warping algorithm. System mode selection is done according to keypad input. It performs communication with computer for system monitoring by Hyper Terminal for debug operations.

This IC has 4 blocks of I/O pins, voltage level of which are controlled by voltage inputs of I/O blocks. OMAP communication block is driven with 1.8V while other three blocks are driven by 3.3V.

6.1.2.2 DSP

In this design, TMS320VC5509A Fixed-Point is used. This IC is used as the communication processor of daughter card. It is used mainly for necessary debugging operations and performs input / output operations between computer and daughter card.

6.1.2.3 GYRO

This sensor is a complete angular rate digital gyroscope with 14-bit resolution from ANALOG devices. Through SPI port, this IC provides access to the rate sensor, an internal temperature sensor, and two external analog signals (using internal ADC). The digital data available at the SPI port is proportional to the angular rate about the axis that is normal to the top surface of the package. Therefore second daughter card is designed for this

IC in order to keep this IC perpendicular to main board. This IC operates on the principle of a resonator gyroscope. There are two poly-silicon sensing structures containing a dither frame that is electro statically driven to resonance. This generates the necessary velocity element to produce a coriolis force while rotating. At both of the outer extremes of each frame, orthogonal to the dither motion, are movable fingers that are placed between fixed pickoff fingers to form a capacitive pickoff structure that senses coriolis motion. Figure-6.8 shows rate sensitive axis.

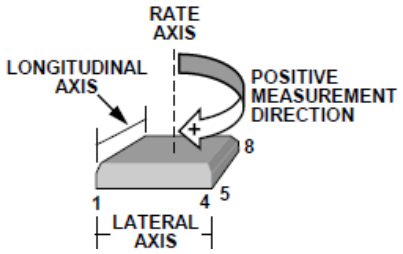

Figure 6.8: Rate sensitive axis of gyro sensor

6.1.2.4 SRAMs

There are 3 1Mb SRAMs on daughter card for possible image storage and debug operations.

6.1.2.5 Boot Sector Flash ROM

This parallel interface memory IC holds boot code of DSP.

6.1.2.6 Serial Flash Memory

This serial interface memory IC holds boot code of FPGA.

6.2 SOFTWARE ARCHITECTURE

This chapter gives detailed information on block algorithms that are used for realization of the thesis.

FPGAs are generally not present inside hardware block of a hand-held device. This is the reason why application software runs mostly on OMAP3530 processor. Software architecture of the system is mainly consisted of 5 algorithm blocks namely:

1. Image formation → Home PC
2. Intermediate view reconstruction → OMAP
3. Occlusion Handling → OMAP
4. Frame Enhancement
    i. Median filtering → OMAP
    ii. Object Dehiscence Removal → OMAP
5. View point determination
    i. Gyro → FPGA
    ii. Key-pad → OMAP
    iii. Touch-screen → OMAP

6.2.1 Image Formation Algorithm

EVM board Linux provides with the following features [5]:

- LCD display interface at VGA resolution (480*640)
- TV display interface at NTSC resolutions on Video Pipelines (only S-Video out is supported, composite out is not supported)
- DVI digital interface at 720P and 480P resolution.
- Graphics pipeline and two video pipelines. Graphics pipeline is supported through fb-dev and video pipelines through V4L2

- Supported color formats: On OSD (Graphics pipeline): RGB565, RGB444, and RGB888. On Video pipelines: YUV422 interleaved, RGB565, RGB888, RGB565X
- Configuration of parameters such as height and width of display screen, bits-per-pixel etc.
- Destination and source color-keying on Video pipelines through V4L2
- Setting up of OSD and Video pipeline destinations (TV or LCD) through sysfs
- Buffer management through memory mapped and user pointer buffer exchange for application usage (memory mapped)
- Rotation - 0, 90, 180 and 270 degrees on LCD and TV output
- Mirroring (except for RGB888)
- LCD backlight control through sysfs interface
- ARGB pixel format on Video2 pipeline and RGBA format on graphics pipeline and global alpha blending

Such features provide user with various applications on image processing, different display opportunities and memory saving.

There exist many standards for RGB (red green blue) pixel data scaling from RGB332 to RGB88 of which numbers for standard correspond to the number of bits for each color components of each pixel [6]. These numbers are aligned respectively, i.e., considering RGB565; it is a 16-bit pixel standard having 5 bits for red, 6 bits for green and 5 bits for blue.

Increment in the number of bits per pixel gives rise to increase in color depth and levels of blending exponentially, therefore greater number of bits per pixel increases the color depth and quality, however, this will in turn results in a need of greater amount of space in the memory for storage of each frame in a video stream [6]. Moreover, considering the fact that true color can also

be represented after blending primary colors from each pixel as represented in figure-6.9, it is sensible to convert RGB888 pixel format to RGB565 [6].

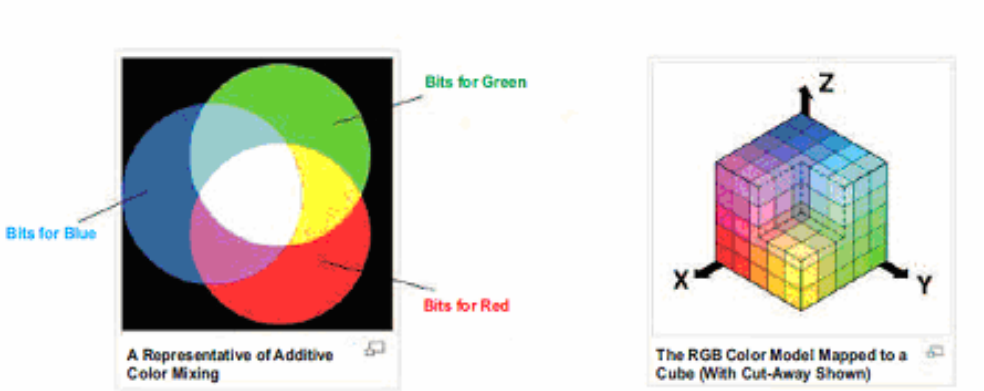

Figure 6.9: Mapping true color to RGB565 from RGB888

In this thesis RGB888 24 bit bitmap files are firstly converted to RGB565 16 bit file format, which is default configuration of display driver. Aim is to save memory and consider each pixel by only one “short” variable, which is due to algorithms used for frame enhancement. In chapter-5, this requirement is explained in details.

6.2.2 Intermediate View Reconstruction

This part of software constitutes main step in realization of user-defined view point video display algorithm. This algorithm is implemented on OMAP3530 microprocessor. Intermediate views according to a user defined view points are constructed using base camera input frames and their corresponding depth maps.

Using same algorithm, another intermediate view is constructed using second camera input frames and their corresponding depth maps. This is due to fill occlusion regions that are occurred at regions visible to virtual camera position but invisible to base camera.

In order to achieve a real time operating display system, this algorithm is optimized. Detailed information on optimization tips is given in chapter-3.

6.2.3 Occlusion Handling

Occlusion handling algorithm is the step applied after intermediate view reconstruction algorithm on second camera input frames.

Although some regions are visible for base camera view point, they can be invisible for virtual camera. Intermediate view reconstruction algorithm handles this problem, because invisible regions are overlapped by objects that are nearer to virtual camera position. However, once a region on background is invisible for base camera but visible for virtual camera position, occlusion handling algorithm is applied on second camera input frames in order to grab additional information of such kind of regions on image.

Occlusion handling algorithms are applied when needed. Application of intermediate view reconstruction for second camera input frames is unnecessary. On the contrary, this algorithm detects regions on second camera input images that are invisible for base camera but necessary for virtual camera position. This is due to speed up software blocks in order to construct a real time operating display system.

6.2.4 Frame Enhancement

These algorithms are based on mainly two stages at the end of intermediate view reconstruction operation. Due to impulse noise occurred at the end of intermediate view reconstruction algorithms, dealing with such problems is explored in details and Median filtering is proposed in the first stage of frame enhancement algorithms. Many proposed median filtering techniques are compared and contrasted. In chapter-5 detailed information on median filtering is given.

Due to lack of pixel amounts on objects that are near to the base camera, after view construction method, separation and spread of pixels belonging same column and having same depth values is observed. This resulted in a mixed form of objects that are near to base camera with background pixels. In figure-6.10, these kinds of conditions are represented.

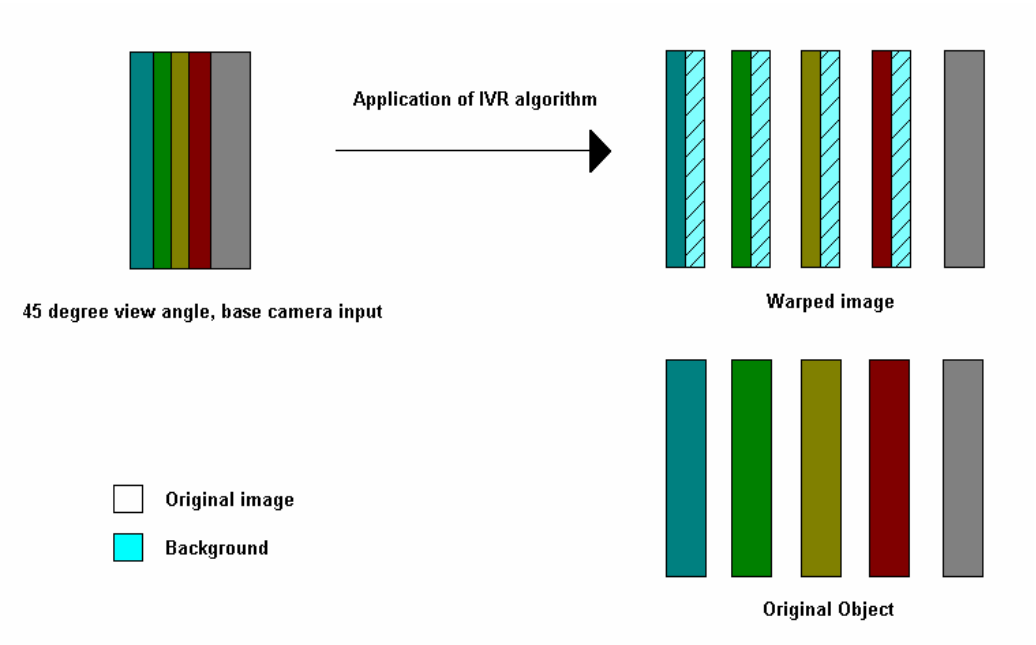

Figure 6.10: Object dehiscence representation

In the second stage of frame enhancement algorithms, background pixel removal from such kind of outputs is discussed in details.

6.2.5 View Point Determination

In this thesis, 3 different methods are used for view point calculation. Inputs for the determination algorithm basically state algorithm type. Different types of inputs for algorithm block are used for pleasure of users. Moreover, variation of view point determination algorithm makes this application program integrate to various hand-held devices, having gyro sensors, touchpad LCD screens or keypad.

6.2.5.1 Gyro Sensor Algorithm

This algorithm is based on calculation of view point relative to an initially assigned position by considering angular rate once user flips-flops the system. Algorithm is performed in FPGA. High speed multiplications, divisions, additions and subtractions are performed by DSP slices that are hardly implemented inside FPGA. Availability of 84 DSP slices became the reason of Spartan XC3SD1800A FPGA family choice [7].

Gyro sensor (ADIS16060) is a yaw rate gyroscope with an integrated serial peripheral interface, which features an externally selectable bandwidth response and scalable dynamic range [8]. The SPI port provides access to the rate sensor, which provides information on angular rate about Z-axis that is normal to the top surface of package [8]. It is possible to increase angular rate measurement range by adding external resistor to the system, which is inversely proportional to the rate range [8]. In this thesis default rate range is used.

In this algorithm, an SPI driver is implemented inside FPGA in parallel, which continuously provides control and rate data information to the main algorithm block inside FPGA. In figure-6.11, timing diagram for angle rate information read cycle is given [8].

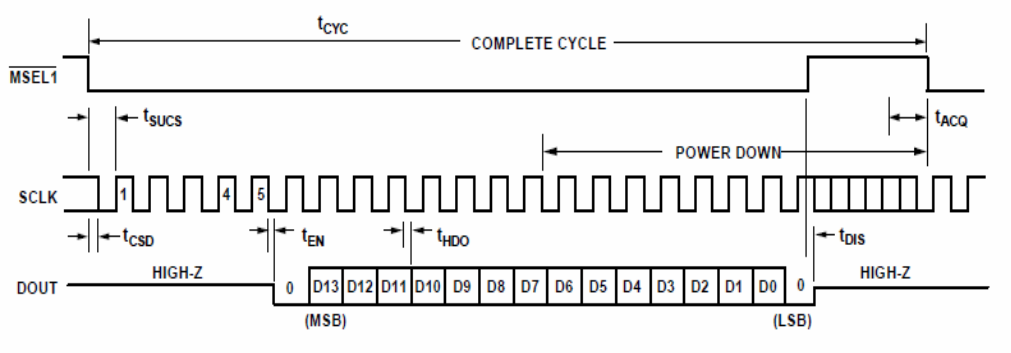

Figure 6.11: Timing diagram for SPI data read

MSEL1 is the data select pin of IC, where SCLK is the transfer clock and DOUT is the available angular rate information. In figure – 6.12, control operation cycle is given [8].

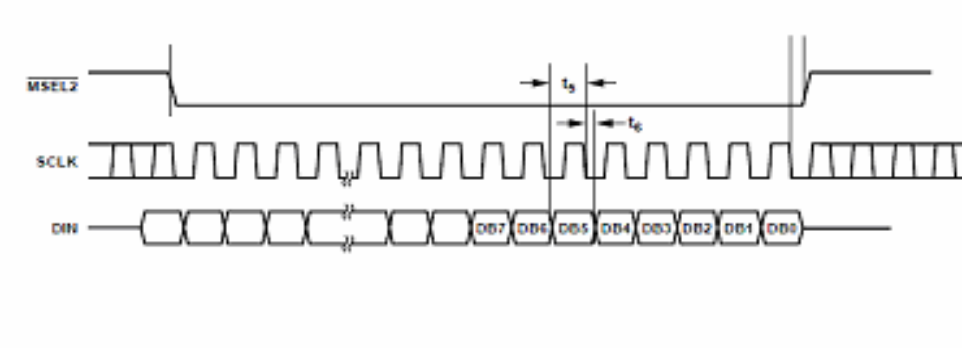

Figure 6.12: Timing diagram for SPI device control

Data provided by the sensor is in an offset-binary format, meaning that the ideal output for a zero rate condition is “8192” codes [8]. If the sensitivity is equal to +0.0122°/sec/LSB which is default sensitivity rate, considering a rate of +10°/sec as an example, results in a change of 820 in the output, of which digital rate output is “9012” [8]. Similarly, once an offset error of -20°/sec is introduced, the output is reduced by 1639 codes, resulting in a digital rate output of 6552 codes as expected [8].

Angular rate information is read continuously by the SPI driver implemented on FPGA. Considering mechanical boundaries of system movement by a user, there is almost no limitations on data transfer and algorithm performance.

Once rate information is available, FPGA calculates new view point continuously once rate is not reversed. This is due to the fact that system continues its movement with a constant angular velocity. Therefore, in this algorithm once new rate information is read, the algorithm assumes continues movement with the given rate until a reverse direction rate is acquired. In figure-6.13, this system is represented [9].

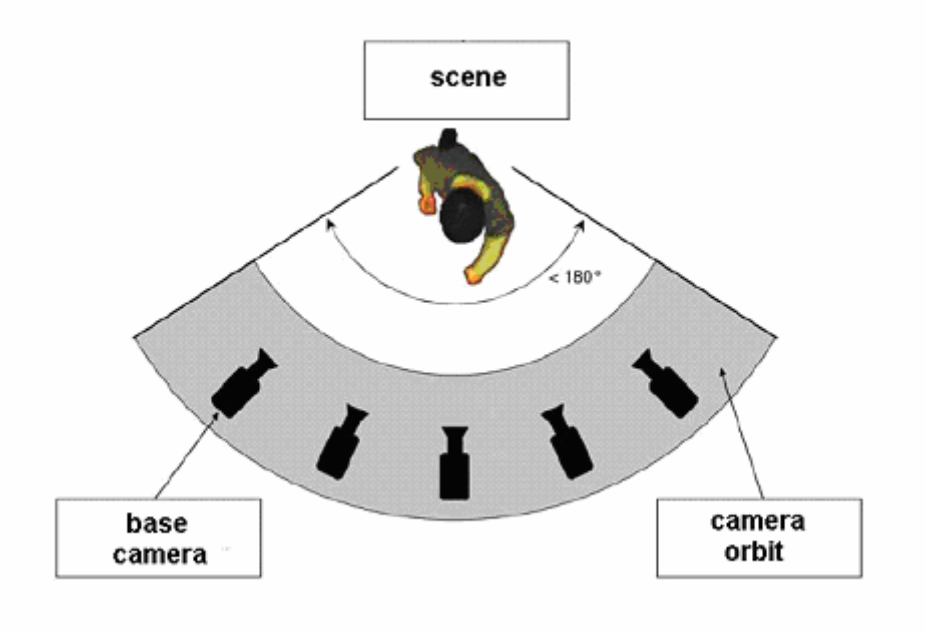

Figure 6.13: Gyro controlled virtual camera orbit

Considering θ to be angular displacement;

$$\theta = \frac{s}{r}$$

where natural units of θ is in radians [10]. Considering Z – axis to be the direction about which rigid system rotates;

$$\omega = \lim_{\Delta t \to 0} \frac{\Delta \theta}{\Delta t} = \frac{d\theta}{dt}$$

where natural units of w is in radians / seconds [10]. Finally angular rate, (angular acceleration) is given to be;

$$\alpha = \lim_{\Delta t \to 0} \frac{\Delta \omega}{\Delta t} = \frac{d\omega}{dt}$$

$$\alpha = \frac{d\omega}{dt} = \frac{d}{dt}\frac{d\theta}{dt} = \frac{d^2\theta}{dt^2}$$

where natural units of α is in radians / seconds^2 [10]. This algorithm has angular rate information for distinct time periods. Assuming 0 radian point at start instant, this algorithm basically determines change in position for distinct periods of time and finally giving an output of new position by adding previous position.

$$\theta_{NEW} = \theta_{PREVIOUS} + \Delta\theta$$

$$\Delta\theta = \alpha\, t^2$$

In general, many companies and driver developers prefer EMIF (external memory interface) communication between FPGAs and processors. This is due to high frequency data interchange considerations. However, in this thesis, i2c communication protocol is preferred because there is no need for huge data exchange. At the instants when main processor ask for view point information, FPGA sends new position data with the given timing diagram in figure-6.14 below [11].

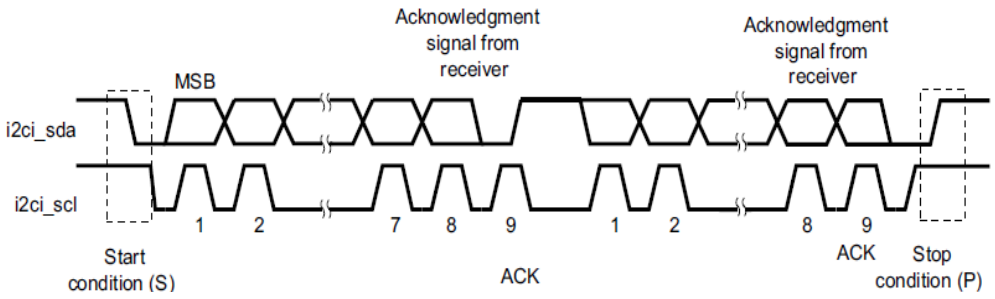

Figure 6.14: I2C port communication timing diagram

Detailed information on processes running on FPGA is given in appendix.

6.2.5.2 Keypad Algorithm

EVM module supports 3x5 keypad as one of the user interface. In this thesis, this feature is used for view point calculation. Once user press left shift key, view point is changed to the left smoothly until user releases the button and once user insists on pressing left shift button, view point is changed until left end is reached. Similarly, pressing right shift key changes view point to the right smoothly until right end is reached.

6.2.5.3 Touch Screen Algorithm

EVM module supports touch-screen with 4 wire touch-screen controller of TSC2046, which is interfaced with main processor OMAP through McBSP port (multi-channel buffered serial port) [4]. Using touch-screen drivers provided by Linux-2.26 embedded operating system, it is possible to read touch-screen information.

In this thesis, this feature is used for view point update. Once user touches the screen, view point is changed to the left end firstly and than backward until the right end is reached and finally comes to the initial point. Changes in view point occur smoothly. This process is depicted in figure-6.15.

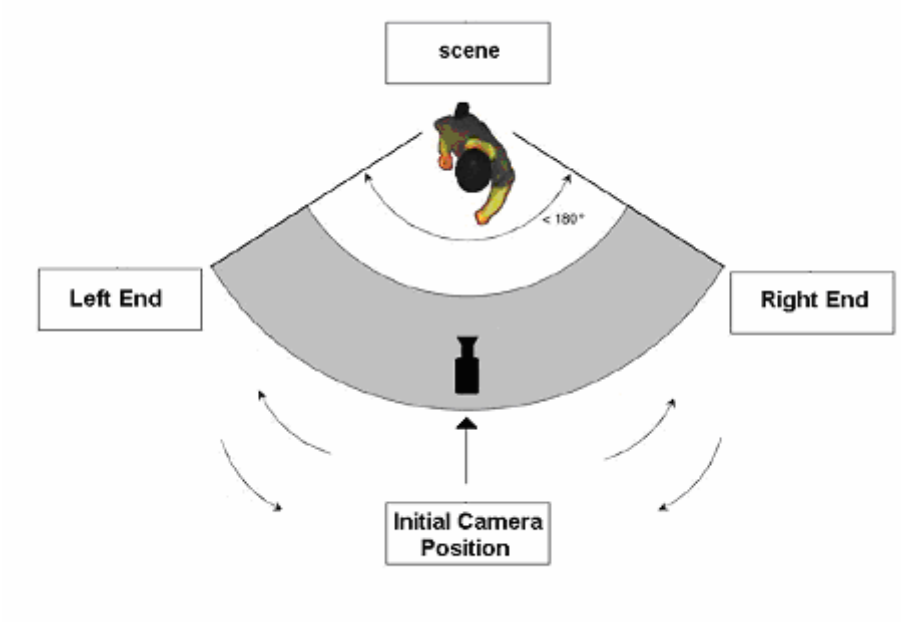

Figure 6.15: Touch-screen controlled virtual camera orbit

6.3 PERFORMANCE of ALGORITHMS on OMAP Cores

In this chapter, performance analysis of algorithms on OMAP cores is performed.. Algorithms were applied on OMAP3530 processor using both cores ARM and DSP separately. Both of the cores are not used simultaneously due to memory collision problem. Both cores perform memory read and write on same bus which causes a need of additional application that controls memory reaches from cores, in kernel space. Such applications are directly related to transferring one or more whole blocks into cache memories by block drivers and devices [29]. In this thesis, character drivers and devices which implements input and output flows are explored [28].

Algorithm performances for occlusion handling and intermediate view reconstruction are given in table-6.1.

Core	**Algorithm**	**Fps**	**Occlusion included?**
Arm only	Floating point, No OcH, FrE	2 / 2 ~ 1	Yes
Dsp only	Floating point, No algorithm	0.66 / 2 ~ 0.33	Yes
Dsp only	Fixed point, No OcH, FrE	12	No
Arm only	Fixed point, No OcH, FrE	18	No
Arm only	Fixed + floating p. No OcH, FrE	7,2 – 7,5	Yes
Arm only	Fixed + floating p. With OcH, FrE	11,5 - 12	Yes

OcH : Occlusion handling algorithm

FrE : Frame enhancement algorithm

Table 6.1: Core performance table

Although, using two cameras requires concern of Rotation matrix as described in chapter-3, using suggested algorithm resulted in a very efficient method for image warping.

Considering algorithm complexity as basic performance measure for the system; given an N x M input frames, suggested algorithm complexity is in the order of $N \times M^2$.

CHAPTER 7

CONCLUSION

In this thesis, view rendering system for hand-held devices with gyro sensors and touch-screen peripheral, is proposed and implemented. The system is applied on OMAP35x EVM board with touch-screen LCD. Two daughter boards are designed and used together with EVM board for gyro sensor application and future work.

First extension board is designed for calculation of view point for intermediate view reconstruction continuously, in order to provide users with view opportunity from different angles and positions. This extension card is consisted of SPARTAN family FPGA with DSP slices for calculation purposes, TMS320VC5509A fixed point DSP for control of the system and additional calculation unit for future purposes, Ethernet controller for communication purposes, SRAMs for memory blocks, touchpad for user interface, LEDs for debugging and UART console for monitoring and controlling purposes. This extension board is designed for future work.

Second extension board is designed as a daughter card for first extension board. This board is consisted of only gyroscope sensor. ADIS16060 from Analog devices is used as single axis response. This additional board is designed in order to align Z-axis perpendicular to EVM board for user

pleasure. This card is connected with an extension connector to first daughter card perpendicularly. This problem can be solved by using 3-axis response gyroscope.

Software architecture of the system is mainly consisted of 5 algorithm blocks as image formation, intermediate view reconstruction, occlusion handling, frame enhancement, and view point determination. Image formation block is implemented on UBUNTU Linux operating system and a pre-processing is done on image frames in order to save memory and consider each pixel by only one “short” variable, which is due to algorithms used for frame enhancement. Intermediate view reconstruction algorithm is optimized for real time video system purpose. Optimization steps are cleared and successfully examined on designed hardware. Occlusion regions are filled successfully by the proposed fast occlusion handling algorithm block after construction of intermediate view. Frame enhancement block is applied as two stages of which first stage is devoted for removal of impulse noise occurred at the end of view reconstruction and second stage is devoted for removal of background pixels that are spread over objects near to camera. Finally view point determination block is realized by 3 methods using gyro sensor, keypad and touch screen facility of EVM board.

7.1 Discussion and Future Work

In this thesis, 256x340 image frames are used for robust intermediate view reconstruction implementation for which view point is arbitrarily determined by user. Up to 12 frames/second are reached which is a very satisfactory result for hand-held devices.

This application is a very entertaining activity as observed after realization of algorithms and watching ballet sequence with depth maps provided by Microsoft research group Sing Bing Kang [19].

These two facts, i.e. using 256x340 image frames and being tied to a research group for video sequences with depth maps, are assumed to be handicaps of proposed method. Although very satisfactory and robust solutions are provided and implemented successfully for hand-held devices, results would not be very pleasant when our methods were applied for a high-definition image frames. This is due to the fact that processor speed would not be sufficient. Moreover it is not always possible to find video frame sequences with corresponding depth maps for two calibrated cameras.

Nowadays, extracting 3D screen structure from an amount of available 2D image content is very popular and a considerable amount of progresses has been made on this issue [18]. Therefore, it is possible to extract depth maps from at least 2 synchronized cameras in order to construct a virtual camera scene. As a result, first handicap can be solved for a mechanically tied and fixed 2 camera system in order to obtain both depth maps and their corresponding image frames. However, this would in turn bring about additional processing blocks for computational system.

Parallel processing is an obligatory solution not only for real time rendering systems of higher resolution image frames but also for additional processing blocks detecting 3D screen information. Moreover, synchronization problem for camera input frames can also be solved by controlling cameras in parallel. And it is an inevitable fact that parallel processing is best realized on FPGAs or CPLDs.

As a future work, our first aim is implementing all software blocks in SPARTAN family FPGA with DSP slices and using either OMAP3530 or TMS320VC5509A fixed point DSP only as a display unit. This is the main reason why first extension card includes additional blocks.

Secondly, we are aiming to construct a fixed and mechanically tied 2 camera system in order to obtain high resolution synchronized image frames. After an intensive research and experiments on 3D screen structure extraction from 2 calibrated images, our aim is to implement suggested algorithms on FPGA on first daughter card. Finally, binding these blocks together on parallel processing unit will provide us with an intermediate view reconstruction system for high resolution real time display systems.

REFERENCES

[1] Y. Bediz, “Automatic Eye Tracking and Intermediate View Reconstruction for 3D Imaging Systems”, MSc Thesis, Middle East Technical University, Ankara, Sept. 2006

[2] S. Ince and J. Konrad, "Occlusion-aware view interpolation", EURASIP J. Image and Video Process., vol. 2008, Article ID 803231, 15 pages, 2008, doi:10.1155/2008/803231

[3] Paul E. Debevec, George Borshukov, and Yizhou Yu, “Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping”, 9th Eurographics Rendering Workshop, Vienna, Austria, June 1998.

[4] “OMAP35x Evaluation Module Hardware User Guide”, Mistral Solutions Pvt. Ltd., Rev. 1.2 May 2008

[5] “OMAP35x EVM Linux PSP User Guide 1.0.2”, Texas Instruments Incorporated, Published 06 OCT 2008

[6] Juan Gonzales, Neal Frager, Ryan Link, "Digital Video Using DaVinci SoC", Texas Instruments, SPRAAN0 - June 2007

[7] "Spartan-3A DSP FPGA Family: Data Sheet", XILINX, DS610 - June2, 2008

[8] "Wide Bandwidth Yaw Rate Gyroscope with SPI (ADIS16060)", ANALOG DEVICES, 2008

[9] Wikimedia, http://upload.wikimedia.org/wikipedia/commons/f/fd/Multicamera.PNG, visited 1 April 2009

[10] D. Smith, "Physics I for Engineers Lecture Notes Chapter 9", Embry-Riddle Aeronautical University, Prescott, 2009

[11] "OMAP3530/25 Applications Processor", Texas Instruments, SPRS507B-February-2008-Revised July 2008

[12] "Architecture and Implementation of the ARM® Cortex™-A8 Microprocessor", ARM Ltd., October 2005

[13] Texas Instruments OMAP™ Family of Products, "OMAP35x Applications Processor IVA2.2 Subsystem Technical Reference Manual", Texas Instruments, September 2008

[14] Texas Instruments OMAP™ Family of Products, " OMAP35xx Applications Processor Interprocessor Communication (IPC) Module Technical Reference Manual", Texas Instruments, February 2008

[15] Texas Instruments OMAP™ Family of Products, “Codec Engine Application Developer User's Guide Technical Reference Manual”, SPRUE67D - September 2007

[16] Xiaoying Li , Baoquan Liu , Enhua Wu, “Double projective cylindrical texture mapping on FPGA”, Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications, June 14-April 17, 2006, Hong Kong, China

[17] A. A. Alatan, “METU EE701 Lecture Notes”, Middle East Technical University, Ankara, 2005

[18] E.Vural, "Robust Extraction of Sparse 3D Points from Image Sequences", MSc Thesis, Middle East Technical University, Ankara, Sept. 2008

[19] Microsoft Research, http://research.microsoft.com/vision/InteractiveVisualMediaGroup/3Dvideo Download/, visited 2 September 2008

[20] Ying-Chieh Chen, Chun-Fa Chang, Zong-Nan Shen, Yong-Min Chen and Hong-Long Chou, “Image-Based Model Acquisition and Interactive Rendering for Building 3D Digital Archives”, Proceedings of 2005 International Conference on Digital Archives Technologies (ICDAT 2005).

[21] William R. Mark , Leonard McMillan , Gary Bishop, “Post-rendering 3D warping”, Proceedings of the 1997 symposium on Interactive 3D graphics, p.7-ff., April 27-30, 1997, Providence, Rhode Island, United States

[22] E. Stringa, A.Teschioni, C.S.Regazzoni , "A classical morphological approach to color image filtering based on space filling curves", IEEE Non Linear Signal and Image Processing Conference, NSIP99, Antalya, Turkey, June1999, pp.351-354.

[23] R. Pandey, “An Improved Switching Median Filter for Uniformly Distributed Impulse Noise Removal”, Proceedings of World Academy of

Science, Engineering and Technology Volume 28, April 2008, ISSN 2070-3740

[24] E. Dinet, F. Robert-Inacio, “Color Median Filtering: a Spatially Adaptive Filter”, Proceedings of Image and Vision Computing New Zealand, 2007, pp. 71–76, Hamilton, New Zealand, December 2007

[25] Gerasimos Louverdis, Ioannis Andreadis and Antonios Gasteratos, "A New Content Based Median Filter", 12th European Signal Processing Conference (EUSIPCO 2004), 6-10 September 2004, Vienna, Austria, pp 1337-1340

[26] K Lawder and P J H King, “Using Space-filling Curves for Multi-dimensional Indexing”, School of Computer Science and Information Systems, Mar. 2001, 16 pages

[27] Hong-Chang Shin, Yong-Jin Kim, Hanhoon Park, and Jong-Il Park, "Fast View Synthesis using GPU for 3D Display", IEEE Transactions on Consumer Electronics, Vol. 54, No. 4, NOVEMBER 2008

[28] Michael Opdenacker, Thomas Petazonni, “Embedded Linux Kernel and Driver Development”, Free Electrons, 3 January 2009

[29] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman, “LINUX DEVICE DRIVERS”, O’Reilly Media, Inc., Third Edition, pp. 6-7, USA, February 2005

[30] Regents of University of California, Berkeley http://bmrc.berkeley.edu/courseware/cs294/fall97/assignment/psnr.html, visited 21 June 2009

APPENDIX A

FPGA PROCESSES FLOW CHARTS

FPGA algorithm basically consists of 3 main parts, namely, gyro-read process, low-pass filter and virtual point detection algorithm and i2c communication with OMAP3530 processor.

i) Gyro - Read Flow Chart

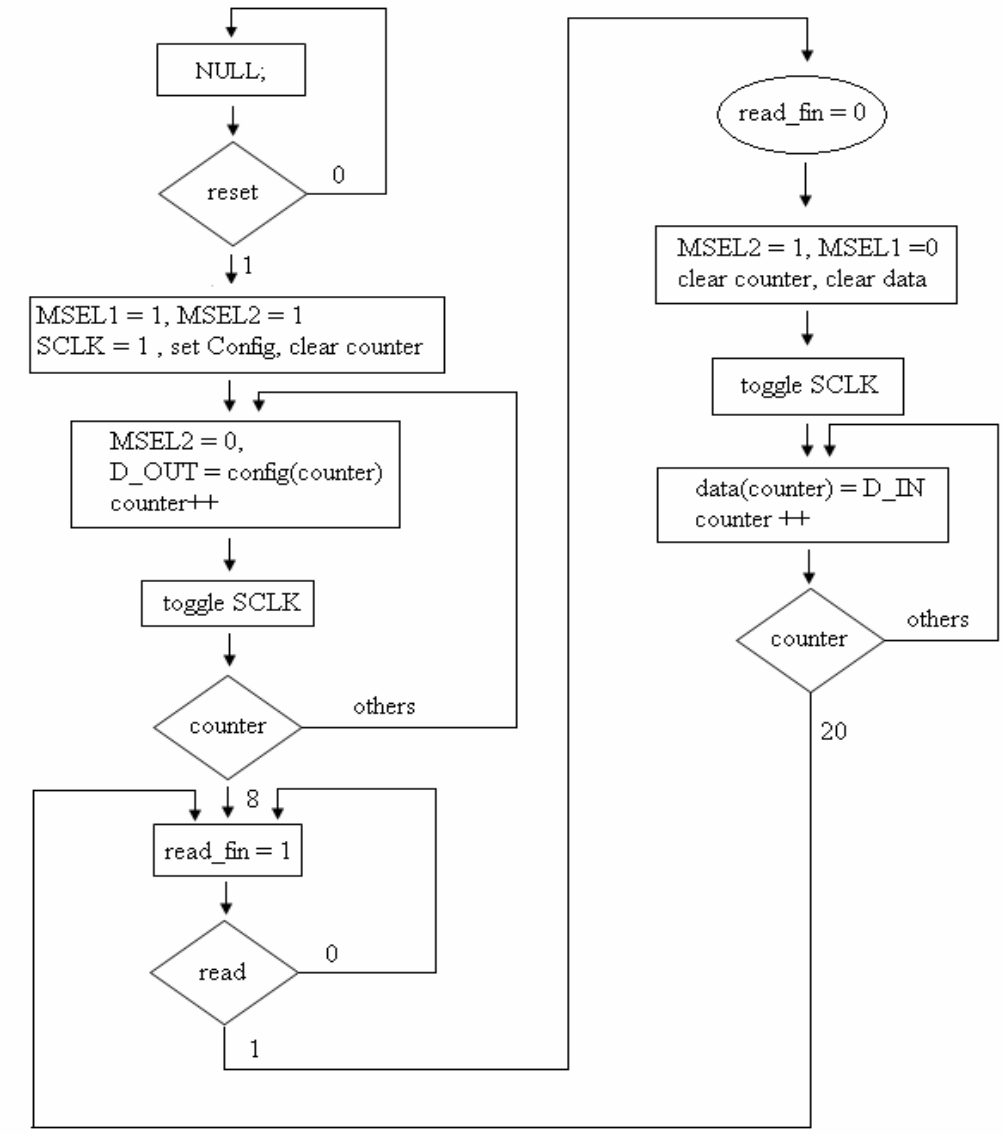

Figure A.1: Gyro-read process flow chart

ii) Virtual Point Detection Flow Charts

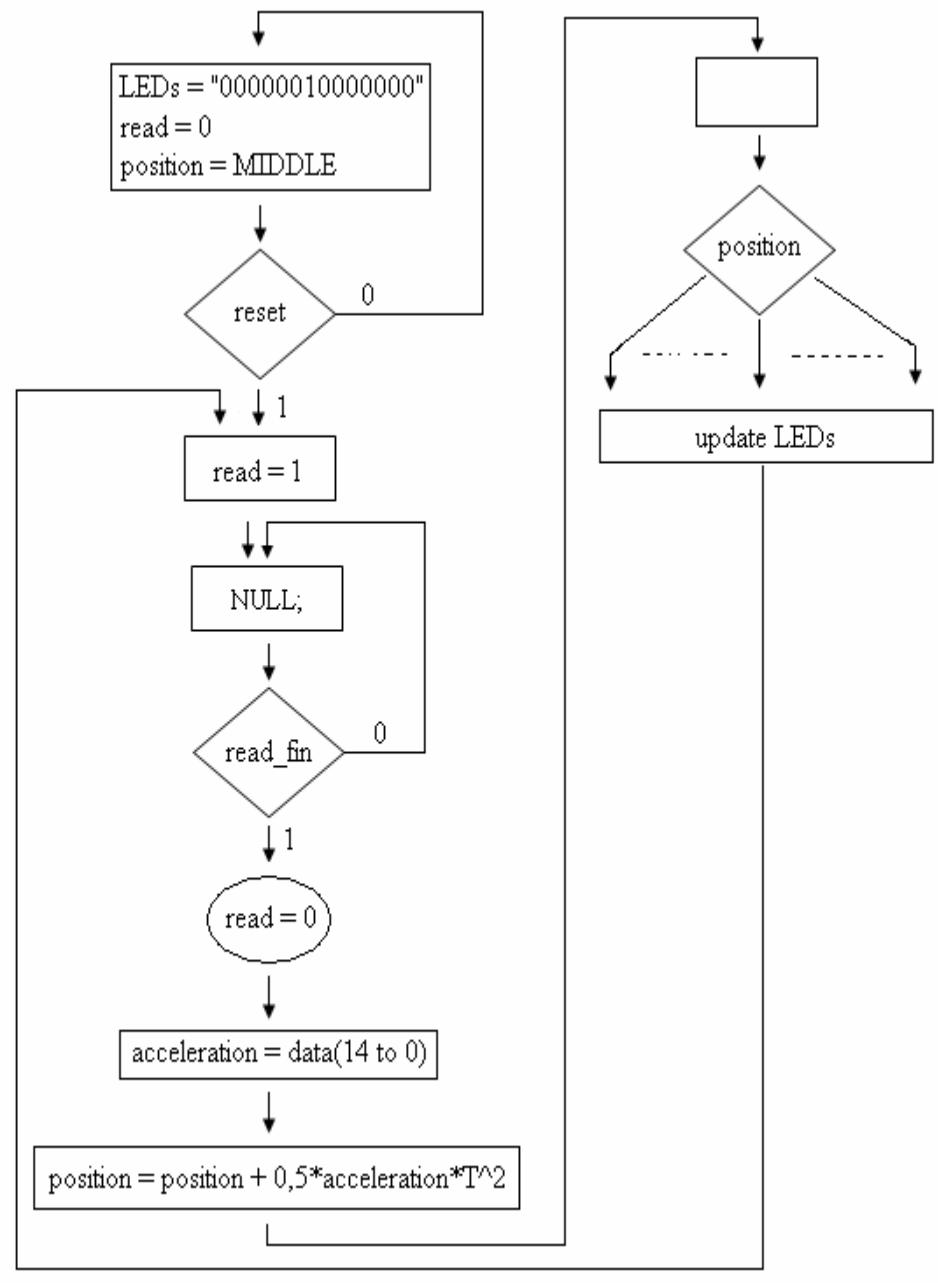

Figure A.2: Virtual point detection flow chart based on look-up table

A second method of virtual point detection based on LED position is given in figure-A.3

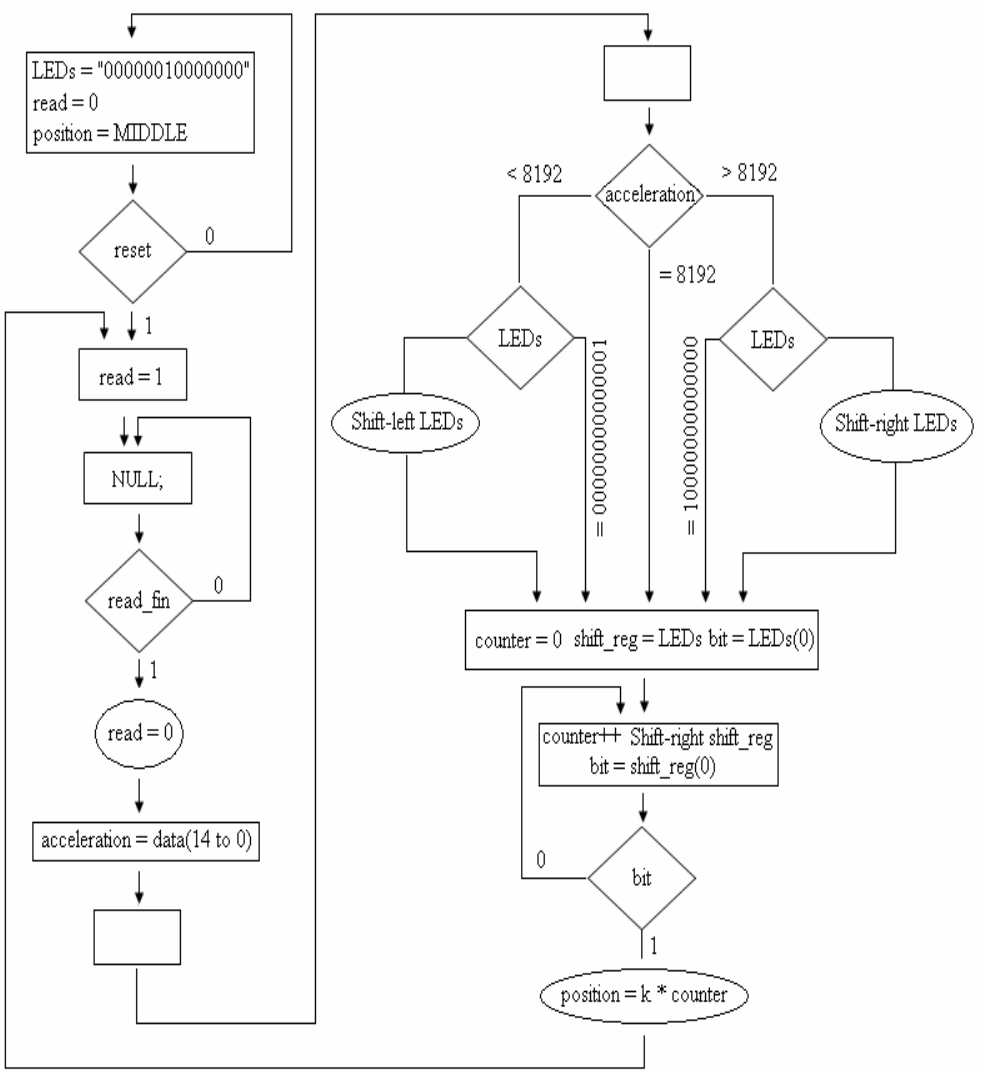

Figure A.3: Virtual point detection flow chart based on LED positions

iii) I2C Communication with OMAP3530

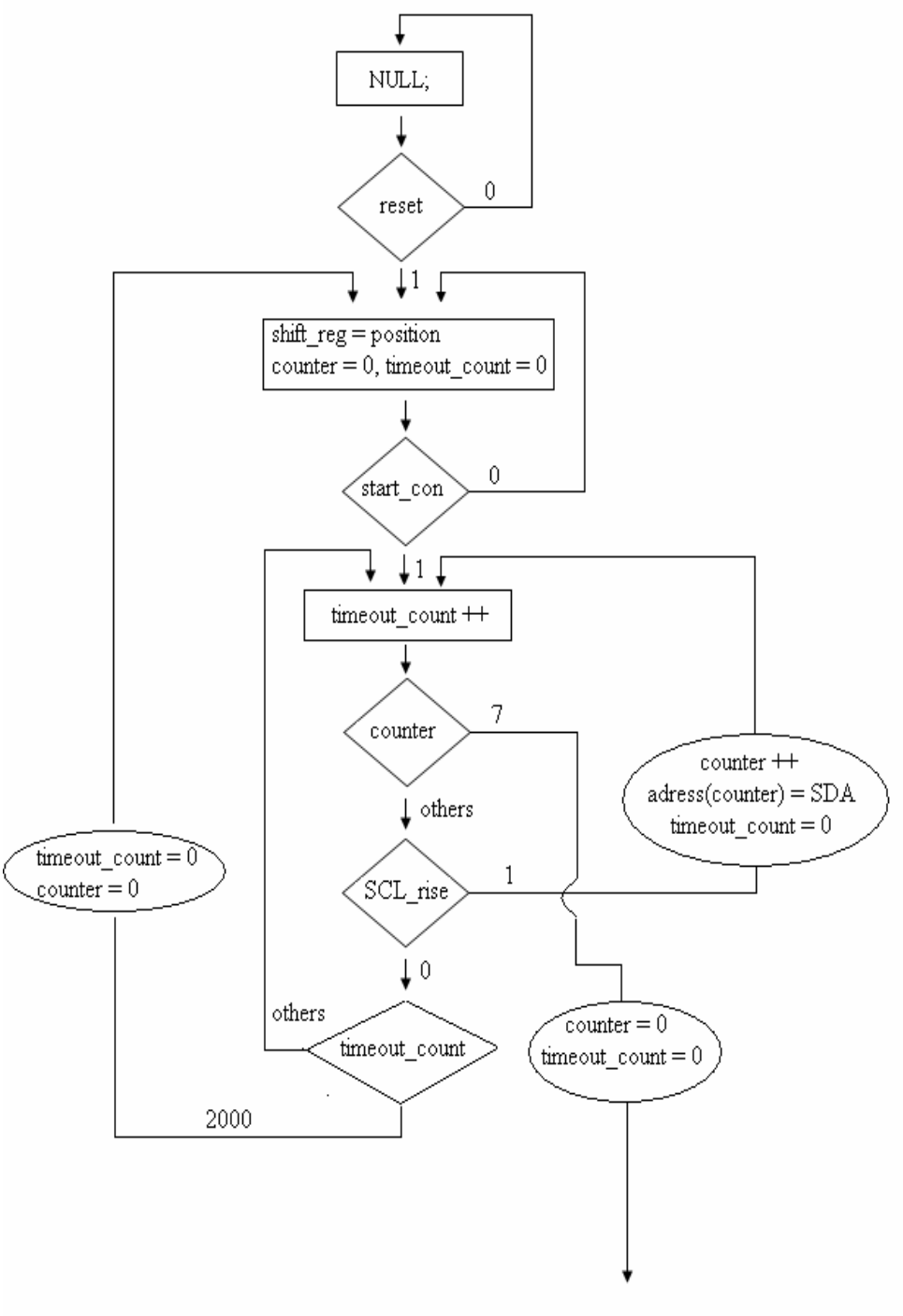

Figure A.4: I2C communication protocol part 1

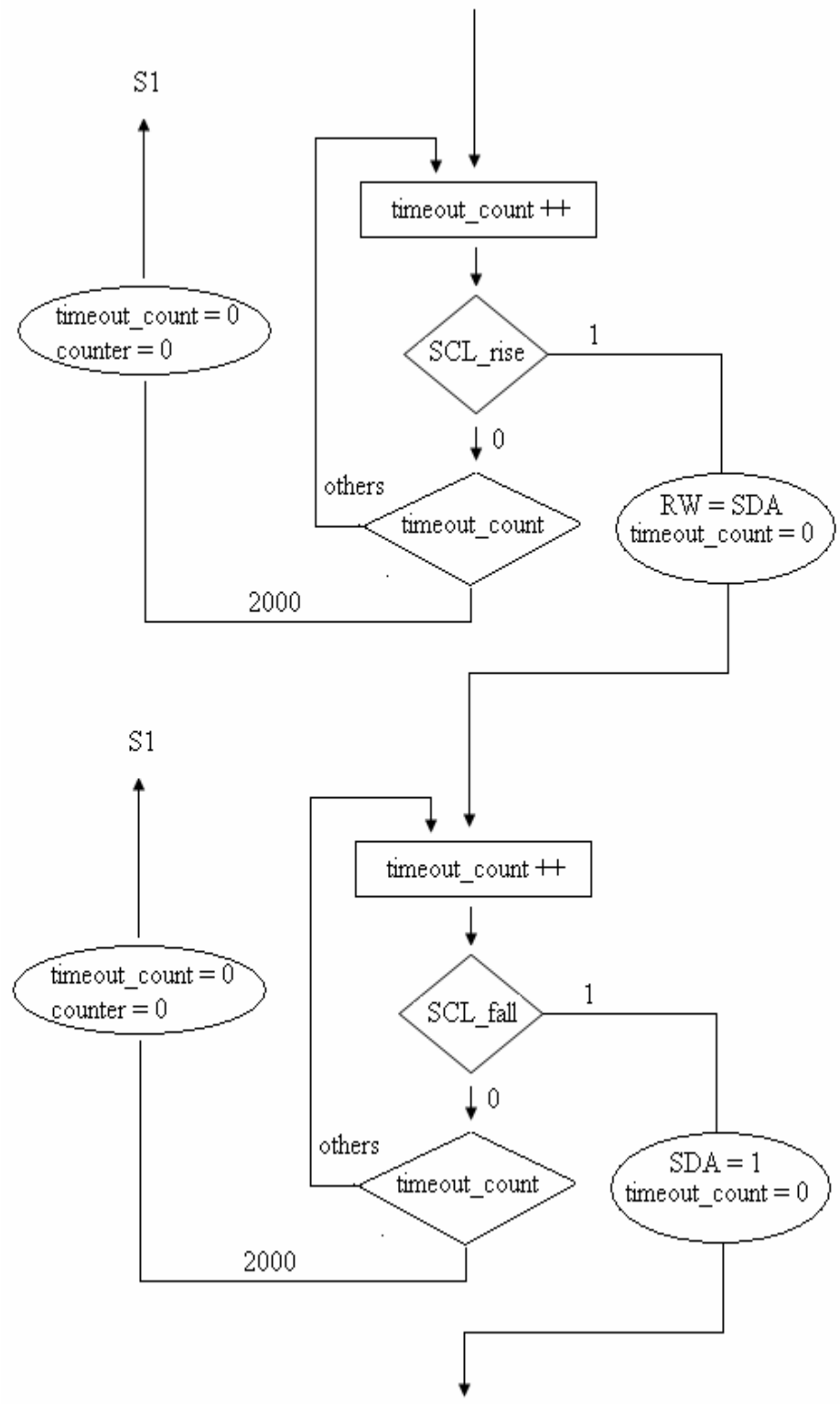

Figure A.5: I2C communication protocol part 2

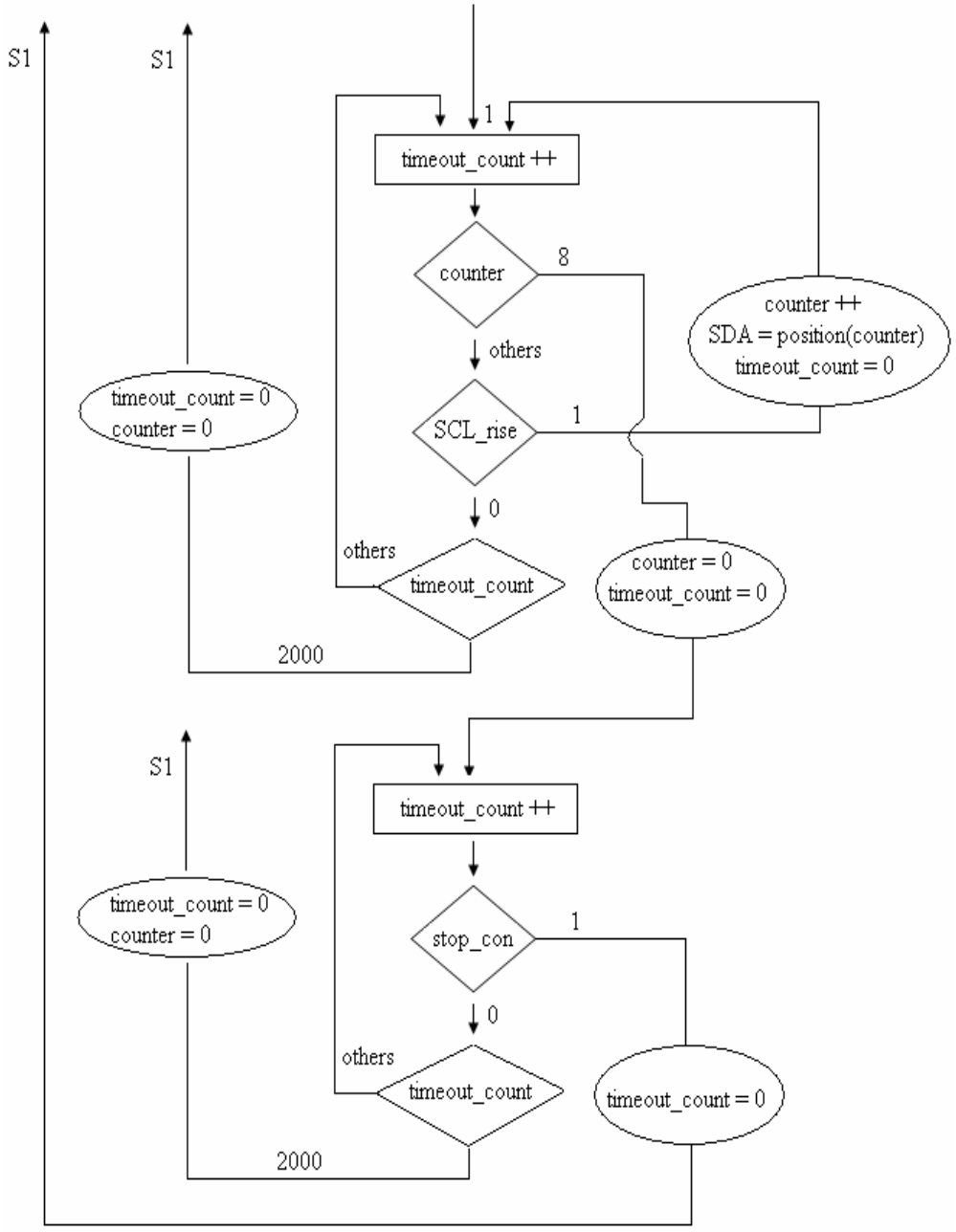

Figure A.6: I2C communication protocol part 3