Real-Time Decoding and Display of Layered Structured Video

by

Tzu-Yun Teresa Chang

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science
at the
Massachusetts Institute of Technology

June 1995

Author_____
Department of Electrical Engineering and Computer Science
May 26, 1995

Certified by_____
V. Michael Bove, Jr.
Associate Professor of Media Technology
MIT Media Laboratory

Accepted by_____
F.R. Morgenthaler
Chairman, Department Committee on Graduate Thesis

Real-Time Decoding and Display of Layered Structured Video

by

Tzu-Yun Teresa Chang

Submitted to the
Department of Electrical Engineering and Computer Science

May 26, 1995

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Electrical Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

With the growth of interactive video applications, many representations of video are being developed with the need of data compression, flexibility, and user interactivity in mind. One such representation is structured video which represents moving images in terms of component parts which can be 2D, 2-1/2D, 3D or layered objects. The component parts are composited according to scripting information to produce the desired image sequence.

This thesis proposal describes the design and implementation of a flexible video decoder for the decoding, processing and displaying of structured video within the Cheops Imaging System. A number of novel features will be added onto the existing decoding system to accommodate more types of structured video representations. A series of experiments will be conducted to demonstrate the types of video the decoder is capable of supporting and to analyze its performance characteristics.

Thesis Supervisor: V. Michael Bove, Jr.
Title: Associate Professor of Media Technology
This work was supported by the Television of Tomorrow consortium.

Acknowledgments

Four score and seven years ago, the year was 1908. No one knew diddly about "Decoding and Display of Layered Structured Video." In 1995, I do. In bringing this topic to the world, I have had lots of help. Those who helped liberate this topic for the world include:

My advisor, Mike Bove, served as a steady source of fresh ideas and suggestions for improvements. His efforts in these areas kept me motivated throughout the research. Without his guidance, this work would not have been possible.

John Wang and Ted Adelson for being so patient answering all my questions on layered video.

Shawn Becker, who always manages to find a solution to my impossible problems. More importantly, he has been a wonderful teacher and a great mentor.

John Watlington, one of the very few souls who can explain what Cheops can and cannot do.

Araz Inguilizian, for being so generous with his time helping me figure out what Rman is all about.

Stefan Agamanolis, whose knowledge of and experience with Cheops helped this project tremendously.

Brett Granger and Katy Evanco for explaining their work to me, so that my work can be accomplished a little more easily.

Greg Haverkamp, a wonderful guy, whose technical writing tutelage helped this thesis to be more readable. He deserves a great job, a tremendous woman, and lots and lots of money.

Karrie, Alison, and Marcie, my wonderful group of friends. Knowing when to work and when to have fun can be tricky; thanks for helping me along this slippery slope.

My parents and my sister, Jeanne, for their unconditional love and for their emotional and financial support.

My boyfriend, Rick, for listening to my incessant whining throughout this whole ordeal and for being the main reason I finished on time.

Contents

1 Introduction **10**

1.1 Structured Video 11

1.2 Structured Video Processing Pipeline 15

1.3 Thesis Overview 18

2 Background Issues **20**

2.1 Represent Moving Image in Layers 20

2.1.1 Motion Analysis: Affine Transformation 22

2.1.2 The Layered Representation 23

2.2 Cheops Hardware 24

2.2.1 The Remap/Composite Card 26

2.3 Cheops Software 28

2.4 Previous Work 29

2.4.1 3-D Objects 29

2.4.2 2-D and 2 1/2-D Objects 30

2.4.3 Composite Unit 30

2.4.4 Scripting Language 30

3 The Layered Decoder **32**

3.1 Intra-Coded Frame 35

3.1.1 IDCT/Q^{-1} 36

3.2 Explicit Transformation Unit 37

3.2.1 Warping 37

3.3 Composite Unit 39

3.3.1 Zbuffering 39

3.3.2 Accumulation/Composite Unit 40

3.4 Error Signals 42

3.5 Scaling and Linear Interpolation 44

3.6 The Interface 45

4 Experimental Results 46

4.1 Experimental Apparatus 46

4.2 Image Quality 47

4.2.1 IDCT/Q^{-1} 47

4.2.2 Src2Dst 49

4.2.3 Dst2Src 50

4.3 Timing Analysis 51

4.3.1 Different Combinations of Processors 52

4.3.2 Analysis on Each Approach 53

4.4 Another Sequence 57

5 Conclusion 61

5.1 Status 61

5.2 Future Work 62

Appendix A: Data Processing Pipeline **64**

A.1 Src2Dst 64

A.2 Dst2Src 67

A.3 Common Elements 69

Appendix B: The User Interface **70**

List of Figures

1-1	The Museum Movie: Man observing the statue	13
1-2	The Museum Movie: A different camera angle	13
1-3	The Museum Movie: Same scene but with zoom	14
1-4	Data processing pipeline of the structured video decoder system	15
1-5	Different configurations of the data processing pipeline	17
2-1	The ball/background sequence	21
2-2	The decomposition of the ball/background image sequence	21
2-3	The reconstructed ball/background sequence	23
2-4	Cheops P2 Processor Board	25
2-5	Remap card in direct read	27
2-6	Remap card in absolute write	28
2-7	RmanConnect and RmanDepend	29
3-1	High level block diagram of the layered decoder	32
3-2	1st, 15th, and 30th frames of the garden sequence	33
3-3	Five intensity maps of the garden sequence	34
3-4	DCT and quantization encoding unit	35
3-5	Dequantization and IDCT decoding unit	36

3-6	Source_to_destination	38
3-7	Destination_to_src	38
3-8	Generic accumulation method	41
3-9	Src2dst using accumulation to composite	41
3-10	Dst2src using accumulation to composite	42
3-11	Generate error signals	42
3-12	Error frame from dst2src	43
3-13	Error frame from src2dst	44
3-14	Scale and interpolation unit	44
4-1	Experimental apparatus	46
4-2	DCT, IDCT-coded image	48
4-3	DCT, Q, Q^{-1}, IDCT-coded image	48
4-4	Original garden layers composited	49
4-5	Src2dst: garden sequence with no error signals	50
4-6	Src2dst: garden sequence with error signals	50
4-7	Dst2src: garden sequence with no error signals	51
4-8	Dst2src: garden sequence with error signals	51
4-9	Mobile sequence: 1st, 15th, and 30th frames	58
4-10	Src2dst: mobile sequence with no error signals	59
4-11	Src2dst: mobile sequence with error signals	59
4-12	Dst2src: mobile sequence with no error signals	60
4-13	Dst2src: mobile sequence with error signals	60
A-1	Vector for src2dst method	65
A-2	Vbuffer for dst2src method	67

List of Tables

2-1	Remap card: Phases and modes	26
4-1	Timing analysis of each type of transfers	52
4-2	Timing analysis on different number of processors	53
4-3	Timing analysis of each src2dst unit	55
4-3	Timing analysis of each dst2src unit	58

Chapter 1

Introduction

In the realm of computing, one may define multimedia as an aggregate digital framework composed of graphical, video, audio, and text information. A distinct advantage of a multimedia environment is its ability to convey each of its various components into a form best suited to its characteristics. The environment uses the graphical, audio and text forms to create an efficient representation of a large body of information. This potential is mainly responsible for the extraordinary growth of demand for multimedia applications. Multimedia technology is revolutionizing the way we represent, share, and exchange information.

Many interactive multimedia applications involve the use of video, such as video mail, video games, and computer-generated movies. There are many advantages of handling video in its digital form. The most notable advantage is the sheer breadth of methods available for manipulation of the image. These methods are realizable because the digital representation of the image is comprised of discrete components known as pixels, each of which may be individually modified. Using the computational abilities of computers to exploit the accessibility of the picture's digital components provides a bounty of potential operations on digital video.

Video or image demands have forced personal computers, workstations and networks to handle enormous amounts of data. Many communication systems simply cannot handle the bandwidth intensive requirement of video, thus producing low transfer rates. Nor do

Note: This overview is based on Joel Adam's definition and discussion of the multimedia field [15].

they have enough storage space for digitized movies. Despite improved hardware, that is, increasing advances in processor speeds, memories, and storage technologies, there is still need for data compression, flexibility, and user interactivity in software. Such improvements are the motivations and goals behind this thesis.

1.1 Structured Video

Structured video must first be defined to understand this thesis. It is the representation of moving images in terms of component parts. The component parts may be objects with uniform depth (2-D), objects with corresponding surface depth values (2 1/2-D), computer graphics objects (3-D) [1], or layered objects with associated intensity, velocity and opacity maps (2-D or 2 1/2-D)[3]. In structured video environments, the component parts are assembled according to scripting information which has the ability to manipulate the objects in a great variety of ways. For instance, objects such as people can be placed into a synthetic background or a 3-D particle based background.

One of the advantages of using structured video to describe an image sequence is the much smaller amount of memory it uses compared to pixel-and frame- based representation. Brett Granger's example of using 1Kx1K, 24-bit color representation on a 60 Hz progressive-scan display will result in 180 megabytes transmitted per second of sequence. By using the structured-video representation, a 1Kx1K color background can be described in three megabytes and an object smaller than the background will obviously require less. The resultant maximum of six megabytes plus a small script will be a tremendous saving when compared to 180 megabytes per second [6].

Flexibility and user interactivity in structured video is demonstrated by the process of shooting and scripting language in the movie, *The Museum*, developed by the Television of Tomorrow Group of the MIT Media Laboratory. However, before explaining those stages, a summary of its plot is as follows: A man walks into a gallery room and looks at the artworks displayed on the walls. Then he sees a statue of a man in the middle of room and becomes fascinated by it because when he views the statue from the front, the statue

appears to become alive, motioning him to come. The man moves closer and closer to the statue and freezes into another statue of a man. The original statue comes alive and walks away.

In the process of shooting, several photographs of the MIT Media Laboratory gallery were taken and a 3-D model of the gallery was extracted using Shawn Becker's semiautomatic 3-D model extraction technique [13]. The two actors, the man and the statue, were shot in front of a blue screen at three different camera angles. The two actors, which are 2-D objects, and the gallery background, which is currently represented as a 2 1/2-D object but can also be a 3-D object, are the three components of this structured video movie. They are stored separately in the computer memory and can be manipulated by the scripting language.

The scripting language allows the user to control the state of the objects and the view and the display parameters, as shown in Figures 1-1, 1-2 and 1-3. The actor's parameters that can be controlled by the script are the actor's position in either 2-D or 3-D, scale, visibility, frame and view. The scale is the size of the actor. The visibility is the transparency or opacity of the actor. The frame is the temporal frame number from the actor's image sequence, while the view is the camera angle at which the user wants to look at the actor.

For display and camera views, the script controls the display size and position as well as all the 3-D view parameters such as camera location, view direction and focal length. If an actor's position is specified in 3-D (as it is in *The Museum* movie), the program uses the camera view parameters to determine the correct position and scale of the actor in the frame.

Figure 1-1 shows the man observing the statue in the center of the gallery. Figure 1-2 shows the same scene from a different camera angle. The user can change the camera angle by simply twisting a knob on the Cheops knob box. Finally, Figure 1-3 shows the same scene again, this time, the scene is zoomed in.

Figure 1-1: Man observing the statue

Figure 1-2: Same scene from a different camera angle

Figure 1-3: Same scene but zoomed in.

1.2 Structured Video Processing Pipeline

Processing Pipeline

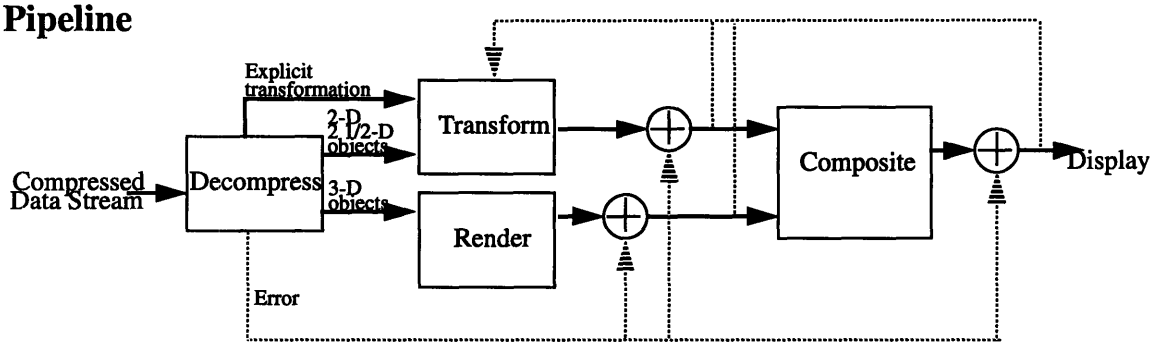

Figure 1-4: Data processing pipeline of the structured video decoder system

Structured video can take on various forms such as 2-D, 2 1/2-D, 3-D, and objects that requires motion transformations such as the layered representations, which will be described in detail in Chapter 2. The Television of Tomorrow Group has been concentrating on the development of an open and flexible structured video decoder system which decodes and displays such a representation.

Each structured video object is processed differently. For instance, 3-D objects require rendering and projection, 2D and 2 1/2-D might need scaling or warping. The generic structured video decoding pipeline [1], depicted in Figure 1-4, supports all these different types of objects. It is assumed that the decoding pipeline is provided with a structured video stream and that the method of encoding is known. The decoding system also features user interactivity, giving the users the ability to insert or relocate objects in the scene or the opportunity to view the scene from a different camera angle.

The following is a list of the major objects and functional requirements that the decoder system should fulfill.

2- D objects: These objects are comprised of pixels arrayed in a common plane. By assuming an uniform plane depth, layering may be accomplished and is handled

through scripted definitions of the objects' data. Transparency allows for the viewing of subsequent layers and is akin to holes placed in the object.

2 1/2-D objects: These are 2-D objects without the assumption of the uniform plane depth. Each pixel takes on a depth value which can be specified by a z-buffer that may be used in compositing.

3-D objects: These are collections of points that possess intensity, x, y, and z values. For the purpose of this thesis, such objects are stored as a collection of particles, but they can also be described as textured-mapped polygons. They require rendering before viewing.

explicit transformations: This refers to the spatial redistribution of an object's points. Transformations may be specified in one of two ways. They may involve an array of transformations to be performed on each point of the object. Also, they may be performed parametrically. Parametric specifications would be used for affine transformations (scaling or rotation). Affine transformations will be described in detail in Chapter 2.

error signals: These are two-dimensional arrays of values which are added to objects that have been transformed or rendered. Their function is to correct for errors that occurred due to the method of encoding.

intra-coded unit: Objects may be compressed (3-D objects may be entropy-coded for compactness; 2-D and 2 1/2-D objects may be DCT-coded). Intra-coded unit will decompressed those objects.

As shown in Figure 1-5, different structured video objects follow different data path in the data processing pipeline. Thus, various types of decoders, such as a hybrid predictive decoder and a combined 2-D, 2 1/2-D, 3-D decoder, can be constructed from the generic pipeline.

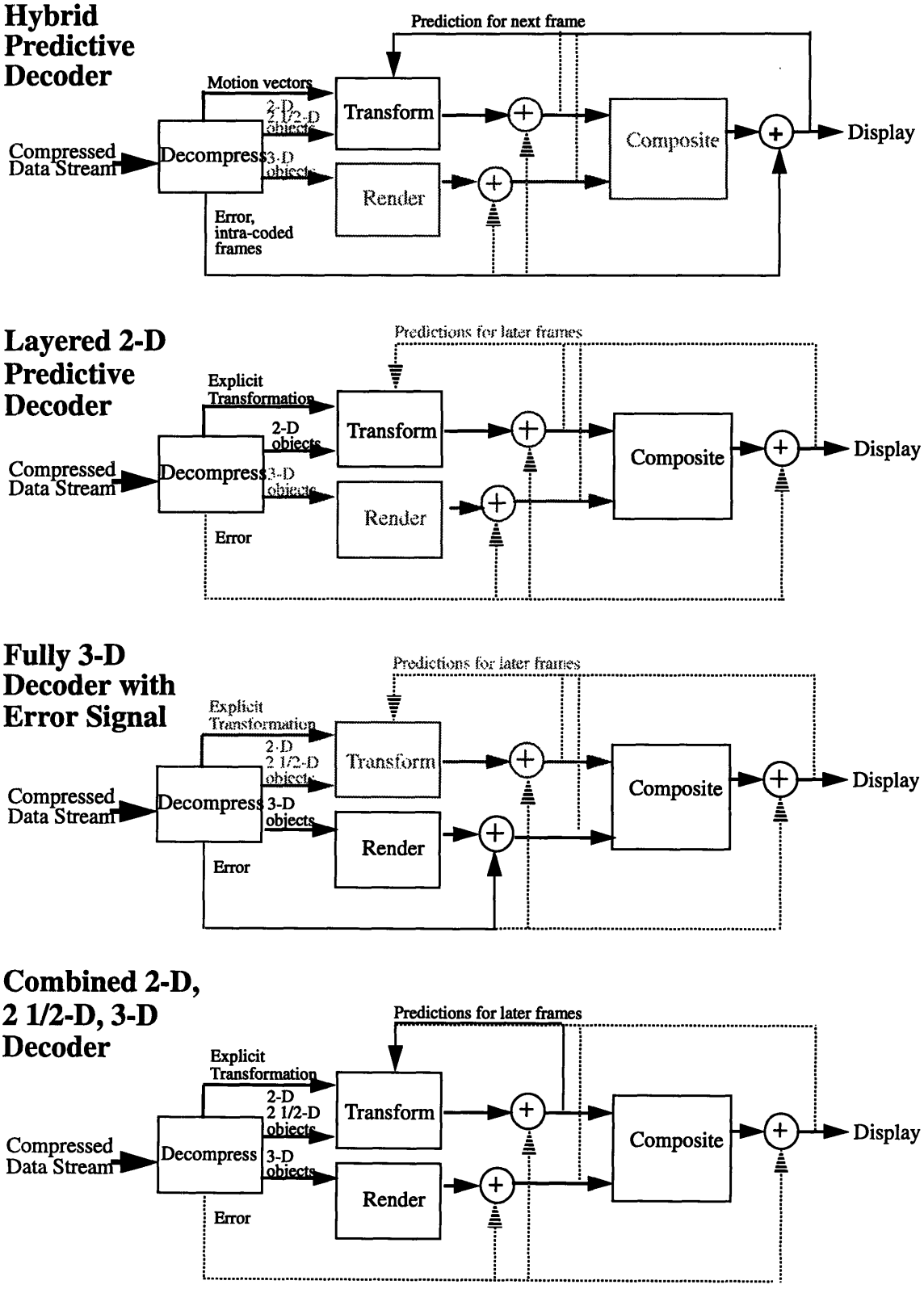

Figure 1-5: The pipeline can decode a variety of coding methods. Gray datapaths are inactive, while dashed datapaths interconnect in an algorithm-dependent manner.

1.3 Thesis Overview

This thesis describes the design, implementation, and use of a layered 2-D predictive decoder, depicted in Figure 1-5, on the Cheops Imaging System. The decoder is incorporated into the existing structured video decoder which exhibits 2-D, 2 1/2-D, and 3-D features. The purpose of this thesis is to add more features to the current real-time decoder for more types of different representations of structured video.

The decoder is able to process and decode a provided known type of structured video stream and display the result. The new layered decoder embodies a number of features such as explicit transformation, predication for later frames, errors signals, intra-coded frame, and compositing unit not found in the current decoding system on Cheops. The layered decoder uses the layered algorithm developed by John Wang and Ted Adelson [3]. It focuses primarily on the software implementation and mainly utilizes the remap and filter cards on the Cheops system.

The remaining portions of this thesis are organized as follows:

Chapter 2 discusses a number of background issues relevant to the layered decoder. The chapter begins by providing a tutorial on the representation of moving images in layers and the Cheops Imaging System. Previous work done on the structured video decoder are then discussed to provide a clearer picture of what has been done and what needs to be added.

Chapter 3 describes the different units: explicit transformation unit, composite unit, error signals, scaling and interpolation units and intra-coded frames. They are the major components that comprise the layered decoder.

Chapter 4 describes a number of experiments developed for the layered decoder to test the decoder's performance.

Chapter 5 draws conclusions from the experiments and suggests future work.

The appendices provide more detailed information about the software of the decoder than what is described in the body of the thesis. Appendix A describes the specifics of how

the data processing pipeline is connected in the current Cheops environment. Appendix B describes the use of functional commands to implement the various layered decoder units.

Chapter 2

Background Issues

This section discusses the many issues which will influence the design of the layered decoder. The section discusses four important background issues: The layered representation of moving images, The Cheops Imaging System: Hardware and Software, and previous work. The subject of layered representation is important to understand the video encoding and decoding process, while the description of the Cheops system is vital to understand how the decoder integrates into the overall system. The last section of the chapter will discuss previous work done on the generic structured video decoder.

2.1 Represent Moving Images in Layers

Layered images is a "mid-level" image coding technique developed by Ted Adelson and John Wang of the Vision and Modeling Group at MIT Media Laboratory[3,4,5]. The technique represents moving scenes with sets of overlapping layers. Each layer contains three maps: the intensity map, the velocity map, and the opacity (alpha) map. The intensity map describes the color or value of the pixels. The velocity map defines how the image moves over time. The alpha map indicates the opacity of the layers. Usually, the layers are ordered by depth. For example, the background will be the first layer, and the foreground will be the second layer. Thus, each layer occludes the one beneath it [1].

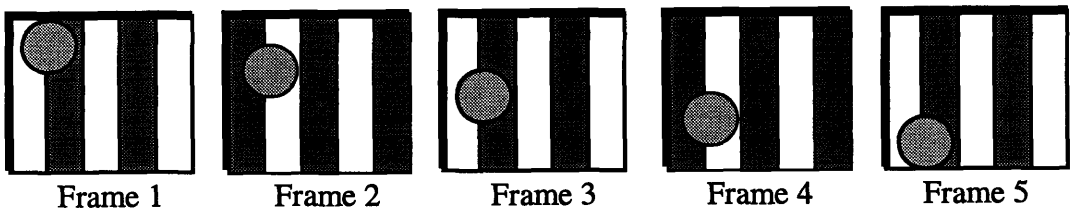

Figure 2-1: The ball/background image sequence.

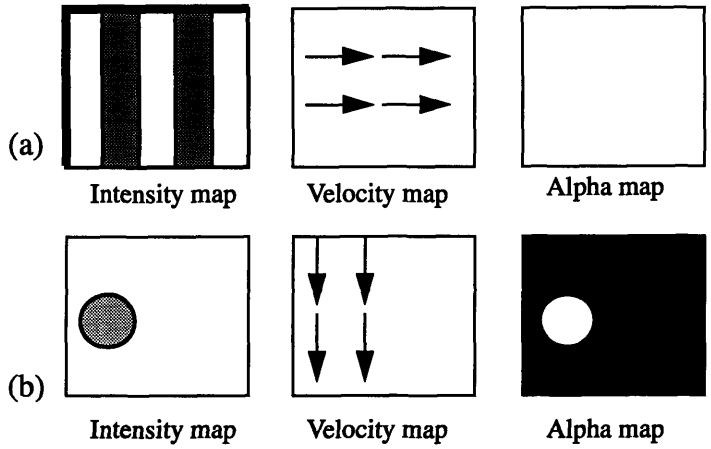

Figure 2-2: The decomposition of the ball/background image sequence. (a) The three maps of the background layer: intensity map is the black and white striped background of the center frame of the sequence, velocity map shows the background is moving horizontally to the right, and alpha map represents the background is opaque. (b) The three maps of the foreground layer: intensity map is the red ball in the center frame, velocity map shows the ball is moving vertically downward, and alpha map shows the ball is opaque and its background (black) is transparent.

Consider a simple moving sequence of a red ball moving against a back and white vertically-striped background, depicted in Figure 2-1. The red ball travels downward along the vertical axis. Independently, the black and white background travels rightward along the horizontal axis. In the layered representation, there will be two layers: the ball and the background. Each layer consists of aforementioned intensity, velocity, and alpha maps, depicted in Figure 2-2. The first layer will have the black and white background as its intensity map, the horizontal motion as its velocity map. Similarly, the second layer will have

the red ball and vertical motion as its intensity and velocity maps, respectively. And the alpha map of the second layer will place the circle in front of the first layer.

2.1.1 Motion Analysis: Affine Transformation

Layered representation uses the idea that motion models from regions that cover the same object will have similar parameters [3]. The affine motion model consists of six parameters, and describes translation, rotation in the plane, zoom, shear and any linear combination of these. All of those effects are commonly encountered in video sequences thus a good analysis to use [3].

Affine motion is defined by the following two equations:

$$V_x(x,y) = a_{x0} + a_{xx}x + a_{xy}y \quad (2\text{-}1)$$

$$V_y(x,y) = a_{y0} + a_{yx}x + a_{yy}y \quad (2\text{-}2)$$

The a's are the affine transformation parameters and V_x and V_y are the velocity components. The process of finding the affine parameters will not be part of this thesis. It is assumed that the affine parameters will already be calculated and will be provided to the decoder. Thus, there will be six affine parameters for each layer in each frame.

The affine parameters for the images used in this thesis are calculated as dx, dy or change in the x and y directions. The equation that finds the destination intensity map, I_N, from the given intensity map, Io, is:

$$I_N(x, y) = I_0(x\text{-}dx_N, y\text{-}dy_N) \quad (2\text{-}3)$$

where

$$dx_N(x,y) = a_{x0,N} + a_{xx,N}x + a_{xy,N}y \quad (2\text{-}4)$$

$$dy_N(x,y) = a_{y0,N} + a_{yx,N}x + a_{yy,N}y \quad (2\text{-}5)$$

2.1.2 The Layered Representation

To gain a better understanding of how the layered representation is used by this thesis, the example of the red ball and striped background image sequence will be continued. If there are five frames of the red ball and striped background sequence as shown in Figure 2-1, the analysis of Adelson and Wang as mentioned previously will give two layers, the red ball and the background, depicted in Figure 2-2.

Each layer will have its associated intensity, velocity, and alpha maps. The intensity map of each layer is usually taken from the center frame of the image sequence. If intensity maps from the first frame of the sequence are used, then warping the first frame to the last frame of the sequence will create a significant amount of errors signals because the algorithm is not exact. Sequentially warping, such as warping from first frame to second, second frame to third, etc., is also not recommended because the second frame already contains some errors or defects from the warping, thus the third frame will contain even more errors including mistakes from second frame. Therefore, intensity maps from the center frame are preferred because they risk the least amount of variation and mistakes as depicted in Figure 2-3. However, since the model is not exact, there will be some kind of defects in the images or that not all of the image changes will be captured. Delta maps or error maps can be added to the results to correct for those mistakes.

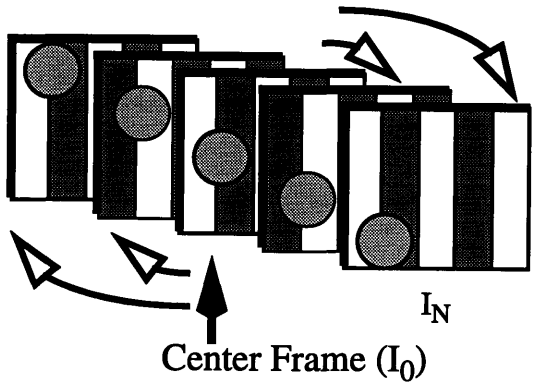

Figure 2-3: The reconstructed sequence is warped from the center frame of the original image sequence.

The layered model not only demonstrates "mid-level" imaging coding techniques which utilizes concepts such as segmentation, surfaces, depth, occlusion, and coherent motion [3], it also greatly reduces the amount of data that describe a moving image sequence. Most of data compression are accomplished by warping from one given frame to another using affine transformation. The data from the model can be further compressed by performing on them discrete cosine transform (DCT) and quantization, which are the two key elements in the Joint Photographic Experts Group Video (JPEG) standard for still-picture image compression.

DCT transforms each color component (RGB, YUV, or CMYK) from the spatial domain to the frequency domain or to a matrix of frequency values. Since most real world images exhibit high spatial correlation, and the spectral content of those images is skewed such that most of the energy is contained in the lower frequencies, the DCT representation will tend to be sparsely populated and have most of the energy in the coefficients corresponding to lower frequencies [14]. Additionally, the visual system is less sensitive to quantization noise at higher frequencies. Then, compression occurs when the frequency coefficients are quantized.

2.2 Cheops Hardware

Video applications present a serious problem in the computing environment especially in the storage and transfer of video information. Full-motion (30 frames/sec) color video sequences overwhelms the capabilities of current computer systems. As an example, a single 24-bit color image with a resolution of 1024x768 pixels occupies about 2.3 Mbytes of storage space. Thus, storing a 15-minute full-motion color video sequence requires 60 Gbytes. Transferring such video sequences is equally as difficult. A good fast hard disk can transfer at about 20 Mbyte/sec, therefore, the 15-minute video will take about 0.7 hours. Many present workstations and networks are simply ill-adapted to the large volume of video data or the bandwidth intensive requirements of video. Fortunately, new systems are being designed and developed with the needs of video in mind such as high-bandwidth

video transfers.

The Cheops Imaging System is a compact data-flow system for real-time processing and display of video. It is designed for high-bandwidth video transfers and its architecture is a prototype for programmable video decoders [2]. The Cheops hardware contains input/memory modules (M1), processing modules (P2), and output/display modules (O1/O2). These modules are connected with one another via three linear buses. One of the buses is the Global Bus, the other two are the Nile Buses which can support high-bandwidth transfers of 120 Mbytes/sec.

The Cheops processor modules abstracts out a sets of basic, computationally intensive stream operations that may be performed in parallel and embodies them in specialized hardware [1]. Three stream transfers may occur at the same time provided that they use different memory banks and stream processors.

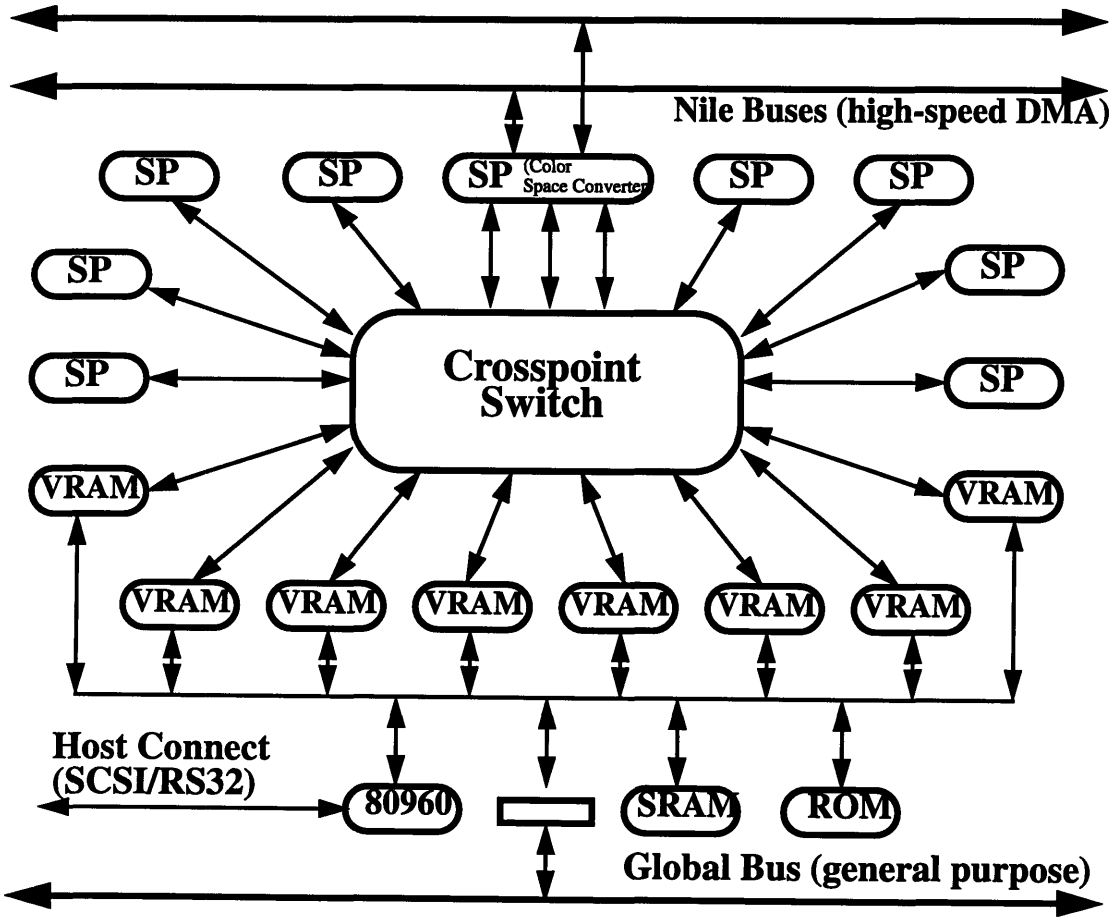

Figure 2-4: Cheops P2 Processor Board

Each P2 processor board, depicted Figure 2-4, utilizes a full crosspoint switch to connect eight memory units and up to eight stream processing units. Since six of these stream processors reside on removable submodules, the hardware can be easily upgraded by inserting new sub-modules. The hardware configuration can also be altered by selecting appropriate sub-modules.

The layered decoder will mainly use the remap and filter processors. The filter unit is used to perform both one-and two- dimensional filtering and multiplication of one stream by another [6]. The remap card, described in the following section, performs image warping and compositing.

2.2.1 The Remap/Composite Card

PHASES	Read	Write	Zbuffer_Write
MODES	Direct	Relative	Absolute

Table 2-1: Phases and Modes

The remap/composite sub-module, which for simplicity will be referred as the remap card, in the Cheops system is heavily used for this thesis. It has two internal memory banks, each is of size 1K by 1K. Memory bank 0 stores intensity values, while memory bank 1 is reserved to store z-values. For the purpose of this thesis, bank 0 is used more frequently. The remap card manipulates data by accepting data and vector positions. It stores and displays data according to the specified phase and mode. Table 2-1 describes the different types of phases and modes available.

PHASES:

Read: The remap card outputs what is stored in its internal memory, usually memory bank 0.

Write: The remap card stores or writes data into its memory.

Zbuffer_Write: The remap card requires a stream of data twice the width of the image

size. The data contains the values of intensity and z-value interleaved. Both values are in shorts (16 bits). And depending on the mode, the zbuffer_write phase also requires a vector of x, y positions. The remap card separates the stream and stores the intensity values in memory bank 0 and z-values in memory bank 1.

MODES:

Direct: In write phase, the remap card only accepts a stream of intensity data. X, Y positions do not have to be supplied. The data will be written into bank 0 as is. Similarly, the data from bank 0 will be read out as is in read phase.

Relative: In relative mode, the remap card accepts a stream of intensity data (i, j) and a stream of position offset vectors (x, y). The data will be written into or read from the position (i+x, j+y) in the internal memory. The size of the offset vector will be the same as the data. The (x,y) offsets are packed into one short where upper byte (8 bits) describe the x position and lower byte the y position.

Absolute: In absolute mode, the remap card again accepts both a stream of data and vector. The vector is twice as wide as the data. Each x and y position is a short, and they are interleaved in the vector. Depending on phase, the remap card will either write data at position (x, y) or read data from position (x,y).

The following figures shows a couple examples of how the remap card can be used:

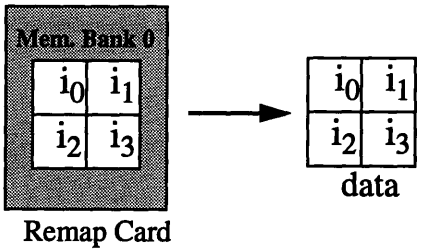

Figure 2-5: In Direct Read, data in the memory get read out as is.

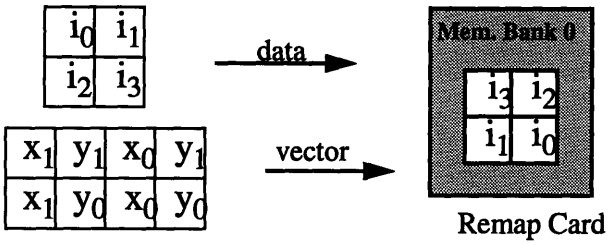

Figure 2-6: In Absolute Write, the remap card requires a set of data and vector positions. The data are written into the internal memory according to the vectors.

2.3 Cheops Software

The NORMAN resource management daemon is developed for Cheops. It manages the stream processors, and flood controllers in the memory unit by maintaining scoreboards and wait queues for each [7]. It uses the parameters provided by the user to set up and enable the hardware. NORMAN attempts to optimize usage of the resources and to parallelize operations. For instance, if three filter units exist in the system configuration, then the three color components (R, G, B) of a color image can be processed simultaneously.

The RMAN (Resource MANagement) library is the user interface for communication with NORMAN [8]. Since Cheops is a stream processing system, the RMAN library contains routines that creates structures which define basic stream operations. The library not only allows the user to define order of the operations and dependencies among the operations, it also gives the user the freedom to customize the operations. However, the user usually needs to specify vital parameters such as dimensions of the image, addresses of the source and destination, and filter taps. Once the user has defined the desired pipeline and the appropriate resources become available, the pipeline will be passed into NORMAN to be executed accordingly. The following shows an example of the dependencies and connections among elements that can be created using RMAN. And in Appendix A, a more detailed description of RMAN's usage will be described.

Given: elements A, B, C, D

RmanConnect(A, B, NULL); RmanConnect(C, D, NULL);

RmanDepend(A, C, NULL);

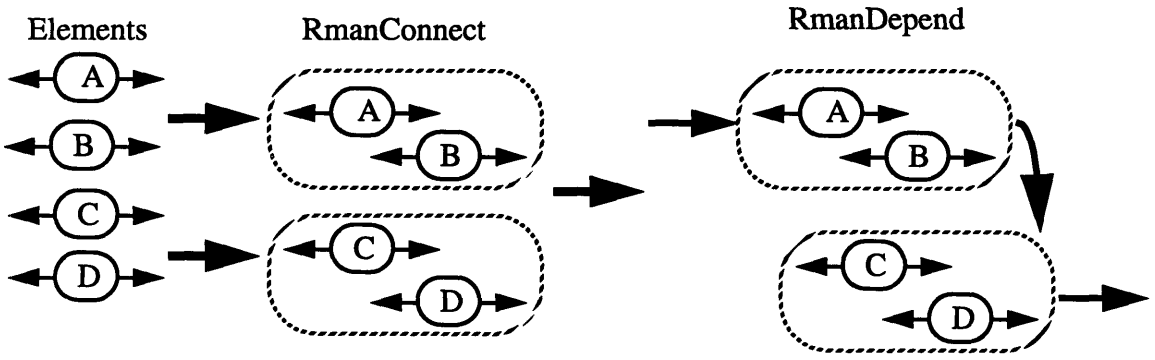

Figure 2-7: RmanConnect and RmanDepend

Elements A, B, C, and D can be a pipeline themselves or some kind of filter process. However, once they're connected as shown in Figure 2-7, then elements A and B can be performed concurrently, whereas C and D can only be processed together after the completion of A and B.

2.4 Previous Work

The generic decoder started by Brett Granger [6] contains modules that process 2D, 2 1/2-D, and 3-D objects. It also has a compositing unit written in assembly code and a scripting language.

2.4.1 3-D Objects

In the existing decoder, a 3-D object is represented as particle databases which is a collection of points where each has an intensity value, and an x, y, and z coordinate that is relative to the object's own origin. The points do not need to be in any particular order because each point is uniquely defined by its coordinate and value. Each 3-D object has its

own transformation matrix that specifies its position and orientation in the world. Once the object has been multiplied by the transformation matrix and thus becoming a 2 1/2-D object, it needs to be scaled correctly to correspond with the effect caused by perspective foreshortening [6]. However, perspective projection of 3-D objects was not implemented due to hardware multiplying/dividing limitation.

2.4.2 2-D and 2 1/2-D Objects

2-D and 2 1/2-D objects are treated identically by the decoder. They are handled similarly to a 3-D object in that they will be placed at a location in the world and transformed, usually scaling and/or translation according to the view parameters which have been establish [6]. Rotation is not allowed for these kind of objects but it can be simulated if several views of the object at different camera angles are available, an example would be *The Museum* movie.

2.4.3 Composite Unit

The compositing unit developed by Granger is written in software because compositing using Remap/Composite card is somewhat slower than the software version. Hardware compositing will be discussed in detail in section 3.3.

2.4.4 Scripting Language

The commands that are supported by the scripting language are: sequence control, user input control, object control, view parameter control, and display window control. The sequence control allows the image sequence to be displayed repeatedly, in a certain amount of time, and etc. The user input control allows the user to use Cheops knobs to change the parameters used in the program. Some of the object control commands are load object, place object at a certain position, and move the object from point A to point B. View param-

eter controls alter camera view, focal length, and etc. Finally display control changes the display window size and its top left-hand corner position.

Chapter 3

The Layered Decoder

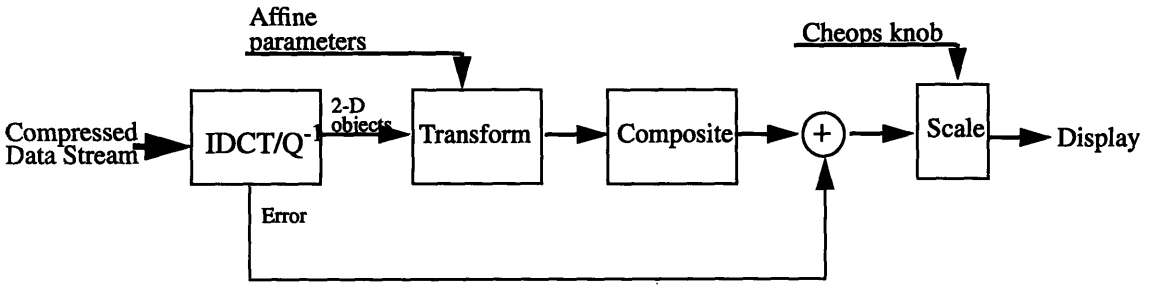

Figure 3-1: High level block diagram of the layered decoder.

The current structured video decoder system comprises of geometry transformations, 3D projections, scaling, positioning, and a software compositing unit [6]. The layered decoder allows more flexibility in the existing system by adding intra-coded unit, explicit transformation, hardware compositing, predication of later frames, and error signals. Thus, the processing decoding pipeline can support more types of structured video formats. A high level block diagram of the layered decoder is depicted in Figure 3-1.

The layered decoder accepts N layers, M frames of affine parameters and M frames of error signals. For a better understanding of the following sections, the MPEG flower garden sequence, three frames of which are shown in Figure 3-2, will be used as an example. The garden sequence is chosen as a layered image sequence because due to the lateral translation of the camera, the sequence undergoes a motion transformation where nearer regions translate faster than the farther regions.

Figure 3-2: 1st, 15th and 30th frames of the garden sequence

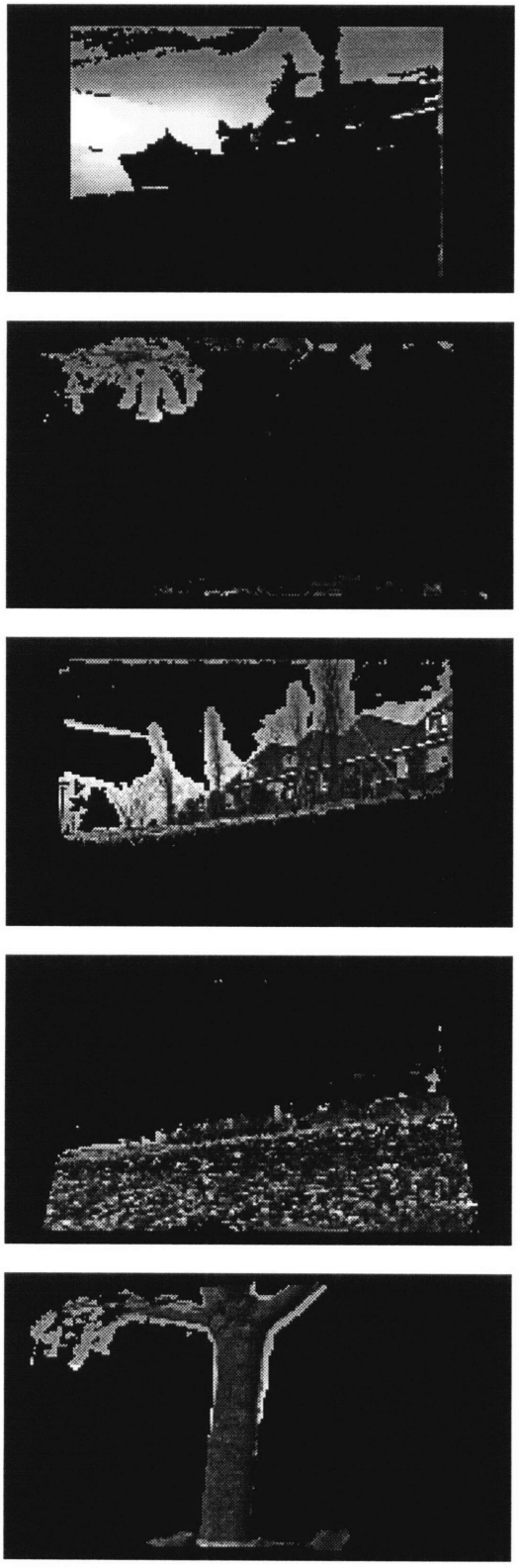

Figure 3-3: Five intensity maps of the garden sequence

For instance, in 30 frames of the garden sequence, the layered algorithm calculates five layers: sky, random noise, house, flowerbed, and tree. Figure 3-3 shows the five intensity maps associated with each layer. The black regions signal transparency in the map.There are also six affine parameters and one error signal frame associated with each frame of the sequence.

During the decoding process, the layered decoder operates in the following manner: First, the intra-coded unit performs IDCT and Q^{-1} on the compressed intensity maps and error frames. Afterwards, the explicit transformation unit uses the affine parameters to warp each intensity map, then the composite unit uses the alpha masks to composite the warped intensity maps into one single frame. The corresponding error frame is then added to the composited image to correct for the defects in the encoding algorithm. Finally, the resulting image is linearly interpolated and scaled if desired and displayed on screen. The whole decoding process is then repeated for the next frame of the sequence.

3.1 Intra-Coded Frame

Many structured video sequences might be compressed when entering the decoding pipeline. For example, a 2-D image sequence may be DCT-coded and quantized for compactness. This thesis introduces an IDCT and Q^{-1} unit to the structured video decoder processing pipeline. However, in order to understand the decoding of DCT-coded and quantized data, the encoding process must first be understood. The following two figures show both the encoding and decoding process.

Encoding

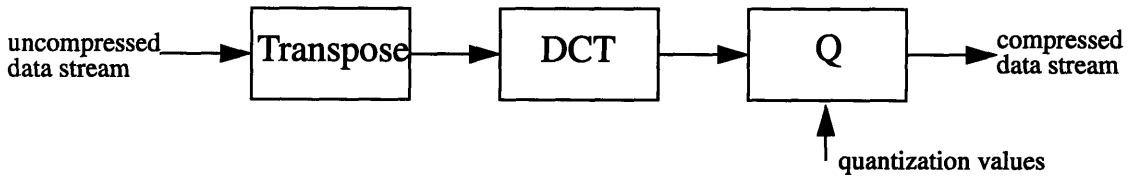

Figure 3-4: DCT and quantization encoding unit

Decoding

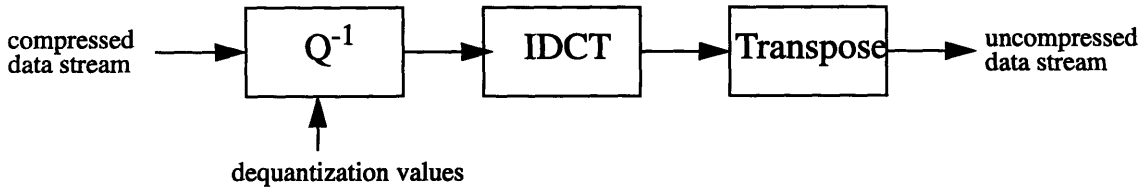

Figure 3-5: Dequantization and IDCT decoding unit

In the encoding process, the transpose engine in Cheops is set to DCT mode to first break the image into 8x8 blocks [16, 17, 18]. Then, the DCT/IDCT processor which resides on the remap card performs discrete cosine transform on the 8x8 blocks. Quantization is applied when the DCT-coded data are multiplied with quantization values.

3.1.1 IDCT/Q^{-1}

$$F^{Q}(u, v) = \text{Round}(F(u,v) / Q(u,v)) \quad (3\text{-}1)$$

or

$$FQ(u, v) = \text{Round}(F(u,v) * 1/Q(u,v))$$

$$F^{Q-1}(u, v) = F^{Q}(u,v) * Q(u,v) \quad (3\text{-}2)$$

If the image sequence that enters the decoder is DCT-coded and quantized, then it must first be dequantized to undo the previous operation, which is quantization. First, the quantization, described in Equation 3-1, occurs when the image in the frequency domain, F(u,v), is divided by a set of predefined quantization values, Q(u,v). Thus, to dequantize the image data, one needs to know the quantization values in order to perform the correct dequantization, described in Equation 3-2.

Quantization and dequantization prove to be difficult to implement using the Cheops system hardware. Although Cheops does contain a processor that performs stream multiplication, the processor is limited by fix-point multiplication which forces one of the two streams to contain numbers of one or less. In this case, the two streams involved are the

DCT-coded coefficients and the quantization values. The quantization values are chosen to be the stream that contains numbers one or less. All quantization values are thus divided by 2^8 or bit-shifted to the right by eight to satisfy that requirement. And once multiplication is accomplished, the appropriate bit-shifts are performed to reverse the quantized value manipulation. Once the 8x8 blocks have been dequantized and IDCT-coded, the transpose unit reconstructs the 8x8 blocks back to its original image format.

3.2 Explicit Transformation Unit

Explicit transformation refers to apply either spatial remapping (i.e. optical flow field, or a set of motion vectors) or parametric remapping (i.e. affine transformation) to 2-D objects [1]. It is useful for objects that require motion vectors to warp them to the next frame.

The transformation unit in the layered decoder takes as inputs, a layered sequence and a set of affine parameters associated with those layers. The transformation unit produces each warped image by warping the given intensity map with its corresponding affine parameters.

3.2.1 Warping

The word warp means to twist or distort a plane. The remap card can accomplish the warping effect in two ways:

Source_to_Destination (forward mapping): The src2dst approach will put every single source data somewhere in the destination. Thus, it is not guaranteed that every single destination pixel will have a value from the source image. The destination image will contain a number of black holes as a result. Error signals are needed to cover up those black holes or correct those mistakes. Figure 3-6 depicts the black hole situation. The remap card achieves the src2dst approach by using absolute write and direct read. In the figure, the goal is to scale a 2x2 image to a 3x3 image. By setting the remap card to absolute write, the vec-

tor is limited to supply position values for only four pixels or the number of pixels in the source. Then, the remap card is set to direct read to read out the contents of the 3x3 image from the memory. There will be five black holes in the destination image because the vector was not able to supply that many positions.

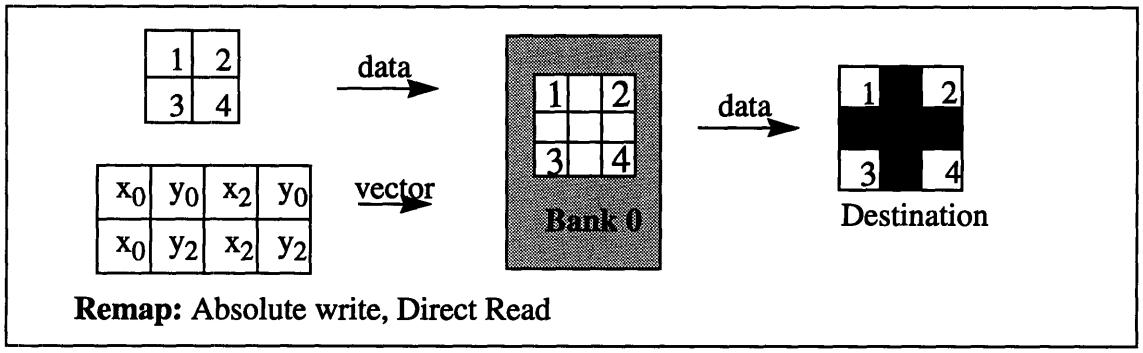

Figure 3-6: The pixel values in the source are mapped to the outer pixels in the destination, thus leaving a cross-shaped black hole in the middle of destination.

Destination_to_Source (backward mapping): The dst2src approach, depicted in Figure 3-7, does not create black holes in the destination image. Every single pixel from the destination image is guaranteed to have an intensity value from the source image. Thus error signals are not necessary unless to correct the mistakes from the warping algorithm The remap card uses direct write and absolute read to achieve dst2src. Once the data has been written into bank 0 of the remap card, then in absolute read mode, the vector supplies source coordinate values for every single pixel in the destination, thus leaving no black holes or no pixel unwritten.

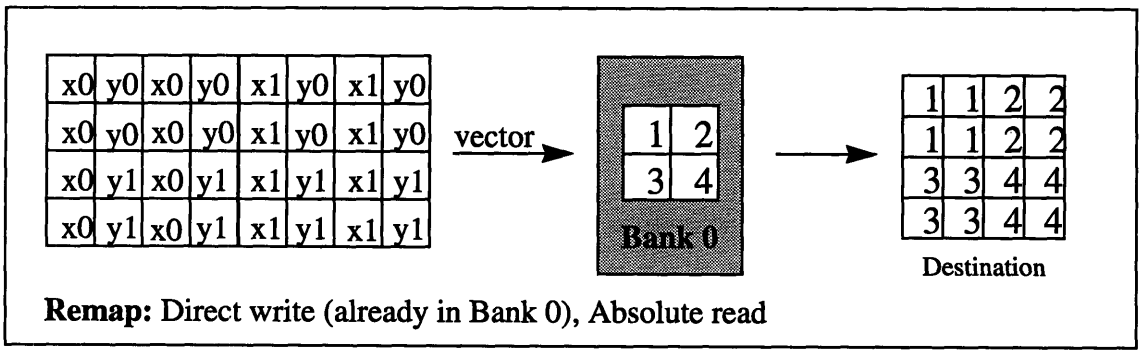

Figure 3-7: The data is directly written into bank 0 in remap card, and read out in absolute mode.

The layered decoder supports both types of warping. Relative mode was not chosen for either method because the positions in the vector are 8 bits each in relative mode, limiting the maximum x and y value to 255. Thus relative mode is not appropriate for image pixels that need to be moved more than 255 pixels in any direction. Since, it is not guaranteed that no pixel will be warped to a place greater than 255 pixels away, relative mode cannot be used. The mode is especially important for the vectors in the composite unit, described in the following section, because there are vector positions bigger than 255.

In the layered image sequence, the vector is created by interleaving $d_{xN}(x,y)$ and $d_{yN}(x,y)$ in equations 2-4 and 2-5. The RMAN library can easily calculate the vector using RmanDualFilter, RmanAdd, RmanDualFilterSetDelays and RmanDualFiltersetSampling. Details of creating the vector is described in Appendix A.

3.3 Composite Unit

Three different composite units have been developed for the structured video decoder. One is implemented in software, and the other two using the same piece of hardware. The first effort to create a composite unit utilized the remap card in z-buffer mode. However, the process was quite slow that an assembly program was created to perform compositing. Now, another method also using the remap card has been created in hope for a faster performance. The following two subsections describe the differences in the two remap-card compositing approaches.

3.3.1 Zbuffering

The remap card is designed with zbuffering or compositing using z-values in mind, where negative z-values force the data to be further away from the user and positive z-values make them appear closer. Thus, given a stream of data which contains intensity and z-value interleaved, and a stream of data which contains the corresponding x and y coordinates interleaved (if direct mode is not selected), the remap card warps the intensity data

according to the given x and y coordinates and gives them depth according to their z-values.

Unfortunately, compositing color objects requires a lot of data rearrangement and transfers. Each channel (i.e.: R, G, B) must be interleaved separately with the same z-values. And since a zbuffer write takes about 8 clock cycles compared to 2 cycles of a direct write, compositing using z-values occupies a lot of time.

3.3.2 Accumulation/Composite Unit

Another approach to accomplish the z-buffering effect without using z-values is the accumulation of layers method. The accumulator exploits the fact that remap card's memory bank size is limited to 1K to 1K. When remap card reads in a vector position less than zero or greater than 1K, it will not write the intensity value anywhere in its memory banks because the vector value is beyond the memory's range. Since the layered intensity maps have black pixels (value of zero) to represent transparency, a mask is created for each map where all black pixels are assigned to a large positive number (greater than 1024) and all pixels with non-zero intensity values are assigned to zero.

Each mask is then added to a stream of positions such that the final result is a vector stream which contains large positive numbers for transparent pixels and calculated warp positions for non-transparent pixels. Then, for every frame, data from each channel (i.e. R, G, B) of each layer and its corresponding vector are written into the remap card in absolute mode and read out in direct mode.

For example, to composite five layers, data from red channel of each layer and its corresponding vector are written into bank 0. Once, the writing process of all five layer is finished, data directly read out from bank 0 will contain the composited information of the red channel of that particular frame. Then all green channels are written in and read out, and finally, blue channels. A high level block diagram of this new accumulating/compositing method is shown in Figure 3-8.

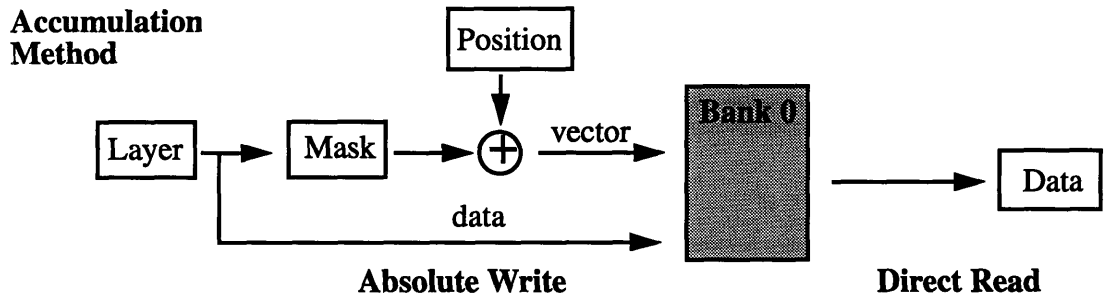

Figure 3-8: Generic Accumulation Method

Both src2dst and dst2src use this accumulation method to composite layers. However, they differ in the layer and position units in the accumulation method. The position units are different because the direction of mapping of data is opposite. One is from source data to destination location, the other from destination location to source location. Fortunately, since affine equations are linear, affine parameters are invertible. If the affine values are known for mapping frame 14 to frame 10. Then, by inverting the affine values, frame 10 can be mapped to frame 14. The following figures, Figure 3-9 and 3-10, show accumulation methods for src2dst and dst2src respectively, where vbuffer is the position stream calculated from the affine parameters.

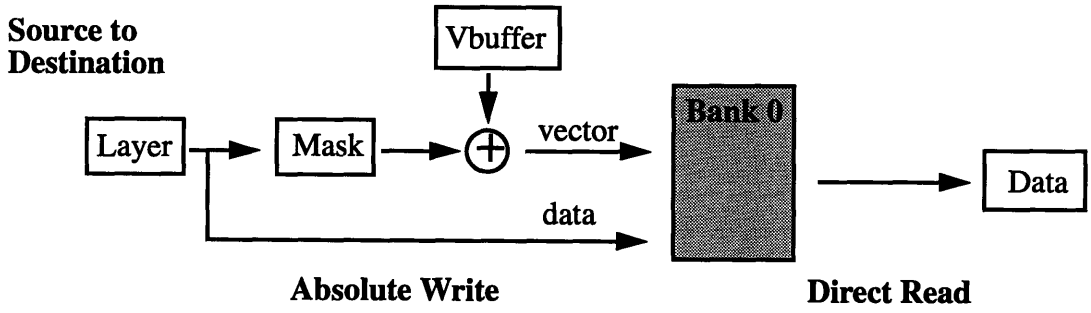

Figure 3-9: Src2dst using accumulation to composite

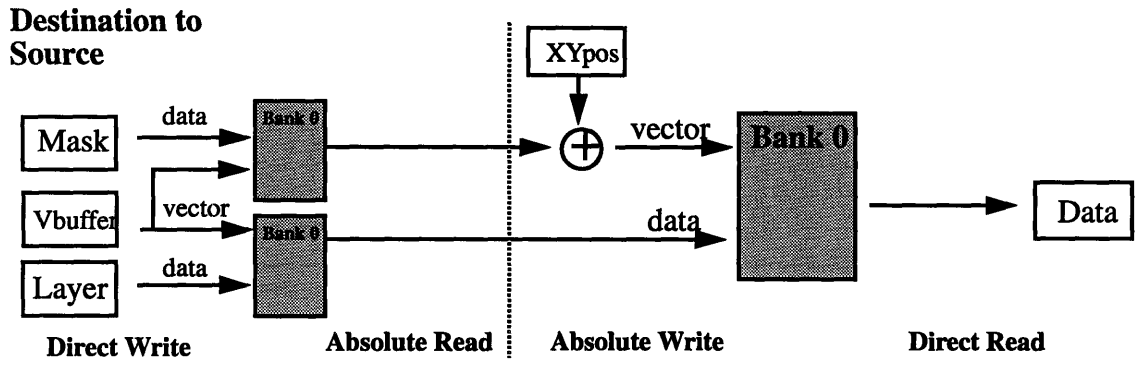

Figure 3-10: Dst2src using accumulation to composite.

Src2dst follows the generic accumulation method very closely. However, dst2src needs additional manipulation on its data and vector. In dst2src, both the layer and its corresponding mask need to be warped with vbuffer, the affine (inverted from the affine parameters used in src2dst) calculated positions. Since vbuffer supplies a source location for every single destination pixel, there will be no black holes. The result from warping the layer is the data going into the accumulation unit. The result from warping the mask has to be added with a stream of x and y coordinates interleaved, XYpos, to become the vector for the accumulation unit. It is important to remember that the mask contains values of zero for non-transparent pixels. Thus, when added to XYpos, the non-transparent pixels will contain the correct XYpos for its position going into the accumulation unit. And the transparent pixels will contain an even larger positive number.

3.4 Error Signals

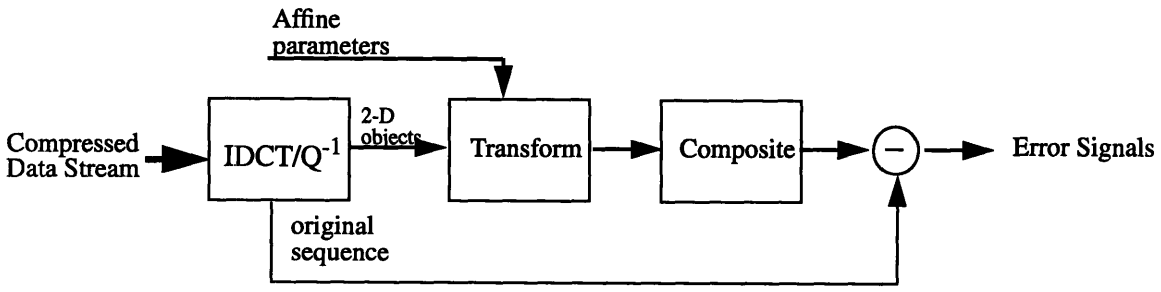

Figure 3-11: Generate Error Signals

Since the explicit transformation may not be sufficient to capture all of the image change, an error signal frame will be added to each composited frame to compensate for the image loss or undesired effect. The error maps can be created by subtracting the decoded image from the original scene, as shown in Figure 3-11. In other words, the encoder produces the error signal sequences and then the decoder adds the error sequences to the decoded image to compensate for the mistakes in the algorithm. It is important to realize that the error signals for the layered objects will be significant. The layered representation algorithm is designed to produce images that appear correct to the human vision system but not necessarily to the computer system.

For instance, if the transformed image is shifted to the left by one pixel, the human eyes can not detect that difference or mistake, but to the computer, the difference is tremendous. Figure 3-12 shows the first error frame for the garden sequence using dst2src method while Figure 3-13 shows the first error frame generated from src2dst method.

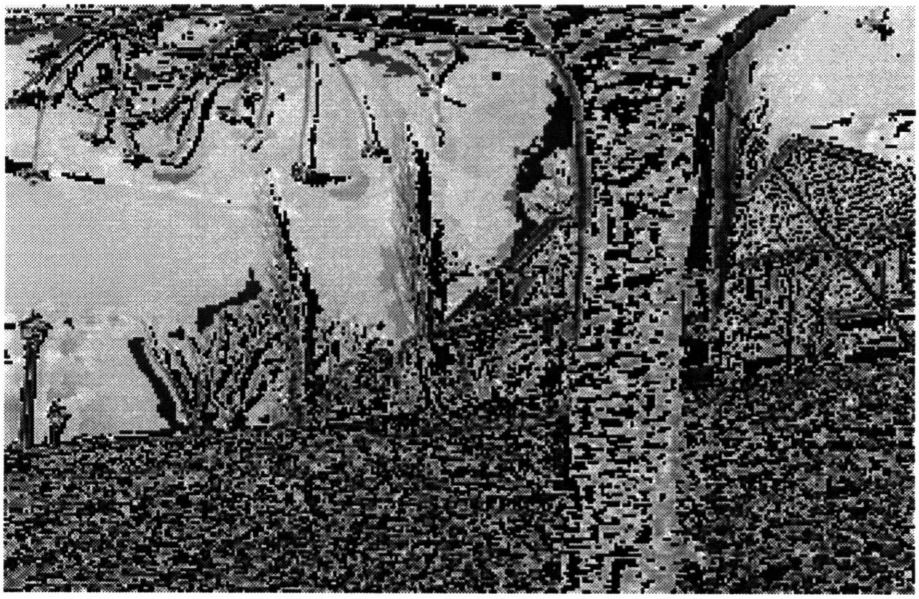

Figure 3-12: Error frame from dst2src

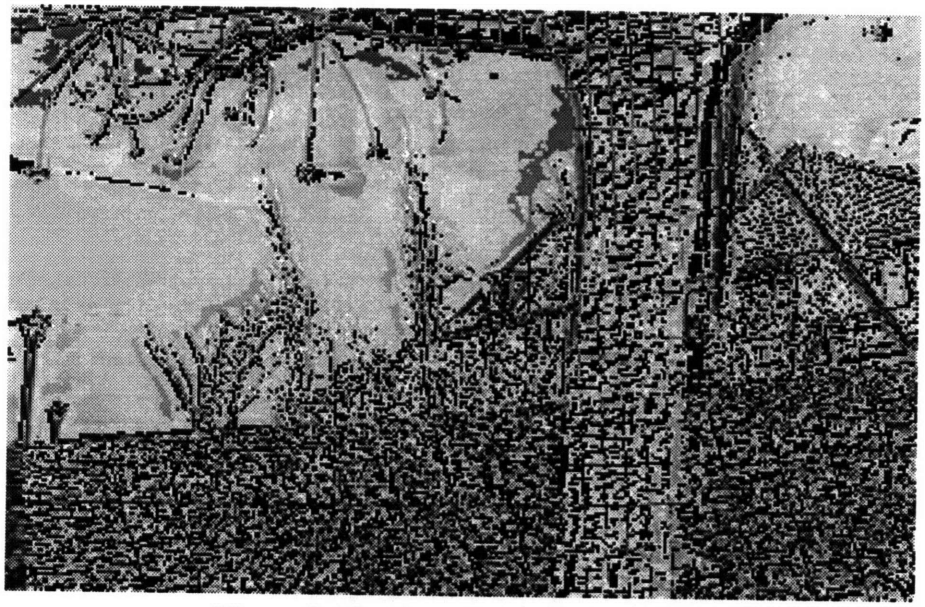

Figure 3-13: Error frame from src2dst

3.5 Scaling and Linear Interpolation

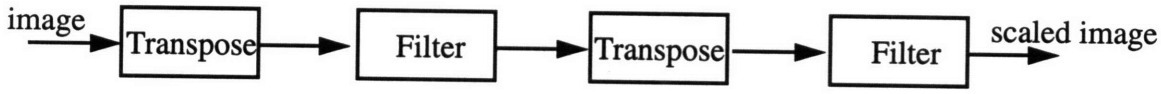

Figure 3-14: Scale and interpolation unit

A linear interpolation and scaling unit is created for the decoder to avoid aliasing in scaling of an image. Aliasing happens when an image is under-sampled and the result does not resemble the original image. Scaling and interpolation is applied after the error frame has been added to a composited frame. The user can use the cheops knob box to change the values of the scaling parameter.

The scaling and interpolation process, depicted in Figure 3-14, is as follows: The image is transposed vertically, filtered vertically, transposed back and filter horizontally. The filter unit in Cheops can scale by rational numbers and linearly interpolate images in one dimension. The interpolation implemented is the triangular interpolation but it could be anything that fits in a 1x16 kernel.

3.6 The Interface

Several new modules were created for the layer decoder. Appendix B contains a detailed description on how these functions can be used to create src2dst and dst2src methods.

Chapter 4

Experimental Results

Three sets of experiments were conducted on the layered decoder. The first set of experiments examines the image quality, specifically, it investigates the effect of dequantization and IDCT on the image and it compares the images from src2dst and dst2src approaches. The second set of experiments analyzes the timing performance of the layered decoder using different number and combination of processors. It again compares src2dst with dst2src. Lastly, a different layer sequence containing rotation of a ball is tried on the layered decoder.

4.1 Experimental Apparatus

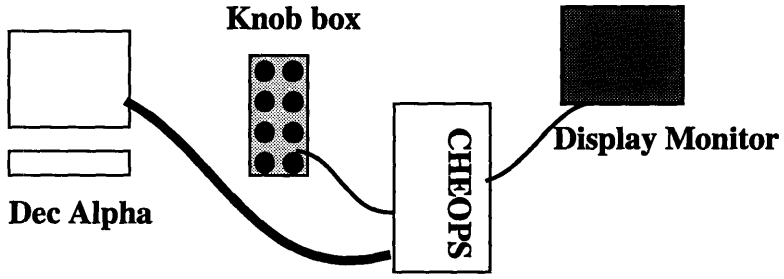

Figure 4-1: Experimental Apparatus

The experiments were conducted on a Dec Alpha which is connect to Cheops via a SCSI. A knob box and a display monitor are also connected to Cheops, shown Figure 4-1.

4.2 Image Quality

The dimensions of the garden sequence are 480x272. Several stages in the layered decoder might deteriorate the image quality. This section examines the causes and the results.

4.2.1 IDCT/Q^{-1}

As mentioned previously in Chapter 3, the purpose of the intra-coded frames is to decrease the amount of data that describe an image. The main advantage is that the reconstructed image is often hard to distinguish from the original. Figure 4-2 and 4-3 show the DCT-coded, IDCT-coded image, and the DCT, Q, Q^{-1} and IDCT reconstructed image, respectively.

The DCT-coded, IDCT-coded image appears quite similar to the original frame, at least to the human eyes, thus, demonstrating the DCT/IDCT processor is working efficiently. Yet, when quantization and dequantization are added in, there are some visible differences, as shown in Figure 4-3. The 8x8 blocks are slightly apparent in the image. The bit manipulations in quantization also have caused too much information to be lost from the original image, thus causing some deteriorations in the reconstructed image.

Additionally, this unit does have a problem performing DCT and IDCT on images that are composed mostly of black pixels. For instance, the transparency in the intensity maps. Black pixels, which have values of zero, are hard to code because they fall right in between positive and negative numbers. Since the DCT/IDCT processor does not offer an exact computation, sometimes the zeros become either positive or negative numbers. Thus, the reconstructed image differs somewhat from the original.

Lastly, error signals are extremely difficult to compress using DCT. Error signals, like noise, usually consist of high frequencies. In the spatial domain, error signals will look like impulses, whereas, in the frequency domain, they will be transformed into unity or a flat line. In order to not lose the content, all the high frequencies must be preserved which defeats the purpose of DCT. As mentioned previously, DCT codes low frequencies and discards high frequencies. Therefore, not much compression will occur on the error signal

frames, especially the errors from the src2dst method. As mentioned before, src2dst approach creates holes in the destination image, thus the differences in signals are even higher.

Figure 4-2: DCT, IDCT image

Figure 4-3: DCT, Q, Q^{-1}, IDCT image

4.2.2 Src2Dst

Figure 4-4: Original garden layers composited

Src2Dst is the warping method which creates holes in the destination image as mentioned in section 3.2.1. Figure 4-4 shows the original five intensity maps composited together. By using the src2dst approach, there are definitely black holes in the image, most visible in the tree. Figure 4-5 shows when no error signals are added, while Figure 4-6 is with the error signals added.

Due to the shorter distance between the tree and the camera in Figure 4-5, the tree appears bigger than the one in Figure 4-4, when the camera has panned away from the scene. Thus, to warp the given layers, Figure 4-4, to Figure 4-5, the tree has to be scaled larger or stretched wider, which implies there will be holes in the tree. The error signals are able to fill in most of those gaps. But the flaw is still slightly apparent. The reason is that the tree in Figure 4-5 is actually larger than the one in the original sequence. The error signals did try to fix the image defects but just at the wrong places since the sizes of the two trees (original and warped) didn't match exactly.

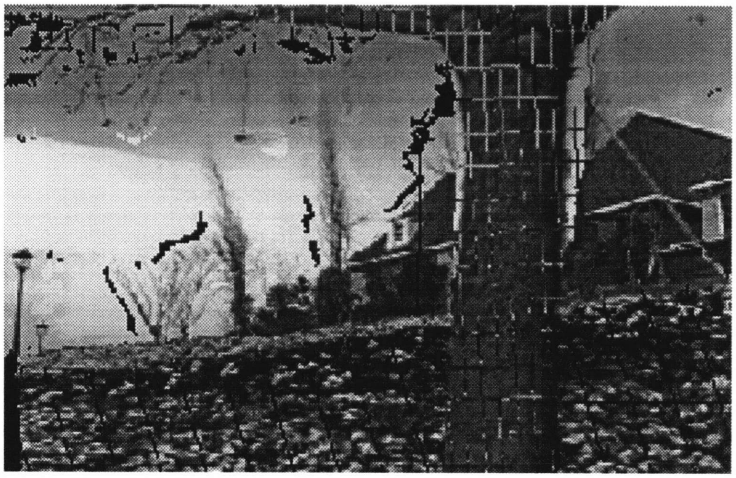

Figure 4-5: Src2dst: garden sequence with no error signals

Figure 4-6: Src2dst: garden sequence with error signals

4.2.3 Dst2Src

Dst2src method doesn't produce any black holes in destination images, as shown in Figure 4-7. And, once the error signals are added, shown in Figure 4-8, the result does not contain any apparent flaws. Overall, dst2src provides a better image quality than src2dst, but, the trade-off of having better pictures is slower processing time as described in the next section.

Figure 4-7: Dst2src: garden sequence with no error signals

Figure 4-8: Dst2src: garden sequence with error signals

4.3 Timing Analysis

The layered decoder is programmed with parallelism in mind. Parallelism as in if more than one transfer occur, the operations will be conducted simultaneously if more than one of the same type of processor are available. Two different types of timing analysis are conducted. The first one uses different combinations of number of processors to examine which combination gives the fastest performance to the dst2src and src2dst approaches.

The second offers a more detailed analysis on each unit of the layered decoder. The table below describes the average time each type of transfers, used in the layered decoder, takes.

Timing Analysis on Each Type of Transfers

Type of Transfer	Avg. Time (mics)	Type of Transfer	Avg. Time (mics)
RmanM1Transfer	4095	RmanRemap	17000
RmanDualFilter	5085	RmanRemapWrite	8460
RmanAdd	5100	RmanRemapRead	8500
RmanDCT	4485	RmanMultiply	2565
RmanTranspose	3100	RmanFilter	2550

Table 4-1: Timing analysis of each type of transfers

4.3.1 Different Combinations of Processors

The different combinations that are tested on the layered decoder are: 1 filter card, 1 remap card; 2 filter cards,1 remap card; and 1 filter card, 2 remap cards. Table 4-2 shows the approximate time to process one frame of the garden sequence which contains five layers. The processes that are involved to produce one frame are: loading one error frame and the five layers from M1 to P2, motion transformation, compositing, adding the error frame, scaling and interpolation. DCT and dequantization are not included as part of the process because they were performed before error frames and layers were loaded into M1. However, the performance of DCT and dequantization will be discussed in the next section. Additionally, appendix A offers a more detailed description of how each approach, src2dst and dst2src, utilizes the remap card and filter card.

Timing analysis of one frame of the garden sequence

	1 Filter, 1 Remap	1 Filter, 2 Remap	2 Filter, 1 Remap
src2dst	0.749 s	0.758 s	0.670 s
dst2src	1.092 s	0.933 s	1.027 s

Table 4-2: Different number of processors on one frame of the garden sequence.

Src2dst approach contains more filter transfers that contain parallel operations. Thus, when there are 2 filter cards available, the time to process one frame reduces significantly since there are more resources. However, there aren't any remap transfers that have parallel operations, hence there is no need to have 2 remap cards as shown in Table 4-2.

Dst2src approach does have parallel operations in both filter and remap transfers. 1filter card and 2 remap cards gives a faster performance than 2 filter cards and 1 remap card. Thus, it's important to understand the cost of each transfer, and how much time each unit takes in the layered decoder.

4.3.2 Analysis on Each Approach

In order to analyze the performances of src2dst and dst2src, the structure of each program must first be examined to understand how filter and remap cards are being used.

Src2Dst

The following is a pseudo-code of the skeleton of the approach:

```
IDCT/Q^-1 images, stores them in M1;
for (every frame) {          /* ramlog or timing starts */
    RmanM1Transfer of error frame;
    for (every channel) {
        for (every layer) {
            RmanM1Transfer of layer frame;
            if (red channel) {
                motion transformation;
                /* calculates vector and store it in vector[layer] */
                RmanRemapWrite layer.R with vector[layer];
```

```
            }
            if (green channel)
                RmanRemapWrite layer.G with vector[layer];
            if(blue channel)
                RmanRemapWrite layer.B with vector[layer];
        } /* end of every layer */
    if (red channel) {
        RmanRemapRead out Seq.R;
        Add in Error.R;
        }
    if (green channel) {
        RmanRemapRead out Seq.G;
        Add in Error.G;
        }
    if (blue channel) {
        RmanRemapRead out Seq.B;
        Add in Error.B;
        }
    Scale and interpolation;
    } /* end of every channel */
    /* if (curr_frame is done) end of ramlog */
} /* end of every frame */
```

The motion transformation unit calculates the warp vector. The composite unit is separated into write and read stages, represented by RmanRemapWrite (absolute mode) and RmanRemapRead (direct read). They are performed separately because the call depends on the channel (R, G, B). Therefore, even if there are two remap cards available, it wouldn't benefit this particular approach, as shown in Table 4-3. However, having two filter cards allows three sets of connections to perform simultaneously in the motion transformation unit, saving three operations. Since, motion transformation is executed N (number of layers) times, 3*N filter operations will be saved.

In addition, the cost for a RmanRemapWrite in absolute mode is 2 hub clock cycles. A RmanRemapRead in direct mode costs 2 cycles. Thus, the time to compute one frame using src2dst method is:

$$\text{Time} = (2*\text{num\_of\_layers} + 2)*\text{num\_of\_channels} \qquad (4\text{-}1)$$

Src2Dst Analysis on One Frame of Garden Sequence (in microseconds)

	1 Filter, 1 Remap	1 Filter, 2 Remap	2 Filter, 1 Remap
IDCT	26,535	20,060	25,875
Q^{-1}	13,995	13,950	10,185
Motion Transformation	62,700	62,715	47,375
Composite (write&read) of one channel	47,655	47,040	48,210
Add Error Frame	5070	5070	5070

Table 4-3: Timing Analysis of each src2dst unit.

Since the DCT/IDCT processor resides on the remap card, two remap cards will give a faster performance to DCT/IDCT than one remap card. However, quantization and dequantization only depends on RMAN operations on the filter card. Thus, quantization benefits more when there are more filter cards on P2. Yet, as shown in the table above, IDCT saves a lot more time than Q^{-1} because remap operations are more expensive than filter operations.

Dst2Src

```
IDCT/Q-1 each error frame and each layer, stores them in M1;
for (every frame) {     /* ramlog/timing starts */
    RmanM1Transfer of error frame;
    for (every layer) {
        RmanM1Transfer of layer frame;
        /* motion transformation calculates vbuffer,
            performs warping and creates vector. */
        motion transformation;
        } /* end of every layer */
    for (every layer)
        RmanRemapWrite TEMP[layer].R using vector[layer];
    RmanRemap Read out Seq.R;
```

```
    Add Error.R to Seq.R;
    for (every layer)
        RmanRemapWrite TEMP[layer].G using vector[layer];
    RmanRemap Read out Seq.G;
    Add Error.G to Seq.G;
    for (every layer)
        RmanRemapWrite TEMP[layer].B using vector[layer];
    RmanRemap Read out Seq.B;
    Add Error.B to Seq.B
    scale and interpolation;
    /* ramlog ends */
}; /* end of every frame */
```

Dst2Src is the less obvious case. First, the motion transformation is different from the one in src2dst. The motion transformation unit here calculates the vbuffer used in warping and the vector used in compositing. It also performs the actual warping. Once every layer has been warped, each channel is composited. This is accomplished by writing the red channel of every layer to the remap card using the appropriate vector, then, reading out the composited red channel. Similarly, all green channels are written into and read out from the remap card. Finally, the same process is applied to the blue channels. Again, error signals are added to the composited image. Scaling and interpolation are then applied. Table 4-4 shows the analysis of each unit using different combinations of processors. DCT, dequantization, and add error signals are not shown because they're independent of the type of method used.

Dst2src has operations that can be performed in parallel using both filter and remap cards. In the motion transformation step, 2*N filter operation can be saved if 2 filter cards are used, whereas, 3*N remap operations can also be save if 2 remap cards are used. As shown in Table 4-1, the filter operations (RmanDualFilter and RmanAdd) require less time than remap operations (RmanRemap, RmanRemapRead and RmanRemapWrite). Thus, using 2 remap cards results in a faster performance than 2 filter cards. Meanwhile, no parallel operations occur in the composite unit, thus the performances are similar.

Each RmanRemap call in the motion transformation unit uses direct write (2 clock cycles) and absolute read (4 clock cycles). Thus, the computation for one frame using 1

remap card is as follows:

Time = (4*6*num_of_layers) + (2*num_of_layers + 2)*num_of_channels. (4-2)

In the above equation, four RmanRemap calls, each costing six clock cycles, occur in each layer in the transformation unit. Then, in the composite unit, a RmanRemapWrite happens in every layer of each channel, plus one RmanRemapRead for each channel. Compared with equation 4-1, dst2src has one extra term from warping, whereas, src2dst warps and composites at the same time. Please refer to Appendix A for more details.

Dst2Src (in microseconds)

	1 Filter, 1 Remap	1 Filter, 2 Remap	2 Filter, 1 Remap
Motion Transformation	132,000	107,385	122,490
Composite (write&read) of one layer	49,995	49,020	49,005

Table 4-4: Timing analysis of each dst2src unit.

4.4 Another Sequence

The mobile sequence, which has dimensions 360x480, exhibits rotation in the plane, not present in the garden sequence, is also used to test the layered decoder. The sequence shows a train and a red with white poka-dot ball moving on a railroad track. The background is composed of a colorful animal-print wallpaper and a calendar. Even though the wall paper and the calendar are considered as two separate objects, they exhibit the same motion since the calendar is nailed to the wall. Thus, they are counted as one layer. The red ball and the train are the other two layers. All in all, there are five layers. However, since the other two layers are comprised mostly of black pixels (transparency), they were not used. Thus, only three layers, wall and calendar, ball, and train, are used.

Figure 4-9: 1st, 15th, 30th frame of mobile sequence

Figure 4-9 shows the 1st, 15th and 30th frame of the mobile sequence. The intensity maps from the 15th frame are used by the layered decoder as intensity maps of each layer. Figure 4-10 shows the result of using src2dst method to warp from the 15th frame to the first frame without adding error signals, while Figure 4-11 has the error frame added in. Figure 4-12 and 4-13 show the results of using dst2src. The results are similar to the garden sequence. Dst2src offers a better image quality, while src2dst gives a faster performance.

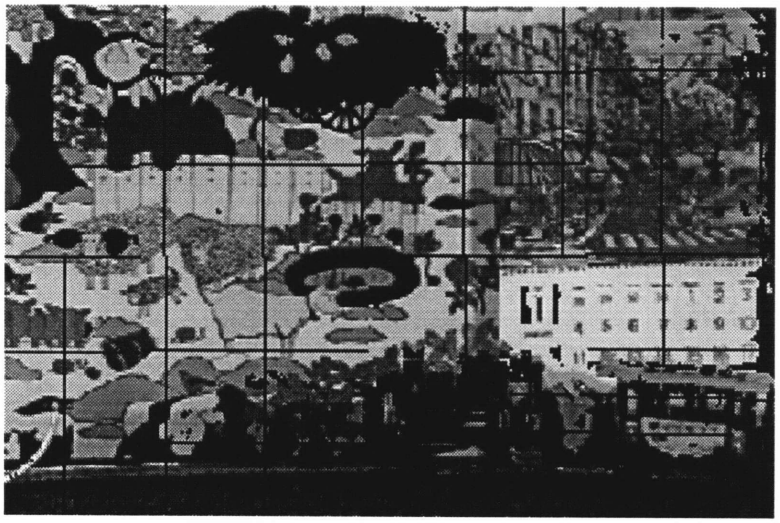

Figure 4-10: Src2dst: mobile sequence with no error signals

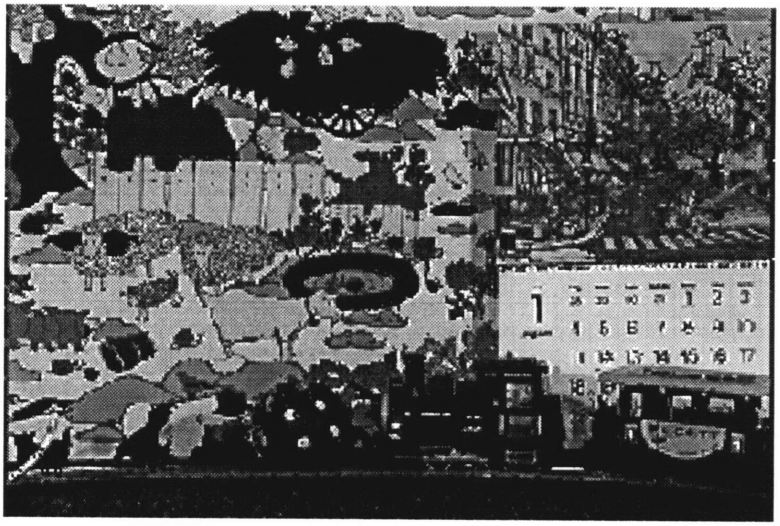

Figure 4-11: Src2dst: mobile sequence with error signals

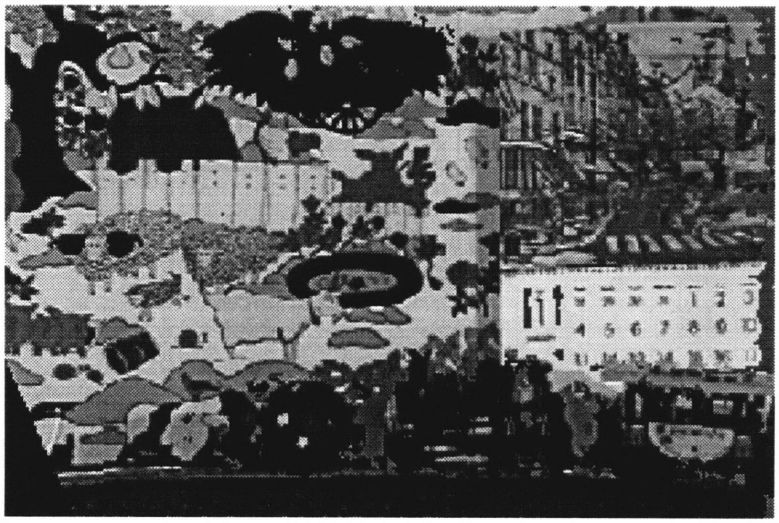

Figure 4-12: Dst2src: mobile sequence with no error signals

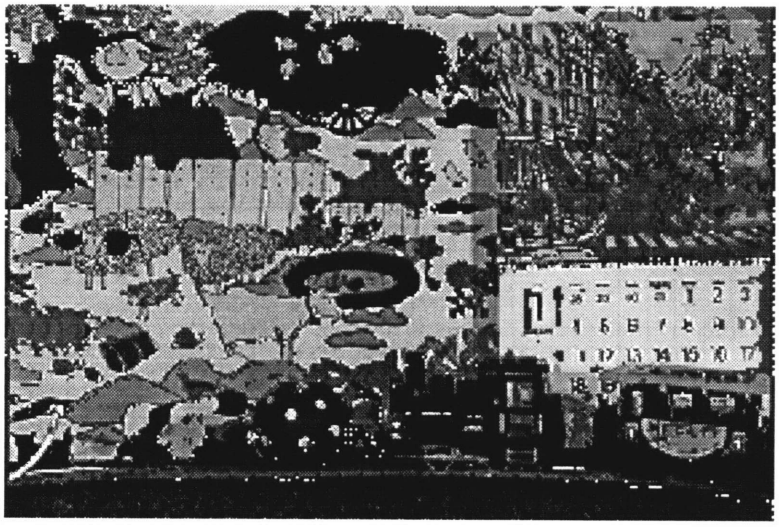

Figure 4-13: Dst2src: mobile sequence with error signals

Chapter 5

Conclusion

This thesis described the design, implementation, and use of a layered 2-D predictive decoder on the Cheops Imaging System. Experiments were conducted to demonstrate the types of video the layered decoder is capable of supporting. The image quality and timing performance were also analyzed. This chapter described the status of the generic structured video decoder with layered decoder feature added and concludes with suggestions for future work.

5.1 Status

The layered decoder has been functioning regularly for the past couple months. It is now being incorporated into the structured video script interpreter. The structured video decoder, shown in Figure 1-4, will then contain most of the features exhibited in Figure 1-5. However, some of the features in the structured video decoder will need improvements.

- The perspective projection in the 3-D decoder is still being done in software due to the difficulties of performing stream division in hardware stream processors. This problem prohibits faster performances when decoding 3-D objects.
- Although the layered decoder has the ability to scale up and down, the tremendous amount of memory it occupies on P2 to perform layer decoding usually leaves not enough memory for upward scaling.

- M1Transfer from M1 to P2 also proved to be a bit tricky to use. Each pixel transferred has 16 bits, with upper byte being the intensity value and lower byte being anything between 0 and 0xFF. Thus, when making any mathematical calculation, those non-zero values from the lower byte will affect the result. Normally, the effect is insignificant except with images that contain objects of the color white. White has the value of 255 for all three color channels. Thus, a small addition error can actually change the value from 255 to 0. To correct this mistake, the sequence can be gainbiased to prevent any pixel to have the value 255. Additionally, when using M1Transferred buffers to find the mask for the layered decoder, the programmer must note that intensity value of zero is not 0 but a value between 0x0000 and 0x00FF.
- The most important problem overall is that the performance of structured video decoder is limited by the number of stream processors present on P2. Three filter cards and three remap cards is the ideal case since color images (three channels) are usually used. Yet, only three processors can be put on P2 at one time. Thus, parallelism is not fully exploited on Cheops.

5.2 Future Work

The implementation of the structured video pipeline is just one aspect being developed on Cheops. Several exciting projects on Cheops are already in the works.

- The audio aspect of Cheops is currently being researched as another master thesis to compliment the visual aspect of Cheops. Structured video movies will be accompanied by synchronized structured sound to present a more complete story to the viewers.
- With the extensive work done on the making of *The Museum* movie comes the realization that a better scripting language is needed The scripting language needs to be more flexible, more user friendly and highly extensible. A more versatile way of controlling the interactions between function calls must be created to produce a faster performance.

- A programmable state machine is being developed to enhance Cheops's hardware programmability. For instance, the state machine can be programmed to perform stream division. Thus, perspective projection on the 3-D decoder done in hardware will be possible.
- Cheops will also have an additional feature other than audio and visual. An input card will allow Cheops to digitize movies in M1.

These features to be added to Cheops will increase the usability and enhance the real-time performance of Cheops. However, more studies must be done on the making structured video movies. For instance, currently, the scripts for the movies are rather simple because the relations among cameras, actors, and lighting are still in the trial and error stage. And with the addition of predication for later frames to the structured video pipeline, more methods other than camera panning should be investigated to develop movies that uses motion estimation. The system should take advantage of the intra-coded frames and motion transformation in order to greatly reduce the number of frames needed to be loaded and stored in M1.

Appendix A

Data Processing Pipeline

The data processing pipeline for src2dst and dst2src are significantly different. Dst2src has a more complicated pipeline as discussed in section 3.3.2. This appendix will first describe the easier src2dst and then attack the more challenging dst2src pipeline. In both cases, the reasoning behind the implementation and the parallelism when more than one stream processor is used will be highlighted. Finally, the appendix concludes with the elements common in both methods. The reader should be familiar with RMAN (resource management interface library) before reading this appendix.

A.1 Src2dst

The general concept behind src2dst is to find a vector for each layer of each frame such that when paired with the corresponding layer in remap card which is set to absolute write mode, the result from direct read is an image warped and composited with what was in the memory. Or simply, the vector indicates to the source where each image pixel should be placed in the destination. The reader should be familiar with Figure 3-9 when reading this section. The following figure shows the process of finding the vector.

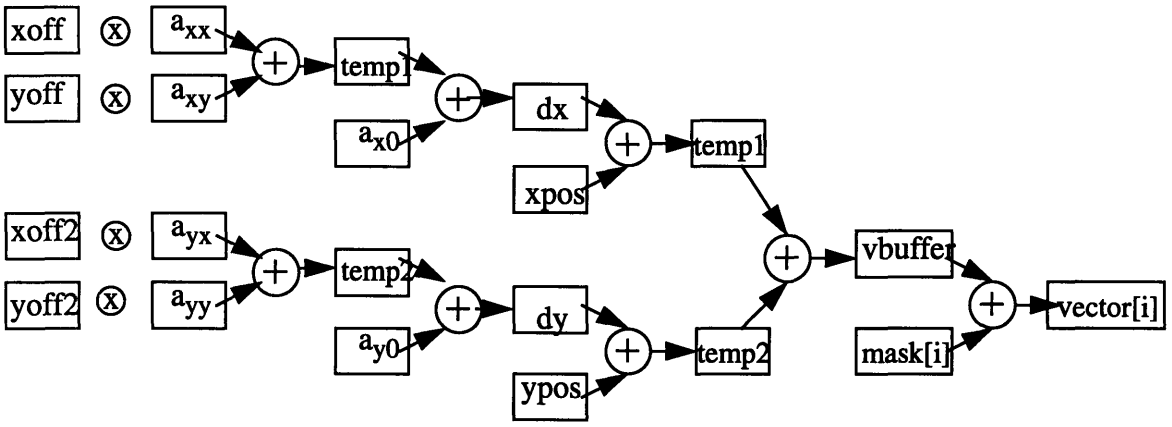

Figure A-1: Vector for src2dst method

a_{xx}, a_{xy}, a_{yx}, a_{yy}, a_{x0}, a_{y0}: The six affine parameters.

xoff: A buffer of size of the image which contains offsets in the x-direction. It shifts the origin of the image (top left hand corner) to the center of image.

yoff: Same as xoff but contains offsets in the y-direction.

xpos: A buffer that contains all the x coordinates.

ypos: A buffer that contains all the y coordinates.

vbuffer: A position buffer twice as wide as the image. It contains x and y interleaved warped-positions.

mask[i]: A position buffer twice as wide as the image. Whereever the image is transparent (black pixels), its (x,y) has a large positive number, and whereever the image is non-transparent, its (x,y) has a value of zero.

Basically, the final temp1 contains $d_{x,N}(x,y)$ and temp2 has $d_{y,N}(x,y)$, as described by Equations 2-4 and 2-5 respectively. They describe the coordinates in the destination the source is to be warped to. And vbuffer contains $d_{x,N}(x,y)$ and $d_{y,N}(x,y)$ interleaved. However, before supplying those destination coordinates to the remap card, vbuffer must be added with the corresponding layer mask to allow the black pixels to be transparent, refer to section 3.3.2.

The parallelism can be seen in the relations between the queue elements.

QueueElements for the Motion Transformation Unit:

F1 = RmanDualFilter(xoff, yoff, a_{xx}, a_{xy}, temp1);

F2 = RmanDualFilter(xoff2, yoff2, a_{yx}, a_{yy}, temp2);

F3 = RmanAdd(temp1, a_{x0}, dx);

F4 = RmanAdd(temp2, a_{y0}, dy);

F5 = RmanAdd(dx, xpos, temp1);

F6 = RmanAdd(dy, ypos, temp2);

F7 = RmanAdd(temp1, temp2, vbuffer);

RmanDualFilterSetDelays(F7, 0, 1);

RmanDualFilterSetSampling(F7, 2, 1, ZERO_PAD, 2, 1, ZERO_PAD, 1, 1);

F8 = RmanAdd(vbuffer, mask[i], vector[i]);

Connections and Dependencies:

RmanConnect(F1, F2, NULL);

RmanConnect(F3, F4, NULL);

RmanConnect(F5, F6, NULL);

RmanDepend(F1, F3, F5, NULL);

RmanConnect(F1, F3, F5, NULL);

RmanDepend(F1, F7, F8, NULL);

RmanConnect(F1, F7, F8, NULL);

If there is only one filter card available on P2, then all these eight operations (F1-F8) will have to be executed sequentially. However, two filter cards will allow F1 and F2 to be done in parallel, followed by F3 and F4, and then F5 and F6. Thus, only five operations will be done sequentially. On the side note, xoff and xoff2 are the same, similarly with yoff and yoff2. In order for F1 and F2 to be done in parallel, all six buffers must be on different memory banks. Thus, the only difference between xoff and xoff2 is that they are allocated in different memory banks on P2.

As seen in src2dst pseudo code in section 4.3.2, src2dst computes every layer of each channel, which means every layer of the red channel will be calculated before the green and blue channels. This particular approach is chosen because vector[layer] is the same for all three channels.

Thus, green and blue channels need not to recalculate vector[layer] for it's already been calculated by the red channel.

A.2 Dst2src

The main difference between src2dst and dst2src is that in dst2src, the image must be warped before going to the composite unit. This results in more frequent use of the remap card. First, to warp the image, dst2src first must find a vbuffer, depicted in Figure A-2, which contains all the source coordinates that destination image will receive its intensity values from. However, it's important to repeat that the six affine parameters in Figure A-2 are the inverse from the parameters in Figure A-1.

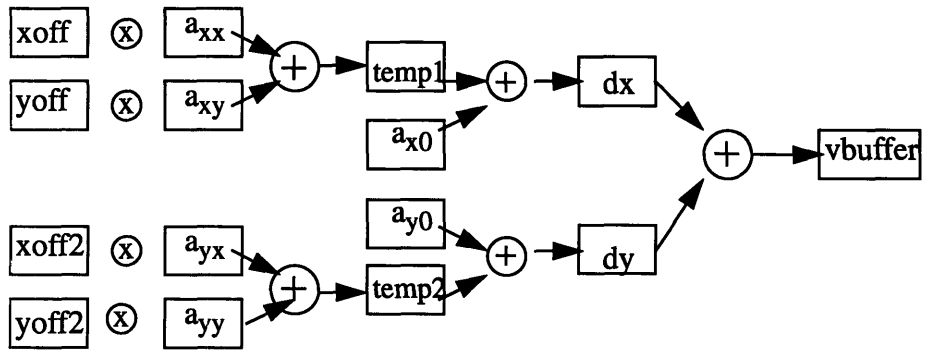

Figure A-2: Vbuffer for dst2src method

Once each layer and mask have been warped with vbuffer, leaving no pixels unwritten in the destination, the result from warping the layer is the data going into compositing, and the result from warping the mask has to be added with XYpos to become the vector for compositing. Please refer to Figure 3-10. XYpos is a buffer of xpos and ypos interleaved. By adding mask with XYpos, all non-transparent pixels in the image will be written into memory, whereas, all transparent pixels will not.

The parallelism is shown below:

QueueElements for the Motion Transformation Unit:

F1 = RmanDualFilter(xoff, yoff, a_{xx}, a_{xy}, temp1);
F2 = RmanDualFilter(xoff2, yoff2, a_{yx}, a_{yy}, temp2);
F3 = RmanAdd(temp1, a_{x0}, dx);
F4 = RmanAdd(temp2, a_{y0}, dy);
F5 = RmanAdd(dx, dy, vbuffer);
RmanDualFilterSetDelays(F5, 0, 1);
RmanDualFilterSetSampling(F5, 2, 1, ZERO_PAD, 2, 1, ZERO_PAD, 1, 1);

R1.R = RmanRemap(layer.R, NULL, dir_wmode, vbuffer, abs_rmode, warped[i].R);
R1.G = RmanRemap(layer.G, NULL, dir_wmode, vbuffer, abs_rmode, warped[i].G);
R1.B = RmanRemap(layer.B, NULL, dir_wmode, vbuffer, abs_rmode, warped[i].B);
R1.M = RmanRemap(mask[i], NULL, dir_wmode, vbuffer, abs_rmode, wmask);
Dup = RmanTransfer(tempbuff1, hdim, vdim, dy, NULL);
F6 = RmanAdd(tempbuff1, dy, hdim, vdim, vbuffer, NULL);
RmanDualFilterSetDelays(ADD2, 0, 1);
RmanDualFilterSetSampling(ADD2, 2, 1, ZERO_PAD, 2, 1, ZERO_PAD, 1, 1);
F7 = RmanAdd(vbuffer, xypos, buffer_hdim, buffer_vdim, vector[i], callback);

Connections and Dependencies:

RmanConnect(F1, F2, NULL);
RmanConnect(F3, F4, NULL);
RmanDepend(F1, F3, F5, NULL);
RmanConnect(F1, F3, F5, NULL);
RmanConnect(R1.R, R1.G, R1.B, R1.M, NULL);
RmanDepend(F1, R1.R, NULL);
RmanConnect(F1, R1.R, NULL);
RmanDepend(F1, Dup, F6, F7, NULL);
RmanConnect(F1, Dup, F6, F7, NULL);

In the steps that calculated vbuffer, F1 and F2 can be done simultaneously, followed by F3 and F4. Meanwhile, R1.R, R1.G, R1.B, and R1.M are connected to be done at the same time. However, even if there were four remap cards, those four operations still can't be done in parallel. There are twelve buffers involved and only six memory banks available. In order for true parallelism to happen, twelve memory banks have to exist on P2. In this particular case, layer.B, warped[i].R, and warped[i].G are in separate banks. Vbuffer is in the same memory bank as layer.R. Layer.G and mask[i] share the same bank, while warped[i].B and wmask occupy the a different bank.

Since the RmanRemap call is being executed as RmanRemapWrite and then RmanRemapRead, RmanRemapWrite of layer.R and layer.G are executed at the same time. RmanRemapRead of vbuffer and warped[i].G and RmanRemapWrite of layer.B happened simultaneously. Similarly, RmanRemapRead of vbuffer and warped[i].B and RmanRemapWrite of mask[i] are done in parallel. Three remap calls are saved when two remap cards reside on P2.

The structure of dst2src's pseudo code also differs from that of src2dst. The image must be warped before entering the composite unit. Then, every layer in the red channel is written into the remap card and read out, followed by the green channel and lastly the blue channel. Dst2src clearly has more steps involved and hence slower than src2dst. This is evident in the results in Chapter 4.

A.3 Common Elements

The two method mentioned above have several units in common: IDCT, Q^{-1}, Add Error Signals, and Scaling. The structure of those units are mentioned in Chapter 3, and the implementations are quite straightforward, with the exception of RmanMultiply in Q^{-1}. The programmer must remember to set the correct number of RmanFilter taps.

Appendix B

The User Interface

Several new modules were written to improve the current decoder. The following functions calls facilitates the use of the layer decoder. This Appendix contains a detailed description on each function.

DCT(short *src, int hdim, int vdim, short *dst):

The function DCT performs discrete cosine transform on src and stores result in dst.

quantize(short *src, short *table, int hdim, int vdim, short *dst):

The function quantize uses table to quantize the values in src and stores result in dst.

dequantize(short *src, short *table, int hdim, int vdim, short *dst):

dequantize uses table to dequantize the values in src and stores result in dst.

IDCT(short *src, int hdim, int vdim, short *dst):

IDCT performs inverse discrete cosine transform on src and stores result in dst.

create_mask(short *src, int hdim, int vdim, short *dst, int mode):

Depending on mode, DST2SRC or SRC2DST, create_mask uses src to create the appropriate mask in dst.

build_affine_pipe(void):

This function builds the pipeline used to perform affine transformation.

calc_affine_vector(float affine_param[6], int hdim, int vdim, short *vbuffer, int mode):

By using affine_param and the mode, this function calculates the vbuffer.

affine(short *src, short *vbuffer, int hdim, int vdim, short *dst, int mode):

affine uses src and vbuffer and calculates the warped version in dst.

build_layer_composite_pipe():

This functions build the composite pipeline for layered video.

layer_composite(short *src, short *vector, int hdim, int vdim, short *dst, int mode):

layer_composite composites src using vector and stores results in dst.

build_add_error_pipe():

This function building the pipeline for adding error frames.

add_error(short *src, short *error, int hdim, int vdim, short *dst):

add_error adds src and error and put the result in dst.

generate_error(short *original, short *seq, short *affine, short *error, int mode);

Generates error frames for the mode selected.

invert_affine(float *affine, float *inverse_affine);

Inverts affine and stores result in inverse_affine.

Bibliography

[1] **Bove, V.M. Jr., Granger, B.D., and Watlington, J.A., "Real-Time Decoding and Display of Structured Video,"** *Proceedings of the IEEE ICMCS '94*, **May 1994, pp. 456-462.**

[2] **Bove, V.M. Jr., and Watlington, J.A., "Cheops: A Reconfigurable Data-Flow System for Video Processing,"** *IEEE Transactions on Circuits and Systems for Video Processing*, **April 1995, pp. 140-149.**

[3] **Adelson, E.H., Wang, J.Y.A., "Representing Moving Images with Layers,"** *IEEE Transactions on Image Processing*, **Vol. 3, No. 5, September 1994, pp. 625-638.**

[4] **Wang, J.Y.A., Adelson, E.H., "Spatio-Temporal Segmentation of Video Data,"** *Proceedings of the SPIE*, **Vol. 2182, February 1994.**

[5] **Wang, J.Y.A., Adelson, E.H., "Applying Mid-level Vision Techniques for Video Data Compression and Manipulation,"** *Proceedings of the SPIE*: Digital Video Compression on Personal Computers: Algorithms and Technologies, **Vol. 2187, San Jose, February 1994.**

[6] **Granger, B.D., "Real-Time Structured Video Decoding and Display," SM Thesis, Massachusetts Institute of Technology, February 1995.**

[7] **Shen, I.J., "Real-Time Resource Management for Cheops: A Configurable, Multi-Task-**

ing Image Processing System," SM Thesis, Massachusetts Institute of Technology, September, 1992.

[8] Various authors, *Cheops Documentation*, MIT Media Laboratory internal memo, February 1993.

[9] Horn, B.K., *Robot Vision*, The MIT Press/Mc Raw Hill, Cambridge, Massachusetts, 1986.

[10] Oppenheim, A.V. and Schafer, R.W., *Discrete-Time Signal Processing*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1989.

[11] Bove, V.M. Jr., "Synthetic Movies Derived from Multi-Dimensional Image Sensors," PhD Dissertation, Massachusetts Institute of Technology, April 1989.

[12] Foley, J.D., van Dam, A., Feiner, S.K., and Hughes, J.F., *Computer Graphics, Principles and Practice*, Second Edition, Addison-Wesley Publishing Company, Reading, MA, 1990.

[13] Becker, S, and Bove, V.M. Jr., "Semiautomatic 3-D model extraction from uncalibrated 2-D Camera Views," SPIE Symposium on Electronic Imaging: Science & Technology, San Jose February 5-10, 1995.

[14] Kermode, R.G., "Building the BIG Picture: Enhanced Resolution from Coding," SM Thesis, Massachusetts Institute of Technology, June, 1994.

[15] Adams, J.F., "The Vidboard: A Video Capture and Processing Peripheral for the ViewStation System," SM Thesis, Massachusetts Institute of Technology, September, 1992.

[16] Huffman, D.A. "A Method for the Construction of Minimum Redundancy Codes," *Proceedings IRE*, Vol. 40, 1962, pp. 1098-1101.

[17] Wallace, G.K. "The JPEG Still Picture Compression Standard," *Communications of the ACM*, Vol. 34, No. 4, April 1991, pp 30-44.

[18] Evanco, K. "An Interactive JPEG Quantizer," MAS814 Final Project, MIT Media Laboratory.