

Copyright © 1994, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

**COMPOSITING AND MANIPULATION OF
VIDEO SIGNALS FOR MULTIMEDIA
NETWORK VIDEO SERVICES**

by

Shih-Fu Chang

Memorandum No. UCB/ERL M94/1

19 January 1994

**COMPOSITING AND MANIPULATION OF
VIDEO SIGNALS FOR MULTIMEDIA
NETWORK VIDEO SERVICES**

Copyright © 1993

by

Shih-Fu Chang

Memorandum No. UCB/ERL M94/1

19 January 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

**COMPOSITING AND MANIPULATION OF
VIDEO SIGNALS FOR MULTIMEDIA
NETWORK VIDEO SERVICES**

Copyright © 1993

by

Shih-Fu Chang

Memorandum No. UCB/ERL M94/1

19 January 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

Compositing and Manipulation of Video Signals for Multimedia Network Video Services

by

Shih-Fu Chang

Doctor of Philosophy in Engineering — Electrical Engineering and Computer Sciences
University of California at Berkeley
Professor David G. Messerschmitt, Chair

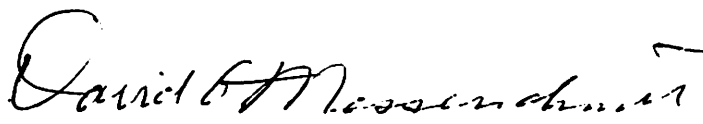
Multimedia network video services, like multi-point video conferencing and multimedia desktop editing/publishing, require real-time high-performance video signal compositing and manipulation. This dissertation investigates three different degrees of freedom for designing video compositing/manipulation systems: *feature, location, and data format*. Our goal is to provide a *systematic* approach to network video compositing/manipulation by integrating the explorations in all degrees of freedom, and by accounting for the interactions among themselves and with other multimedia technologies, in particular video compression.

Representative compositing features include geometrical transformations, linear filtering, opaque/semi-transparent overlapping, pixel multiplication, and arbitrarily-shaped (AS) video objects. We propose a *structured video* model, based on which we present several hierarchical structures for representing compositing functions, and study their restructuring properties.

We characterize various performance factors for different compositing locations throughout the network. We also propose a *shared distributed compositing principle* to match various user/service requirements and optimize the overall system performance in multimedia networks.

By processing less data and avoiding the data format conversion, the compressed-domain compositing approach has the potential to reduce the computational complexity for video compositing/manipulation, and thus the hardware cost. We derive equivalent compositing algorithms in the Discrete Cosine Transform (DCT) domain. We also propose a new decoding algorithm to partially decode Motion Compensated DCT-based compressed video signals to the DCT domain and apply our proposed DCT-domain compositing algorithms in the DCT domain. We compare the computational complexity of this proposed approach to the traditional uncompressed-domain approach both analytically and numerically. Its computational speedup depends on the specific compositing functions and the compression characteristics of video sequences.

We extend our compositing/manipulation techniques to arbitrarily-shaped (AS) image segments. In particular, we study efficient coding schemes for the internal image pixels and the boundary shape. We also propose a new joint approach for shape representation and anti-aliasing along the object boundary.

A handwritten signature in black ink, reading "David G. Messerschmitt". The signature is written in a cursive style with a horizontal line underneath it.

(Professor David G. Messerschmitt)

Table of Contents

	page
LIST of FIGURES	vii
LIST of TABLES	xviii
ACKNOWLEDGEMENTS	xx
CHAPTER 1 Introduction	1
1.1 Technology Framework for Multimedia Systems and Communications	1
1.2 Network Video Compositing	5
1.2.1 Existing Compositing/Manipulation Features	5
1.2.2 Envisioned Advanced Compositing Features	7
1.2.3 Challenges	8
1.3 Degrees of Freedom for Network Video Compositing	9
1.3.1 Different Compositing Features	9
1.3.2 Different Compositing Locations	10
1.3.3 Different Data Formats	11
1.4 Previous Work	12
1.5 Goals and Strategy of Approach	14
CHAPTER 2 Compositing Features	16
2.1 The Structured Video Model	16
2.1.1 Video Objects	17
2.2 Advantages and Disadvantages of Structured Video	20
2.3 Unary Spatial and Temporal Compositing Functions	24
2.4 Compositing of Multiple Video Objects	27
2.5 Compositing of Arbitrarily-Shaped Video Objects	29
2.6 Representations of Compositing Functions	32
2.6.1 Expressions	34
2.6.2 Trees	36
2.6.3 Object-Based Priority and Ordered Lists	38

2.6.4	Constraints	43
CHAPTER 3	Distributed Network Compositing	45
3.1	Background	46
3.1.1	Characteristics of ATM Broadband Networks	46
3.1.2	Related Work on Multi-Source Multi-User Connections	47
3.2	Video Compositing throughout the Network	51
3.2.1	An ATM-Based Multimedia Network Model	51
3.2.2	General Benefits and Drawbacks of Each Possible Compositing Site	52
	<i>Video Source (location A)</i>	53
	<i>Network (location B)</i>	53
	<i>Customer Private Networks (location D)</i>	55
	<i>Final Video Display (location E)</i>	55
3.2.3	Principle of Shared Distributed Compositing	56
3.3	Mapping of Compositing Functions to Network Resources	59
3.3.1	Optimal Resource Allocation — An Open Problem	59
3.3.2	Transformations of Compositing Function Representations	60
3.3.3	Possible Transformations on Trees	64
3.3.4	Possible Transformations on Ordered Lists	67
3.4	Impact on ATM Broadband Network Design	68
3.5	Primitive Performance Analyses	70
3.5.1	Received Video Quality	71
3.5.2	Synchronization and Latency	76
3.5.3	An Example	81
3.6	Summary	82
CHAPTER 4	Video Compositing in the Compressed Domain	83
4.1	Background	84
4.1.1	Review of Motion-Compensated DCT-Based Compression Algorithms	84
4.1.2	Domain Notations	86

4.2	The Conventional Uncompressed-Domain Approach	87
4.3	A New Approach — Compressed-Domain Compositing	88
4.4	Video Compositing in the DCT Domain	90
4.4.1	Overlap	91
4.4.2	Pixel Multiplication	91
4.4.3	Translation	93
4.4.4	Linear Filtering	97
4.4.5	Scaling	97
4.5	Compositing MC-Compressed Video	98
4.5.1	Difficulties for MC-Compressed-Domain Compositing	98
4.5.2	Simplification of MC Data Recalculation	100
	<i>Reducing the Frequency of Recalculation</i>	101
	<i>Calculating New Motion Vectors by Inference</i>	103
	<i>Error Propagation</i>	104
	<i>Simulations and Performance Comparisons</i>	106
4.6	Compositing MC-DCT Compressed Video	108
4.6.1	Partial Decoding of MC-DCT Video	108
4.6.2	Compositing MC-DCT video in the DCT Domain	111
4.6.3	Performance Analyses	113
	<i>Computational Complexity</i>	113
	<i>Image Quality</i>	126
	<i>Other Performance Considerations</i>	129
4.7	Summary	130
CHAPTER 5	Arbitrarily-Shaped Video Objects (ASVO)	132
5.1	Transform Coding of Image Pixels	132
5.1.1	Brute-Force Full-Block Transform	134
	<i>Mirror Image Extension</i>	136
5.1.2	Shape-Adaptive Approach	137
	<i>A New Problem Formulation — Shape-Projected Subspace</i>	138
	<i>Successive Approximation Algorithms Revisited</i>	140
	<i>Adaptive Transform Bases</i>	145
5.1.3	Performance Comparison	148

5.2	Shape Representation	152
5.2.1	Review of Different Encoding Methods	152
5.2.2	A Joint Approach to Shape Coding and Anti-Aliasing	155
5.2.3	Iterative Transformations of Arbitrarily-Shaped Video Objects	159
5.3	Summary	163
CHAPTER 6	Conclusions and Future Work	165
6.1	Conclusions	165
6.2	Future Work	170
6.2.1	Optimal Resource Mappings for Distributed Compositing	170
6.2.2	Video Compression and Compositing Co-Design	171
6.2.3	Model-Based Video Coding	172
6.2.4	Video Format Conversion	173
REFERENCES		175
APPENDIX		183
A.1	Multiplication-Convolution Theorem for the DCT	183
A.2	Transform Domain Filtering	184
A.3	Computational Complexity of DCT-Domain Compositing Operations	186
A.3.1	Matrix multiplication	187
A.3.2	Scaling	188
A.3.3	MC and Inverse MC in the DCT domain	189
A.3.4	Pixel-Wise Translation	189

LIST of FIGURES

- Figure 1-1** (a)A typical multi-point multi-media service on broadband networks. (b)Possible presentation scenarios on user screens. (c)Advanced technologies needed for implementing these cutting-edge applications. 4
- Figure 1-2** (a)Today's typical video conferencing pictures. (b)Concentrated video compositing architecture. (c)distributed video compositing architectures. 6
- Figure 1-3** Envisioned advanced compositing features in future video services 7
- Figure 1-4** A composited video can contain video signals originated from different locations. For example, the image of the weather reporter may come from broadcast video, but the map and logo can be produced locally. 8
- Figure 2-1** Example of illegal pair of video objects, because object B cannot be treated as a unit when composited with another object. (from [Chen93a]) 18
- Figure 2-2** Multi-pass implementation of image rotation. (a)original (b)after y-direction shearing (b)after x-direction shearing. 25
- Figure 2-3** Examples of composited video objects using basic compositing functions in table 1. (from [Chen93a]). 27
- Figure 2-4** Using compositing scripts to specify time-dependent compositing rules among two video objects. An alternative approach is to divide objects into smaller ones (in the temporal sense). The compositing rules for each video object will then become static for the whole dura-

- tion of the video object. 28
- Figure 2-5** The original jagged rabbit and its anti-aliased version. (a) original raw image (b) enlarged area of original image (c) anti-aliased image (d) enlarged area of anti-aliased image. 30
- Figure 2-6** Input/output relations for unary and binary compositing functions of ASVO's. (a) many unary compositing function such as a geometrical transformation treat the α channel just like color channels. (b) But most binary compositing functions have special compositing rules for the α channel. 31
- Figure 2-7** Relationship between different types of representations for compositing functions. 34
- Figure 2-8** A tree representation for the compositing function defined in Equation (2-8). 37
- Figure 2-9** (a) A compositing function including an indecomposable ternary operation and a binary operations. (b) The corresponding tree architecture. 37
- Figure 2-10** There could be different implementations for the same compositing function. (a) the desired effect (b) overlap the first two objects first (c) overlap the last two objects first. 38
- Figure 2-11** Two different tree representations for the same compositing function. Each internal node is associated with an overlap operation (without explicit notations). $+$: opaque overlapping, \oplus : semi-transparent overlapping, U_i : unary transformation, V_i : video objects. 39
- Figure 2-12** The ordered list model representing a hierarchical compositing function. Here we assume only two typical multi-object compositing functions, opaque and semi-transparent overlap. Dotted boundary

encircles video objects with the same priority. The first element, V_0 , represents the virtual background. 43

Figure 3-1 (a) multicasting connections (b) multi-source single user distributed compositing. Combination of these two situations produces the general multi-source multi-user connections. 48

Figure 3-2 A generic ATM-based model for the multimedia compositing systems. CPN represents Customer Premise Network, NPI represents Network-Premise Interface, and S1-S4 are different multimedia data sources. Locations A — E stand for source, network node, intermediate node outside networks, local network node, and destination respectively. Third-party vendors can provide compositing hardware in locations C or D. Some customer premise equipments (labelled as D) can be shared by customers in the same CPN. 52

Figure 3-3 A simple example of multi-point video conferencing. Different computational resource allocation schemes result in different levels of flexibility of user controls. In (a), a central compositing unit (CU) produces a unique presentation. In (b), multiple CU's produce more presentation choices and thus users can have higher control flexibility. 54

Figure 3-4 An example of shared distributed video compositing. Several end users share a common compositing task and complete their displayed scenes in a distributed way.

P: compositing processors. 58

Figure 3-5 A challenging open problem — optimization of mapping compositing functions to actual network resources. 60

Figure 3-6 Possible transformations on the tree structure. (a)associative binary

functions (b_1 - b_2)distributive binary functions (b_3)distributive unary functions with respect to binary functions (c_1)commutative binary functions (c_2)commutative unary functions. 66

Figure 3-7 The corresponding action of applying the distributive property on the implementation tree, from (a) to (b) — merging, from (b) to (a) — splitting. 67

Figure 3-8 If the compositing unit is located in an intermediate node (either within or outside the network), the compression(T)—decompression(T^{-1}) process needs to be duplicated, including the lossy part. 71

Figure 3-9 (a)The quantizer which uses the center point of the decision interval as the reconstruction value. d_i : decision levels, r_i : reconstruction levels. (b) A possible situation in which more distortion is added in the second round of the MC compression process. P_i : reference pixels, O_1 : current pixel, O_2 : recovered pixel after inverse MC process. 73

Figure 3-10 Perform down scaling operation at different locations — source, user, and intermediate site 74

Figure 3-11 Reconstructed images by down scaling at different locations. The compression scheme includes DCT and quantization. (a) uses a uniform quantization step, 36, for all AC DCT coefficients (b) uses the quantization table listed in the JPEG standard. 75

Figure 3-12 A general architecture for intermediate-site compositing unit, which basically includes buffers for input signals, the compositing processor, and the transmitter performing packetizer and transmission. 77

Figure 3-13 A fixed output rate synchronization scheme to compensate the bursty input traffic at the compositing unit. B : Buffer size, T_0 : control

- time. 78
- Figure 3-14 A fixed-output-rate synchronization scheme for multiple variable-rate input streams. 79
- Figure 3-15 Compositing at different locations and their buffer requirement. 80
- Figure 4-1 Video compression methods using the Discrete Cosine Transform algorithm and the variable length code. IDCT: inverse DCT, Q: quantization, VLC: variable length code. 84
- Figure 4-2 The Motion Compensation algorithm and its inverse algorithm. DM: the displacement measurement procedure. FM: frame buffer. Q: quantizer. Q-1: inverse quantizer. 85
- Figure 4-3 Block diagram for hybrid compression methods including the MC algorithm and the DCT algorithm. DM: displacement measurement, FM: Frame Memory. 86
- Figure 4-4 Two different approaches to compressed-input compressed-output video compositing. (a) uncompressed-domain approach (b) compressed-domain approach. 88
- Figure 4-5 Compositing two video sequences in the DCT domain. (*: this procedure is not necessary if the block structures of two input images are matched.) 90
- Figure 4-6 Re-assembling the image blocks of video object B with respect to a new block structure, which mismatches object B's original block structure. The highlighted block, B', is a new block of object B after block re-assembly. It consists of the contributions (B₁₃, B₂₄, B₃₁, and B₄₂) from four original neighboring blocks (B1-B4). 94
- Figure 4-7 Using matrix multiplication to extract a subblock and translate it to the opposite corner. 96

- Figure 4-8** An example showing the problem for compositing directly in the MC domain. Part of the motion area of the background image in the directly-affected area is replaced by the foreground image. Pixels in the indirectly affected area are changed because their reference pixels may be modified through error propagation. d_max is the number of search positions in each direction in the MC algorithm. 99
- Figure 4-9** An example showing the problem for compositing in the MC domain. New prediction errors cannot be calculated in the MC domain since four original reference blocks could be transformed to different blocks in the new image, which is scaled down ($1/2 \times 1/2$) from the original image. 100
- Figure 4-10** The test video sequence for the MC recalculation algorithms. The background and foreground images are originally MC-compressed. The composited image is also MC-compressed by using the proposed 2-point simplified searching algorithm. The displayed image is reconstructed from the MC-compressed composited output. This is frame 3 in Figure 4-14 shown later, which suffers the most severe SNR loss due to this simplified search algorithm. 102
- Figure 4-11** Number of image blocks in the directly-affected area which requires recalculation of the motion vector. The total number of blocks in the directly-affected area is 82. 103
- Figure 4-12** Reducing the number of searched locations to two by using Jain and Jain's assumption. B is the location of the current image block, D is the optimal reference location, and D1 (D2) is the crossing point when we move away from D horizontally (vertically). Based on Jain and Jain's assumption, D1 and D2 are the optimal reference locations

- within the uncovered motion area of B. 104
- Figure 4-13 Number of background pixels outside the directly-affected area affected by error propagation (for the test video sequence shown in Figure 4-10). The number increases because of the motion from the overlap area towards the background object in our test sequence. 105
- Figure 4-14 The average SNR among recalculated blocks within the directly-affected area. The top curve is the original input MC-compressed video, the middle curve uses the full search algorithm in MC recalculation, and the bottom curve uses our proposed simplified 2-point displacement measurement method. Correction of error propagation is used in both the middle and bottom curves. 107
- Figure 4-15 The average SNR among pixels affected by error propagation. We use the full search approach to do the motion measurement here. 108
- Figure 4-16 A new decoding algorithm for the MC-DCT compressed video. The inverse MC algorithm is performed before the inverse DCT, which is opposite to that used in traditional decoders. MCD: MC in the DCT domain. 109
- Figure 4-17 The MC optimal reference block generally overlaps with four blocks in the DCT block structure. 110
- Figure 4-18 Compositing two MC-DCT compressed video sequences in the uncompressed domain. We decode both input videos fully back to the uncompressed domain, composite them pixel by pixel, and then encode the composited video to the compressed format. (d_1, e_1) and (d_2, e_2) are (motion vector, prediction error) for input video streams.

(d', e') is for the composited output video. 112

- Figure 4-19 Compositing two MC-DCT compressed video in the DCT domain. We convert input video to the DCT compressed domain, composite them in the DCT domain, and then convert the composited video to the MC-DCT format. 112
- Figure 4-20 Experimental results of non-zero motion vectors. (a) “Miss USA” sequence (b) “Salesman” sequence.
 α_2 : the percentage of the motion vectors whose components are non-zero in both the x and y directions.
 α_1 : the percentage of the motion vectors whose components are non-zero in only one direction (x or y).
 $\alpha_0: 1 - \alpha_2 - \alpha_1$ 120
- Figure 4-21 original pictures for different compositing scenarios (a) Scene 1 (b) Scene 2 (c) Scene 3. 123
- Figure 4-22 Reconstructed images of Scene I composited in (a) the spatial domain (b) the DCT-compressed domain. All input video and output composited video streams are MC-DCT compressed. 128
- Figure 4-23 A pipelined hardware architecture for transform filtering (proposed by Lee and Lee [Lee92]). 129
- Figure 5-1 Production and processing (e.g., manipulating or compositing) of arbitrarily-shaped video objects. 133
- Figure 5-2 An example AS image segment and the grid lines which separate the image into small blocks. Boundary blocks have part of pixel values defined only. The block structure is for demonstrative purpose and is not of accurate scale. 133
- Figure 5-3 Find the optimal pixel values outside the boundary of image segment

- P, so that the transform spectrum has the most compact energy spectrum. 135
- Figure 5-4 Fill the outside redundant region with the mirror image of the internal contents. (a) original segment. (b) the segment size equals one half of the block size. (c) apply the mirror image recursively when the segment size is not one half of the block size. 137
- Figure 5-5 (a) A partially-filled image block in a 4x4 area. A canonical basis of the subspace is shown in (b). 138
- Figure 5-6 Project the image segment and all representation basis vectors into the subspace spanned over the image shape region only. The dimension of the full block space is more than two and the dimension of the subspace is two. 139
- Figure 5-7 Kaup and Aach's successive approximation method finds the transform basis function with the largest projection in each iteration. 142
- Figure 5-8 A simple image segment and its PSNR in each iteration of the PR successive approximation coding algorithm. The original method finds the minimal residual error *before* quantization, while the integrated method finds the minimal residual errors *after* quantization. We use uniform quantization here. 145
- Figure 5-9 Use the shape information to assist in choosing the optimal transform basis. The shape information is also available at the receiver and thus the correct transform basis can be used to reconstruct the original image. 146
- Figure 5-10 (a) Reshape the image segment into a 1D array and (b) construct its variance-covariance matrix based on the 1st-order Markovian model. 148

- Figure 5-11 Rate/Distortion curves for various transform coding schemes for the image segment shown in Figure 5-2 by using uniform quantizers. (Kaup_snr represents Kaup & Aach's successive algorithm which iterates until the PSNR before quantization exceeds 50 dB.) The average PSNR is computed over the boundary image blocks only. 149
- Figure 5-12 Changing the reference position (in the diagonal direction) of the rabbit image segment within the circumscribing square frame. The resulting quadtree complexity, i.e. the number of nodes, is shown on the right. 155
- Figure 5-13 The 8-connectivity chain code uses 3 bits to represent the pointer from current boundary pixel to the next boundary pixel. The example shown on the right follows a counter-clockwise direction. * stands for the starting point. 156
- Figure 5-14 All possible chain code patterns for a 3 pixels by 3 pixels observed window centered at each boundary pixel. The center pixel represents the current boundary pixel. The chain code runs in the counter-clockwise direction, thus the interior side (the black area) is on the left side. The additional shaded pixel is included as new boundary pixels with fractional α values (from [Takahashi93]). 157
- Figure 5-15 Examples showing the proposed table lookup technique for generating α values for boundary pixels. C_i represents the chain code and α_i represents the α value. The boundary shape is modified in (a), but not in (b). Shaded pixels highlight changes (either in the chain code or the α value). 158
- Figure 5-16 Example pictures showing the anti-aliasing effect by using the traditional approach (uniform low-pass filtering) vs. our proposed table

lookup approach (combination of the use of the chain code and the α value). 159

Figure 5-17 Two different approaches to combining image transformation and anti-aliasing in multimedia applications such as AS image object animation. 161

Figure 5-18 The blurring effect of iterative image transformations. (a) the original object (b) the result after 10 consecutive subpixel translations, by using Approach I in Figure 5-17. (from [Takahashi93]) 162

Figure A-1 Using matrix-vector multiplications to describe the convolution operation. 185

Figure A-2 Matrix multiplication of a sparse matrix with regular pre-matrices and post-matrices. 188

LIST of TABLES

- Table 2-1** Typical unary and binary compositing functions (in both the spatial and temporal dimension) 26
- Table 3-1** Basic restructuring properties of some typical compositing functions. A: associative, C: commutative (C_b for binary, C_u for unary), D: distributive (*binary* with respect to *binary*, or *unary* with respect to *binary*) 63
- Table 4-1** Complexity comparison for different displacement measurement methods. Numbers shown are the total times for evaluating the block distortion function. 101
- Table 4-2** Computational complexity for major compositing functions 114
- Table 4-3** Computational complexity for major compositing operation (with a 8 pixels by 8 pixels block size) 117
- Table 4-4** The non-zero motion vector percentage and compression ratios for test video sequences. The input video is MC-DCT compressed. 121
- Table 4-5** Computation speedup for compositing MC-DCT compressed video in the DCT-compressed domain vs. the uncompressed spatial domain. 124
- Table 4-6** Software prototype for compositing MC-DCT compressed video in the DCT-compressed domain vs. the uncompressed spatial domain. 125
- Table 4-7** Computation speedup for compositing DCT-compressed video (without MC) in the DCT-compressed domain vs. the uncompressed spatial domain. 125
- Table 4-8** SNR of reconstructed images at the compositing unit before and after the re-compression process. The input video is assumed to be MC-DCT compressed. 126
- Table 5-1** Characteristics of several transform coding algorithms for arbitrarily-

	shaped image segments.	152
Table 5-2	Comparisons of space complexity of different shape representations.	154

ACKNOWLEDGEMENTS

I would like to express my gratitude to my adviser, Professor David G. Messerschmitt, for his continuous guidance, support, and help during my pursuit of the Ph.D. degree at Berkeley. I have benefited so much from his enlightened insights and extensive knowledge in academic researches and industrial technologies. I feel extremely lucky to have the chance to work with him.

I also thank other members of my dissertation committee, Professor Domenico Ferrari and Professor David Brillinger, for their invaluable comments on my dissertation. I wish to thank Professor Pravin Varaiya for his support since I arrived in Berkeley.

I wish to acknowledge many of my former and current colleagues at Berkeley, including John Barry, Wen-Lung Chen, Paul Haskell, Masahiko Takahashi, William Li, Louie Yun, Richard Han, Wan-Teh Chang, and Allen Lao. I thank them all for providing kind assistance, discussion, and advice all the time. Thanks also go to Dr. Steve Weinstein and Dr. Andres Albanese. My summer experience at Bellcore has provided me many inspirations on later researches.

In particular, I want to thank my dear wife, Fang-Ru C. Chang, for her long-time persistent caring and support. She has made my Ph.D. career at Berkeley so fruitful and enjoyable. Lastly, I thank my parents and other family members for their continuous emotional support.

Chapter 1

Introduction

1.1 Technology Framework for Multimedia Systems and Communications

Thanks to the evolving technologies for manipulating, storing, retrieving, and transmitting digital information, multimedia applications integrating video, images, graphics, text, and data have become more and more popular. There could be various interpretations of “multimedia”. It can be educational or entertainment applications on computers, such as encyclopedia, almanac, etc. Each can contain articles, graphic drawings enhanced with animations, sounds and video clips for live effects. It can be the computer-based shared workplace among geographically remote locations. Participants exchange documents, drawings, voice, images and perhaps live video signals interactively in real time. In a rough sense, “multimedia” can be defined as the exchange of information through multiple different types of medias, as mentioned above. Examples of multimedia applications also include multi-point tele-conferencing, multimedia electronic mails, multimedia editing and publishing [Adam93, Rosenberg92]. Key technologies supporting high-speed real-time multimedia applications are being investigated as multimedia applications drive for higher performance. Specifically, a complete technology framework supporting high-end multimedia applications needs the following components: *compression, storage, transmission, compositing/manipulation, and authoring* [Cole93, Kretz92]. I will briefly discuss each of these technologies, and particularly the focus topic of this thesis — *video compositing and manipulation*.

Due to the huge amount of data required by continuous-time information, such as video and audio, *compression* is necessary. The information production rate of real-time

video, for example, can range from a million bytes per second to hundreds of million bytes per second. Powerful compression techniques are mandatory especially when resources (such as transmission bandwidth and storage capacity) are limited. Although compression degrades quality to some extent, today's compression techniques can reduce the data by a factor of 25:1 to 50:1 without significant quality degradation. Current research emphasizes multi-resolution video compression, very-low bit-rate video coding, 3-D model-based video coding, and real-time low-cost hardware implementation [Jayant92, Cole93].

When information production becomes easy and pervasive, *storage* of multimedia information becomes more pervasive. The main issues in storage are how to retrieve multimedia information from storage in the real time and how to design internal file structures for different media. The problem becomes more complicated when the storage supports multiple access from two or more users. Many current multimedia applications use CD ROM as the storage device. Its capacity and I/O bandwidth are still very limited (typically 600MB capacity and 300 KB/s), far below the requirement of real-time full-motion video services. The file formats used in different applications are not interoperable among different computing platforms.

Transmission over networks is still a big hurdle for tele-conferencing and computer assisted interactive cooperative applications. Using dedicated real-time channels like circuit-switched networks is too expensive and inefficient. Using multi-access packet switched networks, such as the ATM/BISDN, introduces problems like packet loss and delay jitter in exchange for a more efficient network utilization. Important issues to be considered are real-time performance-guaranteed protocols, packet loss protection/recovery, congestion control, and joint source/channel coding of video signal [Jayant92, Karlsson89, Ferrari92].

Video compositing and *manipulation* is an important technology required in many multimedia applications. By video compositing and manipulation we mean the

manipulation of individual video signals, such as filtering, scaling, geometrical transformation, etc., and the spatial or temporal combination of multiple video streams into a single video stream. These functions are typically used in multi-point video conferencing and multimedia editing. Today, high-speed real-time hardware for video compositing and manipulation are only available in professional studios. Designing reasonable-cost real-time compositing hardware for multimedia network video services remains a big challenge [Lukacs92].

The future success of multimedia applications depends heavily on the customers' perceptions. One key element is the performance of *authoring* tools, which help users to produce multimedia presentations without formal professional training. Currently, primitive tools are available. But significant improvements are needed to allow them to be widely accepted.

There are other technical issues relating to multimedia systems and communications such as interoperability among different platforms, multimedia scheduling in a distributed information environment, and real-time hardware/software support. For a pervasive ubiquitous networked multimedia systems, all the above technologies must be provided. One example utilizing these technology is the multi-point cooperative workplace, as shown in Figure 1-1. Live video signals need to be compressed, transmitted, and composited with locally generated graphics images and remote video signals, which may be stored in distributed information servers.

In this thesis, I focus on the technology of video compositing and manipulation. A *systems* approach is used to explore every degree of freedom of implementing video compositing. Along with the study of individual aspects of this technology, strong interactions between video compositing and other technologies mentioned are observed. I will review existing compositing techniques and describe degrees of freedom in the design

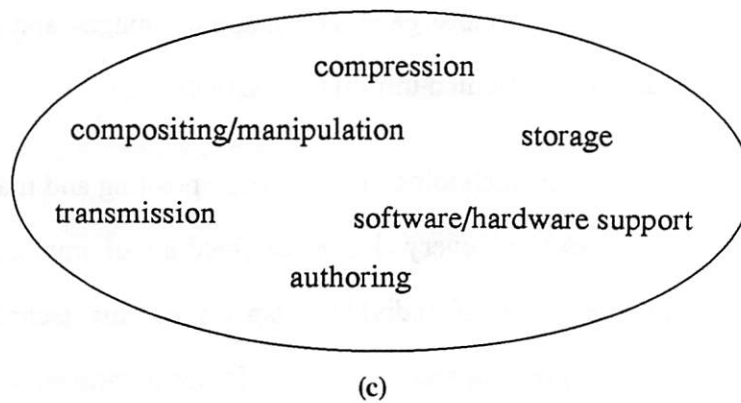
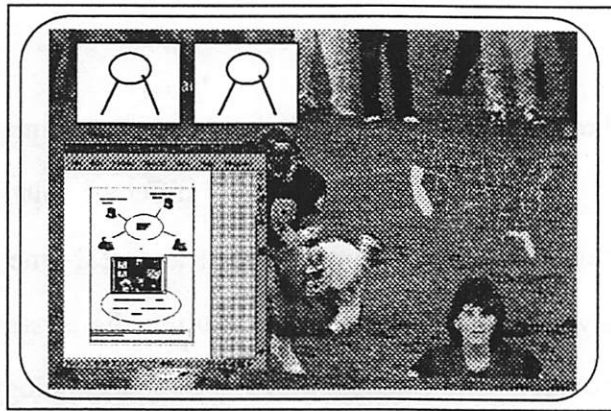
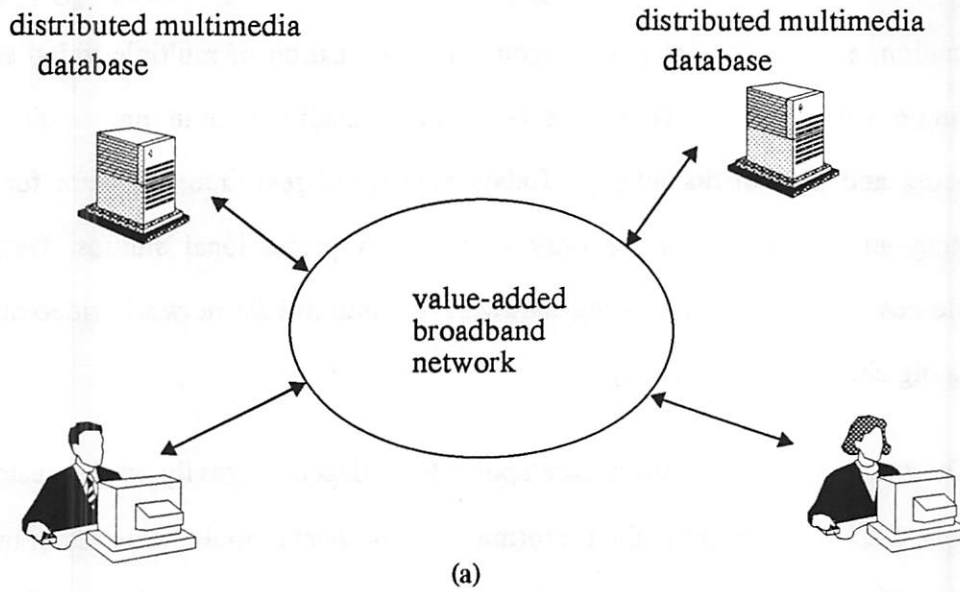


Figure 1-1 (a) A typical multi-point multi-media service on broadband networks. (b) Possible presentation scenarios on user screens. (c) Advanced technologies needed for implementing these cutting-edge applications.

of video compositing in this chapter. Research objectives and strategies are described later in this chapter.

1.2 Network Video Compositing

As mentioned above, video compositing, a process to combine spatially or temporally several video streams into a single displayable video stream, is required in many multimedia services. There might be also some manipulation operations applied on each individual video source before they are combined. However, throughout this thesis we use the term video compositing to mean both the combination of multiple video signals and the manipulation of an individual video source, if there is no contrary indication. In this section, we first review existing compositing techniques. Then, we describe more advanced compositing features which require more innovations.

1.2.1 Existing Compositing/Manipulation Features

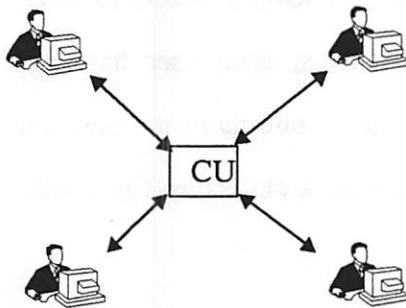
A typical application using video compositing is video conferencing, as shown in Figure 1-2. Video windows showing head-and-shoulder images of participants are composited together and displayed on the screen. [Ahuja92, Rosenberg92]. The compositing units can be dedicated to each individual user or shared by the whole group of users. The so-called *network video bridge* uses a single compositing unit in the network to composite all video signals and then send common composited video to every participant [Lukacs92]. In desktop video conferencing systems, each user has local dedicated compositing hardware or software and thus has more controls of the positions and sizes of the video windows. Figure 1-2 shows the resource architecture for a video bridge and desktop video conferencing.

Most existing video compositing systems only support primitive features such as rectangular video windows, opaque overlapping, scaling, and translation. Some systems

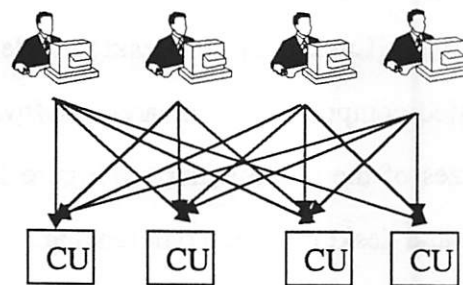
still composite video in the analog domain, ignoring the tremendous potentials of digital technology. Video signals are typically composited in the uncompressed domain. Input video signals are concentrated in the specific location where the compositing unit is located, no matter whether the architecture is concentrated or distributed. These designs are straightforward and perhaps efficient for simple applications. But for future advanced multi-point multimedia network video services, such as that shown in Figure 1-1, significant innovations are needed to achieve reasonable-cost high-speed real-time implementations.



(a)



(b)



(c)

Figure 1-2 (a)Today's typical video conferencing pictures. (b)Concentrated video compositing architecture. (c)distributed video compositing architectures.

1.2.2 Envisioned Advanced Compositing Features

In future advanced video compositing systems, more flexible features should be provided. First, most manipulation operations we now apply on graphics should also be applicable to video. Video objects shown on displays can be arbitrarily shaped for a better model of natural physical objects. Composited video should allow any arbitrary combination of input video signals. Multiple video signals can be semi-transparently mixed. These features are available today only in professional studios [Bennet84, Wolberg90]. But, in the future, users should be able to create special effects easily in multimedia authoring tools. Figure 1-3 shows an example of an envisioned composited picture, which includes arbitrarily-shaped video objects, graphics, texts, and semi-transparent overlapping.

Second, input video signals can come from arbitrary locations, ranging from broadcast stations, professional studios, remote databases, local video jukeboxes, and live



Figure 1-3 Envisioned advanced compositing features in future video services

video cameras. An immediate impact on arranging compositing units is that they should be allocated distributively everywhere throughout the network. Intelligent network management should be able to locate or synthesize the necessary resources when the required hardware resources are not locally available. Figure 1-4 shows a composited image which includes video signals produced at different locations and could be composited in a distributed way.

1.2.3 Challenges

All these envisioned features of video compositing call for innovative algorithm designs and network management schemes. Specifically, how do we perform video compositing operations with different levels of difficulty at a reasonable cost in real time? How do we allocate network resources (bandwidth and hardware) efficiently to implement various advanced video services on broadband networks? Also, for general arbitrarily-shaped video objects, how do we design efficient models of video signals and compositing functions? It is the goal of this work to answer these questions as thoroughly as possible, at least to gain more comprehensive understanding.

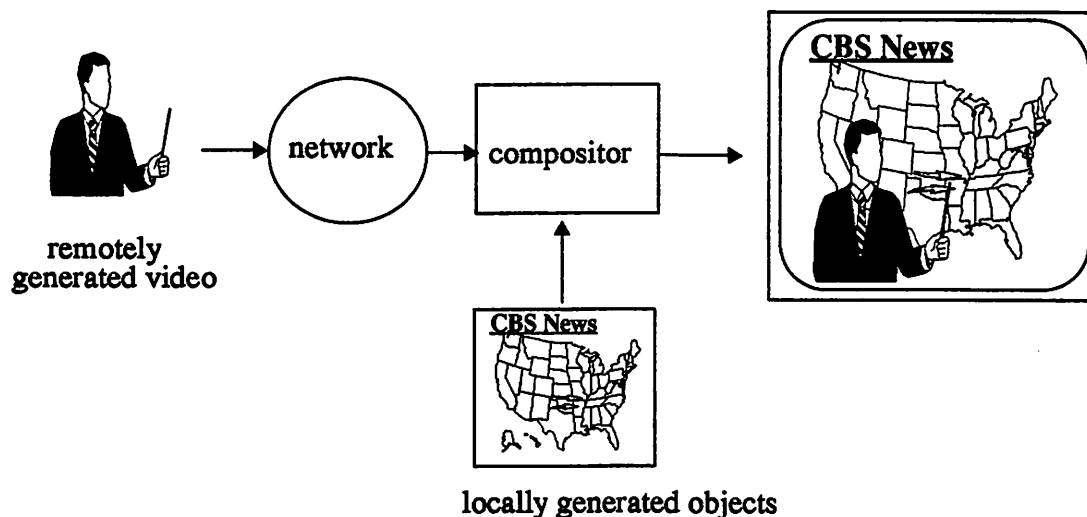


Figure 1-4 A composited video can contain video signals originated from different locations. For example, the image of the weather reporter may come from broadcast video, but the map and logo can be produced locally.

1.3 Degrees of Freedom for Network Video Compositing

In designing network video compositing, we can find several degrees of freedom, which are *feature*, *location*, and *data format*. We consider them to be design freedoms because in each we have flexibility in using different approaches and parameters to satisfy resource constraints and performance requirements. Different approaches in each degree of freedom may have direct or indirect impact on system performance. For example, providing different compositing features affects the final presentation and flexibility of users' control over video objects. Compositing at different locations also indirectly affects users' control and video quality. For specific applications, we want to find optimal solutions in each degree of freedom in order to provide an optimal overall solution for the whole compositing system. Given resource and performance constraints, our goal is to achieve the highest performance/cost ratio. In this section, I will briefly describe the characteristics of each degree of freedom. Detailed investigation in each degree of freedom will be presented in later chapters.

1.3.1 Different Compositing Features

As described in the previous section, the simplest compositing system tiles several rectangular images together to produce a larger image. The input images are not processed further. Despite its simplicity, or perhaps because of it, it is the most popular feature provided in today's video compositing applications (such as video conferencing and some window graphic interface). Emerging desktop computer video conferencing offers users more complicated capabilities for video manipulation, such as changing video window position and size, and overlapping several video windows. In future advanced applications, we can envision additional features like semi-transparent video overlap, arbitrarily-shaped video objects, three-dimensional model-based image synthesis, and so on.

As compositing features become fancier, the required hardware processing power increases. Their impact on hardware complexity varies even for the same compositing feature. For example, the same compositing function may require different computational complexity if implemented in different data formats.

The significance of this degree of freedom is that we may be able to improve the overall system performance by compromising some compositing features. For example, in video conferencing applications, the translation distance and the scaling factors of video windows can be restricted to a fixed size. Users may not feel strongly objective to these restrictions, but the implementation cost would be greatly reduced.

1.3.2 Different Compositing Locations

As distributed video sources spread geographically, the compositing units can also be distributed. Video compositing can be done at the source, the destination, and at intermediate locations in or outside the transport network. For example, television broadcasting companies can composite several video sources before they are transmitted. Cable TV companies composite broadcast programs with their own signals and transmit them to users. Office desktop workstations may composite video signals from remote databases with locally generated graphic images. Different network multimedia services have different characteristics and may prefer different compositing locations. As described in the previous section, a video conferencing system can use centralized or distributed compositing architectures, which affects the final performance. One challenging question is: given an abstract representation of compositing tasks, how do we map it onto the available network resources, which include communication channels and processing hardware. I will discuss the advantages and disadvantages for each compositing location in Chapter 3 and describe some possible approaches for optimizations.

1.3.3 Different Data Formats

By *data format* we mean the compression format of the video signals. The format is important because video signals represent a huge amount of data, which usually requires compression to substantially reduce the required storage and bandwidth. If video signals are in compressed formats, it is potentially beneficial to composite them directly in the compressed domain. For example, if video is composited within the network (such as a video bridge), input video and output composited video are usually compressed. Compositing in the compressed domain can avoid conversion back and forth between the compressed and uncompressed domains. Even at the destination, where output video needs to be in a displayable format, its input video signals may still be compressed. This makes compressed-domain compositing preferable.

However, as mentioned above, the implementation cost for the same compositing function in the compressed domain may be significantly different from that in the uncompressed domain. We need to carefully analyze its impact on the final implementation cost. In addition, there are many different compression algorithms used in today's multimedia applications. Popular algorithms include MPEG for motion pictures, JPEG for still pictures, H.261 for video conferencing, CDI, DVI, Video for Windows and QuickTime for computer-based multimedia applications [Cole93, Le Gall91, Liou91, Wallace91]. However, these standards share some underlying algorithms, such as the Discrete Cosine Transform and Motion Compensation [Netravali88]. Therefore, in our study of compressed-domain compositing, we will focus on the DCT and MC domain (Chapter 4).

1.4 Previous Work

As multimedia applications rapidly emerge, issues related to video compositing have gained more attention from researchers. We briefly describe some of the related work here, but leave more detailed reviews to later context.

In relation to various compositing features, there are mature techniques for still image transformations in the area of computer graphics [Foley90]. These image processing techniques can be considered as unary compositing functions. Multi-object compositing functions such as overlapping have been widely used in window graphic interface and desktop video conferencing applications. Porter and Duff proposed the α channel to perform compositing of digital images [Porter84]. Their approach can be extended to arbitrarily-shaped video objects. We will study the characteristics of these different classes of compositing functions (including spatial and temporal operations). We will also propose efficient representation formats for compositing functions based on our proposed structured video model.

In relation to the choice of compositing locations and issues of distributed compositing, there have been many works associated with different aspects of multi-point multi-media services. Little and Ghafoor studied general compositing processes of stored multimedia objects, assuming a platform of networked distributed databases [Little91]. They also discussed the trade-off among different performance metrics when mapping the compositing process to network resources. However, they did not consider the compression process, which may have significant impacts on the choice of compositing locations. While they focused on stored multimedia database objects, we will consider real-time multi-source multi-user video services as well. In addition, we will analyze the distributed video compositing approach and different compositing locations based on our proposed structured video model, which will be defined in Chapter 3.

Rangan *et al.* described synchronization algorithms and hierarchical compositing architectures for multi-point conferencing [Rangan93]. They found that the hierarchical architecture has higher scalability with regards to the number of conference participants, when compared to the centralized architecture. However, their study focused on a broadcast case where all participants receive the same composited scene. They also failed to consider the compression process.

Schooler and Casner [Schooler92] designed network protocols to support multimedia multi-party connections on heterogeneous wide area networks. They focused on connection control and resource configuration control. Mechanisms such as the resource description language, the distributed locator service, and the resource synthesizer are used to manage heterogeneous resource configurations and achieve compatible services at different end systems. Their work can be considered as study of a lower-level cooperative entities to support our higher-level representation and optimization of compositing processes, which will be discussed in Chapter 3.

In addition, Pasquale *et al.* [Pasquale93] studied the optimization of multi-cast connections based on their proposed loose-coupling principle. In Chapter 3, we will observe some similarity between their multi-cast connection problem and our distributed compositing problem, which is a multi-source, multi-user problem in general. For example, the idea of finding sharable tasks among different users to reduce the implementation cost is similar in both problems.

In relation to the data formats for video compositing, Smith and Rowe [Smith92] investigated the approach to compositing images in the DCT domain, independently of our work. They also found significant computational speedup by using the DCT-domain compositing approach. But they only provided brute-force numerical solutions to a subset of operations. We will derive *analytical* formulae for a larger set of typical compositing operations and will study their computational complexity analytically. Furthermore, we

have extended these algorithms to handle MC-DCT compressed video. Lee and Woods [Lee92] developed some simple compositing operations (such as pictures overlay and text overlay) for subband-compressed images. However, their method needs an additional bounding box to hide the artifact around the overlap boundary.

In summary, there have been many previous works related to both still image compositing and multi-point connections on communication networks, but there have been few attempts to systematically study video compositing on multimedia networks. Although Little and Ghafoor's work is closely related, it did not consider many important aspects of network video compositing.

1.5 Goals and Strategy of Approach

We take a *systematic* approach in exploring different degrees of freedom for implementing network video compositing and interactions among these degrees of freedom. First, we characterize typical and envisioned future compositing operations used in multimedia video services. Second, we study the question of where to perform these compositing operations in the network. Lastly, we study the effects of performing compositing in different data formats, in particular the compressed vs. the uncompressed format. The overall objective is to provide a thorough understanding of different aspects of the video compositing technology. Through performance tradeoff analysis in each degree of freedom, we hope to provide a systematic approach to achieving optimal performance/cost for future multimedia network video services.

In addition, we also study the distinguished features of representation and compositing of arbitrarily-shaped video objects (ASVO). We will describe efficient coding and compositing algorithms for ASVO in Chapter 5.

Most analytical tools in this work are through mathematical derivations of new algorithms and numerical simulations. When subjective video quality is concerned, non-real-time software implementations are employed and video quality is evaluated subjectively.

Chapter 2

Compositing Features

As described in Chapter 1, emerging multimedia applications like multi-point teleconferencing require real-time video compositing and manipulation (video compositing for short). We recognize three different degrees of freedom for implementing video compositing — feature, location, and data format. This chapter covers the first degree of freedom, *compositing features*. We will characterize different types of compositing functions, such as unary and binary operations, and study their efficient representations.

We assume a general definition of video compositing. It can be image transformation techniques widely used in computer graphics. It can also be advanced features used in professional studios. In this chapter, we will propose a *structured video* model which defines video objects and compositing functions used to combine individual video objects. Based on this model, we will discuss the characteristics of different classes of compositing features. We will also survey existing video compositing techniques scattered in different fields and present several hierarchical ways to represent universal compositing functions. These hierarchical structures are useful platforms for later investigation of other degrees of freedom in implementing video compositing. We consider compositing features as one degree of implementation freedom since we can often trade compositing features to obtain implementation flexibility.

2.1 The Structured Video Model

McLean first used the term structured video to represent the concept of separating foreground and background video objects in a video signal and transmitting them

separately [McLean92]. He advocated that substantial compression can be achieved by separately encoding the foreground objects and tracking the motion of the background object. The concept is similar to that of model-based analysis-synthesis video coding [Mussman89, Kunt87, Hotter90].

Although high-compression video coding is one of our objectives, our structured video model should be considered an extension of previously proposed structured graphics models to video [Lantz84]. The main objective of structured video is to present to the user, on a single raster-scanned display, a variety of logically separate *video objects* that could be arbitrarily-shaped and could be *composited* in various ways. The video objects are kept logically separate so as to provide easier manipulation and easier adaptations to heterogeneous hardware platforms and services. For example, in Figure 1-1 of Chapter 1, the composited scene consists of different types of video objects such as rectangular-shaped bitmaps (a football background), irregular-shaped bitmaps (a news reporter), graphics, and text. The news reporter is opaquely overlapped with the background object, while the graphic object is overlapped with the background in a semi-transparent way. By keeping these video objects separate, we can explore tremendous potentials for easy manipulation, efficient compression, flexible user control, and flexible interface to different services. Given a set of video objects, there are many different ways to composite them into a displayed scene. It is also one of our goals for the structured video model to define efficient representations for video objects and their compositing rules.

2.1.1 Video Objects

A *video object* is the representation of the visual appearance, shape, and position of some visible part of an image sequence. Usually, it corresponds to an actual physical object or a logical object contained in a video sequence. For example, the news reporter, the text, and the graphics shown in Figure 1-1 are sampled data of different time-varying

video objects in a particular frame. As time (or the frame sequence) proceeds, the video object can vary its visual appearance (e.g. the head motion of the reporter, changing contents of the text object) and other parameters, such as position and size.

A video object can conceptually be smaller than a single pixel or larger than an entire display. The principle we use to segment a scene into video objects is that all parts of an object use the same rule when composited with other objects. Thus, it is impossible for one object to be displayed as if it were in front of and behind another object at the same time. In Figure 2-1, for example, object B is both in front of and behind another object, and therefore it cannot be treated as a single object. It would be divided into two video objects, possibly at the dotted line. If A and B's two components later moved apart, the two objects into which B was separated could be recombined. Similar ideas for subdividing or merging video objects are discussed in [Lin91].

The definition of a video object is intended to keep the number of compositing rules required in a video system small. Since a video object represents the *largest* object that can use one set of rules, the definition automatically yields a system with no superfluous rules. Further, this definition is simple in that it does not require creating subdivisions within video objects and storing special rules for each of the object's subparts. Video objects are the smallest divisions into which components of a scene need to be divided, and they are the largest divisions that act as homogeneous units. In contrast, in z-buffering systems [Duff85], each pixel has its own associated depth value. Therefore, each pixel can be composited differently from every other, and is a separate video object.

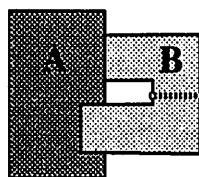


Figure 2-1 Example of illegal pair of video objects, because object B cannot be treated as a unit when composited with another object. (from [Chen93a])

A video object can also correspond to one of many different facets of a video signal. For example, subband coding is often used in separating a video signal into many different frequency bands, each of which has different human perceptual significance. Treating each subband as different video objects, we can process the video signal more efficiently (e.g. compression and transmission) and support users with different display capability. However, this kind of video object separation imposes some constraints on the compositing algorithm and complicates the compositing process. This will be discussed further in the section 2.6.4.

The definition of a video object can be extended hierarchically. A *composited scene* which includes several video objects can in turn be viewed as a video object. To distinguish modified video objects from original video objects produced at the source (e.g. a camera), we call the former *composited video objects*, and the latter *primitive video objects*. These two types of video objects are equivalent during the process of compositing with other video objects. Once several video objects are composited together to form a composited video object, any logical distinctions among the comprising components are removed. Future compositing can no longer separate the component video objects inside the composited video object.

On the user display, the image displayed may consist of several composited scenes just as a computer workstation display consists of several overlapping windows. Here, a displayed scene is composed of a set of several video objects that are related to each other—for example, a video conferencing scene consisting of the video objects representing all of the people in the conference plus the background. As we have mentioned above, each scene and the overall displayed image can be viewed as composited video objects.

2.2 Advantages and Disadvantages of Structured Video

Structured video offers considerable advantages as a model for provisioning advanced video services. We divide the service provision into three distinct aspects: the *vendor* or vendors who provide the video object (they may also be locally stored), the *transport* or network connection from the vendor, and the *user* who interactively manipulates the video. The essence of structured video is to keep the video objects logically separate, even if they share a common storage or transport, and to support the flexibility of compositing them at later stages within the network, such as an intermediate network node or the user display. The efficient hierarchical structure of the proposed structured video model and the generic representation of the compositing functions enable efficient adaptation of hardware resources to dynamically changing user and application requirements.

Structured video can represent video images over a wide range of applications, for example a) full-motion high-definition television, b) conference video, c) videotex, d) interactive video, e) windowing and graphics computer display interfaces, and f) combinations of these types of representations. A standardized representation enables implementation of display signal processing using a set of common modular hardware and software components across a variety of applications. This achieves economies of scale and lower costs. Further, depending on the hardware and software modules installed, a video display can support a variety of services,

Similar to the OSI model of data communications, structured video can provide a common model of video shared among vendors, transport networks, and users. Specific services and applications are readily provisioned by parameterizing this model, thus simplifying the administration of telecommunications networks. A vendor can request the network resources necessary to provision the service. The vendors similarly can query the

user's display platform to determine the standardized resources it has and request the resources it needs.

From the perspective of a vendor, keeping video objects separate enables much greater flexibility and reusability, for example in composing different scenes that reuse common video objects. From the perspective of a user, keeping video objects logically separate all the way to the display enables instantaneous interactive configuration, for example in moving or resizing the participants in a multi-way video conference. The use of standard formats for many image types can support interactive services. Users can combine and customize images from different people and vendors without requiring the sources to interact. A single video stream can be used differently by everyone who receives it without end-users interfering with each other. The transport control traffic necessary to implement the same interactive functionality at the vendor is eliminated.

Structured video has the potential to allow more efficient compression of complex video representations for storage or transport. Data compression is more effective if performed on the separate types of video signals. This is due to the fact that compression algorithms can be tailored to match the statistics of each data type. It is becoming common to separate video into different regions according to classification algorithms for more efficient compression, but this is avoided if the video objects are kept logically separate [McLean92, Kunt85, Mussman89]. It is also possible to add to the structured video representation additional semantic information available at the source, such as panning and zooming information, to simplify and improve the compression. Further, if natural scenes are not overlaid with the high-frequency signals caused by text and graphics, then standard video compression techniques are more effective. Of course, text and simple graphics can be described more efficiently by semantic descriptions (fonts, lines, rectangles, arcs) rather than by bitmaps. Graphics and animation sequences can be transmitted more efficiently by sending the procedure required to generate them rather

than the generated images, if there is sufficient processing at the user display to execute those procedures.

Structured video can also serve as the basis for adapting services to variation in available network and user resources, which is an important practical issue. One video display may have limited resources and be targeted at a limited set of services because of the hardware and software modules it contains. For example, while an interactive full-motion videotex system would require text, graphics, windows, pull-down menus, and full-motion video mixed together, a computer workstation attached to a low-bandwidth network might require only text, graphics, and windows. The vendor and transport will be able to adjust to varying hardware resources at the display. For example, at the expense of interactive flexibility, object compositing can be done at the vendor or within the transport if the display does not possess sufficient processing capability. This can be performed through re-structuring of the compositing functions for optimal mapping, as will be described in section 3.3. In the limit, the minimal display can at least accommodate a single rectangular raster-scanned video, which is also supported by structured video.

The computational resources for the final video presentation can be partitioned arbitrarily on the path from production to final presentation, thereby adjusting to economic constraints for a particular service as well as to the available transmission and storage bandwidth. Distribution or broadcast services will place more processing nearer the source, whereas point-to-point or specialized services will generally place more processing near the display. The representation will adapt easily to a) different transmission media (satellite, fiber, cable), b) display devices with varying processing resources, ranging from simple television displays that present only the pixel map representation to complex workstations that can process all the representations, and c) vendors with widely varying processing resources ranging from reading pixel maps from storage devices through display and graphics engines.

The subjective quality of the presentation can also be adjusted to the transport bandwidth resources. A lower quality can be provided by using only “higher” semantic and graphics representations, with quality up to and including full-resolution HDTV available with the expenditure of additional processing and bandwidth. We also postulate that where limited bandwidth is available, higher subjective quality can be achieved by compressing some video objects more heavily than others, for example in retaining full resolution on a head and shoulders while limiting the resolution of the background.

Structured video has disadvantages. One disadvantage is the higher cost of display platforms, although as mentioned previously even minimal existing platforms can support a degenerate form of structured video. Another is the expansion of data required to transport the hidden portion of video objects, although this can be reduced with more sophisticated flow-control protocols [Gaglianello91]. Lastly, there are limitations in the structured video model relating to its inherently two-dimensional representation (like video itself), as described in more detail in [Lin91].

In summary, the structured video concept of keeping the components used to generate video logically separate past the production process and all the way to the final video presentation if that makes economic sense, is simple yet powerful. In particular, it offers flexibility of reusing and modifying video material at the user display or at any earlier point. Where pictures are actually assembled from different components (foreground, background, text, graphics, etc.), as will be increasingly the case in the future, it is potentially much more efficient to compress the components before combination than after. Further, structured video gives flexibility to adjust subjective quality to bandwidth and processing resources (for example substituting a graphically-generated background when bandwidth is not available for a camera-generated background), and the flexibility to adjust processing resources between the provider and the user (for example in adjusting to bandwidth resources). Finally, structured video is

consistent with many interactive forms of video, where components generated remotely can be combined easily with locally generated elements to form the final video presentation. Centralized interactive services can segment complex tasks into a) time-critical tasks done locally (graphics and animation), b) reading of high-bandwidth background video too expensive to store locally from a central database, and c) non-data-intensive tasks like billing performed remotely.

2.3 Unary Spatial and Temporal Compositing Functions

As mentioned above, compositing functions play an important role in the structured video model to combine several video objects together into a single *composited* video object. We discuss the characteristics of unary compositing functions in this section and leave multi-object compositing to the next section.

Typical unary image manipulation operations used in computer graphics include geometrical transformation (such as translation, scaling, rotation) and some image processing operations (such as linear filtering, median filtering, clipping & concatenation, and thresholding). Geometrical transformations usually involve transforming source pixels into the so-called *target space* and use a weighted filter support to calculate the new target pixel value. Ideal filter designs require an infinite number of filter coefficients with a *sinc* impulse response. But in practice, the filter maybe simple box filters (uniform tap coefficients) or some circularly symmetrical filters such as Gaussian filters. Usually, the wider the filter support is, the smoother the transformed image will be, but the image will also become more blurred. In addition, many geometrical transformations can be implemented by multi-pass approaches. For example, a rotation can be decomposed into column-reserving shearing and row-reserving shearing, plus appropriate scaling, as shown in Figure 2-2. The advantages of multi-pass implementations are incremental processing in each pass and thus much faster processing.

As we mentioned in the last section, each video object may include a sequence of image frames (possibly arbitrarily shaped). If we apply unary operations similar to those described above to the temporal dimension of a video object, we can find many interesting effects. As listed in table 2-1, scaling corresponds to rate conversion, clipping and concatenation correspond to temporal annotation of video sequences, and low-pass filtering corresponds to frame interpolation.

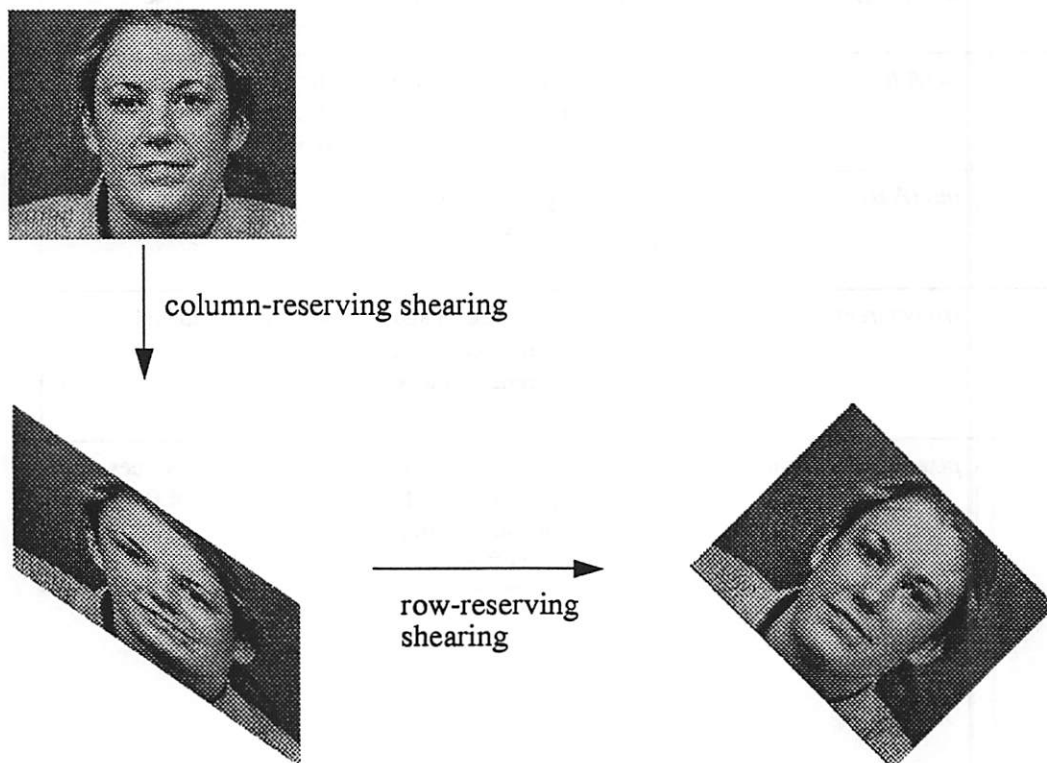


Figure 2-2 Multi-pass implementation of image rotation. (a)original (b)after y-direction shearing (c)after x-direction shearing.

Table 2-1 Typical unary and binary compositing functions (in both the spatial and temporal dimension)

	Function	Spatial	Temporal
<i>unary</i>	<i>translate(A,d)</i>	change the spatial location by <i>d</i>	delay the video object by <i>d</i> time units
	<i>scale (A, s)</i>	scale object A by a factor of <i>s</i>	inter-frame interpolation or extrapolation
	<i>linear_filter(A, h)</i>	filter object A with impulse response <i>h</i>	inter-frame filtering with impulse response <i>h</i>
	<i>flip_x (A), flip_y(A)</i>	flip object A in x or y direction.	time reversal
	<i>Clip(A, M)</i>	extract part of object A, designated by mask M	annotate one subsequence from object A
	<i>rotate(A, r)</i>	Rotate object A by an angle <i>r</i> .	NA
<i>binary</i>	<i>over (A,B)</i>	overlap object A on top of B	display frames of object A on top of frames of object B
	<i>abut(A,B)</i>	put object A and B side by side	concatenate object A and B temporally
	<i>in (A,B)</i>	display part of object A inside object B.	display frames of object A during the duration of object B.
	<i>out (A,B)</i>	display part of object A outside object B.	display frames of object A outside the duration of object B.
	<i>transparent (A, B, τ_1, τ_2)</i>	composite object A and B transparently according to transparency factors τ_1, τ_2 .	same as the spatial domain
	<i>pixel_multiplication(A,B)</i>	multiply pixel values of A with pixel values of B (useful in anti-aliasing and special effects)	multiply pixel values of A with those of B in the temporal dimension

2.4 Compositing of Multiple Video Objects

In this section, we will discuss *binary* compositing functions and in general multi-component compositing functions. Table 2-1 also shows some typical binary compositing functions such as $\text{overlap}(A,B)$, $\text{abut}(A,B)$, $\text{in}(A,B)$, $\text{out}(A,B)$, and so on. Figure 2-3 illustrates some examples.

Again, these operations can be applied to the temporal dimension as well as the spatial dimension. Some operations may look like unary operations, but they actually belong to binary operations. For example, spatially translating a video object actually corresponds to changing the position parameter in an overlap operation with a display reference image frame, which can be viewed as an empty root image. Similarly, temporal shifting of a video sequence corresponds to a temporal mixing operation with a null reference sequence.

For each binary compositing of two video objects, both the spatial and the temporal compositing rules must be specified. The temporal rules specify how to align frames of one video signal to frames of another. The spatial rules specify the spatial relations between corresponding frames. As described in section 2.1.1, the principle we use to segment video objects is that the same compositing rules should be applicable to the

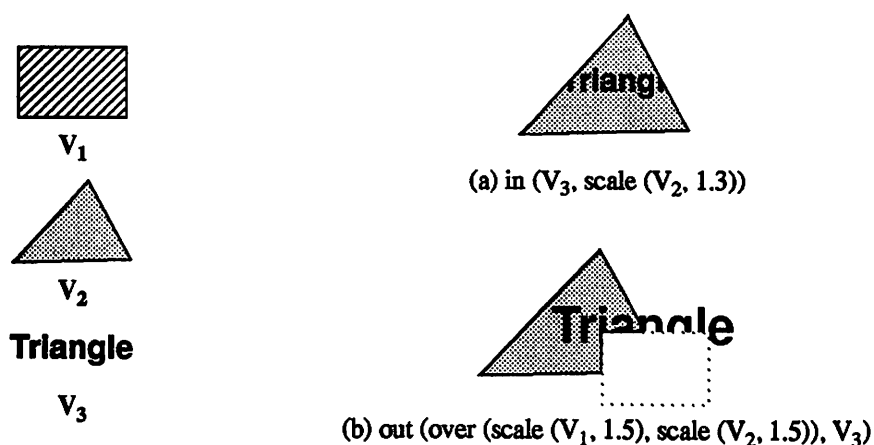


Figure 2-3 Examples of composited video objects using basic compositing functions in table 1. (from [Chen93a]).

whole video object, in both the spatial dimension and the temporal dimension. Unfortunately, this principle also implies that the same spatial compositing rules should be applied in each image frame of the same video object. This imposes some limitations on the permitted compositing operations. Some popular compositing functions such as the *picture in picture* and *dissolve* features available on today's TV sets are still permissible. The former is a time-static operation and the latter can be achieved by simply changing the transparency parameters of input video objects. However, time-dependent operations such as the one shown in Figure 2-4 cannot be supported. In the most general case, a spatial binary compositing operation could vary from frame to frame. Under those situations, we can either divide each video object into smaller ones (in the temporal dimension) and then use the same compositing rule for each segmented video object, or extend the compositing functions to time-dependent representations. For example, in Figure 2-4, we show a *compositing script* comprising of several compositing functions to complete the desired operations.

Theoretically, there are compositing functions that combine more than two video objects into a single one. Some of them can be obtained by simply concatenating binary compositing functions (like overlapping), but some can not¹. However, most practical compositing functions seem to fall into the former category.

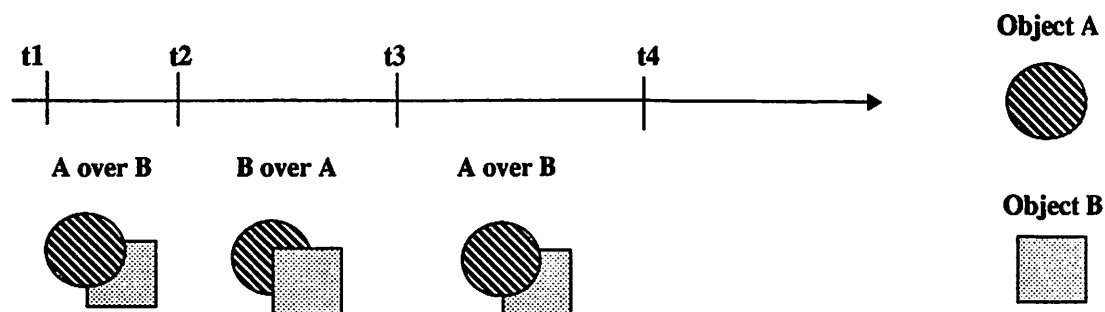


Figure 2-4 Using compositing scripts to specify time-dependent compositing rules among two video objects. An alternative approach is to divide objects into smaller ones (in the temporal sense). The compositing rules for each video object will then become static for the whole duration of the video object.

2.5 Compositing of Arbitrarily-Shaped Video Objects

Compositing of arbitrarily-shaped video objects (ASVO) is used in segmentation coding of video signals [Mussman89] and multimedia editing. Logically, they can be classified into unary and multi-object compositing operations. These objects suffer from complications due to the irregular, jagged boundary of the video objects. We need to perform *anti-aliasing* along the object boundary in order to smooth it. Porter and Duff proposed an α channel to take account of the partial coverage within a pixel area for the boundary pixels [Porter84]. The internal pixels have the α value equal one, external pixels 0, and boundary pixels fractional numbers between one and zero. Figure 2-5 shows an ARVO and its anti-aliased version to illustrate the effect of the α channel. The α values are generated by uniform linear filtering.

Besides anti-aliasing, the α channel can also be used to represent the shape of an irregular-shaped video object. It is 1 for internal pixels, 0 for external, and fractional values for boundaries. It is like an additional channel of the image pixel information, besides the regular color channels. However, since most α values are either 0 or 1 and there is significant redundancy, we should be able to find efficient coding algorithms for the α values (like the run length code), and thus the shape of the object. We will cover the representations and coding of the object shape in Chapter 5.

The additional channel of α values can be treated as a regular color or luminance channel. Figure 2-6 shows the input/output relations for compositing/manipulating the ASVO's. In practice, operations applied to both the α channel and the regular color channel can be the same operations or quite different operations, depending on the compositing functions. For example, in linear geometrical transformations, the same

1. For example, the compositing function could be defined by a non-linear function, $V_{out} = F(V_1, V_2, V_3)$, which cannot be decomposed into binary functions. In turn, the mapping function can be defined by a lookup table.

filtering operations can be applied to the α channel as well as the color channels. For the binary operations described below, however, the calculations for the α channel and the color channels are quite different.

According to Porter and Duff's compositing algorithm, two ASVO's can be composited as follows:

$$P_{\text{new}} = \alpha \cdot P_f + (1-\alpha) \cdot P_b \quad (2-1)$$

where P_{new} , P_f , and P_b are new composited pixel, foreground pixel and background pixel, respectively. The α value of a pixel indicates the coverage percentage of that pixel within one pixel area. This compositing function can be extended to multiple layers incrementally:

$$P_{\text{new}} = \alpha_1 \cdot P_1 + (1-\alpha_1)(\alpha_2 \cdot P_2 + (1-\alpha_2) \cdot (\dots)) \quad (2-2)$$

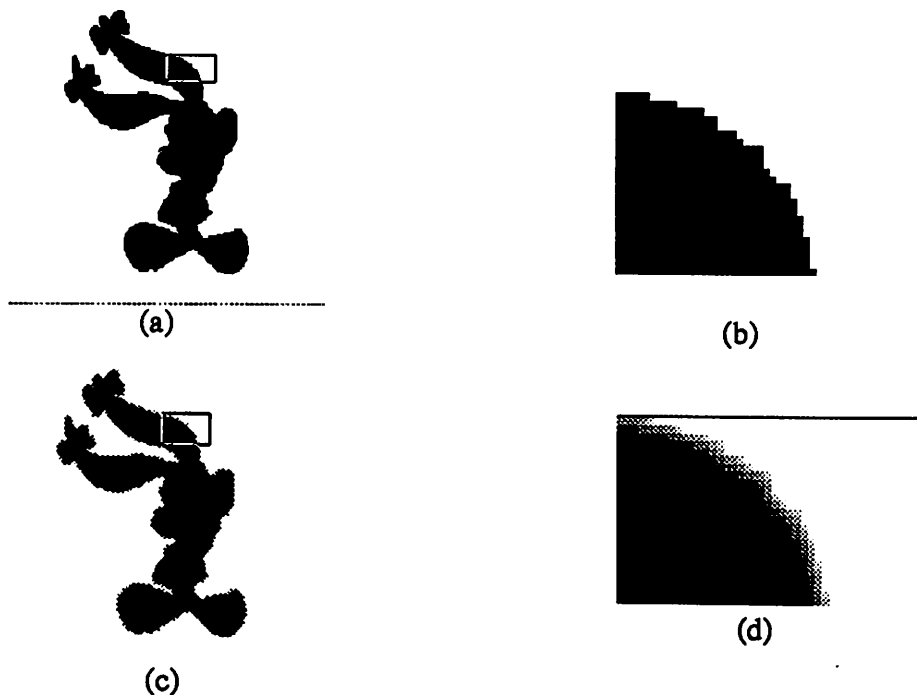


Figure 2-5 The original jagged rabbit and its anti-aliased version. (a) original raw image (b) enlarged area of original image (c) anti-aliased image (d) enlarged area of anti-aliased image.

where the depths of the video objects are assumed in the ascending order. Porter and Duff [1] also use this technique to perform different compositing operations on a pixel scale, such as $\text{over}(A,B)$, $\text{in}(A,B)$, $\text{out}(A,B)$, $\text{atop}(A,B)$ (defined in Table 2-1).

We can combine the α channel with transparent overlapping. Suppose video objects with the same display depth are overlapped semi-transparently; then they can first be combined as in the following:

$$\alpha_{\text{new}} = 1 - (1 - \alpha_1)(1 - \alpha_2) \quad (2-3)$$

$$P_{\text{new}} = P_c / \alpha_{\text{new}} \quad (2-4)$$

$$P_c = \alpha_1(1 - \alpha_1)P_1 + \alpha_1\alpha_2(\tau_1P_1 + \tau_2P_2) + (1 - \alpha_1)\alpha_2P_2 \quad (2-5)$$

to form a single object and later composited with other objects with different depth. τ_1 and τ_2 are the transparency parameters in the semi-transparent compositing function. These formulae can be derived based on the set operations. In [Chen93b], approximations of

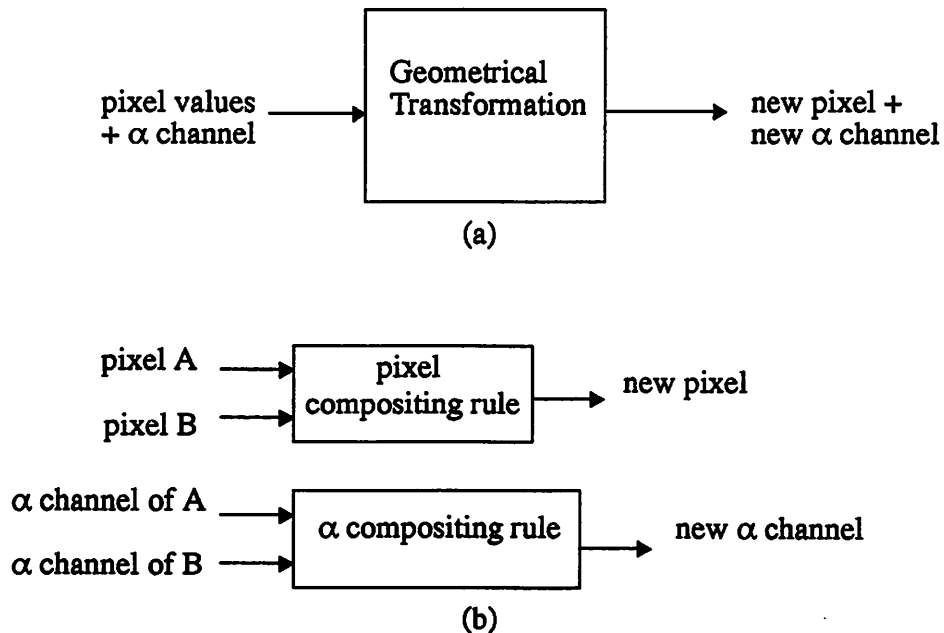


Figure 2-6 Input/output relations for unary and binary compositing functions of ASVO's. (a) many unary compositing function such as a geometrical transformation treat the α channel just like color channels. (b) But most binary compositing functions have special compositing rules for the α channel.

equation (2-5) are taken to achieve incremental implementations. In practice, the depth order of each video object can be represented by a single *priority*. Priority and transparency factors are object-based parameters. By using this object-based priority, we can simplify the compositing process. We will describe the details in section 2.6.3.

There are alternatives for compositing two or more ASVO's. Bitwise operations in hardware can be used to combine simple compositing operations (such as A over B, A and B) [Foley90]. Another technique uses the frame-buffer hardware to implement compositing by table look-up. For example, each image constitutes several bit planes of the table address, and the table contents contain the composited pixel values.

2.6 Representations of Compositing Functions

There have been numerous efforts to standardize the representation formats of multimedia documents and their presentations [Markey92, Kretz92, Ripley89]. In this section, we address the representation problem of the compositing functions. Our goal is not to propose alternatives competing with the fully-defined standardized representation formats. Instead, our goal is to provide an abstract-level framework base on which we can flexibly study other important issues such as mapping compositing functions to distributed processing resources. In addition, the representation methods should conform to the structured video model proposed in Section 2.1.

Particularly, we describe the following representation structures: *expression*, *tree*, and *ordered list*. The first two types of representations have been discussed in the literature [Porter84, Kauffman88, Little91]. But here we define them in the context of the structured video model defined in Section 2.1. The newly proposed ordered list structure has a reduced complexity by assuming an object-based priority parameter and by limiting the compositing functions to opaque and semi-transparent overlapping only. Later, in Chapter 3, we will further investigate restructuring properties based on these representations. Our

goals are to provide efficient representations with enough generality for modeling compositing functions and enough flexibility for adjusting mapping from symbolic representations to hardware resources.

Figure 2-7 shows the relationship between different types of representations for compositing functions. The time-dependent compositing script, as defined in Section 2.4, has the highest generality. It can represent general time-varying temporal or spatial compositing functions applied to individual or multiple video objects. The *expression* structure can be used to represent general time-static compositing functions, including multi-object non-linear compositing operations. Most practical compositing functions can be modeled by the expression representations¹. The *tree* structure is more efficient if the hierarchical property of the compositing functions is emphasized. It has the same level of generality as the expression structure.² The tree representation is particularly useful for distributed or parallel implementations. If we further restrict the allowed compositing functions to overlapping only and assume that each video object is associated with a depth order (denoted as priority τ), the tree structure can be reduced to a one-dimensional *ordered list* structure. As illustrated in Figure 2-7, the inner representations have lower generality for representing compositing functions than the outer ones.

-
1. Some popular temporal effects (e.g. *dissolve*) look like time-varying compositing functions, but actually can be achieved by time-static compositing functions with time-varying compositing parameters.
 2. The tree structure is not constrained to binary trees only. Each internal node can contain two or more child nodes. One extreme case is using a single-level tree to represent the indecomposable compositing functions.

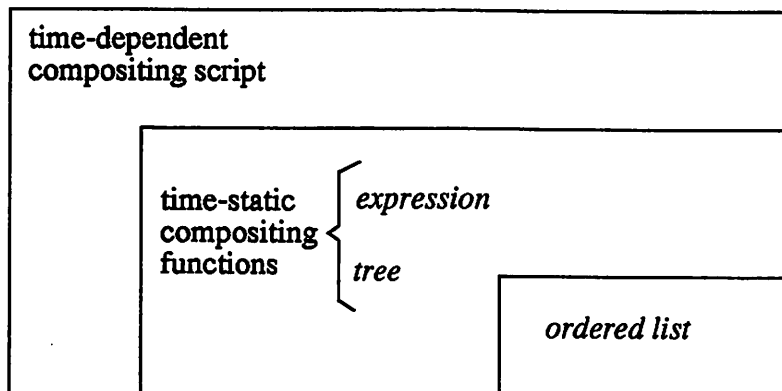


Figure 2-7 Relationship between different types of representations for compositing functions.

2.6.1 Expressions

Expressions can represent any general time-static compositing functions, namely, identical compositing rules for every frame of the video object. A general symbolic expression for representing compositing functions is as follows:

$$V_{\text{composited}} = F(V_1, V_2, \dots, \text{var}_1, \text{var}_2, \dots) \quad (2-6)$$

where V_1, V_2, \dots are video objects, $\text{var}_1, \text{var}_2, \dots$ are variables used by the compositing function to decide how the video objects are composited, and $V_{\text{composited}}$ is the composited video object. The number of video objects can be variable, and so is the number of the compositing parameters. Some compositing parameters, such as location and starting time, can be considered as *attribute* parameters of video objects. Every compositing function must define these attribute parameters of the video objects or use their default values. For a stand-alone video object, these parameters should be viewed as values relative to some reference display device and timing scheme.

In addition to the attribute parameters, different compositing functions have their own compositing parameters. For example, the transparent overlapping function needs the

transparency factor for each involved video object, which is unnecessary in other compositing functions.

As mentioned earlier, the composited results are also video objects. We call them *composited* video objects, in contrast to the *primitive* video objects which are directly produced by a video input device. The interpretation of composited video objects is interesting. It is the goal of the structured video model to keep the logical separation of video objects as far as possible to the destinations to achieve many advantages discussed earlier in section 2.2. But once after the compositing functions are executed, the resulting composited video object cannot be distinguished from a primitive video object. There is no way to extract the ingredient video objects from composited video objects. Therefore, the representation of equation (2-6) is useful only before compositing.

In practice, many useful compositing functions are constructed incrementally and hierarchically. The following is a typical example,

$$V_{\text{composited}} = F(U_g G(U_1 V_1, U_2 V_2), U_3 V_3) \quad (2-7)$$

where F^1 and G are binary compositing functions, U_i are unary compositing functions as those listed in table 2-1, and V_i are video objects. This representation can also recursive since each component video object can be in turn composited video objects and produced by a compositing function like that shown above. However, there are some compositing functions which cannot be decomposed into the above hierarchical format. One example is multi-object non-linear mappings, which are usually defined by look-up tables.

1. Note that we use italic letters to represent compositing functions, among which bold italic letters indicate binary compositing functions and normal italic letters indicate unary functions.

This representation can easily be extended to incorporate more component video objects. The following example shows a practical composited video object with four component objects:

$$V_{\text{composited}} = \text{over}(\text{scale}(\text{over}(V_1, V_2)), \text{over}(\text{scale}(V_3), \text{scale}(V_4))) \quad (2-8)$$

The parentheses indicate the processing order for completing the compositing function. We want to relax this ordering in order to flexibly and efficiently adjust the mapping of the abstract-level representations to actual compositing processors. The motives are significant since on multi-user multi-service networks, application requirements and network configurations are very dynamic. It is important to provide flexible adaptations of compositing systems to dynamic needs. We achieve this flexibility by exploring the characteristic properties of the above generic representations, such as *associative*, *commutative*, and *distributive* properties. Based on these properties, we can restructure the compositing process so that the final structure implies a better mapping to hardware resources. For example, the composited object shown in equation 2-8 can be restructured to a simpler one as follows:

$$V_{\text{composited}} = \text{scale}(\text{over}(\text{over}(\text{over}(V_1, V_2), V_3), V_4)) \quad (2-9)$$

if we assume all scaling factors are the same. This provides potentials for reducing the computational cost since a single scaling operation is used.¹ We will describe the restructuring process in more detail in Chapter 3.

2.6.2 Trees

As described in Equation (2-7), many compositing functions can be described in a hierarchical structure, which can be best described by the *tree* structure. For example, Figure 2-8 shows the tree representation for the compositing function defined in Equation 2-8. Each terminal node represents a component video object; each internal node

1. The actual computational gain depends on the rate reduction ratio of the overlap function.

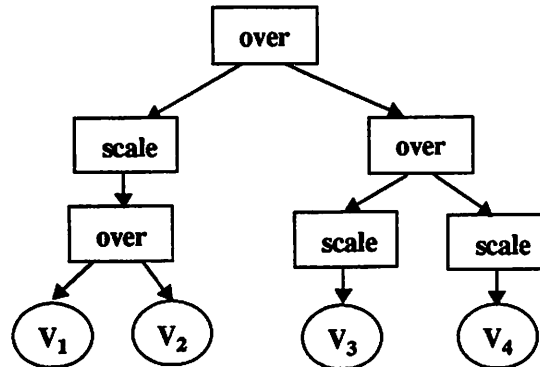


Figure 2-8 A tree representation for the compositing function defined in Equation (2-8).

represents a compositing function used to composite the component video objects defined by the child nodes of this internal node. Note that the tree structure doesn't need to be binary. For example, an indecomposable multi-object compositing function can be represented by an internal node with three or more child nodes. As far as the generality is concerned, the tree structure and the expression structure defined in the last section are equivalent. Figure 2-9 shows an example which includes both binary and ternary operations.

The tree structure is also a natural representation for different implementation procedures. It can clearly specify the order of objects being composited and the way in which unary functions are completed. For example, Figure 2-10 shows two different tree

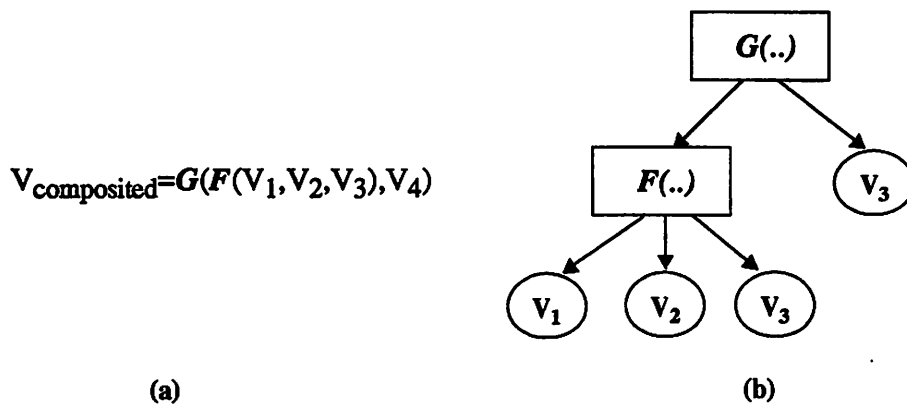


Figure 2-9 (a) A compositing function including an indecomposable ternary operation and a binary operations. (b) The corresponding tree architecture.

representations of opaque overlapping of three objects in different orders. This clear indication of processing orders and possible task clustering is useful in adjusting the mapping from compositing functions to actual processing resources, particularly when compositing functions are implemented distributively or in parallel. For example, Figure 2-11 shows two different tree representations of the same compositing function. One is more suitable for modeling sequential implementation on a single processor while the other one is more suitable for modeling parallel implementation on two processors. In Chapter 3, we will describe the distributed compositing approach for network video services. We will also discuss possible transformations on the tree structure for efficient adjustment of the mapping to practical resources.

2.6.3 Object-Based Priority and Ordered Lists

The expression representation described in Section 2.6.1 can be simplified if we make some assumptions about the compositing rules. For example, as in the *video station* project [Chen93b], we can assign each video object an absolute depth order, called *priority* (τ). Video objects with higher priority cover those with lower priority. Video objects with the same priority are semi-transparently mixed together to produce composited video objects. This absolute depth order is not imposed by the structured video model defined in Section 2.1. Actually, as described in [Lin91], the depth relationship among different video objects can be defined in a relative way (e.g. A on top

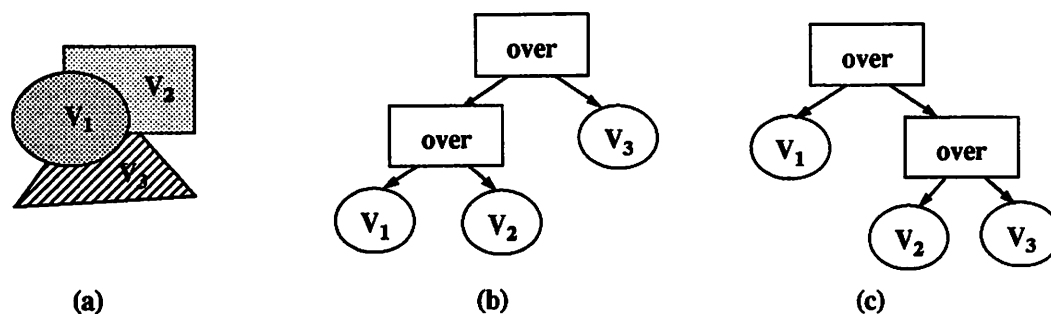


Figure 2-10 There could be different implementations for the same compositing function. (a) the desired effect (b) overlap the first two objects first (c) overlap the last two objects first.

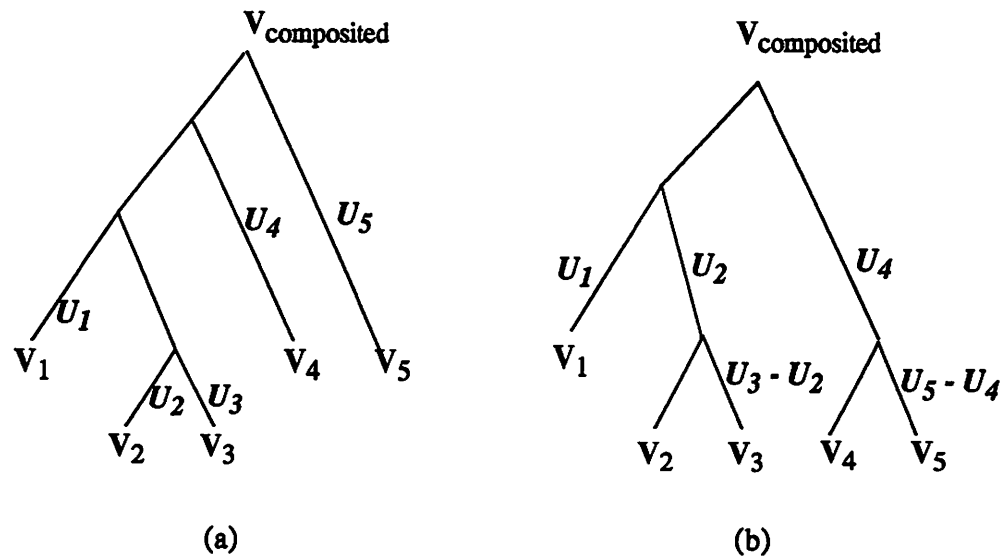
of B, B on top of C, etc.), in an absolute way (using the above absolute order), or based on all possible combinations of video objects. What the structured video model constrains is that all pixels in the same frame of a video object must be displayed in the same depth. But the mechanism of describing the depth information is not specified. Here we choose the above absolute depth parameter in order to simplify the representation complexity.

The above compositing scheme also implicitly constrain the possible multi-object compositing functions to opaque overlapping and semi-transparent overlapping only. We repeat the above descriptions in a symbolic way:

$$\text{When } \tau_1 < \tau_2, V_{\text{new}} = V_1 + V_2, \text{ i.e. } \textit{Opaque Overlapping} \quad (2-10)$$

$$\text{When } \tau_1 = \tau_2, V_{\text{new}} = V_1 \oplus V_2, \text{ i.e. } \textit{Semi-transparent Overlapping} \quad (2-11)$$

where “+” stands for opaque overlap, and “ \oplus ” stands for semi-transparent overlap. In other words, the object-based priority (t) completely defines the inter-object relationship



$$V_{\text{composited}} = U_1 V_1 + (U_2 V_2 \oplus U_3 V_3) + U_4 V_4 + U_5 V_5$$

Figure 2-11 Two different tree representations for the same compositing function. Each internal node is associated with an overlap operation (without explicit notations). +: opaque overlapping, \oplus : semi-transparent overlapping, U_i : unary transformation, V_i : video objects.

(with the supplements of the transparency parameters required in semi-transparent overlapping). Other multi-object compositing functions, such as *abut*, *in*, *out* (defined in Table 2-1), are eliminated.

It is worthwhile to notice some interesting properties of these two compositing functions. The opaque overlap operation is an associative operation, namely,

$$(V_1+V_2) + V_3 = V_1 + (V_2+V_3) , \quad (2-12)$$

but not commutative, namely,

$$V_1+V_2 \neq V_2+V_1 . \quad (2-13)$$

However, opaque overlap and semi-transparent overlap are not associative:

$$(V_1 \oplus V_2) + V_3 \neq V_1 \oplus (V_2+V_3) \quad (2-14)$$

This implies that video sources with equal priorities must be composited before they are composited with other video sources. This kind of restrictions actually constitute an important aspect in compositing systems, denoted as the *constraint* set. There are other constraints such as temporal and spatial binding relations between video objects. Constraints can originate from the video model¹, the compositing functions (like the above processing order), the users' controls, or the video source. The choice of representation methods of compositing functions does not affect the constraint set. The constraint about compositing order described in Equation (2-14) is an inherent property of the overlapping compositing function, not the representation method used. We will discuss more details about the constraint set later.

1. For example, the structured video model imposes a constraint that all pixels in a video object must be composited in the same way.

Another interesting property of the overlapping functions is the *distributive* property. Many useful unary compositing operations (such as geometrical transformations) are distributive with respect to the overlapping function¹, namely,

$$U(V_a) + U(V_b) = U(V_a + V_b) \quad (2-15)$$

That is, the compositing function $U(V_a) + U(V_b)$ can be completed by applying one or more unary transformations on the output of another compositing function,

$V_a + V_b$. We call these two compositing functions *proportional*. Proportional compositing functions are important in distributed implementations of compositing in the sense that they can be implemented on a single processing unit and shared among different users and services. We will elaborate on this issue in Chapter 3.

If we consider only unary operations which are distributive with respect to the overlapping function, every composited scene can be represented by a *canonical* form as follows²,

$$F(V_1, V_2, V_3, V_4, V_5) = (U_{11}U_{12}...)V_1 + (U_{21}U_{22}...)V_2 \oplus (U_{31}U_{32}...)V_3 + (U_{41}U_{42}...)V_4 + (U_{51}U_{52}...)V_5 \dots \quad (2-16)$$

where video objects are composited incrementally with ascending object priority. U_{ij} are a sequence of unary operations applied on object i . The essential concept is that each composited scene has a unique corresponding compositing function, like the one shown above. The inter-object compositing functions can be opaque overlap (“+”) or semi-transparent overlap (“ \oplus ”). Although the compositing functions can be changed to some extent by using restructuring properties such as the associative and distributive properties. The canonical forms are all similar. Users control the layout and appearance of the displayed

-
1. It's possible to artificially design special unary operations which are not distributive with respect to the overlapping function, although they may not be practical.
 2. As mentioned at the beginning of this section, here we allow opaque and semi-transparent overlapping functions only.

scene by changing the object priority, unary transformation of each object, and the position of video objects.

As video objects are defined hierarchically, the compositing function can be defined recursively. For example, the above function can be described as,

$$F(V_1, V_2, V_3, V_4, V_5) = (U_{I1}U_{I2}\dots)V_1 + F_1(V_2, V_3) + F_2(V_4, V_5) \quad (2-17)$$

where $F_1(V_2, V_3)$ composites V_2 and V_3 semi-transparently, and $F_2(V_4, V_5)$ composites objects V_4 and V_5 opaquely.

The above simplified compositing functions can be efficiently characterized by the *ordered list* model, as shown in Figure 2-12. Each video object is represented as a list *element*. Starting from the first element, which has priority zero and represents the virtual background, the priority increases as we proceed along the list. Video objects with the same priority are stored in a branch list, which in turn is an ordered list. The significance of the ordered list representation is in implementation. As shown in equation (2-17), we can complete the whole function by completing any sub-sequence first, followed by compositing the interim results with remaining parts incrementally. Thus, we can grab any portion of the ordered list, complete their compositing operations, put the resulting video object back to the list, and repeat this process until the whole function is completed. However, as shown in equation (2-14), video objects with the same priority must be semi-transparently composited together before they can be composited with objects of different priorities. Thus, branch lists of the ordered list must be grabbed and composited as a unit (however, compositing in the branch list can be incremental).

There is only one unique ordered list representation for each composited scene if we make the assumption about the absolute depth order as described earlier. However, as mentioned above, the compositing functions can be mapped to the physical processing

resources in different ways. The tree structure (described in the last section) is more efficient in representing specific mappings.

2.6.4 Constraints

Constraints are sometimes imposed on video objects' compositing to ensure correct relations between various video objects. For example, the text object and its circumscribing graphic object in Figure 1-4 cannot be displaced independently. The relative position between the weather reporter and the background map cannot be freely adjusted by destination viewers, otherwise the carried information would be incorrect. These special relationships between different video objects may be defined by inputs of video sources, user controls, or hardware limitations. Video sources generate constraints on compositing when video objects are segmented. End users may want to create spatial or logical binding relations between video objects (e.g. multimedia database query). A compositing hardware may have a processing power limit so that it cannot process and composite multiple high-resolution video objects simultaneously in the real time. All these

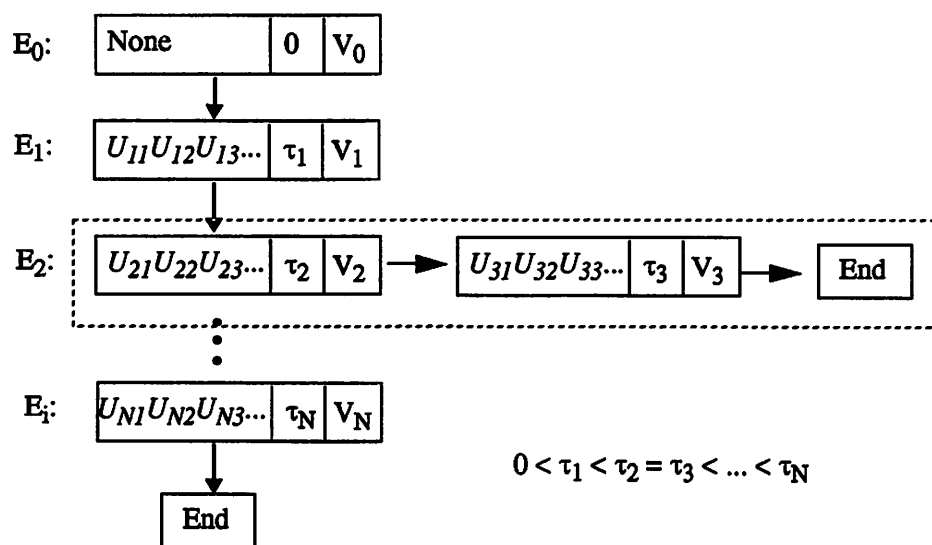


Figure 2-12 The ordered list model representing a hierarchical compositing function. Here we assume only two typical multi-object compositing functions, opaque and semi-transparent overlap. Dotted boundary encircles video objects with the same priority. The first element, V_0 , represents the virtual background.

constraints should be stored in a *constraint set* for each compositing function. All possible implementations of a compositing function must satisfy the associated constraint set, which is a pre-determined domain within which we can seek the optimal implementation of a compositing function. Also, all future modifications of the compositing function (e.g. manipulations of each video object) must conform to the given constraint set. However, users may be allowed to change the constraint set.

The constraint that objects with the same priority need to be composited together before they are composited with other objects can be considered as a *default constraint* for every compositing function when we use the absolute depth order, as described in Section 2.6.3.

Chapter 3

Distributed Network Compositing

As described in Figure 1-1, future multimedia video services include information sources distributed geographically and temporally. The emerging Asynchronous Transfer Mode (ATM) network provides a suitable environment to support these diversified multi-point, multi-media services. In this chapter, we study *distributed network compositing*. In particular, we study the second degree of freedom for implementing the compositing functions — *location*, as described in Chapter 1. The structured video model proposed in Chapter 2 allows us to flexibly and arbitrarily partition the video compositing process along the path from production to the final video display. Compared to the centralized compositing, the distributed compositing approach can provide the flexibility for adjusting different performance tradeoffs (e.g. bandwidth/processing), and the flexibility for adapting video compositing systems to dynamic requirements of different services and users. The distributed compositing approach offers great potential for achieving a higher overall cost-performance ratio.

We will first briefly review the characteristics of the ATM network and discuss its potential for supporting dynamic distributed network compositing. We will describe the general advantages and disadvantages of different compositing locations throughout the network. To satisfy various needs of users and services on multimedia networks, we will propose a *shared distributed compositing* principle for resource allocation. In order to efficiently adjust mapping of compositing functions to network processing resources, we will discuss possible transformations on compositing functions based on the representation formats of compositing functions proposed in Chapter 2. We will leave the optimization of mapping compositing functions to network resources as a challenging open topic, due to the absence of extensive quantitative analyses of various cost functions.

However, we will present some primitive studies of performance factors such as quality, bandwidth, computation, and synchronization. This should be a first step to a concrete complete solution.

3.1 Background

3.1.1 Characteristics of ATM Broadband Networks

As discussed in the literature, the ATM Broadband ISDN (ATM/BISDN) provides an efficient integrated environment for multimedia services including video, audio, and data [Minzer89, Prycker91]. The ATM network uses a unified fixed-sized cell format to transport different kinds of media through networks. Due to progress in high-speed processing and reliable high-speed optical transmission, ATM networks can combine the high transmission speed of circuit-switched networks with the advantages of packet-switched networks, such as dynamic bandwidth utilization, dynamic switching utilization, and integrated service transport¹. Statistical multiplexing of varieties of media sources (such as variable-rate data, voice, video) can greatly increase ATM's utilization efficiency. Also, ATM networks can efficiently support multi-point multi-media connections, which are difficult to achieve in traditional circuit-switched networks.

However, there are still some challenging issues for ATM networks. In particular, for real-time media transmission, issues such as variable delay jitter due to transmission and queueing delays, network congestion due to statistical multiplexing and variable rate traffic, packet loss due to buffer overflow, and service quality guarantees need study. These issues become even more complicated when extended to multi-point real-time interactive services.

1. Although traditional packet-switched networks do not provide integrated service transport.

With respect to network video compositing, the ATM network can support dynamic distributed video compositing, especially, the real-time multi-point multi-connection calls essential for compositing of multi-point distributed multimedia sources. Also, because of the variation in bandwidth between composited image and its constituent, and dynamically changing object areas, the highly flexible bandwidth allocation in the ATM network is essential for flexible partitioning of compositing processes throughout networks. We consider the ATM network as the basic platform for studying distributed compositing in this chapter, though the principle of shared distributed compositing is still applicable to other networks.

3.1.2 Related Work on Multi-Source Multi-User Connections

The problem of distributed multimedia compositing can be categorized in a more general problem domain — multi-source multi-user connections. In Figure 3-1, we show the models for multi-source single-user distributed compositing systems and also multicasting connections. Combining these two cases, the general case is multi-source multi-user connections. There are multiple users on the network and each user can subscribe to multiple multimedia services, each of which may require a multi-to-one compositing process. On the other hand, the same media sources can be accessed by multiple users at the same time. This general multi-source multi-user connection has been studied in the literature at different levels. We will review some related work in this section, focusing on their relations to our network video compositing issues.

Little and Ghafoor have studied compositing of multimedia objects stored in networked distributed databases [Little91]. For each multimedia object, there is a model describing the temporal (e.g. synchronization between constituent signals) and spatial (e.g. overlapping) compositing processes required to produce this multimedia object. They studied the hierarchical representation of the compositing process and discussed the trade-off among different performance metrics when mapping the compositing process to

network resources. This study is very similar to our distributed compositing task, which will be discussed later in this chapter. But it still differs in many aspects. First, they focused on multimedia objects stored in a database, while we consider general real-time video services. Second, we consider compression and its impact on various performance metrics, such as video quality and communication bandwidth, while their analysis did not take compression into account. Third, after representing compositing in a hierarchical way, we study possible ways for restructuring the compositing functions, such as permuting the order of different compositing operations, in order to provide the flexibility for adjusting resource mapping and reduce implementation cost. Lastly, we utilize the *structured video* model of Chapter 2 to separate a video signal into different video objects, such as foreground moving objects and background stationary objects. This logical separation is maintained all the way along the transmission path from the source to the

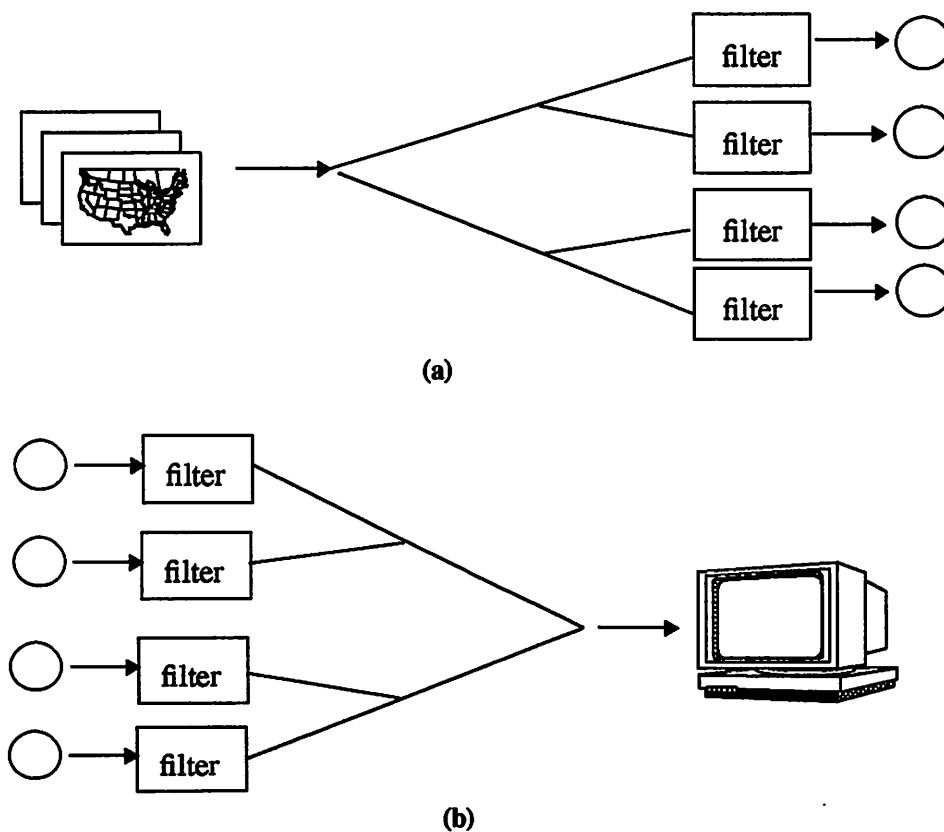


Figure 3-1 (a) multicasting connections (b) multi-source single user distributed compositing. Combination of these two situations produces the general multi-source multi-user connections.

destination so that more efficient compression, transport and display schemes can be utilized.

In addition, Little and Ghafoor concluded that for stored database video, temporal compositing processes¹ are most suitably performed at the destination while the spatial compositing processes can be distributed in the network. This might not be valid for real-time multi-point video services because there are other potential advantages to perform temporal compositing hierarchically throughout the network, such as synchronization.

Rangan *et al.* designed mixing algorithms and hierarchical architectures² for multi-point connections with the absence of globally synchronized clocks [Rangan93]. They found the hierarchical architecture has higher scalability with the number of conference participants, compared to the centralized architecture. This is because each video signal needs to transmit to only one intermediate compositor, and thus the communication and computation load can be more evenly distributed than in the centralized compositing situation. However, they assume that every participant receives the same composited signal broadcast back from the root compositing unit. In advanced multi-point services, each participant can request different compositing operations (such as different screen layout). Therefore, our study has an additional task: to find sharable compositing operations among different users' compositing requests. Also, their performance metrics focus on the synchronization issue only and do not consider the compression process.

-
1. As defined in Chapter 2, temporal compositing deals with the temporal alignment between different media streams.
 2. Note that the hierarchical mixing architecture is comparable to our distributed compositing principle. In their paper, distributed mixing refers to the architecture in which each participant mixes local signal and broadcast signals from all other participants.

Schooler and Casner [Schooler92] designed protocol architectures to support multi-point multimedia connections. Connection managers are located at machines scattered throughout networks and act together to orchestrate multi-point conference connections. Inside connection managers, there is one configuration control unit to communicate configuration information and implement selected services between heterogeneous end systems. They also proposed mechanisms for configuration description language, distributed resource locator, and resource synthesizer, the last of which was originally described in [Nicolaou90]. These concepts are important to our study of distributed compositing. In the process of mapping requested compositing functions to network resources, these configuration management entities such as distributed resource locators are essential in effective resource mapping. Therefore, their work can be considered as complementary to our study in distributed compositing. We propose an abstract model for representation of the compositing process and try to find flexible ways for efficient mapping of compositing functions to network resources. This is at a higher level than actual protocol designs.

Pasquale et al. [Pasquale93] proposed a Continuous-Media Dissemination model for multicasting in packet-switched networks. They advocated the principle of loose coupling between the media source and multiple receivers. Applying the loose coupling principle, they suggest that the source should use embedded coding to send all necessary information to the network and let each user extract the required information by individual filtering. Intelligent network routers should be able to merge paths to different receivers with the same interest (namely media filters) into a single path to reduce communication and computation cost. Similar issues occur in our distributed compositing situation where multiple sources are sent to networks, composited into a single information stream, and then sent to the receiver. But the source-destination relation is reversed. As we described earlier, the most general case will have multi-sources and multi-receivers.

3.2 Video Compositing throughout the Network

As we mentioned before, using the structured video model facilitates the flexibility to arbitrarily partition the video compositing process along the path from production to the final video display. In this section, we discuss where compositing could be done and how to improve the cost-performance ratio by sharing compositing tasks in a multi-user multi-point networked multimedia service.

3.2.1 An ATM-Based Multimedia Network Model

In Figure 3-2, we show a multimedia network model based on the ATM network technology to support dynamic distributed video compositing hardware allocation. The ATM network is composed of a collection of switching nodes which may or may not have the processing capabilities for multimedia compositing. Remote distributed multimedia sources are connected to the network through a network-premise interface (NPI). Within each source node, multiple sources can coexist in parallel, such as parallel disk arrays [Katz90]. Different databases can store different specific types of media such as video, graphics, and text. On the opposite side of the network, individual users are located within perimeters of different Customer Premise Networks (CPN). Each CPN is connected to the backbone network through the network-premise interface, which mainly performs data packet formation, address filtering, and other high-level network functions. In general, users and data sources should be allowed to coexist in the same CPN. For analytic simplicity, we exclude coexistence here without losing generality.

Besides the source (location A) and user (location E) locations, there could be third-party vendors who can access multimedia sources through networks, produce innovative services and then provide them to the users. These third-party vendors can provide compositing equipments remotely outside the broadband network (location C) or

within the local customer networks (location D). In addition, the transport providers may also provide compositing hardware for end users (location B).

3.2.2 General Benefits and Drawbacks of Each Possible Compositing Site

Compositing hardware can be located at any of the locations (A to E) in Figure 3-2. Generally, as the locations of the compositing units (CU) move from sources (location A), through intermediate nodes (location B or C), to the destination sites (location D or E), the number of required CU's increases, the communication bandwidth increases, but the customer control flexibility also increases. In addition, the response to interactive customer control directives becomes more immediate. Therefore, trade-offs exist between

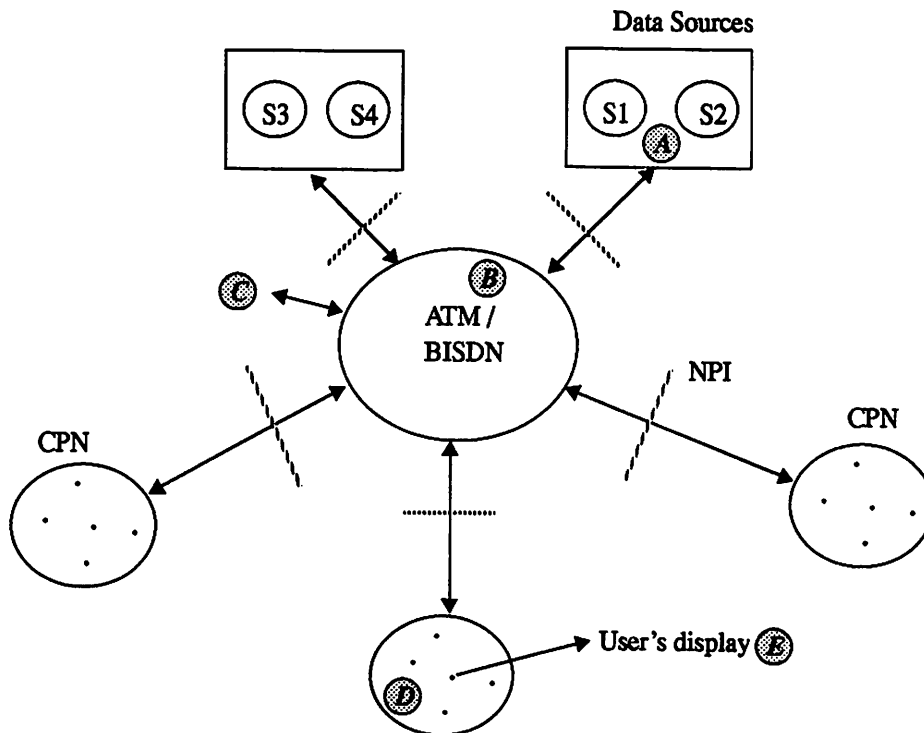


Figure 3-2 A generic ATM-based model for the multimedia compositing systems. CPN represents Customer Premise Network, NPI represents Network-Premise Interface, and S1-S4 are different multimedia data sources. Locations A — E stand for source, network node, intermediate node outside networks, local network node, and destination respectively. Third-party vendors can provide compositing hardware in locations C or D. Some customer premise equipments (labelled as D) can be shared by customers in the same CPN.

different performance metrics, such as hardware cost, communication cost, customer control flexibility, and service quality.

Below we discuss general advantages and disadvantages of placing compositing hardware at each possible location from site A to E. Then, we propose the shared distributed compositing principle in order to match various user and service requirements and improve the overall performance of the video compositing systems.

3.2.2.1 Video Source (location A)

Hardware and transmission costs are kept low if multiple sources are composited together before they are transmitted to users. For example, almost all viewers would want to keep together the graph of a curve and the text labels of its axes.

The shortcoming of this hardware allocation approach is its lack of user controllability. The presentation scenario is determined by the service providers rather than individual customers. To increase the choice of presentations, the source could use multiple processing units to produce a number of different composited video streams. In the extreme case, the source contains one compositing processor per customer. More reasonably, the source could make available two video streams—one with all video objects composited together and another with all video objects separate. If the users are allowed to control the compositing functions by sending commands remotely, a potential long interactive delay is expected.

3.2.2.2 Network (location B)

The so-called *value-added networks* may provide processing capability within them. This approach is similar to the third-party vendor solution but does not require transmission back and forth to the vendor.¹

The compositing hardware can be located in any node within the network. It can be also located in a central node or distributed among several nodes. In Figure 3-3, we show a typical application—*multi-point video conferencing*. The compositing processors receive different video sources from different locations and perform the compositing operations. Then, the composited video is broadcast to every participant. There are trade-offs between the hardware cost and flexibility. In Figure 3-3(a), a single compositing unit (CU) is used to produce a unique presentation, in which all conference participants are of the same size. In Figure 3-3(b), two CU's are employed to produce different presentation formats, one of which has a bigger window for the speaker and smaller windows for the rest. Note these two CU's can share transmission of video sources and some common compositing computations. For example, the computations for compositing the silent participants can be shared.

In addition to the sharing of the hardware cost, the transmission cost can also be shared among customers. For example, the number of communication links required here

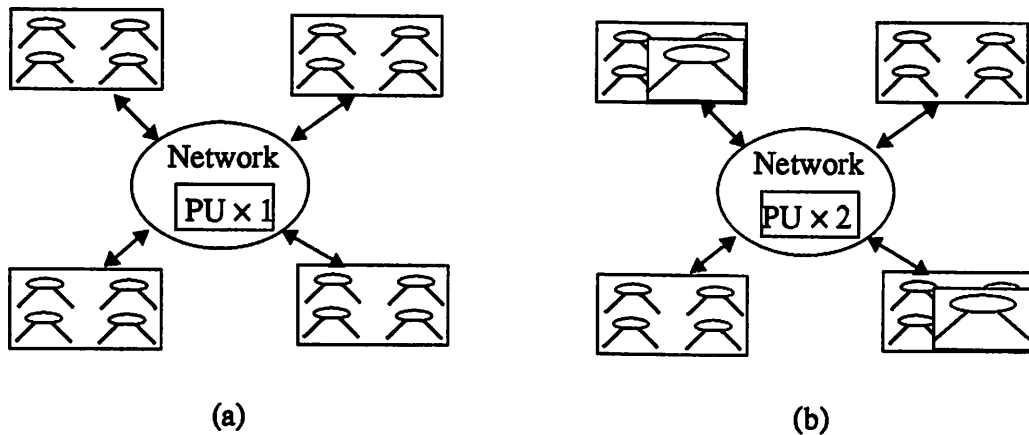


Figure 3-3 A simple example of multi-point video conferencing. Different computational resource allocation schemes result in different levels of flexibility of user controls. In (a), a central compositing unit (CU) produces a unique presentation. In (b), multiple CU's produce more presentation choices and thus users can have higher control flexibility.

1. Currently, U. S. telecommunications network providers are forbidden by law from offering data-enhancement services such as video compositing.

is of the order of N , where N is the number of conference participants. This is much lower than the $N(N-1)$ links required by the allocation scheme that uses dedicated hardware at each participant's site. The number of compositing processors located within a value-added network should be determined by the customers' demand for compositing. Third-Party Vendors (location C or D)

Compositing hardware could be provided by third-party vendors that have access to broadband networks. The hardware could be located in remote areas or within local customer networks. This solution is similar to the location of compositing hardware within the network.

If third-party vendors compete to offer compositing services, a great deal of service innovation should result. For example, some vendors might offer several sports programs in a single presentation. Others could superimpose financial teletext over news or weather programs.

3.2.2.3 Customer Private Networks (location D)

If a number of customers at a site need to see the same display, video compositing should be performed only once. Placing compositing hardware within the local network where the customers are located would be most efficient. These processing units can be provided by third-party vendors (as mentioned in last section) or can be installed as shared customer premise equipments (like shared computer printers and storage devices on local area networks).

3.2.2.4 Final Video Display (location E)

This solution provides maximum flexibility. Every user has dedicated compositing hardware and thus has full control over the presentation on the display with minimum interactive delay. For the previous video conferencing example, this hardware allocation

scheme lets every participant arrange his or her own presentation scenario. The hardware cost is highest with this scheme, simply because so many CU's are required. However, the cost still may not be too expensive if only one (or two) video sources are transmitted every displayed frame and the rest are updated infrequently. For example, today's workstations display many video objects such as text windows and graphics, all of which are updated only infrequently. The communication cost depends on the extent that the requested data sources are shared. For example, if there are only few available data sources but many customers requesting different presentations, then the communication cost per user is low since data transmission is heavily shared. On the other hand, if most customers request the same presentation, then this hardware allocation approach is not as efficient as those which composite video objects in the earlier stages.

3.2.3 Principle of Shared Distributed Compositing

An efficient compositing hardware allocation scheme should take into account both the dynamic requirements of services and users on multimedia networks. First, the characteristics of different multimedia services vary widely. Usually, broadcasting programs (e.g. weather reports) and public shared services (e.g. electronic yellow pages) have fixed compositing scenarios that are determined by service providers. User interactivity is not required in these cases. But for other services such as interactive database query and access, user interactivity is definitely necessary.

From the viewpoint of user diversity, different levels of user controllability are required. Some users may not want to pay for dedicated compositing hardware. Instead, they use the compositing equipment provided by private vendors. For example, vendors may composite two sports programs together, news and stock prices, or multiple movie channels into single streams for the end-users. However, other users may prefer full flexibility of control and enjoy more advanced services such as desktop video editing and publishing.

Therefore, the efficient compositing hardware allocation schemes must reflect different service characteristics and dynamically changing user demands. The allocation must be *distributed* since hardware should be allocated over a widely dispersed area, as opposed to a centralized allocation scheme. The resource allocation must be *dynamic* since different users or services can dynamically reclaim the hardware resources in a time sharing mode. For example, a group of users in the same local area network can dynamically share some compositing units in the same local area network. There could be some public compositing units available in the central broadband network and some private compositing units provided by outside vendors. At the end sites, video source sites can have some compositing units to combine logically bonded video objects before transmission. End users who are multimedia hackers can invest in dedicated compositing resources. By distributing the compositing hardware, a minimal number of compositing resources are required to meet the widely diversified user interactivity requirements; by allocating the hardware dynamically, the hardware cost is further reduced.

The connection complexity, and possibly the computational complexity, can be reduced if a group of users can share some common compositing tasks. These shared compositing tasks can be extracted by some intelligent algorithms in the control entity and completed on a separate compositing unit (within or outside the network), which is shared among the whole group of users. The composited results are then combined with individual remaining parts to produce different scenes for each user. We call it *distributed* because compositing functions are completed distributively by many processors and *shared* because each composited processor may be shared by many users.

There are two ways to interpret this concept of hardware sharing. First, from the viewpoint of network service providers, given arbitrary compositing scenarios of multiple users and services, how do we allocate the compositing processors throughout the network to satisfy these compositing tasks at a minimal cost? Using the compositing model

described in section 2.6, each compositing task is associated with some abstract representation, such as the expression representation or ordered-list representation. Based on these representations of compositing functions, the service agent should try to find the most profitably shared compositing tasks (e.g. long identical segments or segments shared by many users in the ordered-list representation), and then allocate appropriate hardware to implement these common compositing tasks. In the example shown in Figure 3-4, n users share a common compositing task which contains m input video sources. The connection complexity is reduced from $n \cdot m$ to $n+m$. However, the service provider needs to assume abundant hardware resources and should be able to perform sufficiently fast hardware reallocation.

Another perspective of this concept of shared distributed compositing is that given limited amount of hardware resources at the disposal of network service providers, how do we maximize their capability in terms of the variety of supported compositing programs? Network providers can collect the past usage statistics of user compositing programs and implement the most popular programs on the available hardware. In this way, users' requests can be satisfied to the maximal extent by using limited hardware resources.

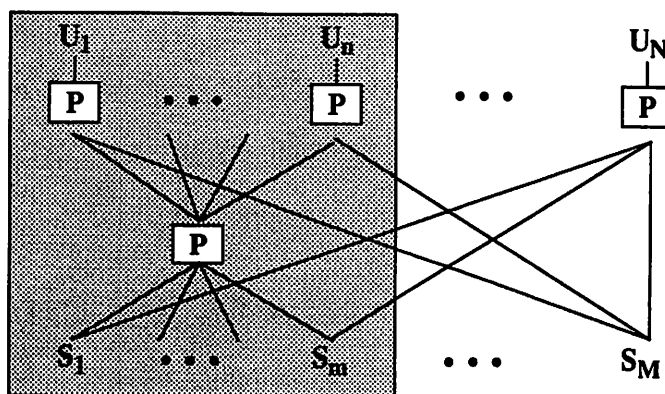


Figure 3-4 An example of shared distributed video compositing. Several end users share a common compositing task and complete their displayed scenes in a distributed way. P: compositing processors.

An important issue emerging immediately is how to find beneficial sharable compositing tasks among many users or services. Abstract-level hierarchical representations of compositing functions proposed earlier in Chapter 2 will prove to be useful here. In the next section, we will explore some abstract-level methods for restructuring and partitioning the compositing functions in order to flexibly adjust their mapping to network resources (e.g., bandwidth and processing).

3.3 Mapping of Compositing Functions to Network Resources

3.3.1 Optimal Resource Allocation — An Open Problem

As described in section 2.6, each compositing function can have many different implementations, each with its own performance and cost. Figure 2-10 and 2-11 (in Chapter 2) illustrate two typical implementation examples. In this section, we try to address a challenging problem — given limited network resources (including bandwidth and compositing hardware), and diversified requirements of multiple users and services, how do we organize and partition the compositing functions appropriately, and then map them onto suitable network resources? In optimization of mapping compositing functions to network resources, different performance weighting from different users and services also need to be considered. Figure 3-5 shows a block diagram describing this challenging open problem.

Our goal is not to design any optimization algorithm for any specific network environment or application. Instead, based on the abstract-level representation we proposed in Chapter 2 and the shared distributed compositing principle described above, we focus on general approaches for effectively adjusting mapping of compositing functions to network resources and flexibly tuning various performance tradeoffs.

3.3.2 Transformations of Compositing Function Representations

As defined in section 2.6, a general compositing function can be represented by an expression as follows,

$$V_{\text{composited}} = F(V_1, V_2, \dots, var_1, var_2, \dots) \quad (3-1)$$

where V_i are video objects and F is the compositing function. Most of useful compositing functions (as those defined in Table 2-1) can be defined hierarchically.¹ In particular, they usually can be decomposed into binary operations. The following is an typical example,

$$V_{\text{composited}} = F(U_g G(U_1 V_1, U_2 V_2), U_3 V_3) \quad (3-2)$$

where U_i are unary compositing functions applied on individual video objects. We focus on these decomposable compositing functions only in this section.

We have shown that there are many different implementations of each compositing function. For example, overlapping of three objects can be completed with either the first

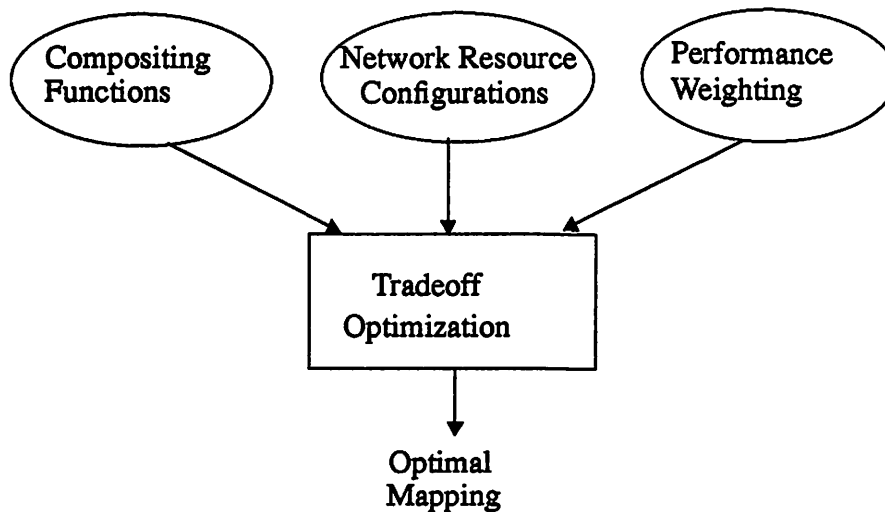


Figure 3-5 A challenging open problem — optimization of mapping compositing functions to actual network resources.

1. Generality of different representations of compositing functions are discussed in Section 2.6.

two or the last two objects overlapped first. It is this implementation flexibility that enables transformation of structured video representation and searching for its efficient mapping to network hardware in order to match the dynamic needs of applications and resources. In this section, we discuss several properties of the expressions for video compositing.

A binary compositing function, F , is *associative* if

$$F(F(V_1, V_2), V_3) = F(V_1, F(V_2, V_3)) \quad (3-3)$$

Examples of associative compositing functions are *overlap*¹, *transparent*, and *in*(V_1, V_2). The binary compositing function *out*(V_1, V_2) is not associative. Definitions of these binary functions can be found in Table 2-1. The significance of the associative property is that it provides the freedom of an arbitrary compositing order. For example, if video components are generated incrementally along a bus network, it is natural to composite these component objects one by one sequentially. On the other hand, if sources of component objects are grouped into several locations, parallel compositing maybe more efficient. In terms of hardware complexity, sequential compositing may facilitate efficient implementations like pipelined compositing architectures. But if sequential compositing is distributed among many different locations, the end-to-end latency will be long.

A binary compositing function, F , is *commutative* if

$$F(V_1, V_2) = F(V_2, V_1) \quad (3-4)$$

Examples of commutative compositing functions are *transparent*² and *pixel multiplications*. *Overlap*, *in*(V_1, V_2), and *out*(V_1, V_2) are not commutative. When combined with the associative property, the commutative property can determine whether multiple video

-
1. Note that here we assume an absolute order for overlap. If relative-order overlap is used, the final composited result may depend on the order we apply the overlap rules [Lin91].
 2. If we also change the associated transparency factors.

objects can be composited in any arbitrary order or not. The associative property enables compositing of any subset of video objects before they are composited with others. The commutative property enables changing the compositing order of any specific video object. This is important since video objects may come from many different remote locations and their geographic locations does not necessarily match the specified compositing order. If the specified compositing order is unchangeable, video objects may need to be sent to a central node for compositing, or some distant video objects may need to be sent to the same node and composited together before they are transmitted to another distant location.

Note that for unary operations, the commutative property can be defined as $U_1(U_2(V)) = U_2(U_1(V))$, where U_1 and U_2 can be the same operation. For example, scaling and translation are commutative.

The distributive property is defined for a unary or binary function with respect to a binary function. Unary operation U is *distributive* with respect to a binary function, G , if

$$U(G(V_1, V_2)) = G(U(V_1), U(V_2)) \quad (3-5)$$

A binary function, F , is *distributive* with respect to a binary function, G , if

$$F(G(V_1, V_2), V_3) = G(F(V_1, V_3), F(V_2, V_3)) \quad (3-6)$$

and

$$F(V_1, G(V_2, V_3)) = G(F(V_1, V_2), F(V_1, V_3)) \quad (3-7)$$

Several distributive relations are listed in table 2.

The first advantage of using the distributive property is the reduction of redundant operations by extracting the common operations shared by two video objects, resulting in the reduction of computation in most cases. But the actual amount of computation

reduction depends on the specific operations applied. For example, simple non-overlapping compositing of two objects does not reduce the total amount of data and thus does not reduce computation when a common operation is extracted. On the other hand, using the distributive property to apply the same operation on both component objects before they are composited together may be advantageous in some cases. For example, scaling down the component video objects before they are transmitted to a network node for compositing can save some transmission bandwidth compared to transmitting the full-sized video components and doing down-scaling at the network node.

Table 3-1 Basic restructuring properties of some typical compositing functions. A: associative, C: commutative (C_b for binary, C_u for unary), D: distributive (*binary* with respect to *binary*, or *unary* with respect to *binary*)

		Binary (G)				Unary			
		Over	Transparent	In	Out	Scale	Translate	Rotate	Flip
Binary (F)	Over	A ^a ,D	D	D	D	(not applicable)			
	Transparent	D	A, D, C_b	D	D				
	In	D	D	A	D				
	Out	D	D	D	^b				
Unary (U)	Scale	D	D	D	D	C_u	C_u		C_u
	Translate	D	D	D	D	C_u	C_u		C_u
	Rotate	D	D	D	D			C_u	C_u
	Flip	D	D	D	D	C_u	C_u	C_u	C_u

- a. Opaque overlapping is associative if we assume an absolute depth order. If we use the relative depth order, then it's not associative.
- b. $(S1 \text{ out } S2) \text{ out } S3 = S1 \text{ out } (S2 \text{ over } S3) \text{ or } S1 \text{ out } (S3 \text{ over } S2)$.

These basic properties for restructuring the compositing functions facilitate an efficient and systematic approach to matching the structured video implementations to dynamic application requirements. For example, if transmission bandwidth is the most important resource, we should perform compositing at nodes closer to the sources and

complete rate-reducing operations (e.g. down scaling) at the source sites or close to the source sites. If the computational complexity is of most concern, then finding the simplest computation form should be pursued. On the other hand, if users want the most responsive control, then keeping video objects separate all the way down to the user sites should be most beneficial. Besides these considerations, there is a challenging issue of optimizing the mapping of several structured video representations for many different services at the same time. A suboptimal mapping for single-user services may become the optimal mapping when multiple services are considered together. In a multi-user heterogeneous-service situation, the above restructuring techniques are useful for finding the sharable compositing processes among different users and different services to reduce the overall implementation cost.

3.3.3 Possible Transformations on Trees

As mentioned earlier, there could be many different ways for completing the same compositing function. The tree structure is suitable for representing each practical partition and mapping of hierarchical compositing functions to network resources. It can also clearly indicate the order in which the constituent objects are composited. In this section, we describe the possible tree-based transformations corresponding to the associative, commutative, and distributive properties mentioned above. Given a compositing function represented in the tree format, these transformations are useful for adjusting the mapping and partition of the compositing function to processing resources. Note that we focus on the compositing functions which can be decomposed into binary operations only.

Another implication of the tree-based transformations is related to adding new services (and thus their associated compositing functions). New compositing functions may affect the efficiency of the mapping of the existing compositing functions and mandate restructuring of their mappings to network resources. For example, new sharable

compositing tasks may be created when new compositing functions are added. Restructuring the existing mappings is needed to take advantage of these new sharable compositing tasks. Knowing possible transformations on the tree structure is important for restructuring the mapping of compositing functions.

Figure 3-6 shows all possible transformations on the tree representation. Definitions and examples of these transformations can be found in the last section. The *associative* property of binary compositing functions allows pair-wise grouping of three objects in different ways. One binary function is moved from one branch to the other branch of another binary function of the same kind. The *distributive* property has two different types — *binary with respect to binary* and *unary with respect to binary*. Both these two transformations propagate the common compositing operation from the top of an internal node (i.e. a binary function) down to both of its child nodes. Actually, the former can further be separated into two different types based on the order of the binary functions, as shown in Figure 3-6(b₁) and (b₂). Lastly, the commutative property allows swapping the order of two objects of a binary function, or swapping the order of two unary functions of the same video object.

Figure 3-7 shows an example illustrating the distributive property and its corresponding tree transformations. The distributive property is equivalent to *merging* two identical blocks from under into a single block on the top of another binary compositing block, as shown in Figure 3-7, or the other way around (*splitting*). As we described at the beginning of this section, these transformations are useful for adjusting the mapping of compositing functions to processing resources. However, finding the optimal mapping of each compositing function requires extensive studies of various cost factors, some primitive analysis of which will be described in Section 3.5.

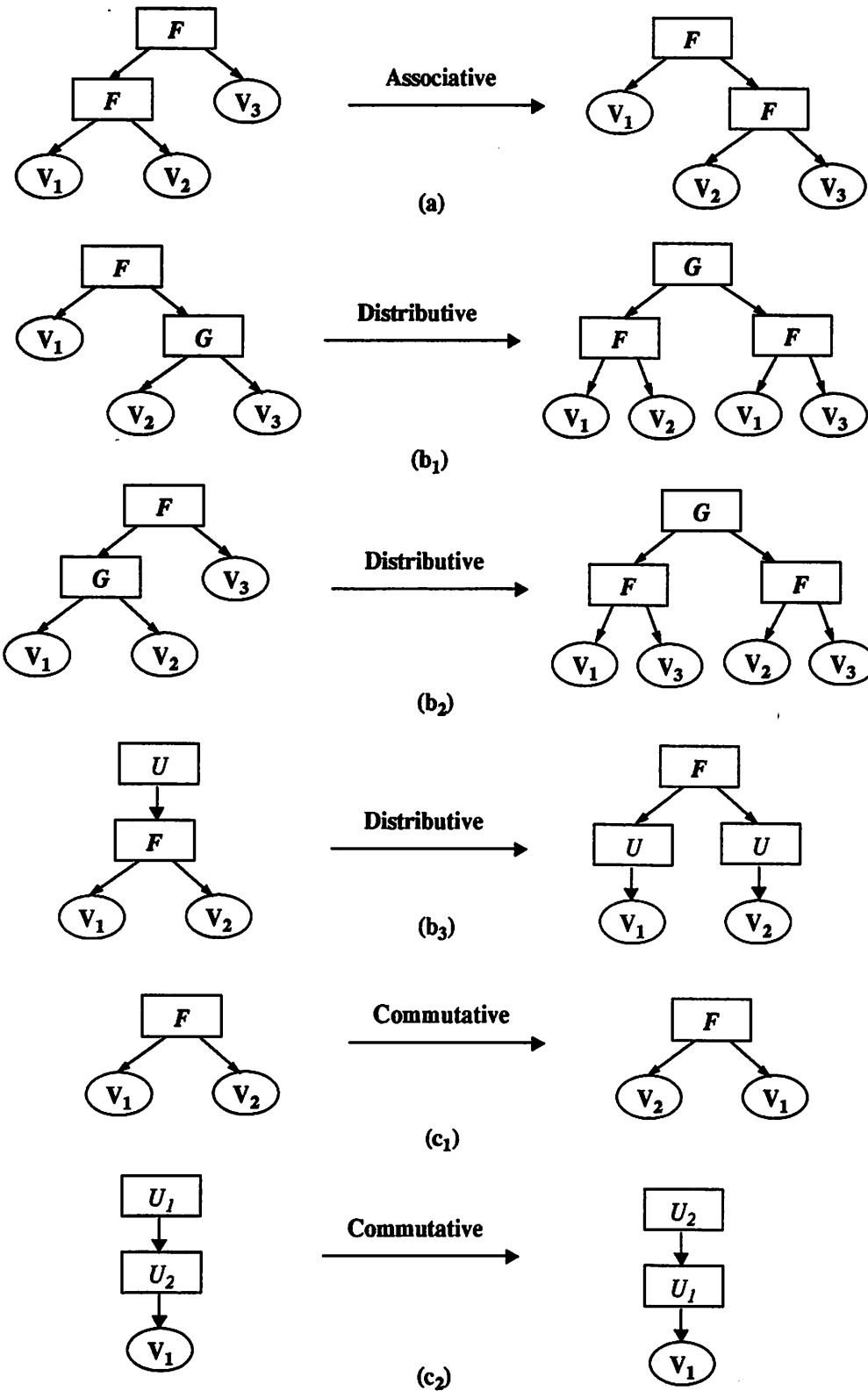


Figure 3-6 Possible transformations on the tree structure. (a) associative binary functions (b₁-b₂) distributive binary functions (b₃) distributive unary functions with respect to binary functions (c₁) commutative binary functions (c₂) commutative unary functions.

3.3.4 Possible Transformations on Ordered Lists

The ordered list model is proposed in section 2.6.3 to represent the compositing tasks when we use the object-based depth order (priority τ) and restrict the compositing functions to opaque and semi-transparent overlapping of arbitrarily-shaped video objects only. One ordered list could be like the one shown in Figure 2-12. Video objects are linked together according to their depth order parameter, the priority. By comparing the ordered lists of different compositing functions, we can find identical or proportional compositing segments. Proportional compositing segments are those which can be derived from one base segment by some simple unary compositing operation, as defined in equation (2-15). Identical segments need to be executed once only. Proportional compositing segments can share hardware by implementing the base segment first, followed by unary operations to generate each individual compositing segment. Complexity reduction increases as the shared segment is longer or more compositing functions share the same segment. In order to find the most profitable shared compositing segments, we need to consider various weighting factors such as computational complexity of each unary operation, bandwidth requirement, number of users sharing the same compositing segment, etc. Some primitive analysis results will be described in Section 3.5.

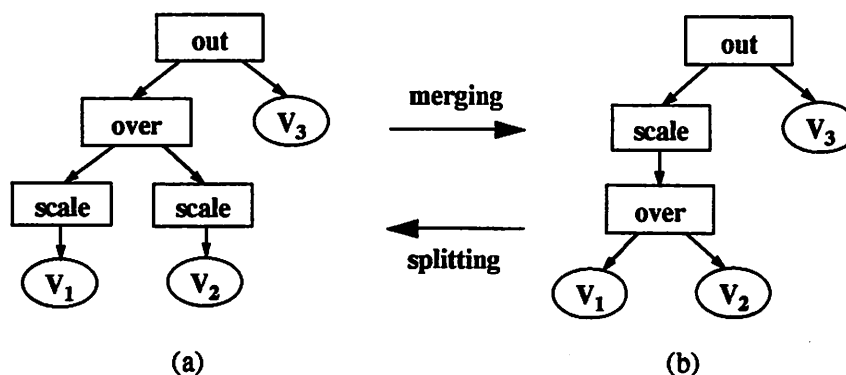


Figure 3-7 The corresponding action of applying the distributive property on the implementation tree, from (a) to (b) — merging, from (b) to (a) — splitting.

Since we assume an absolute depth order, the opaque overlapping used here is associative. Therefore, given an ordered list, we can process different segments of the ordered list in an arbitrary order. That means we can grab any sub-lists arbitrarily and implement them first. This flexibility is important for finding the sharable compositing tasks between multiple compositing functions, as described above. Note that the priority item in each list element is used to determine the order of video objects only, it does not involved in the process of finding the sharable compositing segments. Lastly, because the opaque overlapping is not commutative, we can not switch the positions of the list elements.

3.4 Impact on ATM Broadband Network Design

Distributed and dynamic hardware allocation schemes impose many challenges on ATM networks. We now discuss some important features that ATM networks must provide to support advanced multimedia services.

- **Support All Locations for Video Compositing Hardware**

As mentioned above, there are many different types of applications that are best supported with different locations for compositing hardware. The ATM network must provide an open access environment that allows compositing hardware in all locations to function identically. The network must be able to transmit signals in multiple formats, including compressed video and video that contains embedded compositing information.

- **Multi-Connection, Multi-Point Calls**

Networks must allow one user to establish multi-connection calls to other sites (to get multiple video sources or different media streams). This allows third-party vendors or customer network equipment to composite video for end-users. Also, networks must support multi-point calls such as for multi-point video conferencing. Furthermore, some users may want to set up these two types of multimedia calls at the same time.

- **Dynamic Connection Establishment and Bandwidth Allocation**

Dynamic connection establishment and bandwidth allocation allow services that composite a varying number of sources. For example, in interactive services like interactive database access or desktop video editing, some video streams may be requested or removed conditionally based on user inputs. This requires a dynamic scheme to establish or tear down connections and allocate transmission bandwidth.

- **Heterogeneous Information Access**

For example, instead of receiving compressed video sources and decompressing them locally, a user might want to have a video signal decompressed somewhere within the network but not care where. That is, a user might request a connection to the lowest-cost or least heavily loaded video compositing vendor. The network has to keep track of these resources, calculate the user's cost for these resources, and be able to allocate and reclaim the resources.

The resource allocation problem is very similar to that encountered in designs for multi-input, multi-output parallel processors. All issues such as scheduling, inter-processor communications, and synchronization need to be solved in order to obtain an efficient implementation. For example, in a multi-point video conference, implementation cost can be reduced if video compositing can be completed hierarchically.

- **Identify and Eliminate Redundant Traffic Flows**

In a wide-area network, many databases and compositing processors can be shared among different users and services. Usually, much video transmission can also be shared. For example, if seven million people all watch the Super Bowl football game, there is no need to establish a connection between each user and the game's originating studio. Single connections can be made to local distribution centers, and users can receive their signal from them. Another example is that when compositing hardware is allocated distributively

within a network, common subcomponents or group of subcomponents of different final presentations can be shared among different processing units. If the network can identify and eliminate parallel connections that carry identical data, then more network resources are available for other services.

The identification of parallel traffic flows can be handled during routing. When deciding whether or not to accept a requested call, the network can check if the new call carries identical data to an existing call. Every time a request arrives to modify an existing connection, the network again must check if any links become or cease to be redundant.

3.5 Primitive Performance Analyses

The above approach of distributed compositing and hardware sharing needs to be justified in terms of performance figures such as *implementation complexity*, *user controllability*, and *quality of service*. Some general high-level discussions of different performance metrics have been described in several sections. To reiterate briefly, the communication cost and computational cost can be reduced if compositing is done near the source location and shared among multiple users or services. But the penalty is that user controllability is also reduced. Distributed hierarchical compositing can alleviate the problem of overwhelming traffic and computational load at a single node, and thus boost scalability of the multi-point connections.

In practice, there are always some exceptions to the above trend. For example, if the compositing operation increases the data rate of the video signal (such as spatial or temporal resolution up-scaling), then processing at the final destination can avoid greater demand of the communication bandwidth. But in general, most compositing operations produce a reduced data rate (such as overlap, down-scaling), or at most maintain the output rate more or less equivalent to the net input rate (such as strictly temporal compositing).

In this section, we further the study of performance comparison at different compositing locations and of the distributed compositing approach. We are particularly interested in two important issues related to real-time video services — received video quality and synchronization of isochronous traffic.

3.5.1 Received Video Quality

The received video quality is heavily affected by the compression process, which is mandatory in most network transmission of video signals. The approach of distributed compositing will require more than one iteration of the compression—decompression process. If the compression process is lossy (such as quantization), then there is a potential that the quality degradation due to tandem lossy compression will be increased. Figure 3-8 shows a block diagram to illustrate the error accumulation problem. This issue is also very important in the field of stored compressed video retrieval and playback. The actual significance of the accumulated error depends on the compression algorithm in use. For example, the widely used Discrete Cosine Transform (DCT) is error-accumulation free if we ignore the accumulation of round-off computational errors. It is because that the transform coding algorithm maps the recovered data to the same quantized values in the second round of compression. Therefore is no additional quality degradation in the second round of compression process.

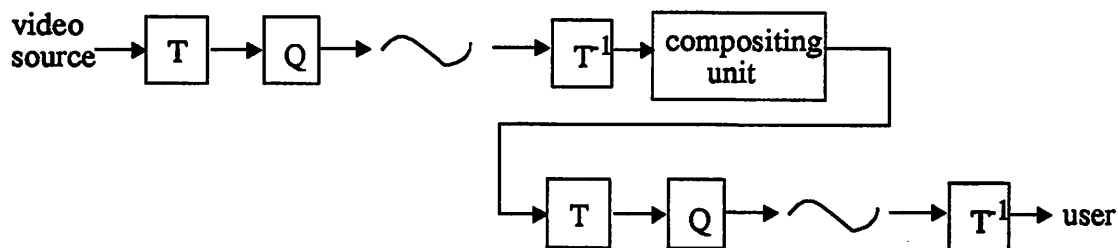


Figure 3-8 If the compositing unit is located in an intermediate node (either within or outside the network), the compression(T)—decompression(T^{-1}) process needs to be duplicated, including the lossy part.

However, for another popular compression algorithm, motion compensation (MC), image distortion could get worse and worse after iterative compression, if quantizers are not designed appropriately. To elaborate, if the quantizer uses the center point of each decision interval as the reconstruction value, as shown in Figure 3-9, then it is possible that the reconstruction value of the quantizer is bigger than the input sample value, i.e. the error term of the MC algorithm. When we re-apply the MC process to the reconstructed video, a different motion vector might be favored and thus additional noise is introduced in quantizing the new error term. For example, in Figure 3-9, O_1 stands for the current image pixel and $P_1 - P_4$ stand for possible reference pixels in the previous image frame. Suppose in the first MC process, P_1 is chosen as the optimal reference value and the error between P_1 and O_1 is calculated and quantized. The recovered value from the inverse MC, denoted as O_2 in Figure 3-9, can be calculated as follows

$$O_2 = P_1 + Q(O_1 - P_1) \quad (3-8)$$

where Q stands for the quantizer. Since the quantizer uses the center value of the decision interval as the reconstruction value, the reconstructed pixel value, O_2 , could be more distant from P_1 than O_1 and moved into the decision cell of another reference pixel, say P_2 . Therefore, P_2 will be chosen as the new reference pixel and new quantization noise will be introduced when the second MC compression process takes place. The actual amount of error accumulation depends on the video sequence and the quantizer. However, using on one non-uniform quantizer from [Netravali88] and some real video sequences, we did not find serious image degradation due to the above error accumulation problem associated with the MC compression algorithm.

The above error enhancement problem can be avoided if the quantizer always pulls the input values inward, namely, using the lower decision level to be the reconstruction level. Thus, the reconstructed value from the inverse MC process will never exceed the original cell chosen in the first MC compression process. The same motion vector and

error term can be used in the second and later compression processes and no additional distortion is added.

The significance of above image quality degradation issue will become more complicated if we further perform some compositing functions before the second compression process, i.e. in the block of compositing unit in Figure 3-8. Unlike the above observations, the net accumulated distortion may become quite significant. This is mainly because that composited image may have different pixel values with the original image and thus has to suffer distortion from another round of lossy compression. For example, in Figure 3-10, we show the block diagrams illustrating three different approaches of doing down scaling at different locations — source, user, and intermediate node. We use the DCT algorithm plus uniform quantization as the compression algorithm.

Figure 3-11 shows the reconstructed pictures by using different compositing locations. If we use a coarse quantization table (36 for each AC DCT coefficient), we can clearly see that the intermediate node approach suffers the most serious degradation, with the source site approach better, the user site approach best. The objective PSNR figure is 33.2 dB, 33.7 dB, and 38.0 dB respectively. However, if we use the example quantization

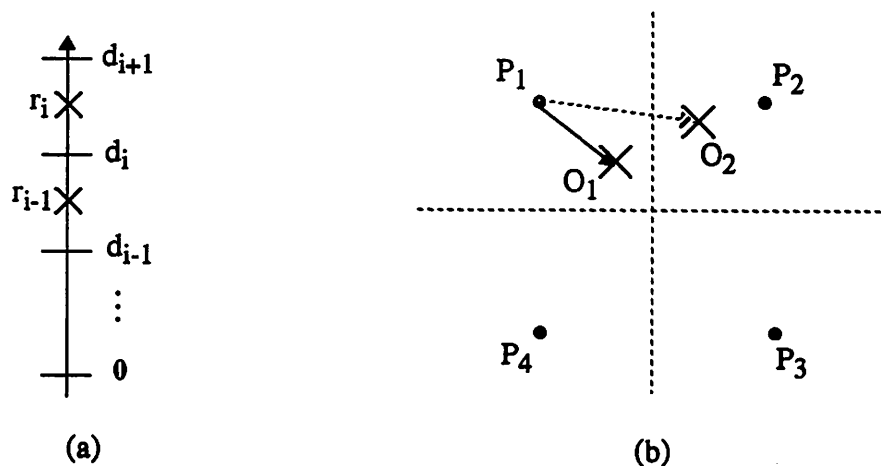


Figure 3-9 (a) The quantizer which uses the center point of the decision interval as the reconstruction value. d_i : decision levels, r_i : reconstruction levels. (b) A possible situation in which more distortion is added in the second round of the MC compression process. P_i : reference pixels, O_i : current pixel, O_2 : recovered pixel after inverse MC process.

table listed in the JPEG standard [JPEG91], the subjective quality seems to be more comparable among these three different approaches, although the objective PSNR figures are still quite different — 34.7 dB, 34.9 dB, and 42.3 dB respectively. This maybe because the SNR is high enough to conceal the effect of significant SNR difference.

The above comparisons imply that the image quality degradation caused by repeated compression heavily depends on the compositing operations performed between consecutive compression processes, the quantizer designs, and of course the compression algorithm. We need to take this issue into account when we use the approach of distributed compositing and avoid serious image quality impairment caused by this error accumulation problem.

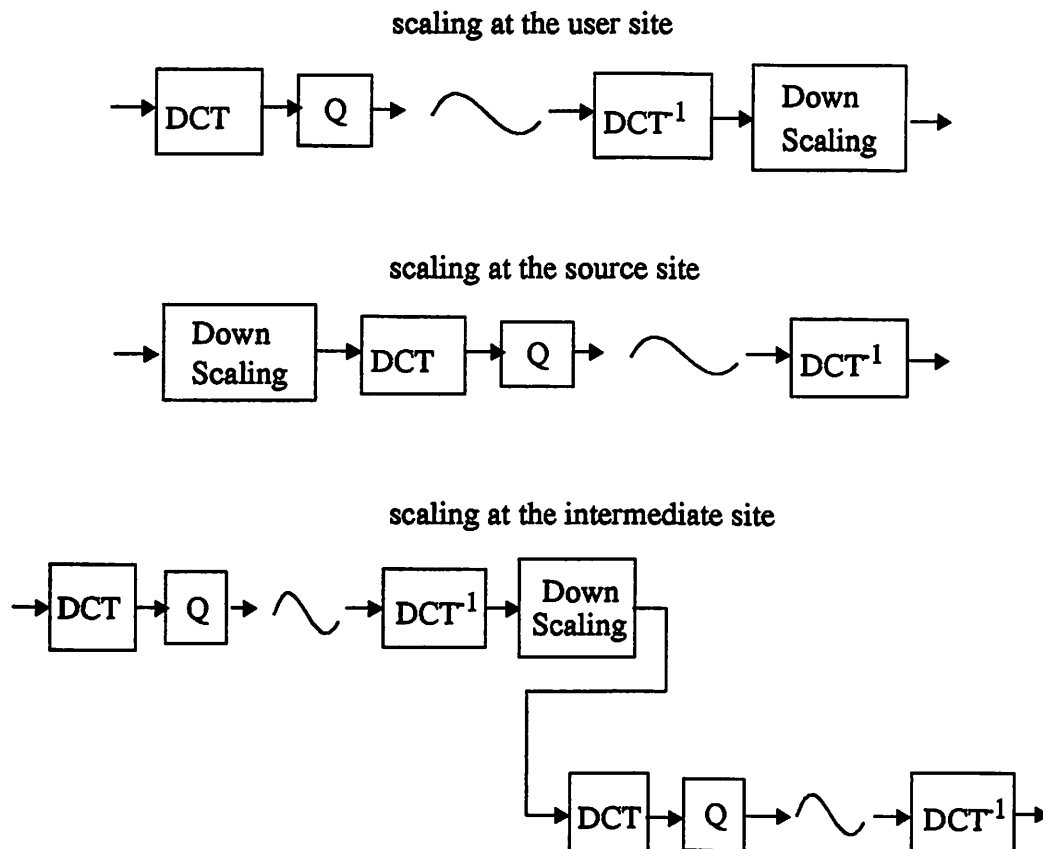


Figure 3-10 Perform down scaling operation at different locations — source, user, and intermediate site

Another interesting phenomenon worthy of discussion is that the user-site compositing produces much higher quality (at least in terms of the PSNR figures) than the alternative of compositing at the source site. This is mainly because of the reason that the down scaling operation used in this example basically includes a low-pass filtering process, which filters out lots of high-passband quantization noise at the receiver if down

Uniform Quant



(a)

JPEG



Figure 3-11 Reconstructed images by down scaling at different locations. The compression scheme includes DCT and quantization. (a) uses a uniform quantization step, 36, for all AC DCT coefficients (b) uses the quantization table listed in the JPEG standard.

original	user site
source site	intermediate site

scaling is done after quantization. Therefore, from the perspective of received video quality, it seems doing down scaling at the user site is more preferable. However, one strong counter argument to this statement is that using user-site down scaling actually needs to send twice as many samples as that of the source-site compositing approach. If the image has been scaled down at the source site, we can change to use a finer quantizer (i.e. more bits per sample) to raise the received image quality. This will make a fairer comparison since the distortion level will be compared at the same transmission rate. However, in many cases, fixed quantizers are used in compression hardware. Under that circumstance, the advantage of higher video quality by performing compositing operations at suitable locations is still worthy of pursuit

3.5.2 Synchronization and Latency

Each intermediate compositing unit may introduce additional latency, such as extra buffer delay. On the other hand, the distributed compositing approach may alleviate some practical difficulty in multi-source synchronization, which is mandatory on networks with delay jitters such as the ATM cell-based networks. The delay variation will be less severe if the compositing hardware is located closer to the sources. Also, synchronization among multiple sources should be easier when there are fewer input sources. We will discuss the impact of distributed compositing on synchronization and latency in this section.

Figure 3-12 shows the general architecture for an intermediate-site compositing unit. It basically includes buffers for input source data, the actual compositing processor, and the transmitter which perform data packetization and transmission. The overall additional delay, D , can be broken down as follows:

$$D = t_{fill} + t_{comp} + t_{tx} + t_{jitter} , \quad (3-9)$$

where t_{fill} is the delay time for filling a video frame memory, t_{comp} is the computation time for completing the compositing function, t_{tx} is the delay time for packetization and trans-

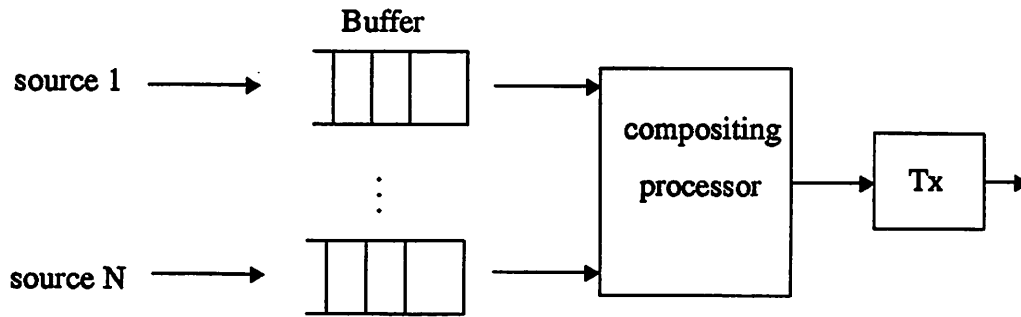


Figure 3-12 A general architecture for intermediate-site compositing unit, which basically includes buffers for input signals, the compositing processor, and the transmitter performing packetizer and transmission.

mission, and t_{jitter} is delay variance between multiple input video sources. We include t_{fill} because many compositing functions (such as image flipping) need to be implemented by a frame-by-frame method. In other words, each image frame needs to be stored before being processed. But for some compositing functions (such as linear filtering) which do not need to operate on the whole image frame, this term can be removed. The last term, t_{jitter} , takes account of delay variance of packets from different video sources. Early packets need to wait for late packets of other video sources in order to be composited into a single video signal. On ATM cell-based networks, delay jitter always exists and efficient synchronization schemes among multiple streams are required.

Typically, video signal has a refresh rate of 30 frames per second. Therefore, the first term (t_{fill}) is about 33ms. To achieve real-time compositing, the computation time (t_{comp}) and the transmission time (t_{tx}) both need to be within one frame duration. As for the delay jitter term (t_{jitter}), it greatly depends on the actual network designs, traffic conditions, and of course the location of the compositing unit. Usually, this delay jitter is compensated by some buffer scheme and control time, which controls the starting time of processing the input data. Figure 3-13 shows a graph of accumulated input and output rate and corresponding buffer status. We assume a fixed output rate in this example. When the accumulated input rate drops below the output rate, underflow will occur. When the accumulated input rate exceed the output rate plus the buffer capacity, overflow will

happen. Minimization of underflow probability favors longer control time, but at the cost of larger buffer size in order to minimize the overflow probability. One alternative of synchronization is to process the packet and send it out whenever it is available. This approach will remove the underflow problem, but still need the buffer to accommodate bursty input rate.

In general, the required buffer size and the control time are directly proportional to the delay variance of the input traffic. If the compositing unit has multiple input video sources, then the synchronization scheme need to allocate buffers for each input video source. In practice, buffers for different input source can be put on different small modules or combined onto a bigger module, as long as their logical separation is kept. The required buffer size depends on not only the delay variance of each individual input stream, but also the delay variance between different input streams. In Figure 3-14, we show the accumulated input and output rate of a fixed-output-rate synchronization scheme.

The approach of distributed compositing combines video sources in a hierarchical way. Thus, at each intermediate compositing node, there are less input video sources than that in the centralized approach. Figure 3-15 illustrates approaches of compositing four

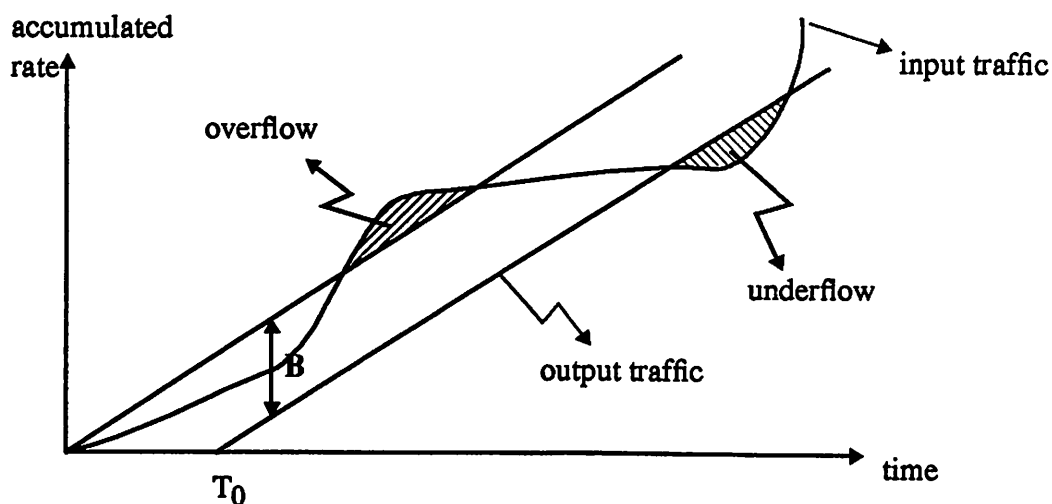


Figure 3-13 A fixed output rate synchronization scheme to compensate the bursty input traffic at the compositing unit. B: Buffer size, T_0 : control time.

video objects at different locations — user, intermediate node, and source. The user-site approach sends all video sources to the destination and does compositing there once. Each video source requires a buffer size B_1 . The intermediate-site approach composites video hierarchically and requires two stages of synchronization. The requires buffer size is $2 \cdot B_2 + 2 \cdot B_3 + B_4$. The source-site approach composites video before they are sent out. This will be feasible only when the video sources are located close to each other. Each input video source still needs a buffer, B_5 . At the receiver, a single buffer, B_6 , is needed to compensate the delay variance introduced in the network. Actually, if we allow the compositing unit to move away from the source site, this architecture can be extended to model the centralized compositing approach widely used in today's centralized multi-point conferencing. The centralized compositing unit broadcasts the composited result back to each participant.

If we assume the delay variance is approximately proportional to the transmission distance and the number of sources being composited together, then the following relations should hold: $\{B_2, B_3, B_4\} < B_1$, $B_5 < B_1$, $B_6 < B_1$. Therefore, the user-site

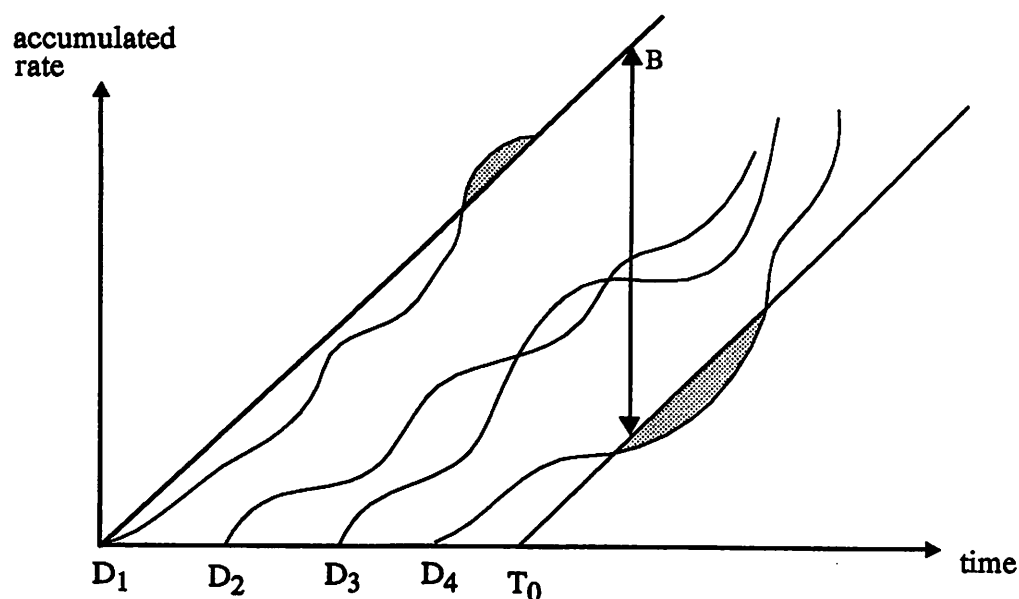


Figure 3-14 A fixed-output-rate synchronization scheme for multiple variable-rate input streams.

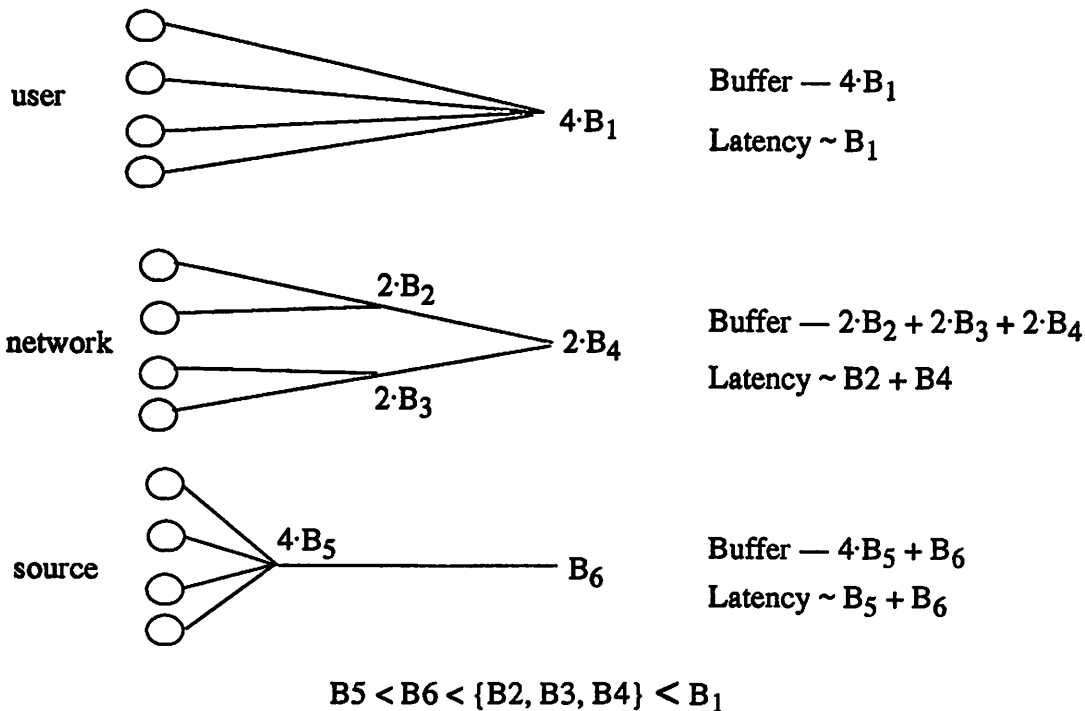


Figure 3-15 Compositing at different locations and their buffer requirement.

approach will require the most buffer resources and the source-site approach the least. The intermediate-site distributed approach will require a moderate amount of buffer resources, depending on the actual network routing scheme. But if in fact each link in the intermediate-site approach is equivalent to the links in the user-site approach, then the intermediate-site approach may turn out to perform worse in terms of the total buffer requirement and the latency. Therefore, in order to take utilize the maximal benefits of distributed compositing approach, efficient routing schemes need to be employed. It is worthy of reminding that the distributed compositing approach provides the possibility of sharing common compositing tasks among different users and services.

In practice, the maximal tolerable latency is usually application dependent. For example, typical interactive conferencing applications can tolerate a maximal latency in the order of 100-150 ms [Rangan93, Karlsson89], which is only about 3-5 frame durations

for a typical display refreshing rate. Given this tight delay budget, we need to consider the above impacts on latency carefully when designing the distributed approach.

3.5.3 An Example

Let us take the following compositing function as an example to discuss the most suitable compositing location in different situations.

$$scale(A, 1/2) + scale(B, 1/2) + scale(C, 1/4) + scale(D, 1/4) + scale(E, 1/4) \quad (3-10)$$

If communication bandwidth is the resource of most concern, then each video object should be scaled down before transmission; namely, the scaling operation should be done at the source site. If the computational complexity is most critical, then we can apply the distributive property described in section 3.3.2 to restructure the compositing function as follows,

$$scale(A + B, 1/2) + scale(C + D + E, 1/4) \quad (3-11)$$

The computation cost can be reduced if the total data rate is reduced after overlapping.¹ The actual implementations can be centralized or distributed. For example, operation A+B and C+D+E can be implemented in some intermediate nodes so that they can be shared by other users, and the down scaling operations can be left at the user destination site. Doing down scaling at the user site is also beneficial to the received video quality, as described in the last section.

If users need to have strong control over the compositing process, then compositing at the location closest to the user is more preferable. For example, users may request special compositing constraints such as video frame rate and resolution. The control function will be most responsive and flexible if the compositing unit is at the user

1. This is not necessarily true if the input video objects are combined without any overlapping, such as *A abut B*.

site. However, a remote compositing unit at some intermediate node can still satisfy the same control request at the cost of some latency.

3.6 Summary

This chapter investigates the approach of *shared distributed network compositing*. By distributively allocating the compositing hardware throughout the network, we can flexibly meet various demands of different users and services. Different network multimedia applications have different characteristics and thus may prefer different compositing locations. By sharing the identical hardwares for implementing the common compositing functions among different users and services, we can reduce the overall implementation cost.

In order to flexibly adapt video compositing to various user and service needs, and find beneficial sharable compositing tasks among different users and services, we study efficient restructuring of compositing functions. In particular, we consider associative, commutative, and distributive properties of expressions, and their corresponding operations on trees. We also consider extraction of common sub-lists from different ordered lists.

We consider the ATM network as the basic platform for distributed compositing because of ATM's efficiency of flexible resource allocation, service integration, and multi-point connections. We study advantages and disadvantages of different compositing locations within the network. Concerned performance factors include quality, complexity (computation and bandwidth), synchronization, and latency. As shown in the example of the previous subsection, the most suitable compositing location for each specific application strongly depends on its most concerned performance metrics.

Chapter 4

Video Compositing in the Compressed Domain

In the previous chapter, we discussed the issue of where compositing should be done. In this chapter, we turn to another important issue — the *data format* of the video signal. Due to its information-intensive nature, video is usually transmitted in compressed form over networks, and stored in compressed form in video databases. Given compressed video signals, we have two possible approaches to compositing video. First, we can decompress the video signal and perform compositing in the uncompressed domain, i.e. with the original raw video data format. Second, we can derive equivalent compositing algorithms in the compressed domain and directly composite compressed video in the compressed domain [Chang 92a, Chang92b, Chang93]. In this chapter, we will show that many typical compositing functions can be performed in the compressed domain more efficiently (with less computation) than in the uncompressed domain. The specific compression format we study is the widely used Motion Compensated Discrete Cosine Transform (MC-DCT) format. The above statement is particularly true if video is composited in the network, where both input and output video signals are compressed.

Performing video compositing in the compressed domain is new. There is some independent related work. Smith and Rowe studied some compositing algorithms in the DCT domain only [Smith93]. They only covered some subset of operations like linear combination and pixel multiplications. Also, they used a brute-force approach without deriving underlying mathematical formulae. But their numerical simulation results showed advantages for compressed-domain compositing similar to those revealed by our work. Lee and Lee derived the transform-domain one dimensional filtering algorithms and proposed a suitable pipelined hardware architecture for implementation [LeeJ92]. Their derivation is a little different from ours, but the result for linear filtering is the same. The compressed-domain compositing approach can also be extended to other compression algorithms, such as subband coding. Lee and Woods proposed some subband-domain algorithms for simple operations such as picture in picture and text overlay [LeeY92],

although their algorithms introduce some artifacts such as border boxes along the overlap boundary to alleviate a ringing effect.

We will derive transform-domain algorithms for representative compositing functions described in chapter 2. Our derivations are based on the linearity of these compositing functions and the orthogonality of the transform coding algorithms. To extend the proposed techniques to hybrid non-linear compression algorithms such as Motion Compensation plus Transform coding, we propose a new approach to partially decoding the compressed video to the transform domain and, then apply our compositing techniques in the transform domain. The performance is studied both analytically and numerically.

4.1 Background

4.1.1 Review of Motion-Compensated DCT-Based Compression Algorithms

Many compression standards use Motion Compensation [Netravali79] and Discrete Cosine Transform [Ahmed74]. These standards include JPEG for still images [Wallace91], MPEG for multimedia video [Le Gal91], H.261 for video conferencing [Liu91], and some HDTV proposals [Jurgen91]. In this section we briefly review the operations of the DCT algorithm and the MC algorithm. Because of their popularity, we will limit our study of compressed-domain approaches in these compression algorithms, although most of our proposed techniques can be extended to other orthogonal transform coding algorithms.

Figure 4-1 shows a block diagram for intraframe DCT in combination with variable length coding (VLC).

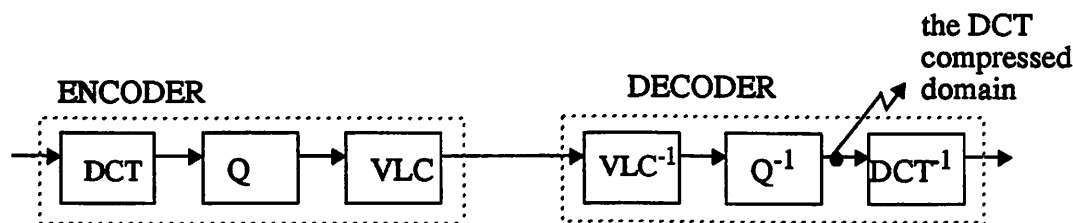


FIGURE 4-1 Video compression methods using the Discrete Cosine Transform algorithm and the variable length code. IDCT: inverse DCT, Q: quantization, VLC: variable length code.

Images are segmented into small blocks of N pixels by N pixels, which can be represented by an $N \times N$ matrix. The DCT A_c of the image block A can be computed as follows

$$A_c = C A C^T, \text{ where } C(i, j) = \begin{cases} \frac{1}{\sqrt{N}} & i = 0 \\ \sqrt{\frac{2}{N}} \cos \frac{\pi(2j+1)i}{2N} & i \neq 0 \end{cases} \quad (4-1)$$

where the rows of C form a set of orthonormal basis functions. Its elements, $A_c(i, j)$, represent the spectrum of the original image block, A , at different spatial frequencies. Elements with larger index values represent the higher-frequency components.

Usually, the DCT coefficients, $A_c(i, j)$, are quantized and then run-length coded (RLC) to take advantage of long sequences of zero's. The high-frequency coefficients are considered less important and thus the coefficients are quantized more coarsely. This results in correspondingly longer zero sequences at the high-frequency region. Lastly, the statistically-based variable-length code (VLC), such as the Huffman code or arithmetic code, is applied to exploit any remaining data redundancy.

In Figure 4-2, we show the block diagram for the MC algorithm. The MC algorithm basically includes two steps— displacement measurement (DM) and error calculation. The displacement measurement procedure searches for the *optimal reference block* in the previous frame and is much more computationally intensive than the error calculation. On the feedback path, the previous frame image is reconstructed as the reference frame in the displacement measurement process.

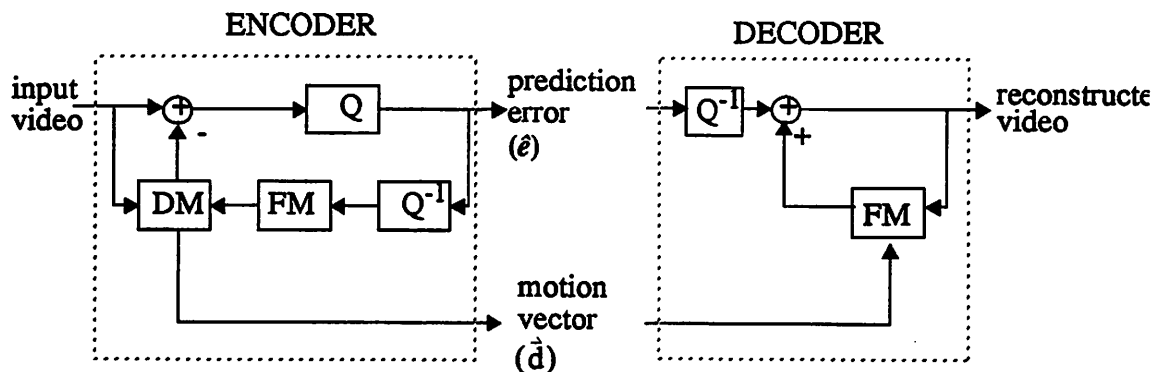
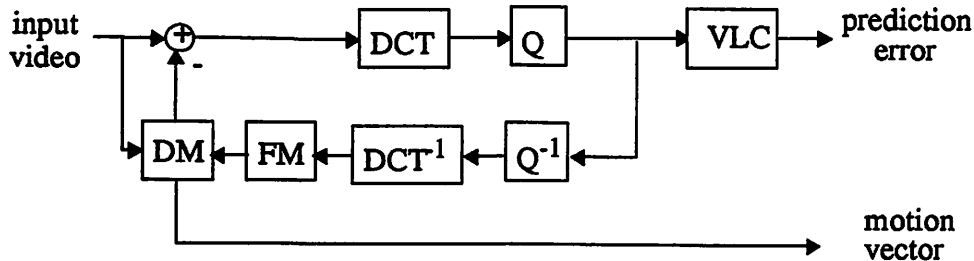


FIGURE 4-2 The Motion Compensation algorithm and its inverse algorithm. DM: the displacement measurement procedure. FM: frame buffer. Q: quantizer. Q^{-1} : inverse quantizer.

In hybrid interframe coding algorithms, the MC algorithm is usually integrated into the DCT-based compression system as shown in Figure 4-3. The MC algorithm is performed first to remove the temporal dependence between frames. The prediction errors are transformed to the DCT coefficients, quantized, and then variable-length encoded.

Encoder:



Decoder:

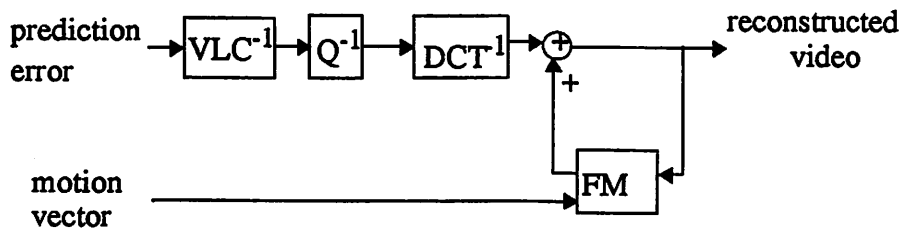


FIGURE 4-3 Block diagram for hybrid compression methods including the MC algorithm and the DCT algorithm. DM: displacement measurement, FM: Frame Memory.

4.1.2 Domain Notations

For clarity, we define the following terms explicitly here. The *uncompressed* domain contains uncompressed images or reconstructed images. If not otherwise specified, we usually assume a simple pixel map data format. The input and reconstructed video in the above compression algorithms are assumed to be in this format. Sometimes, we also refer to this domain as the *spatial* domain, because pixel values are usually defined on spatial coordinates, except for some confusing situations such as subband coding, where spatial-to-spatial compression algorithms are used.

The *DCT-compressed* or *DCT* domain contains the data format after the DCT compression process. The image data is represented by its corresponding DCT coefficients. In addition, as mentioned above, DCT coefficients are usually quantized in

the encoder. The quantizer converts continuous input values into discrete output symbols; the inverse quantizer reconstructs continuous values (usually with a finite resolution) from received symbols. When we talk about the DCT compressed domain in the following, we refer to the reconstructed DCT coefficients calculated from the inverse quantizer, such as the data format at point A in Figure 4-1.

Note that a video signal can be transformed into the DCT domain either by applying the DCT or by partially decoding the interframe MC-DCT compressed video, as described later. In the following, we will also use the term “*transform* domain” to describe the frequency domain for the general orthogonal transform, such as the Discrete Sine Transform, the Discrete Fourier Transform, and the DCT.

The *MC-compressed* or *MC* domain contains the compressed data format after the MC algorithm. Basic components include the motion vectors (\vec{d}) and the prediction errors (\hat{e}), as shown in Figure 4-2. The *MC-DCT-compressed* or *MC-DCT* domain contains the video signals after the interframe coding algorithms shown in Figure 4-3. It can be considered as the MC-compressed data of the DCT coefficients.

4.2 The Conventional Uncompressed-Domain Approach

In networked or stored video applications, input video is usually compressed. The output video can be compressed or uncompressed, depending on the functionality of the output device. For example, video compositing devices within the network have both input and output video compressed, while a user-site workstation needs to produce uncompressed displayable video. In either case, input compressed video is always converted back to the uncompressed domain before being composited together in today's applications [Lukacs92, Rangan93]. A decoding process is required for each input video signal before they are composited together, as shown in Figure 4-4(a). The advantage of this approach is that uncompressed-domain compositing algorithms are available and well known. However, the overhead of decompression of each video input may become quite

high for advanced high-resolution video services. Also, after decompression, the compositing unit needs to run at the pixel rate, which may be too high to afford.

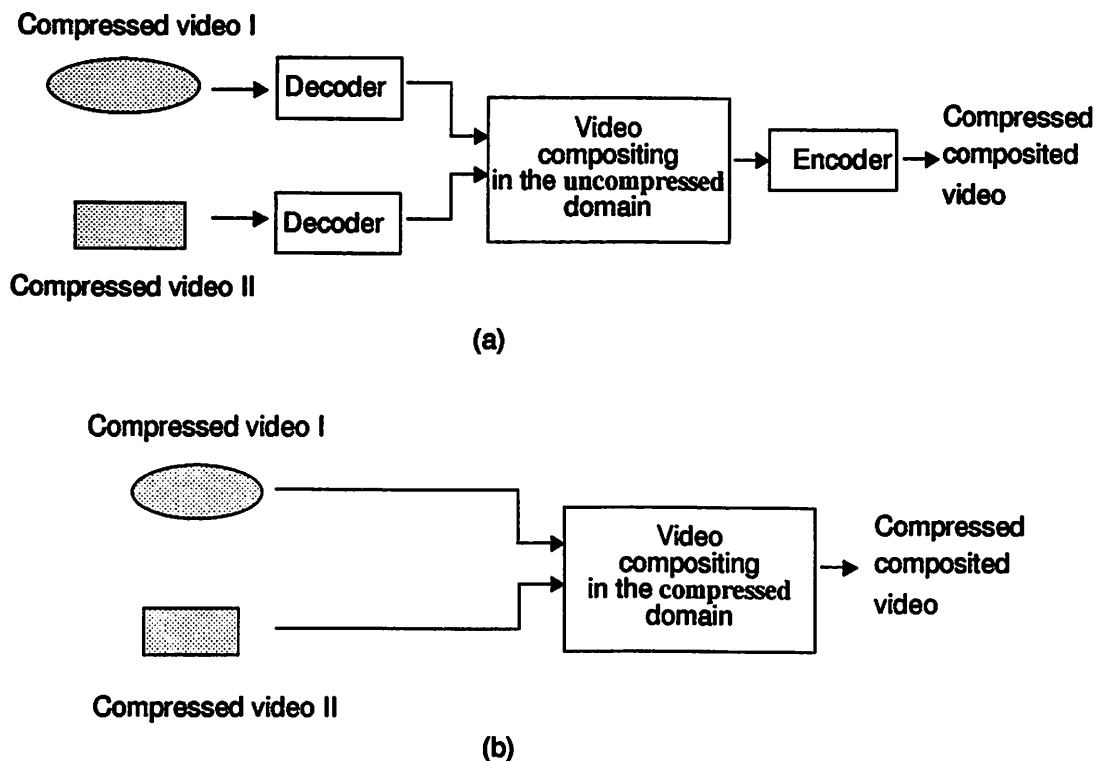


FIGURE 4-4 Two different approaches to compressed-input compressed-output video compositing. (a) uncompressed-domain approach (b) compressed-domain approach.

4.3 A New Approach — Compressed-Domain Compositing

As shown in Figure 4-4(b), for compressed-input compressed-output video compositing, a very natural alternative is to directly composite video in the compressed domain and produce compressed composited output video. The conversion back and forth between the compressed format and the uncompressed format is avoided. Further, the amount of data in the compressed video is usually much less than that of the uncompressed video. Thus, the associated computations can be greatly reduced.

Another advantage for compositing in the compressed domain is *scalability*. For example, a single compressed video stream may be used by many users with different

levels of compositing processing power. Users with lower compositing power can process signal components with the highest significance in the compressed domain only (e.g. lower-order DCT coefficients, or lower bands in subband coding) and get as high a video quality as possible within the limit of the hardware capability. This prioritized rank of signal significance is not available in the uncompressed domain.¹ Further, in real-time constrained implementations, this selective processing principle can be applied to throw away insignificant signal components in the worst-case source behavior (such as the peak-rate period of variable-bit-rate video sources) or network behavior.

For compressed-input uncompressed-output situations such as the user-site workstations, the compressed-domain compositing approach is still potentially preferable because of the much lower data amount and the scalability in the compressed domain. By using the latter advantage, lower-end hardwares can still get satisfactory real-time video by processing the most important signal component only.

Based on the above discussions, the compressed-domain compositing approach is most appealing to situations like compositing video in networks (e.g. video bridges), compositing compressed video from video databases (e.g. image/video editing), and filtering of networked video by third-party service providers (e.g. cable companies). One common point for all these situations is that the input video is already compressed, which justifies the need for using the compressed-domain approach.

The feasibility of compositing in the compressed domain heavily depends on the compression algorithms in use. We will derive compositing algorithms for typical compositing functions in the DCT domain. These techniques can be applied to general orthogonal transform coding algorithms, such as DFT or DST (Discrete Sine Transform). We will also describe a method to extend these techniques to the MC-DCT domain, which is the base system for many compression standards, such as MPEG, H.261, and several HDTV proposals.

1. Once video is decoded back to the uncompressed domain, it will have a full resolution and will require a full-resolution compositing cost. Of course, subsampling in the uncompressed domain can be used to reduce computations when hardware power is limited. But good subsampling would require extra computations (such as anti-aliasing) compared to importance screening directly in the compressed domain.

4.4 Video Compositing in the DCT Domain

Given the DCT coefficients of two input images, Figure 4-5 shows how to composite them in the DCT domain. The block alignment procedure may sometimes be necessary because the block structures¹ for different images may not generally match unless some restrictions on translations are enforced. Even if all video source encoders use the same block structures, receivers can move images around arbitrarily and thus change their relative positions. Alignment in the spatial domain is trivial since all relative spatial positions are known. In the next subsection, we will propose a method to align blocks in the DCT domain.²

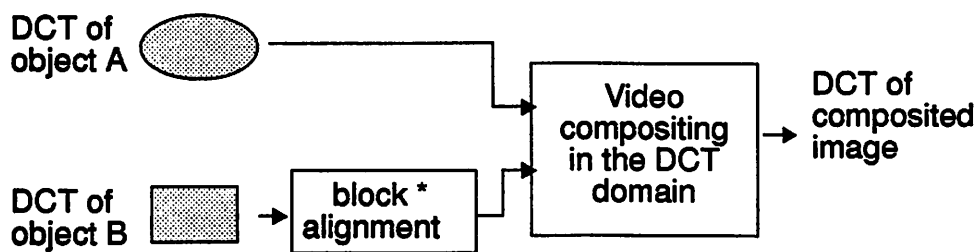


FIGURE 4-5 Compositing two video sequences in the DCT domain. (*: this procedure is not necessary if the block structures of two input images are matched.)

We describe one set of basic image manipulation primitives — overlap, scaling, translation, linear filtering, and pixel-multiplication, and their DCT-domain equivalents in this section. Most video services require only a subset of these. Note, these algorithms can be applied to other transform domains, like DFT and DST.

1. By *block structure*, we mean the grid lines used to segment the images into small rectangular blocks. For example, in Figure 4-6, the block structures of images A and B are not matched.

2. Note that in Figure 4-5, we assume that composited output is in the compressed format. If uncompressed images are needed, an inverse DCT block is required at the end.

4.4.1 Overlap

Opaque overlapping of two video objects requires substituting pixels of the foreground video object for those of the background object. *Semi-transparent overlapping* requires a linear combination of the foreground and background pixels, i.e.,

$$P_{\text{new}}(i,j) = \alpha \cdot P_a(i,j) + (1-\alpha) \cdot P_b(i,j) \quad (4-2)$$

where P_{new} , P_a , and P_b are new pixels, foreground pixels and background pixels [Porter84]. Since it's a linear operation, we can apply the same technique in the DCT domain, i.e.,

$$\text{DCT}(\bar{P}_{\text{new}}) = \alpha \cdot \text{DCT}(\bar{P}_a) + (1 - \alpha) \cdot \text{DCT}(\bar{P}_b) \quad (4-3)$$

where \bar{P} is used to represent the block-wise operation of the DCT.

4.4.2 Pixel Multiplication

If the α coefficients vary from pixel to pixel, semi-transparent overlapping becomes a pixel-wise operation, i.e.,

$$P_{\text{new}}(i,j) = \alpha(i,j) \cdot P_a(i,j) + (1-\alpha(i,j)) \cdot P_b(i,j) \quad (4-4)$$

This operation is similar to another useful compositing operation called **pixel-multiplication**, i.e.,

$$P_{\text{new}}(i,j) = P_a(i,j) \cdot P_b(i,j) \quad (4-5)$$

Pixel-multiplication is required in situations like *subtitling* (adding text on top of an image), *anti-aliasing* (for removing the jagged artifacts along the boundaries of irregularly-shaped video objects), and special-effect *masking* (with special graphic patterns). To compute the pixel-wise multiplication in the DCT domain, we derive a multiplication-convolution relationship for the DCT similar to that for the Discrete Fourier Transform (DFT), except that the order for the convolution is increased to $2 \cdot N$ points. We leave the proof in the appendix and present the final result here. Suppose X_c is the DCT of image block X . First, we form an extended symmetrical version of the DCT coefficients as the following

$$\hat{X}_c(k_1, k_2) = \begin{cases} X_c(k_1, k_2) / C(k_1, k_2) & k_1, k_2 = 0, \dots, N-1 \\ X_c(-k_1, k_2) / C(-k_1, k_2) & k_1 = -N+1, \dots, -1, k_2 = 0, \dots, N-1 \\ X_c(k_1, -k_2) / C(k_1, -k_2) & k_1 = 0, \dots, N-1, k_2 = -N+1, \dots, -1 \\ X_c(-k_1, -k_2) / C(-k_1, -k_2) & k_1, k_2 = -N+1, \dots, -1 \\ 0 & k_1 = N, \text{ or } k_2 = N \end{cases}$$

$$\text{where } C(k_1, k_2) = \begin{cases} \frac{1}{2} & k_1 = k_2 = 0 \\ 1 & \text{otherwise} \end{cases}$$

(4-6)

Then, the *multiplication-convolution theorem* can be described as below.

If

$$y(i, j) = x(i, j) \cdot h(i, j), \quad i, j = 0, \dots, N-1$$

then

$$\hat{Y}_c(k_1, k_2) = \frac{1}{2N} \cdot \sum_{l_1, l_2 \in [-N, N-1]} [\hat{X}_c(l_1, l_2) \cdot \hat{H}_c(((k_1 - l_1))_{2N}, ((k_2 - l_2))_{2N})] \times \alpha(k_1 - l_1) \alpha(k_2 - l_2)$$

$$k_1, k_2 = -N, \dots, N-1$$

$$\alpha(k) = \begin{cases} 1 & k \in [-N, N-1] \\ -1 & \text{otherwise} \end{cases}$$

(4-7)

For convenience we use the notation “ $((n))_{2N}$ ” to denote $(n \text{ modulo } 2N)$. If we ignore the α variables, the above equation is equivalent to a 2D $2N$ -point circular convolution. In other words, we expand the $N \times N$ DCT coefficients of an image block to a symmetric $2N \times 2N$ extended block. The pixel-wise multiplication of two image blocks in the spatial domain

corresponds to the 2-dimensional $2N$ -point circular convolution-like operation of the extended blocks in the DCT domain.

4.4.3 Translation

As mentioned earlier, each input video object is associated with a particular block structure for the block-wise transform coding like DCT. All the above compositing operations assume that the block structures of the input video objects are aligned. Therefore, the indices of the DCT coefficients are matched. However, this assumption is not valid when input video objects are allowed to move to arbitrary positions. In applications like video conferencing, users may wish to control the position of each video window on the screen. We describe the general translation operation in this section.

Two types of translation should be discussed separately— *block-wise* and *pixel-wise* (i.e. arbitrary-position) translation. If we restrict both the horizontal and vertical translation distance to be an integral multiple of the block width, the DCT coefficients are always aligned with the same block structure. Moving a video object just updates the position of the origin point of the video object. Compositing operations like overlapping and pixel multiplication can be performed in the DCT domain as described earlier with a fixed block structure.

However, if we allow translation by an arbitrary number of pixels, the block structure of the input video objects could be mismatched. Figure 4-6 illustrates the mismatch of block structures of objects A and B, which could be caused by moving one of these two objects arbitrarily. In the spatial domain, this mismatch problem is trivial. We decompress two video objects by using the inverse DCT (with respect to their individual block structures), calculate the composited image in the spatial domain, and then re-compress the composited by using the DCT if the DCT-compressed-format output is required. The final block structure can be chosen from the block structure of any input video object or the block structure for the background image of the final composited scene. As shown in Figure 4-6, suppose we want to composite object A and object B, and we choose the block structure of object A as the final block structure. If we re-align object B with respect to the block structure of object A, a new image block (say B') of object B contains contributions from four original neighboring blocks (B_1 - B_4), namely the lower-

left corner (B_{13}) of block B_1 , the lower-right corner (B_{24}) of block B_2 , the upper-right corner (B_{31}) of block B_3 , and the upper-left corner (B_{42}) of block B_4 . If we supplement these four contributions with zero's in a way illustrated in Figure 4-6 to form N pixels by N pixels image blocks, the new block can be calculated by the following summation,

$$B' = B_{13} + B_{24} + B_{31} + B_{42} \quad (4-8)$$

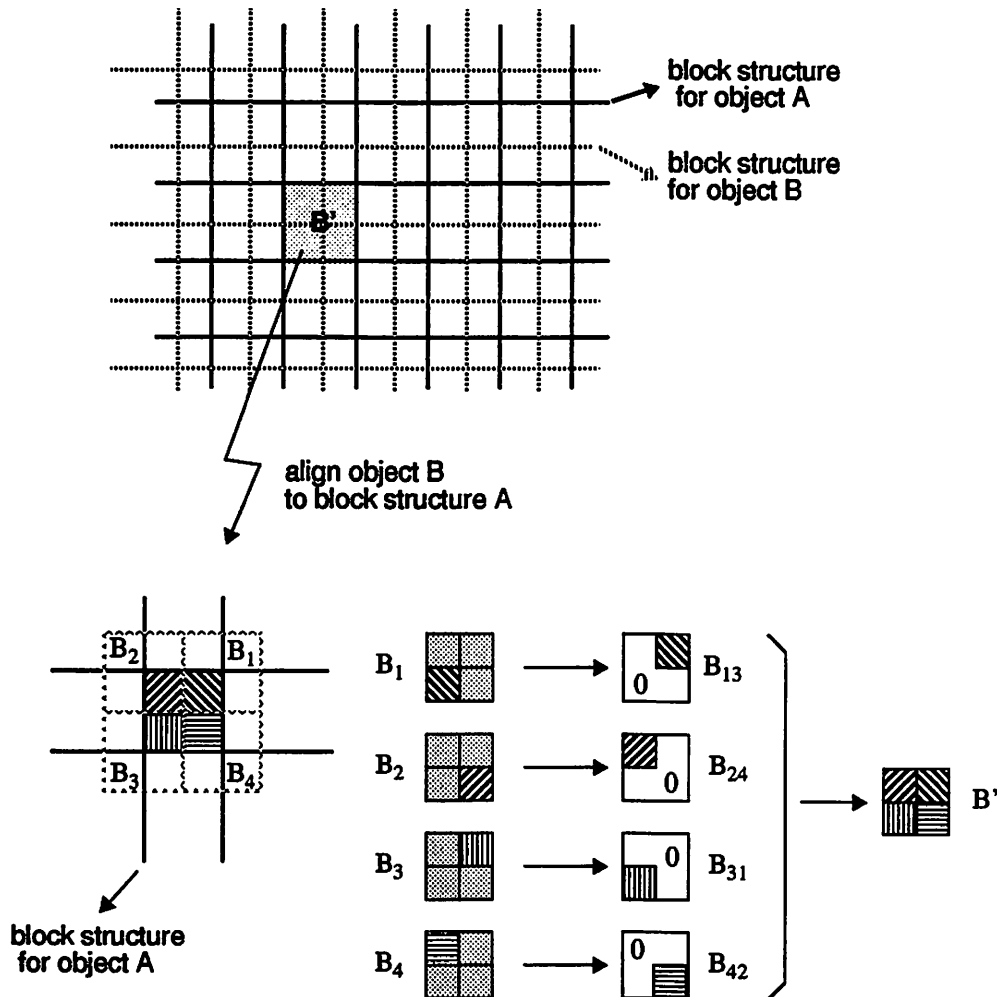


FIGURE 4-6 Re-assembling the image blocks of video object B with respect to a new block structure, which mismatches object B's original block structure. The highlighted block, B' , is a new block of object B after block re-assembly. It consists of the contributions (B_{13} , B_{24} , B_{31} , and B_{42}) from four original neighboring blocks (B_1 - B_4).

This method cannot be directly applied in the DCT domain. We cannot simply assembly four subblocks from the DCT coefficients of the original neighboring blocks to form the DCT coefficients of the new block (B'). Instead, the correct DCT coefficients of the new block should be calculated as follows,

$$\text{DCT}(B') = \text{DCT}(B_{13}) + \text{DCT}(B_{24}) + \text{DCT}(B_{31}) + \text{DCT}(B_{42}) \quad (4-9)$$

Therefore, if we can find the relationship between the DCT coefficients of subblocks ($B_{13} - B_{42}$) and the DCT coefficients of the original blocks ($B_1 - B_4$), then we can calculate the DCT coefficients of the new block directly from the DCT coefficients of the original blocks. In other words, the conversion processes back and forth between the DCT-compressed domain and the spatial domain can be eliminated.

Figure 4-7 shows a mathematical model for calculating the contribution from an original block (e.g. B_{42} from B_4) in the spatial domain. Note that the upper-left corner of B_4 is extracted, supplemented by zeros, and moved to the lower-right corner. This process is necessary since the upper-left subblock of the original block B_4 will appear as the lower-right subblock of the new block B' . As shown in Figure 4-7, we can use the following operation to calculate the contribution from the original block B_4 :

$$B_{42} = H_1 B_4 H_2, \text{ where } H_1 = \begin{bmatrix} 0 & 0 \\ I_h & 0 \end{bmatrix} \quad H_2 = \begin{bmatrix} 0 & I_w \\ 0 & 0 \end{bmatrix} \quad (4-10)$$

I_h and I_w are identity matrices with size $h \times h$ and $w \times w$ respectively, where h and w are the number of rows and columns extracted from block B_1 . As shown in Figure 4-7, multiplying B_1 with a pre-matrix H_1 extracts the first h rows and translates them to the bottom;

multiplying B_1 with a post-matrix H_2 extracts the first w columns and translates them to the right.

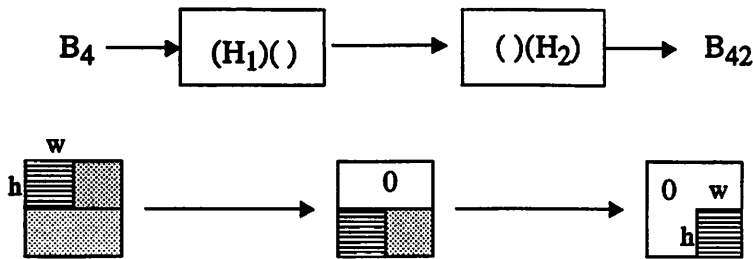


FIGURE 4-7 Using matrix multiplication to extract a subblock and translate it to the opposite corner.

It can be shown that all unitary orthogonal transforms such as the DCT are distributive to matrix multiplication, i.e.,

$$\text{DCT}(AB) = \text{DCT}(A)\text{DCT}(B) \quad (4-11)$$

Thus, we can compute the DCT of B_{42} directly from the DCT of B_4 , i.e.,

$$\text{DCT}(B_{42}) = \text{DCT}(H_1)\text{DCT}(B_4)\text{DCT}(H_2) \quad (4-12)$$

Summing all contributions from four corners, we can obtain the DCT coefficients of the new block B' directly from the DCT of old blocks $B_1 - B_4$. In other words,

$$\text{DCT}(B') = \sum_{i=1}^4 \text{DCT}(H_{i1}) \text{DCT}(B_i) \text{DCT}(H_{i2}) \quad (4-13)$$

The DCT of H_{i1} and H_{i2} can be pre-computed and stored in memory, since there are a finite number of them. The required computation is reduced since many elements of $\text{DCT}(B_i)$ are typically zeros. We will discuss the computational complexity in section 4.6.3.1.

4.4.4 Linear Filtering

Two-dimensional separable linear filtering can also be done in the DCT domain [Chang92a, Lee92, Chiprasert90, Ngan80]. Linear filtering of images in the horizontal direction can be achieved by multiplications with post-matrices:

$$Y = \sum_i X_i H_i \quad (4-14)$$

where X_i is the input image block, H_i is the filter coefficients represented in the block form, and Y is the output image block. Each output image block have contributions from several input blocks. The number of contributing input blocks depends on the length of the filter kernel. Since the DCT algorithm is distributive to matrix multiplication, we can calculate $DCT(Y)$ in the following way

$$DCT(Y) = \sum_i DCT(X_i) DCT(H_i) \quad (4-15)$$

Similarly, image filtering in the vertical direction can be achieved by multiplication with pre-matrices. Detailed derivation of the transform-domain filtering algorithm can be found in the Appendix.

4.4.5 Scaling

Another important image manipulation technique is **scaling**. Each pixel in the final scaled image is a linear combination of several neighboring pixels in the original image [Foley90]. Thus, it can be treated in a way similar to linear filtering. For example, if we use the simple box area averaging method to implement the $1/2 \times 1/2$ down scaling operation [Weiman80], a new block can be computed as $H_1 B_{11} W_1 + H_2 B_{21} W_1 + H_1 B_{12} W_2 + H_2 B_{22} W_2$, where B_i are the original neighboring blocks, H_i are the vertical scaling matrices, and W_i are the horizontal scaling matrices. For example,

$$H_1 = (W_1)^T = \begin{bmatrix} 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (4-16)$$

$$\mathbf{H}_2 = (\mathbf{W}_2)^T = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 \end{bmatrix} \quad (4-17)$$

for a block width of 4 pixels. The same linear operations can be performed in the DCT domain as described above.

4.5 Compositing MC-Compressed Video

As shown in Figure 4-2, the video signal in the MC-compressed domain basically is composed of two components: motion vector (d), and prediction error (e), denoted as the *MC data*. Given the MC data of two input video streams, it would be desirable to compute the new MC data of the composited video directly without converting the video back to the uncompressed format. Unfortunately, the MC compression does not have the same linear and orthogonal property as that of the above transform coding algorithms. In most compositing situations such as overlapping and scaling, compositing in the MC-compressed domain is not feasible. In this section, we illustrate this difficulty by using two examples: overlapping and scaling. Given the MC data of the input video streams, we need to convert them back to the uncompressed domain and re-compress the composited video into the MC domain (i.e., *MC data recalculation*) if the compressed-format composited output is required. In order to prevent that the *MC data recalculation* process becomes the most computationally-dominant process, we propose the *inference principle* to calculate the new motion vector of the composited video stream.

4.5.1 Difficulties for MC-Compressed-Domain Compositing

In this section, we explain the reason why the MC-compressed video needs to be composited in the uncompressed domain. Figure 4-8 shows an **overlapping** example. We assume both the foreground and background objects are compressed in the MC format and their block structures are aligned. The MC data of the foreground object can be used for the composited video since it is not affected by the background object. However, part of

the background object is obscured by the foreground object. We need to separate the background object into two different areas: the *directly affected area* and the *indirectly affected area*. The difference comes from the prediction image block from the previous frame used in the MC algorithm. In the directly affected area, part of the *motion area*¹ of each background image block is replaced by the foreground object. If the motion vector is from the foreground area, the prediction image block is destroyed. Thus, the MC data becomes invalid and needs to be recalculated. In the indirectly-affected area, the motion area of each image block is not overlapped by the foreground object. However, since the image pixels in the motion area in the previous frame could be modified through *error propagation* (we will elaborate on this in Section 4.5.2.3), the MC data may also become invalid. In other words, the MC data of the composited video cannot be directly obtained from the original MC data of two input video streams. Both the foreground and background video needs to be reconstructed in the uncompressed domain so that the uncompressed-domain MC algorithm can be reapplied to calculate the new MC data for the composited video.

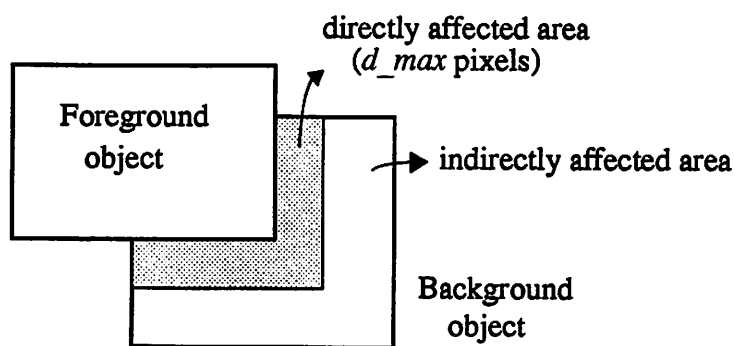


FIGURE 4-8 An example showing the problem for compositing directly in the MC domain. Part of the motion area of the background image in the directly-affected area is replaced by the foreground image. Pixels in the indirectly affected area are changed because their reference pixels may be modified through error propagation. d_max is the number of search positions in each direction in the MC algorithm.

Another example is shown in Figure 4-9. Suppose we want to scale down an image by a ratio of 2 to 1 on each side. Each new image block is transformed from four original image blocks. The motion vectors of these four original blocks could be different from one

1. The motion area is the area in the previous frame where we search for the optimal reference block. For most blocks, it forms a square region of width $(2 \cdot d_max + 1)$ pixels, centered at the current block location.

another and their reference blocks generally will not be transformed into the same image block. Although the new motion vector can be inferred from the original four motion vectors, uncompressed images still need to be reconstructed so that the prediction errors can be recalculated.

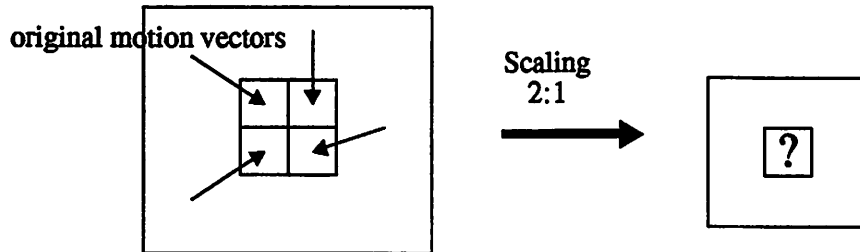


FIGURE 4-9 An example showing the problem for compositing in the MC domain. New prediction errors cannot be calculated in the MC domain since four original reference blocks could be transformed to different blocks in the new image, which is scaled down ($1/2 \times 1/2$) from the original image.

4.5.2 Simplification of MC Data Recalculation

As discussed in the last section, all MC-compressed video needs to be decoded first so that the new MC data of the composited video output can be recalculated by using the MC algorithm. But the computation cost for a full-search MC algorithm could become the dominant part in the compositing process, compared to other operations such as MC decoding and compositing operations. The displacement measurement procedure in the MC algorithm searches for the optimal reference block in the previous frame and is quite computation-intensive. Assuming we use the block-based MC algorithm and the full block-matching displacement measurement procedure, each image block of the current frame requires $(2 \cdot d_{max} + 1)^2$ evaluations of the block distortion function, where d_{max} is the maximum displacement allowed in each direction. Using the simplified displacement measurement method proposed by Jain and Jain [Jain81], the computational complexity can be reduced to the logarithmic order. In table 4-1, we list the numbers indicating how many times we need to evaluate the block distortion function for different MC parameters. However, if we need to recalculate the MC data for every block in the

background object as described above, the incurred computational overhead for the whole frame is still very significant.

Table 4-1 Complexity comparison for different displacement measurement methods. Numbers shown are the total times for evaluating the block distortion function.

d_max	4	5	6	7	8	9	10	11	12	13	14	15	16
full search	81	121	169	225	289	361	441	529	625	729	841	961	1089
Jain & Jain ^a	13	13	13	13	17	17	17	17	17	17	17	17	21

a. Jain and Jain's algorithm needs to search at least $(\log_2 d_{\max}) \cdot 4 + 5$ locations.

Thus, it is our goal to simplify the MC recalculation process in order to keep low-cost compositing feasible. The general principles are first to reduce the frequency of MC recalculation, and second, if MC recalculation is needed, to simplify the complicated displacement measurement process by inferring the motion vectors from the original input motion vectors. In the following subsections, we will use the overlapping situation shown in Figure 4-8 as an example to verify the effectiveness of these two principles. This *inference principle* can be applied to other compositing situations. For example, applying the same approach to the down-scaling case shown in Figure 4-9, new motion vectors can be inferred from old motion vectors by interpolation.

4.5.2.1 Reducing the Frequency of Recalculation

The first principle for simplifying the MC recalculation process is to reduce the number of image blocks which require the recalculation of the MC data. For example, in the opaque overlapping situation shown in Figure 4-8, the foreground object is totally unaffected and its MC data does not need to be modified. For background image blocks in the *indirectly-affected area*, we can assume the old motion vectors are still valid and only the prediction errors need to be updated. For image blocks in the *directly-affected area*, we need to check their motion vectors. If the motion vector comes from the foreground area, that means the original optimal reference block is destroyed and thus a new motion vector needs to be calculated.

We use a video test sequence (the ping-pong sequence) to simulate the MC recalculation. We assume that a 360 pixel x 288 pixel area out of the whole 720 pixel x

480 pixel area is covered by a foreground object (the Miss America image). An example composited picture is shown in Figure 4-10. Block size is 8 pixels by 8 pixels and the



FIGURE 4-10 The test video sequence for the MC recalculation algorithms. The background and foreground images are originally MC-compressed. The composited image is also MC-compressed by using the proposed 2-point simplified searching algorithm. The displayed image is reconstructed from the MC-compressed composited output. This is frame 3 in Figure 4-14 shown later, which suffers the most severe SNR loss due to this simplified search algorithm.

maximum displacement (d_{max}) is 8 pixels. The block distortion function is the mean square error. For each new frame, we check the image blocks in the directly-affected area of the background object that if their MC data needs to be recalculated. If so, we use the full block-matching method to find a new optimal reference block. To further reduce the computational complexity, we can assume that there is no strong interrelation between the foreground and background image contents and thus the search area can be reduced to the uncovered background motion area only.

Figure 4-11 shows the number of blocks which require the MC data recalculations. There are 82 blocks contained in the affected area. About 5-15% of these blocks require recalculation of their MC data. Therefore, compared to using the straightforward MC recalculation for every block in the background object, the frequency of MC recalculation is greatly reduced. Of course, this number changes from frame to frame and depends on

the composition scenario and the specific video sequence we use. If most motion occurs in the overlapped area, then the frequency of recalculation increases.

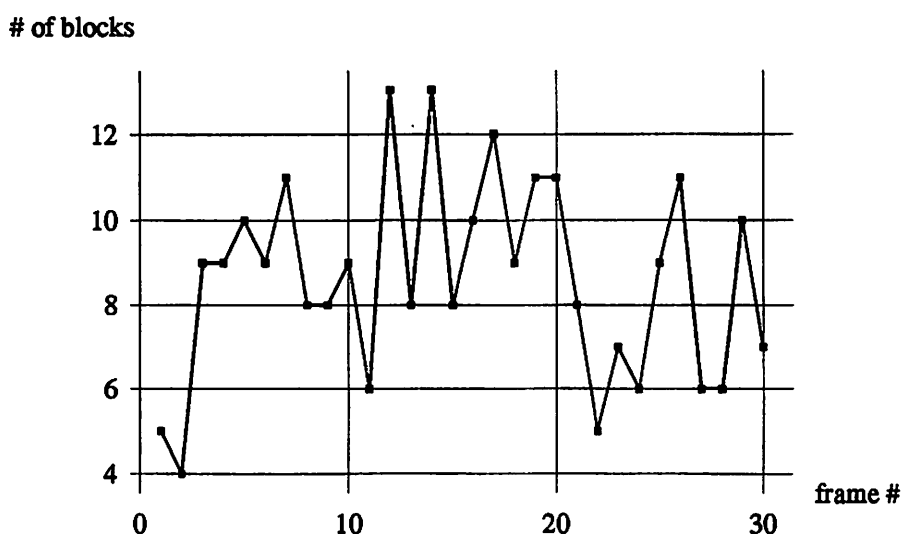


FIGURE 4-11 Number of image blocks in the directly-affected area which requires recalculation of the motion vector. The total number of blocks in the directly-affected area is 82.

In order to re-apply the MC algorithm, we need to convert the MC compressed video object back to the spatial domain. Note that, although we need to recalculate the motion vectors for blocks in the directly-affected area only, we still need to convert the whole uncovered background region to the spatial domain because every pixel in the background object area may be referenced by future pixels in the directly-affected area. We need their spatial uncompressed values because the MC recalculation is performed in the spatial domain.

4.5.2.2 Calculating New Motion Vectors by Inference

To further reduce the computation cost associated with the MC recalculation in the affected area, we will apply the second principle mentioned above — inferring new motion vectors from the old ones. This method will greatly simplify the displacement measurement procedure, which is the most complicated process in the MC algorithm.

To reduce the computational complexity, Jain and Jain's algorithm uses a two-dimensional binary search approach by assuming that the block distortion function is

monotonically increasing along the horizontal or vertical direction when we move away from the position of the optimal reference block. As shown in Figure 4-12, for block B in the current frame, the block distortion value measured at the optimal reference location D is less than those measured at location D_1 and D_2 . Based on Jain and Jain's assumption, the distortion function will increase if we move from D_1 to any location on the right side of D_1 , or if we move from D_2 to any location below D_2 . Suppose now the optimal reference location D is covered by a foreground object, which in general is not related to the background object. Thus, it is reasonable to assume the new optimal reference location will come from the uncovered background region (i.e. the shaded area in Figure 4-12). Based on Jain and Jain's assumption about the distortion function, either D_1 or D_2 will be the new optimal reference location in the uncovered background region. We need to check these two locations only to find the lowest distortion in the uncovered motion area. This is 144.5 times less complicated than the full block-matching procedure when $d_max=8$, which requires searching $(2 \times 8 + 1)^2 = 289$ possible locations.

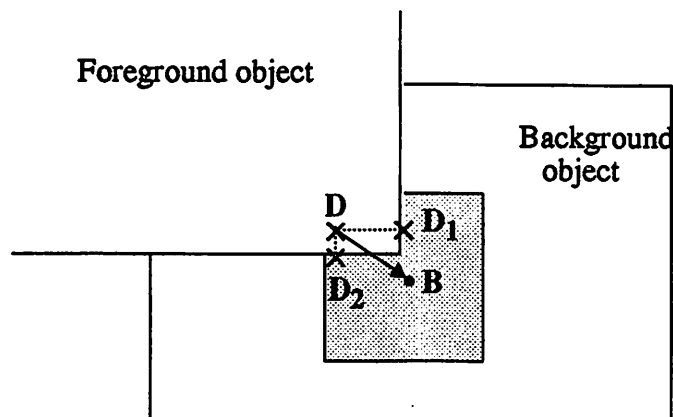


FIGURE 4-12 Reducing the number of searched locations to two by using Jain and Jain's assumption. B is the location of the current image block, D is the optimal reference location, and D_1 (D_2) is the crossing point when we move away from D horizontally (vertically). Based on Jain and Jain's assumption, D_1 and D_2 are the optimal reference locations within the uncovered motion area of B.

4.5.2.3 Error Propagation

The above two methods only handle the pixels inside the directly-affected area of the background image. For background pixels outside, we can reasonably assume their motion vectors are unchanged but the prediction errors may need to be updated because their prediction pixels may come from inside the directly-affected area and need to be

recalculated. This change of prediction values will be propagated to more outside area when the video sequence proceeds.

Suppose block B in frame n is the optimal reference block of block C in frame n+1. If the MC data for block B is recalculated due to overlapping, then quite possibly the reconstructed value for block B will be changed (say from \hat{B} to \tilde{B}). If we do not recalculate the MC data for blocks outside the affected area, the difference between \hat{B} and \tilde{B} will be propagated to those outside blocks which use block B as the reference block. *Error propagation* will continue until re-synchronization. In Figure 4-13, we show the number of pixels affected by this error propagation effect. We can see that the number of affected pixels has an increasing trend as the video sequence proceeds. This is due to the motion towards the lower-right direction in our test sequence. Later we can find the average SNR loss among the affected pixels ranges from 2.38 to 6.19 dB.

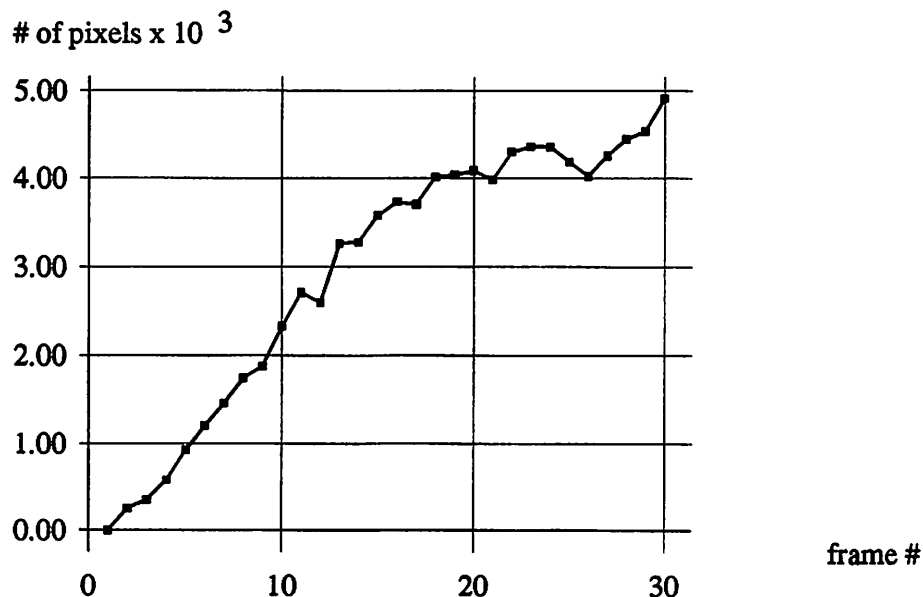


FIGURE 4-13 Number of background pixels outside the directly-affected area affected by error propagation (for the test video sequence shown in Figure 4-10). The number increases because of the motion from the overlap area towards the background object in our test sequence.

We should be able to partially rectify the error propagation problem by at least updating the prediction errors, though the motion vectors are not recalculated. Suppose $b(t)$ is the current picture pixel at time t , $\hat{b}(t-1)$ is the optimal reference pixel at time $t-1$, $\hat{e}(t-1)$ is the quantized prediction error. The reconstructed pixel value, $\hat{b}(t)$ can be calculated as

$$\hat{b}(t) = \hat{b}(t-1) + \hat{e}(t). \quad (4-18)$$

Suppose $\hat{b}(t-1)$ in the above equation is changed to a different value, say $\tilde{b}(t-1)$. To update the prediction errors, we compute the difference between $\hat{b}(t-1)$ and $\tilde{b}(t-1)$ (denoted as Δ) and compensate it from the prediction error. The new prediction error becomes $\hat{e}(t)+\Delta$, which can be then quantized to $\hat{e}_2(t)$. The new reconstructed pixel value is

$$\tilde{b}(t) = \tilde{b}(t-1) + \hat{e}_2(t) = \tilde{b}(t-1) + Q(\hat{e}_1(t) + \Delta), \quad (4-19)$$

where $Q(\cdot)$ stands for the quantization process.

Actually, the new reconstructed pixel value, $\tilde{b}(t)$, is not necessarily a better approximation to the true pixel value, $b(t)$. It depends on the quantizer and the specific content of the video sequence. But conceptually, it should more or less improve the average quality. For the ping-pong game test sequence, the above error-correction method works fairly well. Especially, abruptly severe loss caused by error propagation can be removed with this method.

4.5.2.4 Simulations and Performance Comparisons

As described above, we use three methods to reduce the computation cost for compositing MC-compressed video — (1) reducing the number of blocks in the directly-affected area which need recalculation of their entire MC data, (2) simplifying the displacement measurement process, and (3) using the original motion vector for blocks in the indirectly-affected area. In this section, we examine their effect on recovered video quality by simulations.

We use a non-uniform quantizer in our simulations. The larger the prediction error is, the more noise is inserted (since the probability for small errors is higher than that for large errors). In other words, the quantization noise increases with the difference between the reference image and the current image.

Whenever MC recalculation takes place, more noise is introduced even the full-search displacement measurement is used. This issue is very similar to the error

accumulation issue we discussed earlier in Section 3.5.1, with the difference here that the prediction values could well be modified, while in Section 3.5.1 the same reference image frame is used.

In Figure 4-14, we show the average SNR among the directly affected blocks which require recalculation of their entire MC data. The average SNR is decreased by 2.88 ~ 5.45 dB if full-search displacement measurement is used. The additional SNR loss due to the proposed two-point displacement measurement procedure (in addition to the loss caused by the full-search MC recalculation) is small (within 1 dB) for most frames. Significant loss (about 5 dB) occurs in very few frames, such as frame 3. The reconstructed picture of frame 3 is shown in Figure 4-10, which at least subjectively does not illustrate server quality degradation.

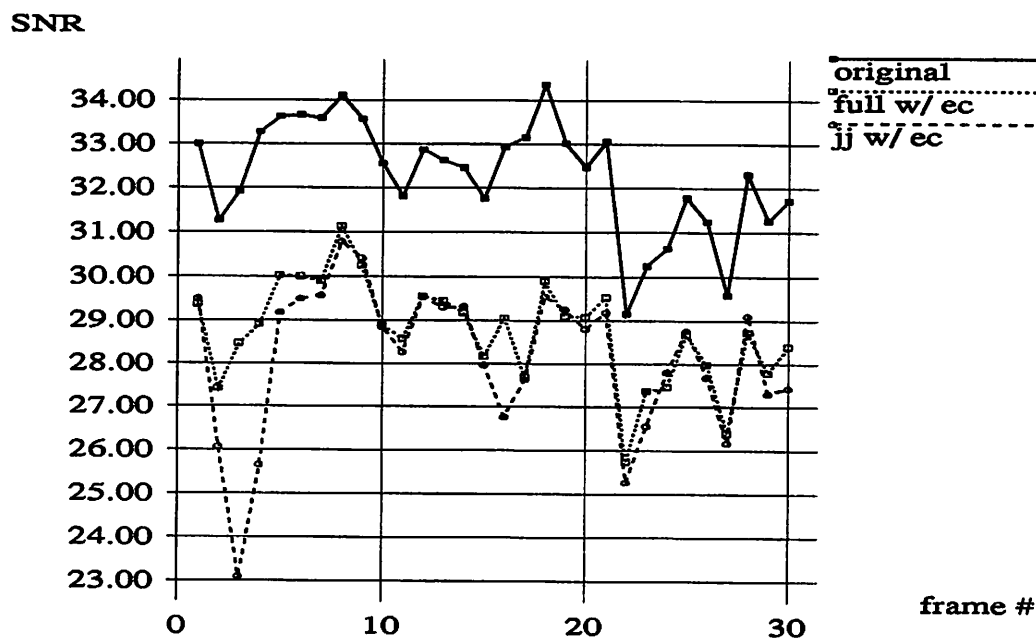


FIGURE 4-14 The average SNR among recalculated blocks within the directly-affected area. The top curve is the original input MC-compressed video, the middle curve uses the full search algorithm in MC recalculation, and the bottom curve uses our proposed simplified 2-point displacement measurement method. Correction of error propagation is used in both the middle and bottom curves.

For the indirectly-affected area, the average SNR loss among pixels affected by the error propagation effect ranges from 2.38 to 6.19 dB. Using our proposed error correction method, we can remove abruptly severe loss in several frames, such as frame 24 and 26.

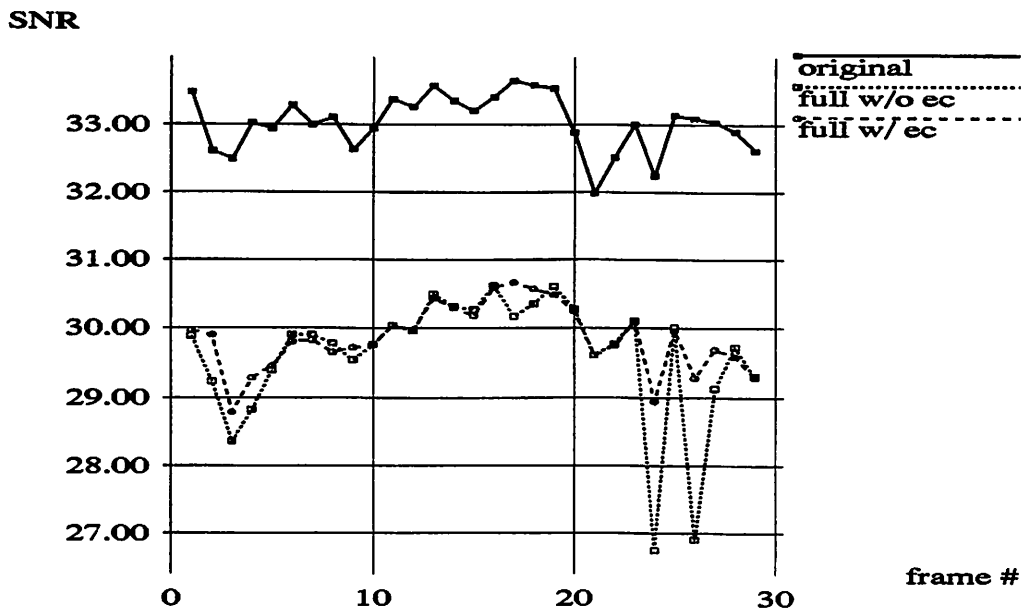


FIGURE 4-15 The average SNR among pixels affected by error propagation. We use the full search approach to do the motion measurement here.

4.6 Compositing MC-DCT Compressed Video

4.6.1 Partial Decoding of MC-DCT Video

From the discussions above, we know that video compositing can be done in the DCT compressed domain, but not in the MC-compressed domain. In order to keep the benefits of compressed-domain compositing, our philosophy is to perform as little decoding as possible in order to keep video at least in some “compressed” format, rather than in a fully-decoded uncompressed format.

However, the traditional MC-DCT decoder first decodes the DCT part and then the MC part. The non-linear MC algorithm is in the bottom of the compression stack. Our proposed approach is to swap the order of the decoding stack, namely inverse MC first, followed by inverse DCT. Then, we can apply video compositing in the DCT compressed

domain by using techniques derived in section 4.4. Note that the encoding stack doesn't have to change. We will describe the new decoding approach in the following.

As shown in Figure 4-3, the decoding process for MC-DCT video can be described as follows:

$$P_{\text{rec}}(t, x, y) = \text{DCT}^{-1}(\text{DCT}(e(t, x, y))) + P_{\text{rec}}(t-1, x-d_x, y-d_y), \quad (4-20)$$

where P_{rec} is the reconstructed image, e is the prediction error, and d is the motion vector. We skip quantization of $\text{DCT}(e)$ for simplicity here. With a simple reordering, we can change this to

$$\text{DCT}(P_{\text{rec}}(t, x, y)) = \text{DCT}(e(t, x, y)) + \text{DCT}(P_{\text{rec}}(t-1, x-d_x, y-d_y)). \quad (4-21)$$

That is, the inverse MC procedure is performed before the inverse DCT, as shown in Figure 4-16. The inverse MC procedure is simply an addition operation, which adds the DCT coefficients of the prediction errors back to the DCT coefficients of the reference image. The DCT coefficients of the first image frame can be obtained from the intraframe coding of the initial frame in every video segment.

In other words, the inverse MC procedure is performed in the DCT domain, which is denoted as “ $(\text{MCD})^{-1}$ ” in Figure 4-16. The $(\text{MCD})^{-1}$ procedure first locates the optimal reference block (by using the received motion vector) and then adds the DCT coefficients of the optimal reference block to the DCT coefficients of the prediction errors. The received MC-DCT compressed video signal is converted back to the DCT domain.

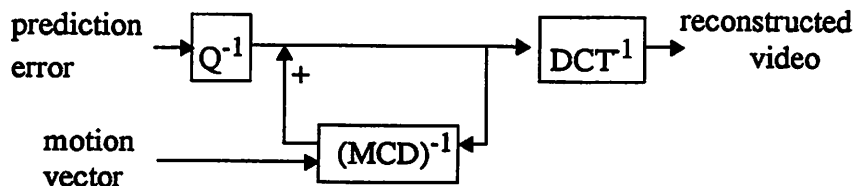


FIGURE 4-16 A new decoding algorithm for the MC-DCT compressed video. The inverse MC algorithm is performed before the inverse DCT, which is opposite to that used in traditional decoders. MCD: MC in the DCT domain.

However, the calculation of the DCT coefficients of the optimal reference block is not trivial. The motion vector of each block could be an arbitrary number of pixels, and

not the fixed block boundary used in the DCT algorithm. The optimal reference block generally overlaps with four blocks whose DCT coefficients are available from the previous frame, as shown in Figure 4-17. Calculating the DCT of the optimal reference block is equivalent to that for the pixel-wise translation discussed in Section 4.4.3, in which we need to calculate the DCT coefficients of a new arbitrary-position block from the given DCT coefficients of overlapping blocks. We can use the linear operation shown in equation (4-13) to obtain the DCT of the reference block, i.e., $\text{DCT}(P_{\text{rec}}(t-1, x-d_x, y-d_y))$. Then, using equation (4-21), we can perform the inverse MC algorithm in the DCT domain and partially decode the MC-DCT compressed images to the DCT domain.

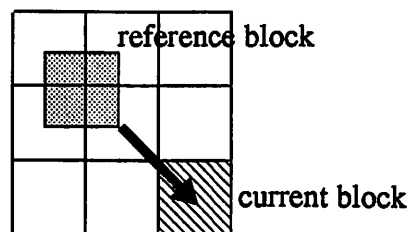


FIGURE 4-17 The MC optimal reference block generally overlaps with four blocks in the DCT block structure.

The same approach can be used in the forward motion compensation process when the composited image needs to be compressed again. The prediction error can be calculated as follows

$$\text{DCT}(e(t, x, y)) = \text{DCT}(P_{\text{rec}}(t, x, y)) - \text{DCT}(P_{\text{rec}}(t-1, x-d'_x, y-d'_y)) \quad (4-22)$$

where d' is the new motion vector for the composited video sequence. New motion vectors of the composited images can be obtained with inferences from the original motion vectors, as described in section 4.5.2.2. Using this simplified method to re-calculate new motion vectors can reduce the computations of the MC algorithm dramatically.

If the motion vectors are zero (as for DPCM interframe coding) or integral multiples of the block width, the block structure alignment procedure is not necessary. Motion compensation in the DCT domain requires simple additions only, as in the spatial domain. If one motion vector component (d_x or d_y) is zero or integral multiples of the

block width, the DCT coefficients of the new block can be computed from the DCT coefficients of only two original overlapping blocks, rather than four blocks, i.e.,

$$\text{DCT}(\text{new block}) = \text{DCT}(H_1)\text{DCT}(B_1) + \text{DCT}(H_2)\text{DCT}(B_2) \quad (4-23)$$

The computational complexity will be discussed later.

It is worth mentioning that our proposed decoding method for MC-DCT video can be directly used to perform format conversion, such as from the MPEG format to the JPEG format. Figure 4-16 shows how the MC part in the MPEG format can be stripped without affecting the DCT part.

4.6.2 Compositing MC-DCT video in the DCT Domain

Once MC^{-1} is performed and all input images are converted to the DCT domain, we can apply the techniques described in section 4.4 to composite them directly in the DCT domain. For comparison, Figure 4-18 shows the block diagram for compositing two MC-DCT video sequences in the uncompressed spatial domain. Both input videos are decompressed fully first, composited in the uncompressed domain, and then the final composited video is compressed again for further transmission. Figure 4-19 shows the proposed approach for partially decoding both input video streams into the DCT compressed domain, compositing them in the DCT domain, and then re-encoding the final DCT-domain composited video using the MCD encoder. All conversions back and forth between the DCT and spatial domains are eliminated. The MC and inverse MC algorithms are performed in the DCT domain, rather than in the spatial domain. Using the heuristics

mentioned above (section 4.5.2) for obtaining new motion vectors prevents the final MC step from becoming dominant in terms of the computational complexity.

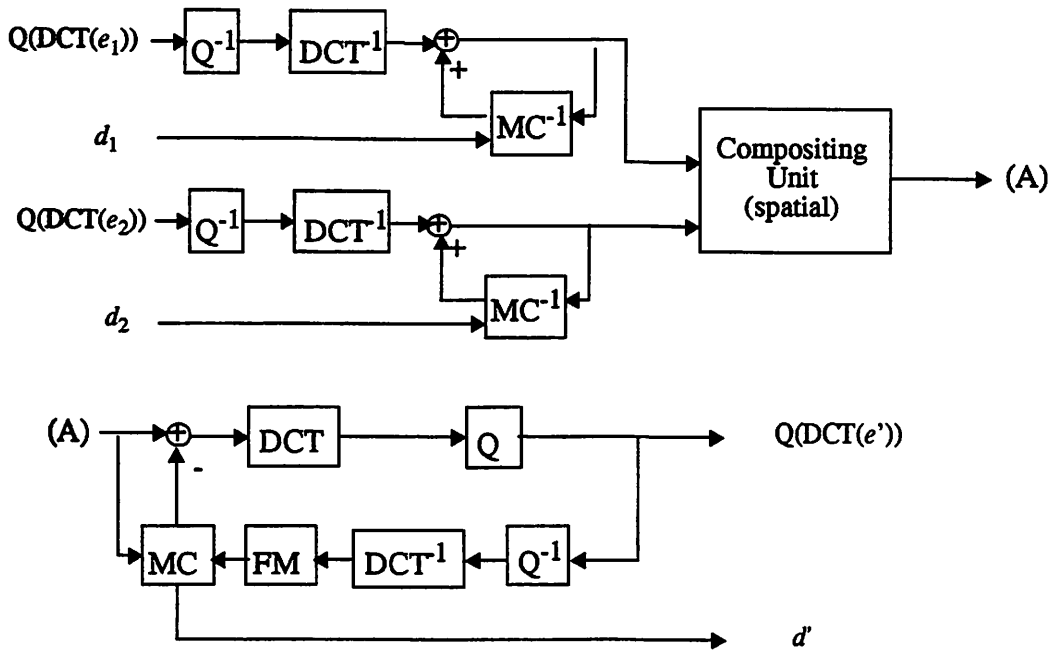


FIGURE 4-18 Compositing two MC-DCT compressed video sequences in the uncompressed domain. We decode both input videos fully back to the uncompressed domain, composite them pixel by pixel, and then encode the composited video to the compressed format. (d_1, e_1) and (d_2, e_2) are (motion vector, prediction error) for input video streams. (d', e') is for the composited output video.

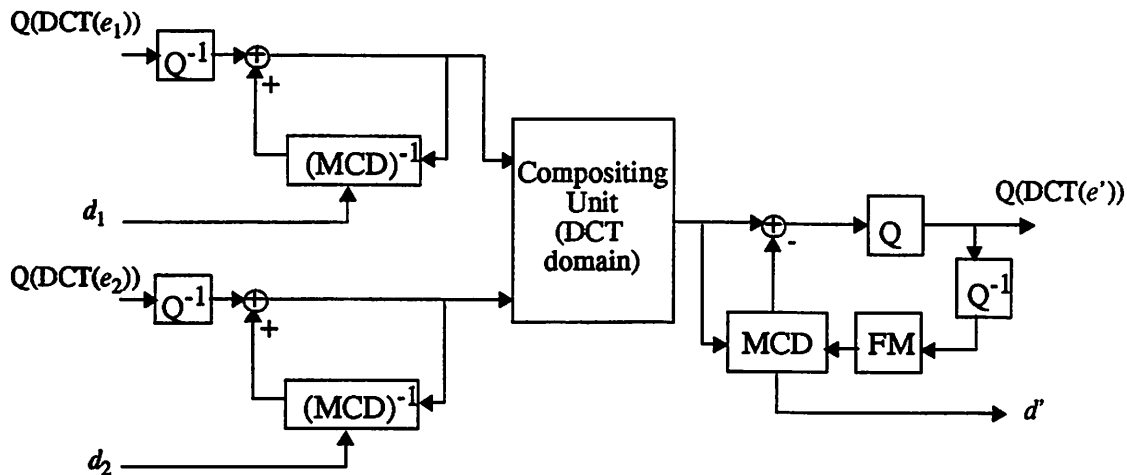


FIGURE 4-19 Compositing two MC-DCT compressed video in the DCT domain. We convert input video to the DCT compressed domain, composite them in the DCT domain, and then convert the composited video to the MC-DCT format.

4.6.3 Performance Analyses

We will analyze some performance metrics of the DCT-compressed-domain compositing approach both analytically and numerically in this section. Considered performance factors include computational complexity, recovered video quality and latency. Possible hardware architectures will be briefly discussed. We simulate some envisioned compositing scenarios, such as video conferencing scenes, by non-real-time software prototyping of proposed compositing algorithms.

4.6.3.1 Computational Complexity

The computational speedup of using the DCT-domain compositing algorithms in comparison to the spatial-domain approach depends on the compositing features supported in the applications, such as overlapping, block-wise or pixel-wise translation, scaling, and arbitrary-shaped video objects. It also strongly depends on the number of zero DCT coefficients, which in turn depends on the compression ratios of the input images. Generally, pixel-wise operations are more efficient in the spatial domain than in the DCT domain. But since the DCT-domain approach can avoid the conversion back and forth between the spatial and DCT domains, and it processes much less data if the compression ratio is high, the DCT-domain approach could provide a net efficiency gain for many cases involving pixel-wise operations.

For compression systems including the MC algorithm, we can use the approach described in Figure 4-19 to perform MC, inverse MC, and compositing operations all in the DCT domain. The overhead of converting all images to the DCT domain is the block boundary adjustment process required in the MC and inverse MC operations in the DCT domain (which will be called MCD and MCD^{-1} in the following context). We analyze the computational complexity of these operations in this section. Later, we compare these analyses to the simulation results.

In table 4-2, we list the number of multiplications and additions required in each major operation, such as the DCT, DCT^{-1} , MCD, MCD^{-1} , pixel-wise translation, quantization, and inverse quantization. We leave the detailed derivations to the Appendix. An important property we use for the DCT-domain operation is that the run-length-code (RLC) of the quantized DCT coefficients can indicate the position of the non-zero values

so that we can skip the redundant operations for zero values. Note that the compression ratio parameter (β) shown in the table simply means the ratio of the number of original DCT coefficients to the number of non-zero DCT coefficients after quantization. It's different from the overall compression ratio, which also takes VLC and MC into account.

Table 4-2 Computational complexity for major compositing functions^a

	operation	# of multi. / pixel ^b	# of add. / pixel
DCT Domain:	MCD, MCD ⁻¹	$(4/\beta + 2/\sqrt{\beta}) \cdot N \cdot \alpha_2 + (2/\beta) \cdot N \cdot \alpha_1$	$[(4/\beta + 2/\sqrt{\beta}) \cdot N + 3] \cdot \alpha_2 + [(2/\beta) \cdot N + 1] \cdot \alpha_1 + 1$
	scale 1/2×1/2	$(1/\beta + 1/(2\sqrt{\beta})) \cdot N$	$(1/\beta + 1/(2\sqrt{\beta})) \cdot N + 3/4$
	scale 1/3×1/3	$(1/\beta + 1/(3\sqrt{\beta})) \cdot N$	$(1/\beta + 1/(3\sqrt{\beta})) \cdot N + 8/9$
	pixel-wise translation	$(2/\beta + 2/\sqrt{\beta}) \cdot N$	$(2/\beta + 2/\sqrt{\beta}) \cdot N + 3$
	pixel multiplication	$N^2/(\beta_1 \cdot \beta_2)$	$N^2/(\beta_1 \cdot \beta_2)$
	semi-transparent block-wise overlapping	$< (1/\beta_1 + 1/\beta_2)$	$< 2 \cdot (1/\beta_1 + 1/\beta_2)$
Spatial Domain:	FDCT, FDCT ^{-1 c}	$2 \cdot \log_2 N - 3 + 8/N$	$3 \cdot (\log_2 N - 1) + 4/N$
	MC, MC ⁻¹	0	1
	scale 1/2×1/2	1/4	3/4
	scale 1/3×1/3	1/9	8/9
	pixel multiplication	1	0
	semi-transparent block-wise overlapping	1	2
Common:	opaque block-wise overlapping	0	0
	Inverse Quantization	1/ β	0
	Quantization	1	0

a. Notations:

β : Total # of the DCT coefficients / # of the non-zero DCT coefficients, e.g. (percentage of non-zero DCT coefficients)⁻¹. This is different from *compression ratio*, which is generally defined as (# Of bits for the original image / # of bits for the compressed image). But these two terms usually grows proportionally. We will use them interchangeably in the context, except in the calculation of computational complexity.

α_2 : the percentage of image blocks which need block boundary adjustment in both directions, i.e., both d_x and d_y are not integer multiples of the block size.

α_1 : the percentage of image blocks which need block boundary adjustment in only one direction, i.e., one of d_x and d_y is integer multiples of the block size, and the other one is not.

N: the block width (height), e.g. N=8 in our experiments.

b. the normalized complexity is calculated by dividing the overall computations with the number of pixels in the original image.

c. Using the fast DCT algorithm of Chen & Smith[Chen77]. If we use the 2N-point FFT approach, the computational complexity will be doubled.

The complexity of the MCD (or inverse MCD) algorithms depends on both the compression ratio (β) and the percentage of image blocks that need block boundary adjustment (α_2 and α_1), as defined in Table 4-2. When both the motion vector components d_x and d_y are not integer multiples of the block width, we need to calculate the DCT coefficients of the reference block from the DCT coefficients of four original blocks. The computational complexity is $(4/\beta + 2/\sqrt{\beta}) \cdot N$ multiplications per pixel, whose derivation will be described in the Appendix. If one of the motion vector components is zero or an integer multiple of the block width, then the DCT coefficients of the reference block can be constructed from two original blocks only. The complexity is $(2/\beta) \cdot N$ multiplications per pixel. Pixel-wise translations in the DCT domain also require the block structure adjustment in both directions. But since some matrix multiplications can be shared, its complexity is reduced to $(2/\beta + 2/\sqrt{\beta}) \cdot N$ multiplications per pixel. As discussed in section 4.4.5, the scaling operation, like linear filtering, can be implemented by multiplications with a pre-matrix and a post-matrix in the DCT domain. The computational complexity for the $1/2 \times 1/2$ down scaling operation in the DCT domain is $(1/\beta + 1/(2\sqrt{\beta})) \cdot N$ multiplications per pixel. Similarly, the complexity for the $1/3 \times 1/3$ down scaling operation in the DCT domain is $(1/\beta + 1/(3\sqrt{\beta})) \cdot N$ multiplications per pixel. The reduction is due to the fact that more image blocks share a single matrix multiplication. The detailed derivation is described in the Appendix.

For the spatial-domain operations, the major computations are from the conversion process, i.e., the DCT algorithm and its inverse process. Chen's fast algorithm [Chen77]

has a complexity of $2 \cdot \log_2 N - 3 + 8/N$ multiplications per pixel, which is about one half of that using the $2N$ -point FFT [Clarke85]. There have been many new fast DCT computation methods reported in literatures [Narasimha78, Lee84, Chan91], but all share a similar complexity order. For the $1/2 \times 1/2$ down-scaling operation, the required computational complexity is $1/4$ multiplication per pixel. ($P_{\text{new}} = (P_{11} + P_{12} + P_{21} + P_{22})/4$ by using the simple box area averaging algorithm described in [Weiman80]). It becomes $1/9$ multiplication per pixel for the $1/3 \times 1/3$ down scaling operation.

For the quantization process, each coefficient needs one multiplication (multiplied by $1/\text{quantization step}$). In the inverse quantization process, each non-zero coefficient needs one multiplication. In other words, the computational complexity is $1/\beta$ multiplications per pixel.

One interesting note is that the major component for the compressed-domain operations increases linearly with the block width, N , while it increases with the order of $\log_2 N$ for the uncompressed-domain approach (although when the block size increases, the image compression ratios may change as well.) Therefore, the compressed-domain approach is more suitable for cases using a small block size, which is usually true in image compression.

We use a typical block width of 8 pixels and some hypothetical values for the compression ratio (β) and non-zero motion vector percentages (α_2 and α_1) to compute some numerical figures for comparison, shown in table 4-3. We find that the MCD and inverse MCD algorithms are quite complicated in the worst case, i.e. $\alpha_2 = 100\%$. Even with a compression ratio of 16, the required complexity is still higher than that for the spatial-domain operations. However, for typical head-and-shoulder images, the non-zero motion vector percentages will be much smaller, due to the still background and the slow motion of the foreground person. With a moderate percentage ($\alpha_2 = 25\%$, $\alpha_1 = 50\%$), the MCD (MCD^{-1}) operations are less complicated than the fast DCT when the compression ratio is equal to or higher than 8. For video conferencing images which include a flat background, this compression rate is feasible. If the non-zero motion vector percentages are zero, the MCD and MCD^{-1} are equivalent to the original MC and MC^{-1} in the spatial domain. No extra computations are needed.

**Table 4-3 Computational complexity for major compositing operation
(with a 8 pixels by 8 pixels block size)**

	operation	$\beta=4$	$\beta=8$	$\beta=10$	$\beta=16$
DCT domain:	MCD, MCD ⁻¹ ($\alpha_2=100\%$, $\alpha_1=0$)	16 × 19 + ^a	9.66 × 12.66 +	8.26 × 11.26 +	6 × 9 +
	MCD, MCD ⁻¹ ($\alpha_2=25\%$, $\alpha_1=50\%$)	6 × 8.2 +	3.41 × 5.61 +	2.87 × 4.07 +	2.0 × 4.2 +
	MCD, MCD ⁻¹ ($\alpha_2=0$, $\alpha_1=0$)	0 × 1 +	0 × 1 +	0 × 1 +	0 × 1 +
	scale 1/2×1/2	4 × 4.75 +	2.41 × 3.16 +	2.06 × 2.81 +	1.5 × 2.25 +
	scale 1/3×1/3	3.33 × 4.22 +	1.94 × 2.83 +	1.64 × 2.53 +	1.17 × 2.06 +
	pixel-wise translation	12 × 15 + ^b	7.66 × 10.66 +	6.66 × 9.66 +	5 × 8 +
	semi-transparent block-wise overlapping	< 0.5 × < 1 +	< 0.25 × < 0.5 +	< 0.2 × < 0.4 +	< 0.125 × < 0.25 +
Spatial domain:	DCT, DCT ⁻¹	4 × 6.5 +			
	MC, MC ⁻¹	0 × 1 +			
	scale 1/2×1/2	0.25 × 0.75 +			
	scale 1/3×1/3	0.11 × 0.89 +			
	semi-transparent block-wise overlapping	1 × 2 +			
Common:	opaque block-wise overlapping	0 × 0 +			
	Inverse Quantization	0.25 × 0 +	0.125 × 0 +	0.1 × 0 +	0.06 × 0 +
	Quantization	1 × 0 +			

a. The numbers shown are # of real multiplications per pixel and # of real additions per pixel.

- b. The complexity for the pixel-wise translation seems to be very high here. But in many video conferencing scenes, the input video will be first scaled down by some factor before being translated. In this case, the significance of intensive computation associated with pixel-wise translation is small compared to the complexity of other operations.

- **Worst-Case Considerations**

The DCT-domain operations produce variable throughput, as opposed to the constant throughput for the uncompressed-domain approach. The higher the compression ratios and the lower the non-zero motion vector percentages the input images have, the faster the compositing unit can composite the input images in the compressed domain. For real-time implementations which need to consider the worst-case situation, this variable throughput may be a shortcoming. But in the DCT domain, we have the advantage that we can skip the high-order DCT coefficients whenever the maximal processing delay bound is exceeded. The image quality will be hurt as little as possible since the high-order coefficients are usually less subjectively important. For example, in compositing H.261 compressed video sequences, given the non-zero motion vector percentages of the input images, we can decide how many high-order DCT coefficients we need to drop in order to meet the processing delay bound for the real-time compositing requirements. This decision can be made based on the relationship between the computational complexity and the compression ratios shown in table 4-2. High-order DCT coefficients need to be dropped only when necessary. For input images with high compression ratios and low non-zero motion vector percentages, all DCT coefficients can be processed in time and no additional image impairment is introduced. In essence, this approach decides the amount of non-trivial DCT coefficients based on the hardware processing capability, while in constant-rate video encoders, rate-based criteria are employed.

- **Techniques to Reduce the Computational Complexity**

As we can see in table 4-2, the most complex operations in the compressed domain are the MCD (inverse MCD) algorithms, and the pixel-wise translation. This is mainly due to the need for computing the block structure realignment. However, for the MCD and inverse MCD operations, if the non-zero motion vector percentages, i.e. α_2 and α_1 , are decreased, computations can be greatly reduced. When α_2 and α_1 are both equal to zero, the interframe coding is equivalent to the DPCM interframe coding algorithm, in which

the inverse MC algorithm can be easily computed in the DCT domain by using simple additions. In the regular MC algorithm, these percentages vary with different video sequences. For example, Figure 4-20 shows some experimental results of the non-zero motion vector percentage. The “salesman” video sequence has more image blocks with zero motion vectors than the “Miss USA” sequence. One way to reduce the α_2 and α_1 percentages is to modify the MC encoder to give some preference to the reference block with zero motion vectors. For example, instead of searching for the optimal reference block with the minimal block distortion, we can add a rule forcing the optimal reference block to have a block distortion value smaller than 80% of that for the zero-motion block. Otherwise, the zero-motion block is selected as the reference block. We refer to this algorithm as the *modified MC (80%) algorithm* in the following and show that it has much smaller non-zero mv percentages than the original MC algorithm, as shown in table 4-4. For example, the non-zero motion vector percentages (α_2 , α_1) for one test frame of the “Miss USA” video sequence can be reduced from (52%, 38%) to (30%, 19%). However, the effect of this suboptimal MC algorithm on other compression performance factors (like compression ratio and image quality) may need to be further considered.

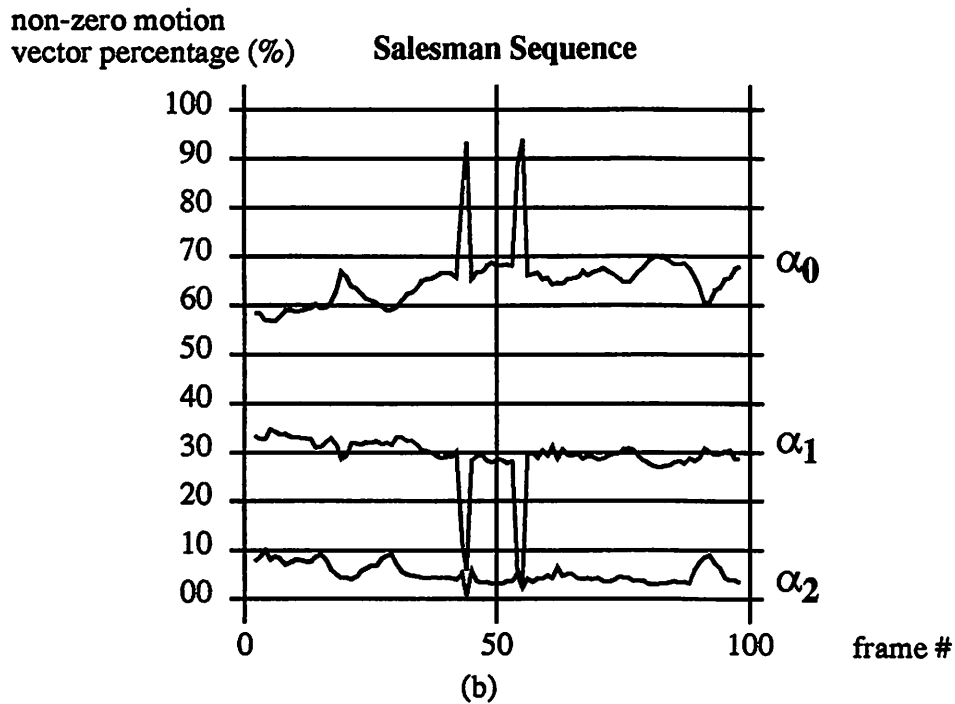
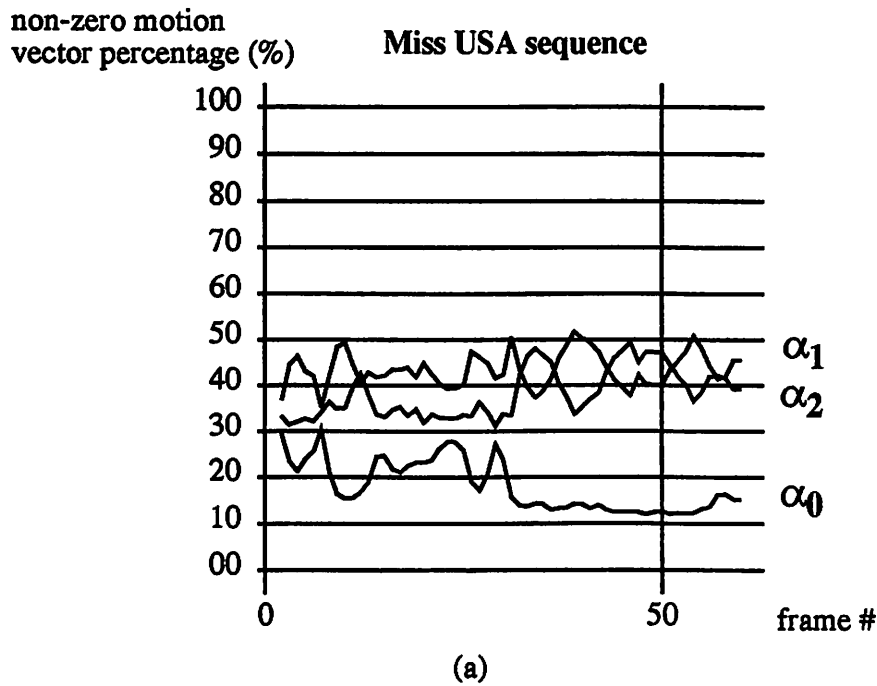


FIGURE 4-20 Experimental results of non-zero motion vectors. (a) "Miss USA" sequence (b) "Salesman" sequence.

α_2 : the percentage of the motion vectors whose components are non-zero in both the x and y directions.

α_1 : the percentage of the motion vectors whose components are non-zero in only one direction (x or y).

α_0 : $1 - \alpha_2 - \alpha_1$

Table 4-4 The non-zero motion vector percentage and compression ratios for test video sequences. The input video is MC-DCT compressed.

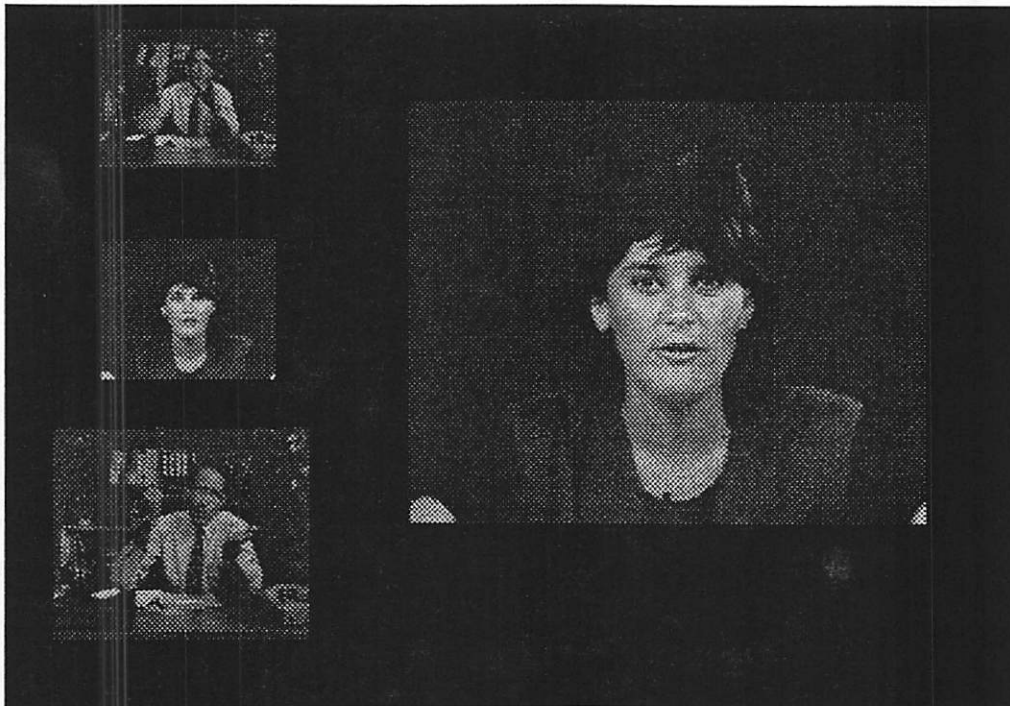
	“Salesman” frame 121	“Miss USA” frame 31	composited scene 1	composited scene 2	composited scene 3
α_2, α_1 (regular MC) ^a	8%, 19%	52%, 38%	3%, 9%	5%, 12%	43%, 38%
α_2, α_1 (modified MC (80%)) ^b	6%, 7%	30%, 19%	2%, 5%	3%, 7%	24%, 18%
β	7.69 ^c	20.0 ^d	9.09	7.69	9.09

- a. The regular MC algorithm uses the full-search block-based displacement measurement. The block size is 8 pixels \times 8 pixels and the maximum motion distance in each direction is 8 pixels in our simulations.
- b. The modified MC algorithm uses the rule that the selected reference block has a block distortion less than 80% of that for the zero-motion block. Otherwise, it uses the zero-motion block as the reference block.
- c. The first figure is for the reference image frame (e.g. frame 120), and the second figure is for the reconstructed frame after the MCD^{-1} algorithm (e.g. frame 121). The actual compression ratio is different from β , because overhead such as the run length code and the end-of-block delimiters are not included in calculating β .
- d. The low percentages for the Miss USA sequence are mainly due to its flat dark background.

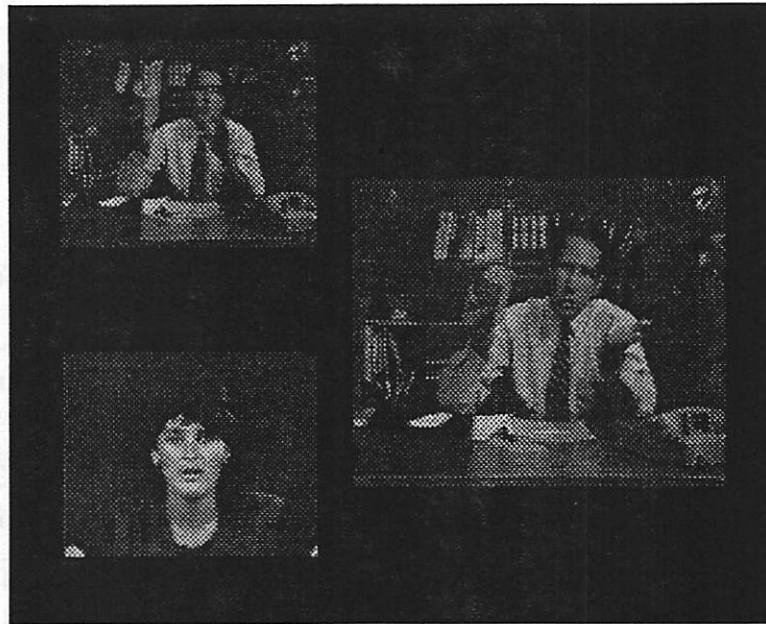
Another possible technique to reduce the computational complexity in the DCT domain is to combine a sequence of operations into a single operation. For example, for MC-DCT compressed images, we may need to perform $F(DCT(e) + G(DCT(P_{ref})))$, where G represents the MCD^{-1} operation and F represents the scaling operation. Since they are both linear operations, we can apply the distributive law to change the above formula to $F(DCT(e)) + H(DCT(P_{ref}))$, where H represent the composite function $F \cdot G$. The computations for function F can be reduced since the non-zero coefficient ratio for $DCT(e)$ is usually higher than that without motion compensation, e.g., $DCT(P_{ref})$. However, for generality, we do not use this method in the following simulations.

• Simulations

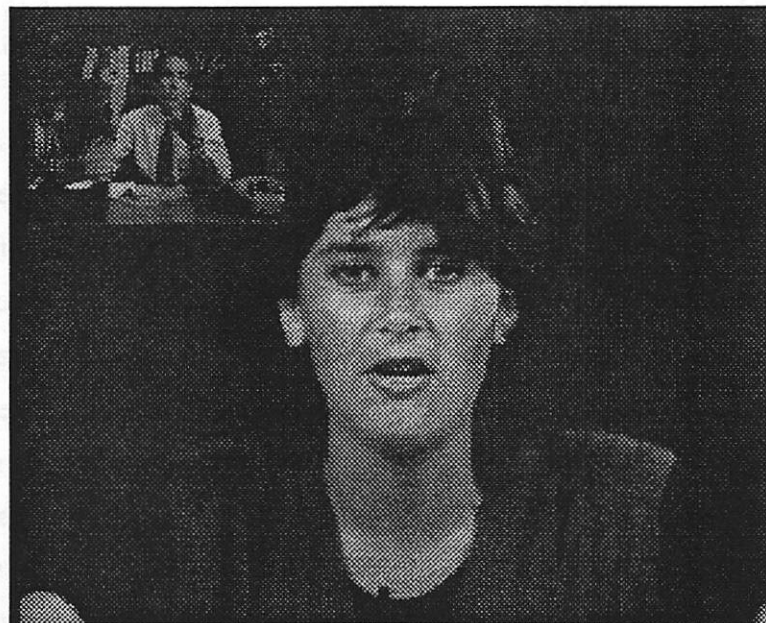
We have simulated different compositing scenarios, which are depicted as Scene 1, 2, 3 in Figure 4-21. In Scene 1, input images are of the size of 360 pixels \times 288 pixels, which is approximately the CIF format, and the composited output is of a larger size of 696 pixels \times 480 pixels, which is envisioned to be the size of the final video display. In Scenes 2 and 3, both the input sequences and the output sequence are of the same size (360 pixels \times 288 pixels). The input sequences are all head-and-shoulder images, which are most common in a video conference. We assume each video source is compressed with the MC-DCT algorithms, and the compositing unit needs to generate the composited video in the same compressed format. We tested several different quantization tables, including those listed in the JPEG and MPEG standards. The compression ratios and non-zero mv percentages for these images are shown in table 4-4.



(a)



(b)



(c)

FIGURE 4-21 original pictures for different composing scenarios (a)Scene 1 (b) Scene 2 (c) Scene 3.

The computational complexities for each compositing scenario are shown in table 4-5. Compositing features demonstrated here include overlapping, scaling with different ratios and block-wise translation. If the regular MC algorithm is employed, the DCT-domain approach is faster than the spatial-domain approach by about 10% to 40%. If we use the modified MC algorithm (80%), the speedup can be further increased.

Table 4-5 Computation speedup for compositing MC-DCT compressed video in the DCT-compressed domain vs. the uncompressed spatial domain.

	Spatial-domain compositing	DCT-domain compositing (regular MC)		DCT-domain compositing (modified MC—80%)	
	# op./pixel	# op./pixel	speedup	# op./pixel	speedup
Scene 1	33.28 mul. 57.57 add	26.63 mul. 38.96 add	1.25 1.48	19.69 mul. 29.02 add	1.69 1.98
Scene 2	17.99 mul. 32.53 add	16.10 mul. 23.89 add	1.12 1.41	12.80 mul. 19.29 add	1.41 1.69
Scene 3	13.41 mul. 23.39 add	13.76 mul. 20.39 add	0.97 1.15	9.23 mul. 14.06 add	1.45 1.66

We implemented the above three compositing scenarios in C programs on a SUN SPARC I workstation. The CPU time needed is reported in table 4-6. Both regular MC algorithm and the modified MC algorithm are tested. The resulting speedup figures are approximately in line with the theoretical predictions shown in table 4-5. The modified MC algorithm can reduce the complexity of the MCD and MCD^{-1} algorithms and extend the speedup by 10% to 20%. If we adopt a stronger preference towards the zero-motion blocks in the MC algorithm, e.g. using a stronger preference factor lower than 80%, the speedup will be further increased. Another way to boost up the speedup is using larger quantization steps at the cost of heavier quality loss. This step could be necessary when the non-zero motion vector percentages are high, and the real-time implementations are required.

Table 4-6 Software prototype for compositing MC-DCT compressed video in the DCT-compressed domain vs. the uncompressed spatial domain.^a

	Spatial-domain compositing	DCT-domain compositing (regular MC)		DCT-domain compositing (modified MC—80%)	
	CPU time ^b	CPU time	speedup	# op./pixel	speedup
Scene 1	40.72 sec.	31.46 sec.	1.29	25.68 sec.	1.59
Scene 2	22.53 sec.	20.29 sec.	1.11	18.02 sec.	1.25
Scene 3	16.57 sec.	14.69 sec.	1.13	11.53 sec.	1.44

a. The DCT-domain implementation is not fully optimized yet. Some sharable matrix multiplications are duplicated. The spatial-domain approach uses the fast DCT algorithm by Narasimha and Peterson [Narasimha78]. Its complexity is about the same as that of Chen's fast algorithm.

b. CPU time on a SPARC I machine.

If the original images are compressed without the MC algorithm, namely DCT-encoded only as in the JPEG standard, the computational speedup by using the DCT-domain algorithms can be increased greatly, as shown in table 4-7, since the complicated MCD operation is not required. The net computational complexity can be reduced by a factor ranging from about 3 to 6.

Table 4-7 Computation speedup for compositing DCT-compressed video (without MC) in the DCT-compressed domain vs. the uncompressed spatial domain.

	Spatial-domain compositing	DCT-domain compositing	
	# op./pixel	# op./pixel	speedup
Scene 1	33.28 mul 50.35 add	9.66 mul 11.23 add	3.45 4.48
Scene 2	17.99 mul 28.53 add	7.36 mul 8.01 add	2.44 3.56
Scene 3	13.41 mul 20.39 add	3.3 mul 2.89 add	4.06 7.06

4.6.3.2 Image Quality

As discussed earlier, compositing within the network requires a recompression process at the output of the intermediate compositing unit. Since the composited output video is different from the input video, this recompression process will introduce video quality impairment, in addition to the initial quality loss caused by lossy compression at each source (even if we use the error-accumulation-free compression algorithms such as the DCT). This problem has been described as the error accumulation problem of the iterated compression process in Section 3.5.1. Note that the quality loss due to the recompression at the compositing unit is unavoidable regardless of whether we use the spatial-domain or DCT-domain compositing approaches. Table 4-8 shows the SNR values for reconstructed images at the compositing unit before and after the recompression process. As described earlier, the SNR loss strongly depends on the compositing operations performed in the compositing unit. For example, the extent of change in Scene 2 is larger than that in Scene 3. Therefore, the SNR loss caused by re-compression is also much larger in Scene 2 (7.7 dB) than in Scene 3 (1.2 dB).

Table 4-8 SNR of reconstructed images at the compositing unit before and after the re-compression process. The input video is assumed to be MC-DCT compressed. ^a

	Before re-compression	After re-compression	
		Spatial-domain Compositing	DCT-Domain Compositing
Scene 1	31.5 dB	28.8 dB	28.5 dB
Scene 2	34.7 dB	27.0 dB	26.3 dB
Scene 3	30.7 dB	29.5 dB	29.5 dB

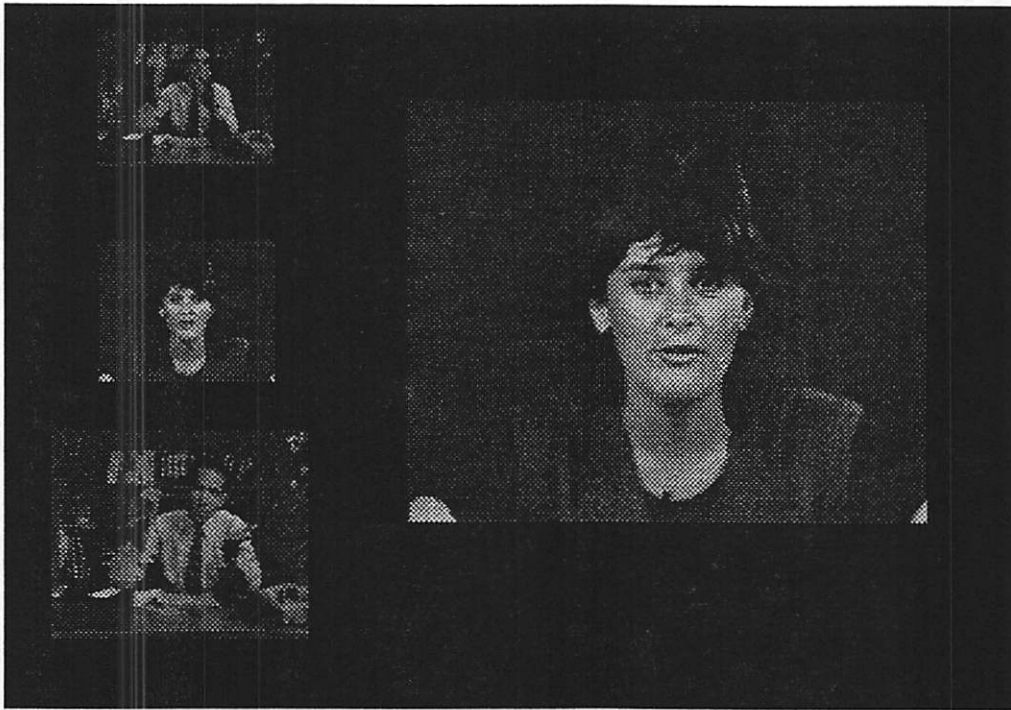
a. We use the regular MC algorithm here.

For DCT-compressed-domain compositing, there is a minor additional quality impairment due to the need to apply thresholding in every intermediate operation. For example, the reconstructed DCT coefficients of an input video from its MC-DCT compressed format does have a run-length-code format, as that available in the input compressed stream.¹ Therefore, in order to remove insignificant DCT coefficients and their associated computations, we apply a simple thresholding after the input video is

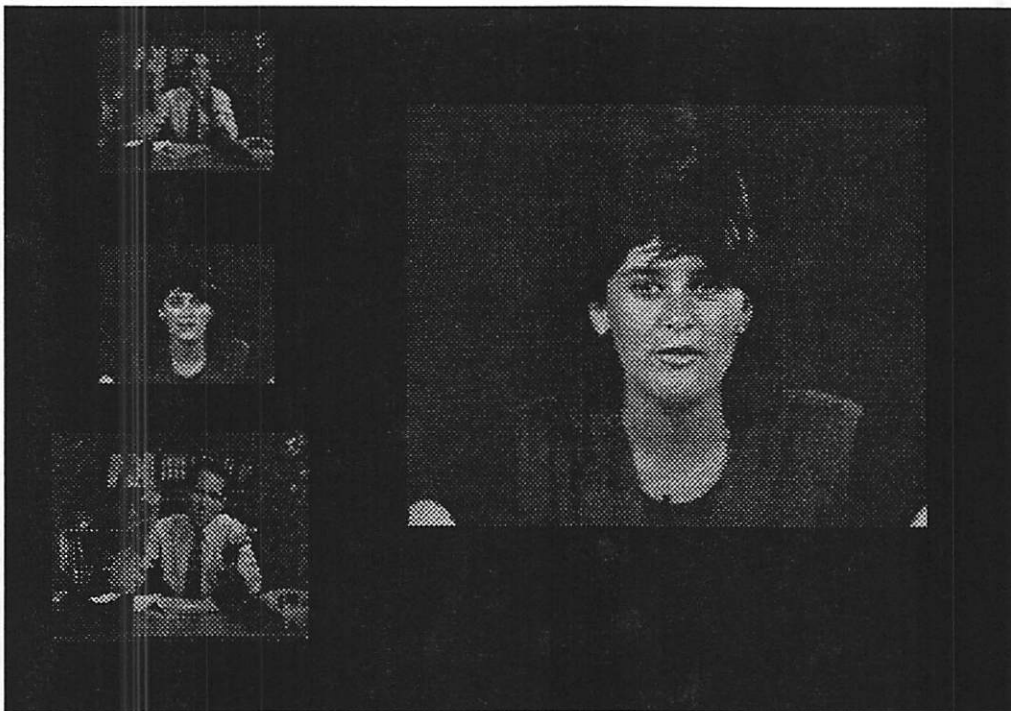
converted to the DCT domain.¹ The thresholding levels can be the quantization table in some intraframe transform coding algorithm, such as the JPEG table, or simply a constant value for each coefficient. From table 4-8, we can see that the additional image quality degradation caused by this thresholding is very minor, about 0 ~ 0.7 dB. There is no noticeable difference in subjective quality comparison. The reconstructed composited images for Scene 1 are shown in Figure 4-22. For the image shown in Figure 4-22(a), compressed input images are converted back to the spatial domain and composited pixel by pixel in the spatial domain. For the image shown in Figure 4-22(b), images are composited in the DCT-compressed domain. The final composited images are both transformed to the MC-DCT format, quantized, and run-length encoded. The images shown here are reconstructed from the final compressed data. We can see that both images composited in the uncompressed domain and the DCT domain suffer from some quality degradation due to their incurring lossy quantization twice.

1. The run-length-code of an MC-DCT compressed stream is for the DCT coefficients of the MC prediction errors, rather than the whole reconstructed image.

1. Note, unlike the original compression methods, DCT coefficients larger than the threshold values are not quantized. Only small DCT coefficients are truncated to zero.



(a)



(b)

FIGURE 4-22 Reconstructed images of Scene I composited in (a)the spatial domain (b)the DCT-compressed domain. All input video and output composited video streams are MC-DCT compressed.

As we mentioned earlier, the modified MC algorithm can effectively reduce the non-zero mv percentage and thus reduce the computations for the DCT-domain operations. We find that the SNR values of reconstructed images using the modified MC algorithm are very close to those using the regular MC algorithm.

4.6.3.3 Other Performance Considerations

Besides computational complexity and video quality, the compressed domain approach also has impacts on *hardware implementations*. The uncompressed-domain approach fully decodes video signals to the uncompressed domain and composites video pixel by pixel. Therefore, the compositing process is performed at the pixel rate. For the DCT-domain compositing approach, the operation rate can be reduced to the block level. For example, Lee and Lee propose a pipelined hardware architecture for transform domain filtering [Lee92], as shown in Figure 4-23. The input/output rate is block by block, which is much lower than the pixel rate.

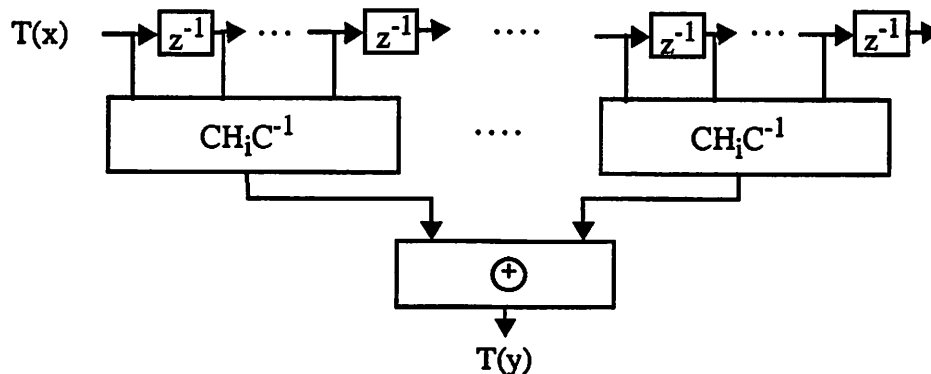


FIGURE 4-23 A pipelined hardware architecture for transform filtering (proposed by Lee and Lee [Lee92]).

Another aspect of the hardware cost is the buffer size required. The buffer size will directly affect the output latency, which needs to be kept small so that interactive services can be achieved. For MC-DCT compression algorithms, the most significant delay comes from the frame buffer in the (inverse) motion compensation part and the buffer for the variable coder/decoder, such as the run length code and the entropy code. The compressed-domain compositing approach proposed in this chapter still needs these buffers. For example, the DCT-domain motion compensation algorithm needs a frame

buffer to store the DCT coefficients of the previous image frame. The proposed compressed-domain compositing approach can only potentially reduce the overall latency through the reduction of the processing delay, rather than the removal of the frame buffers.

4.7 Summary

We explored the freedom of performing video compositing with different data formats. We have designed efficient algorithms in the DCT-compressed domain for many compositing operations such as overlapping, translation, scaling, pixel multiplication, and linear filtering. These compressed-domain algorithms can be applied to other orthogonal transform algorithms, like the DFT transform and the DST transform. In applications that require compressed input images, our proposed approach can reduce computations, compared to the straightforward approach which convert compressed images back to the spatial domain and composite them in the spatial domain. The computational speedup depends on the compression ratios of the input images, which can be adjusted to achieve the minimal overall system cost. To extend these DCT-domain algorithms to motion-compensated DCT-based images, we propose a new decoding algorithm to convert the MC-DCT compressed images to the DCT domain and composite them in the DCT domain. This new decoding algorithm can also be applied to direct conversion between image compression formats like between JPEG and MPEG.

The proposed DCT-domain compositing approach can reduce the required computations by 60% ~ 75% for DCT-compressed images, 10% ~ 23% for MC-DCT-compressed images in different simulated scenarios. The speedup factor depends on the compression ratios and the non-zero motion vector percentages. Namely, the compositing throughput is variable, varying from image to image. To meet the real-time requirement for most video applications, we can use a processing delay constraint to decide what percentage of high-order DCT coefficients should be skipped. For example, in the compositing unit within the network, if the input image has a high non-zero motion vector percentage, it can skip some high-order DCT coefficients in order to reduce computations and satisfy the maximal delay bound. If the input non-zero motion vector percentage is low, then all non-trivial DCT coefficients are processed and the processing delay bound is satisfied automatically. This freedom of reducing processing time by sacrificing least

important image components is not available in the spatial domain, as discussed in section 4.3.

Chapter 5

Arbitrarily-Shaped Video Objects (ASVO)

In previous chapters, we assume that the video objects can be arbitrarily shaped (AS). For example, in multimedia editing systems, users can create arbitrarily-shaped video objects manually or by segmentation algorithms. Users can manipulate each individual object or composite multiple video objects together. In the so-called analysis-synthesis (or object-oriented) video coding algorithms, AS video objects are segmented (by image analysis algorithms) and transmitted separately [Mussman89, Kunt87]. Separate objects are composited at the receiver to reconstruct the original scene. Figure 5-1 is a block diagram illustrating production and processing of AS video objects. The anti-aliasing process is necessary for smoothing the object boundaries to remove the jagged artifact. As described in Chapter 2, we use an additional α channel to perform anti-aliasing.

In this chapter, we will investigate how to efficiently produce the α value and to modify it after video objects are transformed or composited. This chapter also focuses on the *representation* of the ASVO, where both the *pixel values* and the *shape* are encoded. The former specifies the internal intensity variation while the latter specifies the boundary information. Regarding pixel values, we consider only intraframe transform coding. We propose a joint approach for representing the shape and calculating the α value.

5.1 Transform Coding of Image Pixels

In this section, we design efficient representations of the image pixel values to achieve good compression and image quality. In particular, we consider block-wise transform coding of the image content, such as the widely used Discrete Cosine Transform

(DCT). One immediate advantage of using the transform code is that existing codecs can be used to process AS video signals, as well as traditional rectangular video signals.

In block-wise transform coding algorithms, images are separated into small blocks with fixed size, say N pixels by N pixels. Figure 5-2 shows an example of an AS video object (Miss USA) and illustrates the concept of block structure. All pixel values in internal blocks are fully defined. The traditional DCT algorithm can be used to encode these blocks efficiently. For the boundary blocks, however, the pixel values are not completely defined. One straightforward approach is to fill zero values outside the boundary, and treat the resulting block as before. A drawback of this approach is the significant increase of the high-order transform coefficients, which may seriously degrade the compression performance. We will focus on transform coding techniques capable of

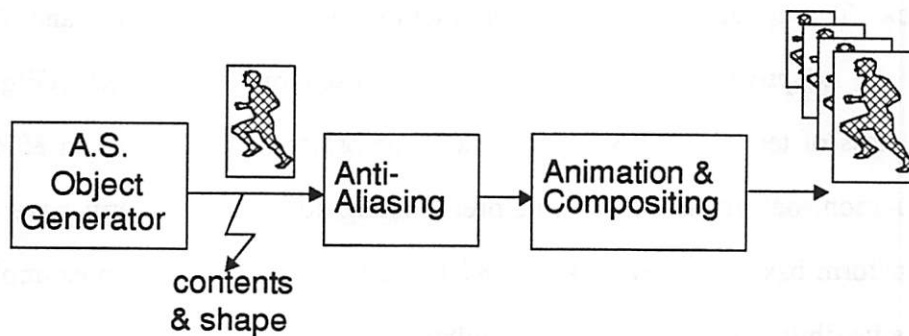


FIGURE 5-1 Production and processing (e.g., manipulating or compositing) of arbitrarily-shaped video objects.



FIGURE 5-2 An example AS image segment and the grid lines which separate the image into small blocks. Boundary blocks have part of pixel values defined only. The block structure is for demonstrative purpose and is not of accurate scale.

treating general arbitrarily-shaped image blocks, including both fully-filled and partially-filled image blocks. Note that we will use the term “AS image segment” to denote both the entire AS image frame, such as Miss America in Figure 5-2, and one AS image region within each partially defined boundary block. The distinction should be clear in the context.

We investigate two classes of transform coding techniques — *brute-force full-block transform* and *shape-adaptive transform*. The first class explores innovative ways of filling the redundant data outside the boundary in the boundary blocks and followed by the full-block DCT. The zero-stuffing method and traditional band-limited extrapolation techniques belong to this class [Jain89, Soltanian-Zadeh93]. The second class of transform methods changes the transform basis functions adaptively based on the shape of the input block. The iterative approximation method proposed by Kaup and Aach [Kaup92] and the adaptive orthogonal transform proposed by Gilge *et al.* [Gilge89] belong to this class of techniques. We propose a shape-projected domain as an efficient problem formulation, on which we can interpret existing adaptive transform bases and derive new transform bases. We derive a new KLT-like transform basis as an example to demonstrate the flexibility of the proposed formulation.

Afterwards, we compare the performance of different transform coding techniques and illustrate the tradeoff among the compression performance, computational complexity, and codec complexity.

5.1.1 Brute-Force Full-Block Transform

As mentioned earlier, image segments are separated into small blocks, e.g. N pixels by N pixels each. For AS image segments, boundary blocks usually have pixel values partially defined. Let $P(x,y)$ represent the pixel values within this N pixel \times N pixel block area, called R . Let B represent the occupied region within the block, as shown in

Figure 5-3. A partially-filled image block has $P(x,y)$ defined within region B . The brute-force full-block transform coding technique fills up the redundant area outside the boundary and then utilize the traditional block-wise transform coding.

Once the image data, $P(x,y)$, is extended to the full block, we can use traditional block-wise transform coding to represent the block as follows,

$$P(x,y) = \sum_i a_i \cdot f_i(x,y) ; x,y \in R, \quad (5-1)$$

where f_i 's are *basis functions* defined on the full-block area, R . That is, $P(x,y)$ is transformed to a set of coefficients a_i , which can be used to completely or partially reconstruct the original image. For the purpose of compression, we would like to use as small number of coefficients as possible to obtain an accurate reconstruction, $\hat{P}(x,y)$. The resulting error term is defined as

$$e = \sum (P(x,y) - \hat{P}(x,y))^2, \quad x,y \text{ in } B \quad (5-2)$$

Note that the summation is executed over the occupied region only because error terms outside the boundary are discarded when we apply the shape information at the receiver.

If we fix the choice of basis functions, e.g. use $N \times N$ DCT basis functions, the objective can be interpreted as finding the optimal $P(x,y)$ values outside region B so that the transform coefficients, a_i , present the highest energy compaction. The concept is illustrated in Figure 5-3. However, it is difficult to quantitatively formulate this abstract

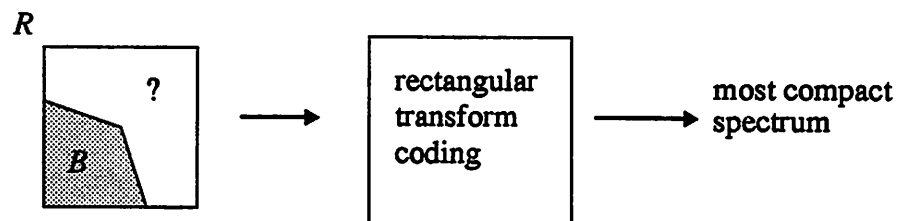


FIGURE 5-3 Find the optimal pixel values outside the boundary of image segment P , so that the transform spectrum has the most compact energy spectrum.

property of *energy compactness*. An example discussed in [Kaup92] uses the entropy definition

$$-\sum_i \left(\left(\frac{|f_i|}{\sum |f_i|} \right) \cdot \log \left(\frac{|f_i|}{\sum |f_i|} \right) \right) \quad (5-3)$$

to emulate the energy compactness of the transform spectrum. The problem with this definition is that the final choice usually ends up with few large spectrum components, which may cause overflow problems, though the spectrum “entropy” is low. Furthermore, optimization for minimizing the entropy is difficult.

5.1.1.1 Mirror Image Extension

Despite the difficulty in quantifying the compactness of spectrum coefficients, the approach of filling the region outside the boundary with optimal redundant data does provide us the freedom to optimize the transform spectrum. The simplest method to augment a partially defined image segment into a full block image is stuffing zero’s outside the image boundary. However, this may introduce sharp edges on the boundary and thus high-frequency components in the transform spectrum.

The traditional band-limited extrapolation approach assumes the input signal has a limited bandwidth the same size as the defined signal samples [Soltanian-Zadeh93, Jain89]. However, the goal of this approach is to find remaining undefined signal samples rather than data compression. The resulting spectrum may not be compressible at all¹. That is, all spectrum coefficients turn out to be non-trivial. In addition, the spectrum coefficients are usually obtained by solving a linear equation system. For a fixed spectrum passband, the associated linear equation system may be singular and have no solutions.

1. Compression comes from the quantization process which truncates small transform coefficients to zeros.

One promising alternative is to extend each image segment with its “*mirror image*” outside the image boundary. This approach has been used to improve the compression performance of subband coding [Karlsson89b]. Figure 5-4 shows a partially defined block in one dimension. In general, the defined pixels may not occupy exactly one half of the block. We may need to duplicate the given pixel values several times and truncate it at the block boundary. For a 2D image segment, we can apply this 1D mirror image extension in one direction first, and then in another direction. This mirror-image extension technique is simple but efficient. Its compression performance will be described later.

5.1.2 Shape-Adaptive Approach

As described in equation 5-2, we are only concerned with reconstruction errors within the image boundary, i.e. errors within the covered region B . An equivalent but perhaps more efficient approach to finding the optimal representation of AS image segments is to perform optimization only in the subspace defined over region B , denoted by S_B . Basically, we project the AS image segment and all basis functions into the

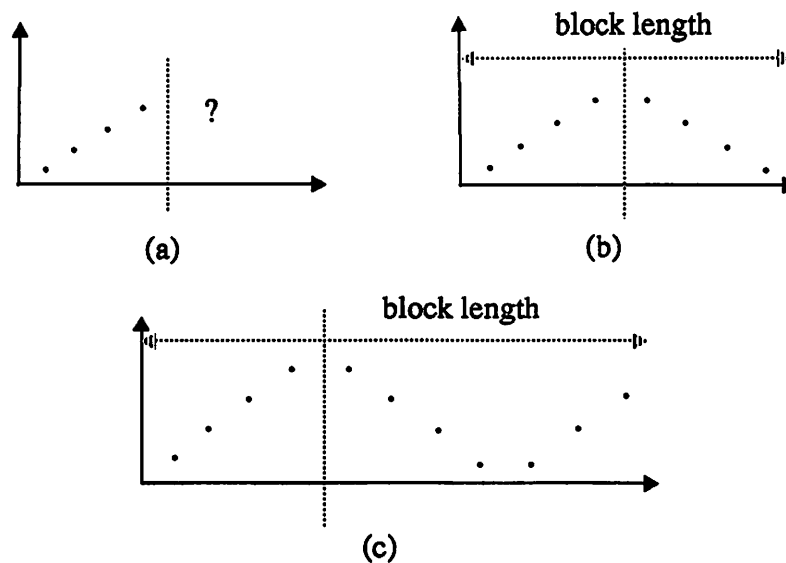


FIGURE 5-4 Fill the outside redundant region with the mirror image of the internal contents. (a) original segment. (b) the segment size equals one half of the block size. (c) apply the mirror image recursively when the segment size is not one half of the block size.

subspace S_B and find the optimal representation there. The redundant pixel values and their associated errors outside the boundary can thus be automatically ignored. However, since the subspace varies with the image shape, the optimal transform bases for different image shapes may also be different. This is the reason why this is called the *shape-adaptive* approach.

5.1.2.1 A New Problem Formulation — Shape-Projected Subspace

Instead of filling data outside the image boundary and applying a full-block rectangular transform, we can focus on the defined image pixels only, i.e. $P(x,y)$ values within region B . Mathematically, let's define S_R as the linear space spanned over the entire square block R , and S_B as the subspace spanned over the irregular region B . For example, in Figure 5-5, space S_R has a dimension equal to 16, while the dimension of subspace S_B is equal to 4. One possible basis for subspace S_B is shown in Figure 5-5(b). Each basis matrix has a single non-zero element.

Every arbitrarily-shaped image segment, $P(x,y)$, can be considered as a vector in S_B . To completely represent this vector, we need to find a set of linearly independent vectors, say $\{b_i\}$, in S_B and describe $P(x,y)$ as a linear combination of b_i 's. The distinction

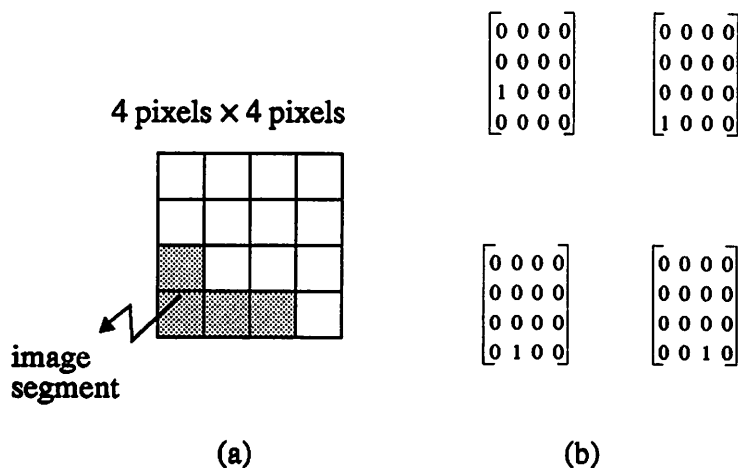


FIGURE 5-5 (a) A partially-filled image block in a 4x4 area. A canonical basis of the subspace is shown in (b).

between this approach and that in the previous section is that the entire problem domain now is confined only in the subspace S_B . We don't have to worry about the redundant data outside the image boundary, i.e. vector component outside the subspace S_B . If we want to use traditional block-based transform bases, say f_i (e.g. the DCT basis), we can project these basis functions into subspace S_B ,

$$\hat{f}_i = \text{Project}(f_i, S_B) \quad , \quad (5-4)$$

and describe vector $P(x,y)$ as a linear combination of \hat{f}_i 's. However, usually we will have too many basis functions and they will not be orthogonal. In actuality, for orthogonal systems, this projection simply removes the components of f_i orthogonal to subspace S_B .¹ Figure 5-6 illustrates an example when the dimension of S_B equals two.

The important issue that remains is to find the optimal representation of the image vector in subspace S_B such that we can use a small number of coefficients to reconstruct the image segment vector with a satisfactory quality. The above formulation does provide a very flexible platform to derive new transform bases and evaluate their performance. However, one disadvantage of this approach is that if we use shape-adaptive transform bases, then there is an overhead for constructing the new transform basis for each different

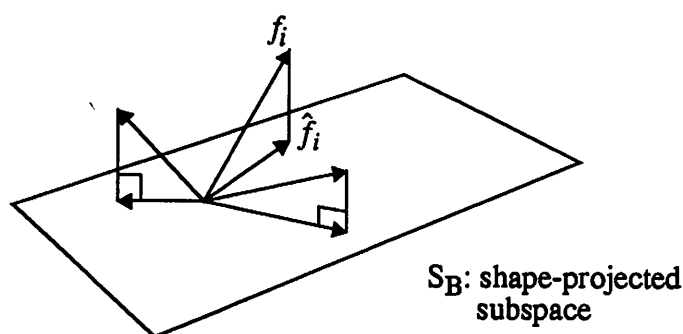


FIGURE 5-6 Project the image segment and all representation basis vectors into the subspace spanned over the image shape region only. The dimension of the full block space is more two and the dimension of the subspace is two.

1. Another interpretation of projection is to force all those component values of f_i outside S_B to be zero.

shape at the receiver. Note that for AS video objects, the shape information is also transmitted to the receiver. Therefore, we can reconstruct the correct transform basis at the receiver based on the received shape information. We do not have to transmit the transform basis functions.

An alternative of using the shape information to improve the coding efficiency is to adaptively change the mapping procedure for calculating the transform coefficients according to the varying shape, but still use a fixed transform basis (such as DCT). In this case, the overhead is only in the encoder. The decoders need not to be modified. There is no overhead for modifying the transform basis at the decoder and existing decoders can be used without any modification.

We will describe some known shape-adaptive approaches and our new proposal utilizing the above concepts in the following subsections.

5.1.2.2 Successive Approximation Algorithms Revisited

Using the existing full-block 2D DCT basis to represent arbitrarily-shaped image segments is attractive since existing decoders for rectangular images can be used without modifications. However, as described earlier, the shape-projected DCT basis functions, $\{\hat{d}_i\}$, are generally not orthogonal and linearly dependent. There are multiple solutions for equation 5-1 if $\{\hat{d}_i\}$ are used as the basis functions. Instead of finding a fully accurate representation, Kaup and Aach [Kaup92] proposed a successive approximation method to calculate only the most significant coefficients. However, the computational overhead of iterative approximations may be significant. Also, the number of unorthogonal transform basis functions stays the same even when the image segment size is small.

In this section, we first briefly review Kaup and Aach's approach based on our shape-projected subdomain formulation. Then, we apply their technique to constant-rate

and constant-quality compression. Some subtle issues imposed by quantization of transform coefficients are also addressed.

- **Perfect Reconstruction vs. Non-Perfect Reconstruction**

If a linear representation can reconstruct the original function without any error, it is a *perfect-reconstruction* (PR) representation. Otherwise, it is a *non-perfect-reconstruction* (non-PR) representation. If the subspace spanned by the representation basis functions can not contain the image segment vector, then the PR property cannot be achieved. As mentioned earlier, the shape-projected DCT basis functions $\{\hat{d}_i\}$ form a linearly dependent but complete set¹ of vectors in the shape-projected subspace. We should be able to choose m linearly independent vectors from the projected DCT vectors to form a basis in the subspace and achieve the PR property, where m is the dimension of the shape-projected subspace. The issue is to find the basis that can produce the best energy compaction.

Kaup and Aach used a successive approximation algorithm to iteratively project the image vector to each basis function and choose the basis function with the largest projection in magnitude in each iteration, i.e.,

$$Project(r(n), \hat{d}_{opt}) = \underset{i}{Max} (Project(r(n), \hat{d}_i)) \quad (5-5)$$

$$r(n+1) = r(n) - Project(r(n), \hat{d}_{opt}) \quad (5-6)$$

where $r(n)$ is the residual error in the n th iteration, $\{\hat{d}_i\}$ are the shape-projected transform basis functions (e.g., the shape-projected DCT basis functions), and \hat{d}_{opt} is the optimal basis function with the largest projection in each iteration, as illustrated in Figure 5-7. Note that the same basis function could be chosen repetitively since $\{\hat{d}_i\}$ are not orthogo-

1. A set of vectors is *complete* in a linear space if they can span the entire space.

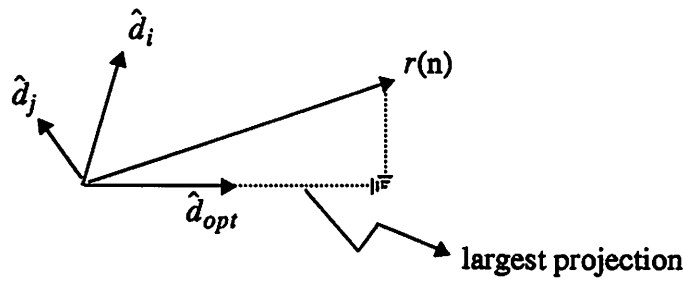


FIGURE 5-7 Kaup and Aach's successive approximation method find the transform basis function with the largest projection in each iteration.

nal. However, one problem is that the number of transform coefficients may exceed m (the image segment size) without achieving the PR property.

One way to achieve the PR property in this iterative algorithm is to accumulate the chosen basis functions in each iteration and project the image segment to the entire set of chosen basis function, rather than to a single basis function only. That is, the following operations are performed during each iteration.

$$D_i(n) = \text{Union}(D_{opt}(n-1), \hat{d}_i) \quad (5-7)$$

$$\text{Project}(r_0, D_{opt}(n)) = \text{Max}_i(\text{Project}(r_0, D_i(n))) \quad (5-8)$$

$$r(n) = r_0 - \text{Project}(r_0, D_{opt}(n)) \quad (5-9)$$

where *Union* is simply adding a vector to a vector set, that is, $\text{Union}(\{d_1, d_2, \dots, d_i\}, d_{i+1}) = \{d_1, d_2, \dots, d_{i+1}\}$, $D_{opt}(n)$ is the entire set of optimal basis functions accumulated from iteration 1 to n , $r(n)$ is the residual error after n iterations, and r_0 is the initial residual error (i.e., the original image segment). In each iteration, we keep the set of basis functions chosen from last iteration and add an additional basis function to minimize the residual error (i.e. maximizing the projection). The dimension of subspace spanned by the basis functions is incremented by one in each iteration¹. Note that in order to find the optimal basis function during each iteration, we need to project the image segment vector to a large

1. This is true until the residual error becomes zero.

number of possible set of basis functions (i.e., $D_i(n)$ of Equation 5-7), each of which requires solving a complete system of linear equations. This computational overhead is significant.

Another interpretation of the above PR iterative approximation algorithm is that during each iteration we not only add a new basis function, but also make the remaining unchosen basis functions and the residual error orthogonal to the chosen basis functions by projection, i.e.,

$$\hat{d}_{opt}(n) = \text{Max}_i(\text{Project}(r(n), \hat{d}_i(n))) \quad , \forall \hat{d}_i \notin D_{opt} \quad (5-10)$$

$$D_{opt}(n) = \text{Union}(D_{opt}(n-1), \hat{d}_{opt}(n)) \quad (5-11)$$

$$\hat{d}_i(n+1) = \hat{d}_i(n) - \text{Project}(\hat{d}_i(n), \hat{d}_{opt}(n)) \quad , \forall \hat{d}_i \notin D_{opt} \quad (5-12)$$

$$r(n+1) = r(n) - \text{Project}(r(n), \hat{d}_{opt}(n)) \quad (5-13)$$

where $\hat{d}_{opt}(n)$ is the new basis function added to the chosen set D_{opt} in iteration n . Essentially, we reduce the dimension of the residual error and remaining basis functions successively. During each iteration, all remaining unchosen vectors are made orthogonal to the chosen set by Equation 5-12. Therefore, the optimal basis function in each iteration is simply the one with the largest projection of the residual error, as shown in Equation 5-10. The complex process of iteratively solving a complete linear equation system in Equation 5-8 is avoided.

This approximation algorithm can achieve the PR property after m (the image segment size) steps, since only linearly independent basis functions are chosen. Also, the residual error decreases faster than the non-PR approximation method described above with some extra computational overhead.

- **Constant Rate vs. Constant Quality**

The above successive approximation algorithm successively increases the number of coefficients and reduces the residual error. As discussed, the residual error always decreases to zero after m steps for the PR approximation but not for the non-PR approximation. In practice, the number of coefficients used is determined by the available output transmission capacity of the encoder, the acceptable reconstructed image quality, and the affordable processing power of the hardware resources. Rate control can be easily achieved by limiting the number of generated coefficients. Quality control can be performed by measuring the final residual energy. Lastly, the computational complexity depends on the number of iterations performed. These controls are further complicated by quantization of the transform coefficients, which will be discussed in the following.

- **Quantization**

Transform coefficients are usually further quantized to increase the compression ratio. Small coefficients may be truncated to zero after quantization. Thus, after quantization, the proportionality between the recovered image quality and the number of iterations may become invalid. The reason is twofold. First, small coefficients obtained in later iterations are truncated to zero. They will not increase the recovered image quality level. Second, existing coefficients may be changed when new coefficients are added (particularly in the PR approximation technique). These changes may cause the quantized approximation more distant from the perfect representation and thus increase the prediction error. Figure 5-8 shows the peak signal-to-noise ratio (PSNR) of a simple image segment for each successive approximation. The PSNR after quantization begins to drop after four iterations. One way to avoid this problem is to integrate the quantization into the optimization process. Namely, change equation 5-8 to the following

$$Project(r_0, D_{opt}(n)) = \underset{i}{Max} (Quantz (Project (r_0, D_i(n)))) \quad (5-14)$$

In other words, we choose the basis function with the largest projection *after* quantization. This increases the computational complexity, but the recovered image quality, as shown in Figure 5-8, becomes monotonically non-decreasing and generally higher than that obtained from the original approach. Another way to avoid the quality decline due to over-iteration is to end iteration when quality after quantization starts to drop or reach a preset quality goal.

5.1.2.3 Adaptive Transform Bases

Intuitively, the spatial statistics of an AS image segment varies with its irregular shape. Thus, it requires different optimal transform bases. For example, image pixels of a single line may prefer a 1D DCT basis while a square image block may prefer a 2D DCT basis. In this section, we describe an approach which uses adaptive transform bases based on the shape information of the input image segment. As shown in Figure 5-9, the shape information is also available at the receiver, where the correct transform basis can be used to reconstruct the original image signal. It is again worth mentioning that we do not need to transmit the transform basis functions to the decoder.

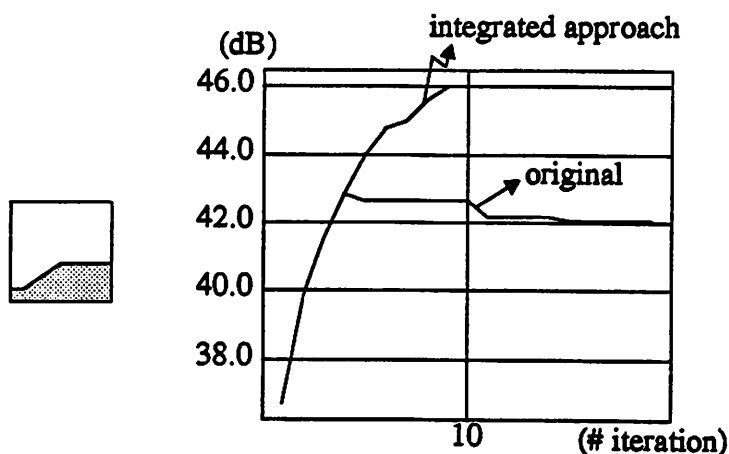


FIGURE 5-8 A simple image segment and its PSNR in each iteration of the PR successive approximation coding algorithm. The original method finds the minimal residual error *before* quantization, while the integrated method finds the minimal residual errors *after* quantization. We use uniform quantization here.

• Existing Orthogonal Transform Bases

Finding the transform coefficients is simplified if the basis functions form an orthogonal set, in which case the coefficients can be obtained by projection. (If the basis functions are not orthogonal to each other, then we need to solve a complete linear system.) Also, orthogonal basis functions usually perform better in separating the signal energy in the transform spectrum and thus improve the compression performance.

One way to construct an orthogonal transform basis is to reshape the arbitrarily-shaped image segment into a 1D array and apply the 1D DCT basis. The DCT is known to be close to the optimal Karhunen-Loeve Transform (KLT) if the image has high spatial correlation. However, except for single-line shapes, arbitrarily-shaped image segments usually do not have exact 1D spatial correlations. Furthermore, the dimension of the 1D DCT basis changes with the image segment size. This will also make the codec design complex.

Another way to construct orthogonal basis functions in the subspace S_B is to use the Gram-Schmidt algorithm, as proposed in [Gilge89]. The Gram-Schmidt algorithm can extract an orthogonal subset of functions out of a larger set of arbitrary functions. One possible initial set of functions for the Gram-Schmidt algorithm is the traditional 2D DCT

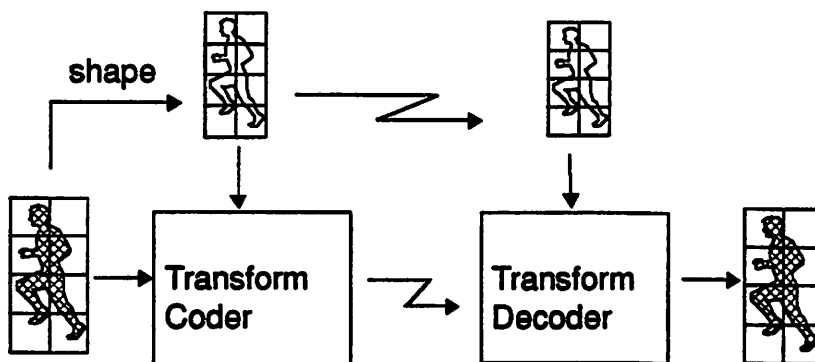


FIGURE 5-9 Use the shape information to assist in choosing the optimal transform basis. The shape information is also available at the receiver and thus the correct transform basis can be used to reconstruct the original image.

basis. Suppose the dimension of the full block is n and the dimension of subspace S_B is m ($m \leq n$). Let d_i 's represent the original DCT basis function and \hat{d}_i 's represent their projected version in the subspace S_B . It can be shown that $\dim(\text{span}\{d_i\})=n^1$, $\dim(\text{span}\{\hat{d}_i\})=m$, and $\{\hat{d}_i\}$ are linearly dependent if $m < n$.

Actually, in the Gram-Schmidt algorithm, we still have a great deal of flexibility in choosing different orthogonal subsets from a larger set of functions. In later simulations, we start from the DCT basis functions with the smallest zonal order². The final choice of orthogonal basis depends on the input image shape.

• New Orthogonal Transform — KLT-Like Transform

The KLT can be shown to be the best transform algorithm for the rectangular image segments if the spatial statistics of the input images are known. The DCT can be derived from the KLT if the image assumes a first-order Markovian model with high spatial correlation [Jain89]. We propose a new transform basis using this implication. Using the same assumption of a first-order Markovian model, we can find the variance-covariance matrix for an arbitrarily-shaped image segment. For example, if the image segment has m pixels, then we can rearrange the image segment, $P(x,y)$, to a 1D array of m elements, and define a $m \times m$ variance-covariance matrix, C , with

$$C_{ij} = (\lambda_1)^{|k-l|} \cdot (\lambda_2)^{|p-q|} \quad (5-15)$$

where λ_1 and λ_2 are the correlation coefficients in x and y direction, $P(k,p)$ is the i -th element in the 1D array, and $P(l,q)$ is the j -th element in the 1D array. Figure 5-10 shows an

-
1. $\dim(\text{span}\{d_i\})$ stands for the dimension of the vector space spanned by the vector set $\{d_i\}$.
 2. The *zonal order* is often used in coding the transform coefficients. The transform coefficients are transmitted in order of increasing spatial frequency (starting from the upper-left corner) [Netravali88].

example of a 4-pixel segment in a 4×4 image block. For simplicity, we assume that λ_1 equals λ_2 in later simulations.

Using a technique similar to that for deriving DCT from KLT, we can set the correlation coefficients λ_1 (λ_2) to a value close to unity (e.g. 0.9) and find the eigenvectors of the above variance-covariance matrix, C . Since the above variance-covariance matrix is real and symmetric, it has an orthonormal basis of eigenvectors if it is non-singular (i.e., λ_1 and λ_2 are not both 1) according to the real Schur decomposition theorem [Golub89]. Hopefully, these KLT-like transform bases can encode AS image segments as well as the DCT basis for the traditional rectangular image blocks. We will show the compression performance of this technique in the next section.

5.1.3 Performance Comparison

In this section, we use the irregular shaped image segment shown in Figure 5-2 (Miss USA) as the test case to simulate the performance of various transform coding schemes described in this paper. Only the partially defined boundary blocks (8 pixels × 8 pixels each) are used. As discussed earlier, it is difficult to have a quantitative measure of energy compactness of a transform spectrum. Instead, here we try to evaluate the rate-

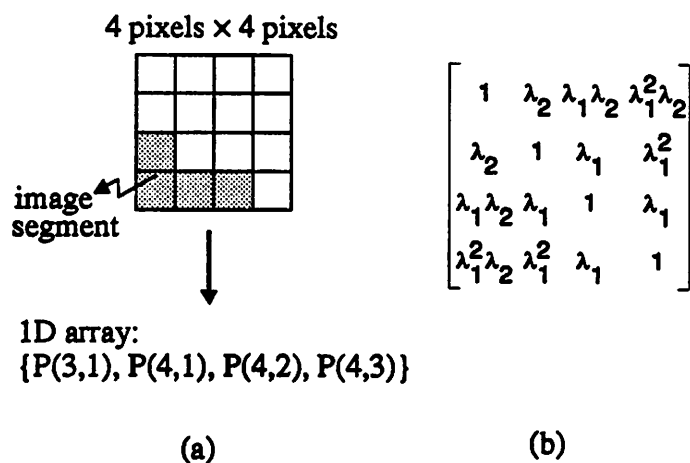


FIGURE 5-10 (a)Reshape the image segment into a 1D array and (b)construct its variance-covariance matrix based on the 1st-order Markovian model.

distortion performance of each transform scheme. The distortion is measured by the peak signal-to-noise ratio (PSNR) of the recovered image. The rate is represented by the compression ratio, i.e. the number of pixels inside the image segment divided by the number of non-zero transform coefficients after quantization. The results are shown in Figure 5-11. Note that uniform quantizers are used.

There are three different groups of coding schemes in Figure 5-11. The first group uses adaptive transform bases (section 5.1.2.3), including the 1D DCT, the proposed KLT-like transform basis, and DCT-based orthogonal transform bases proposed by Gilge *et al.* These algorithms change the transform basis when the image segment shape is changed. The transform bases are orthogonal and complete in the shape-projected subspace S_B . Therefore, the perfect reconstruction property is assured if the transform coefficients are not quantized. At the decoders, the adaptive transform basis can be recalculated in real time or pre-calculated and stored in memory in advance. However, for the latter case, the required memory storage could be quite large due to the wide variety of possible shapes.

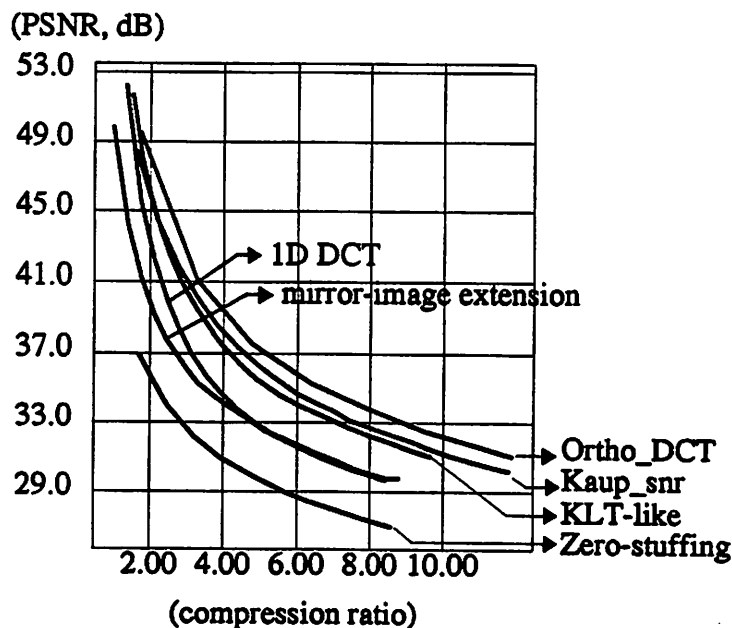


FIGURE 5-11 Rate/Distortion curves for various transform coding schemes for the image segment shown in Figure 5-2 by using uniform quantizers. (Kaup_snr represents Kaup & Aach's successive algorithm which iterates until the PSNR before quantization exceeds 50 dB.) The average PSNR is computed over the boundary image blocks only.

The second group of algorithms include modified versions of the successive approximation proposed by Kaup and Aach [Kaup92]. As discussed in section 5.1.2.2, the iteration process can be based on the output quality or rate constraints. For example, a quality-based scheme may iterate until the PSNR reaches 50 dB; a rate-based scheme may iterate until the number of transform coefficients exceeds 25% of the number of the original pixels. On average, for the same performance level, the quality-based schemes need fewer iterations than the rate-based schemes. The reason is that the quality-based schemes can adapt to the local activity of individual image blocks and spend more computations on busy image blocks than on flat ones. The overhead of this successive approximation algorithm is only in the encoders. Existing decoders can be used to reconstruct the image segment without any modifications. Note that this group of algorithms can achieve perfect reconstruction (PR) at some cost in extra computation, as discussed in 5.1.2.2. The PR iterative scheme usually has a slightly higher quality than the non-PR iterative schemes at the same compression rate.

The third group of coding algorithms directly extend the image segments into full image blocks and apply the traditional 2D DCT algorithm. Two results are shown in Figure 10 — zero-stuffing and mirror-image extension proposed in Section 5.1.1.1. After augmentation, the image segments are treated as the regular rectangular image blocks. No overhead is introduced while perfect reconstruction is assured.

From the R/D curves shown in Figure 10, we can see that the adaptive-basis schemes (the 1st group) and the iterative schemes (the 2nd group) outperform the most straightforward scheme (i.e., zero-stuffing) by a quality difference of 5-10 dB. The only exception is the 1D DCT, which suffers a lower performance (about 3-4 dB difference) at high compression ratios compared to other complicated schemes. This is a reasonable result since an arbitrarily-shaped 2D image segment usually does not have the spatial correlations similar to those found in a 1D image sequence.

In order to avoid severe computational overhead, we use the non-PR iterative scheme in our simulations. However, a large number of iteration (20 iterations in average) is still required for the iterative method to achieve the performance shown in Figure 5-11. During each iteration, the residual vector needs to be projected to 64 possible basis vectors. The computational overhead is still quite significant.

A satisfactory performance is observed for the proposed mirror-image extension method. It can achieve a 3-4 dB compression gain over the zero-stuffing method without any significant overhead.

Table 5-1 lists some major characteristics and compression performance for these coding algorithms. This comparison should be useful for system-level designs. If the processing resources are abundant, fancy algorithms like adaptive or iterative methods can be used to improve the reconstructed image quality. Otherwise, we can use simple mirror-image extension technique to achieve a fairly good reconstructed image quality. In addition, both adaptive and iterative algorithms require revision of the current codec hardwares, but the mirror-image extension technique is compatible with existing hardware.

Table 5-1 Characteristics of several transform coding algorithms for arbitrarily-shaped image segments.

	Transform Bases	Iterative Computations	Perfect Reconstruction (PR)	Compression Gain ^a
Orthogonal_DCT	adaptive, orthogonal ^b		Yes	6-12 dB
KLT-like	adaptive, orthogonal		Yes	5-10 dB
1D DCT	adaptive, orthogonal		Yes	2.7-7 dB
Kaup & Aach's iterative method	static	Yes ^c	Possible	5-10 dB
Mirror-image extension	static, orthogonal		Yes	2.7-4 dB
Zero-stuffing	static, orthogonal		Yes	—

a. This is the gain of the average PSNR (in comparison to the zero-stuffing methods) at the fixed compression rate. Note that the average PSNR is computed over the boundary image blocks only.

b. orthogonal with respect to the shape-projected subspace

c. 20 iterations for the R/D curve shown in Figure 5-11

5.2 Shape Representation

5.2.1 Review of Different Encoding Methods

For ASVO manipulation and compositing, the shape information is needed in addition to the image pixel values. In Chapter 2, we mentioned using the α channel to represent the irregular region of an ASVO. It is a pixel map data structure, with one α value for each pixel. Usually, the α value is equal to one for internal pixels, zero for external pixels, and between 0 and 1 for boundary pixels.

There are other possible data structures for representing irregular shapes, such as run length code, quadtree, and chain code (or so-called boundary code) [Foley90, Dyer90, Freeman74]. Each has different advantages and disadvantages. Samet wrote a good survey

paper on the quadtree and related hierarchical data structures [Samet84]. The run length code and the chain code both have the advantage of compact representations. The run length code is particularly useful for raster displays. The chain code, however, can carry local geometrical information along the boundary (such as turns, straight lines). Quadtree has been used in the hierarchical representation of graphics and colored images [Strobach91, Samet88]. It also provides an efficient structure for image processing such as searching, superposition, intersect, and geometrical transformation [Hunter79, Samet88].

In terms of space efficiency, we list the space complexity of different data structures in Table 5-2.¹ We use the 8-connectivity chain code; thus, each chain code requires 3 bits. The worst-case quadtree complexity is based on the bound theorem derived by Hunter [Hunter79]. We also use the rabbit image segment shown in Figure 2-5 as a test example to obtain some experimental figures, listed in the third column. The chain code has the lowest complexity, while the pixel map has the highest. The quadtree has some overhead due to internal nodes. But we can reduce the overhead to some extent by using pointerless representations, such as the linear quadtree and the DF-expression [Samet84].

1. Here we compare the complexity of binary pictures only, although these data structures can be extended to colored pictures or multi-valued α values.

Table 5-2 Comparisons of space complexity of different shape representations.^a

	complexity	example
Binary Map	1 bit per image pixel	1 bit/pixel
RLC	q bits per boundary pixel	0.127 bits/pixel
Quadtree	# of nodes $\leq 16q-11+16p$ ^b	0.092 bits/pixel ^c
Chain Code	3 bits per boundary pixel	0.048 bits/pixel

- a. Assume the AS image segment has an 2^q pixel by 2^q pixel size.
- b. The complexity of quadtree depends on the shape of the object and the choice of the reference point. q is defined as above; p is the perimeter of the object.
- c. Based on our simulations, the rabbit segment has about 3000 quadtree nodes. We use the DF-expression representation, in which each quadtree node requires two bits.

It is interesting that the complexity of quadtree depends on the reference position where image segment is embedded into a square area. Li *et al.* have studied this issue and proposed a normalized quadtree with respect to translation [Li82]. For example, in Figure 5-12, we show the relationship between the space complexity of an example AS image segment (the rabbit) and the choice of the reference point. We observe a variation of about 22% when the reference point shifts. The variation extent heavily depends on the AS image segment. For example, for a simple rectangle, the variation can be as high as 400%. Therefore, in order to optimize the quadtree complexity, it is worthwhile to try several different reference locations. In addition, as illustrated in Figure 5-12, the complexity shows an approximate periodicity (usually with period of approximate 2 or 4). Therefore, we can search possible positions in one period only, and the computational overhead should be insignificant.

Our objective is not to propose new data structures for AS image segments. Instead, we use the chain code described here to help in another important process — *anti-aliasing*.

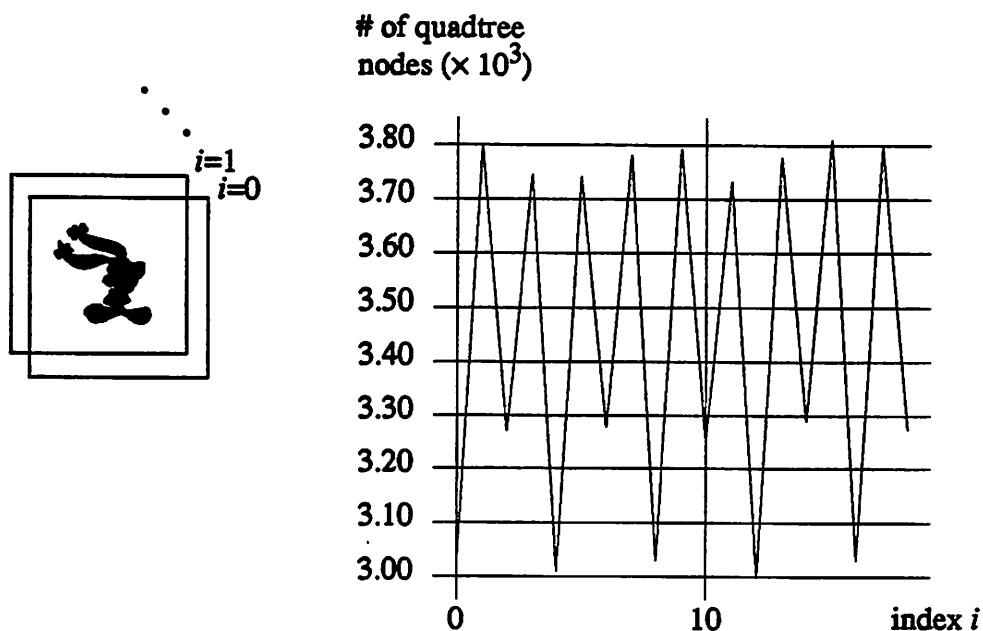


FIGURE 5-12 Changing the reference position (in the diagonal direction) of the rabbit image segment within the circumscribing square frame. The resulting quadtree complexity, i.e. the number of nodes, is shown on the right.

5.2.2 A Joint Approach to Shape Coding and Anti-Aliasing

In section 2.5, we described how to use the α channel to perform anti-aliasing along the object boundary to remove the “jagged” effect, but we did not explain how to produce the α values in the first place. For graphic objects like curves and polygons, their α values can be generated mathematically. However, for general irregularly-shaped image segments, the mathematical representation is not available. Given the binary pixel map representation of an AS image segment (0 for external pixels and 1 for internal pixels), how do we calculate the α values near the boundary in order to smooth the boundary appearance?

One straightforward approach is to blindly apply low pass filtering on the original binary α values of the AS image segment to produce new α values. The resulting α values are 1 inside the boundary, 0 outside the boundary, and between 0 and 1 near the boundary. The boundary is blurred to some extent depending on the tap length of the filter. However,

this approach does not consider the local geometry variation along the object boundary. Some types of object boundaries may not need anti-aliasing, such as straight horizontal or vertical boundaries. By blindly applying low pass filtering, we introduce unnecessary blurring effect along those boundaries.

One possible approach for rectifying this problem is to calculate the α values based on the chain code representation of the object shape. Intuitively, the chain code carries the local geometrical information about the object boundary. For example, the 8-connectivity chain code uses 3 bits to indicate the relative direction between the current boundary pixel and its next boundary pixel, as shown in Figure 5-13.

If we look at a fixed-sized window circumscribing each boundary pixel, there are a finite number of possible patterns of chain codes, and the corresponding local geometry, in the observed window. In Figure 5-14, a 3 pixel by 3 pixel window and all possible chain code patterns are illustrated. The center pixel is the current pixel, while the additional shaded pixel is the new pixel to be included as the new boundary pixel. We modify the α values for both the current pixels and the newly included boundary pixels to fractional numbers in order to smooth the boundary. In essence, we approximate the local boundary

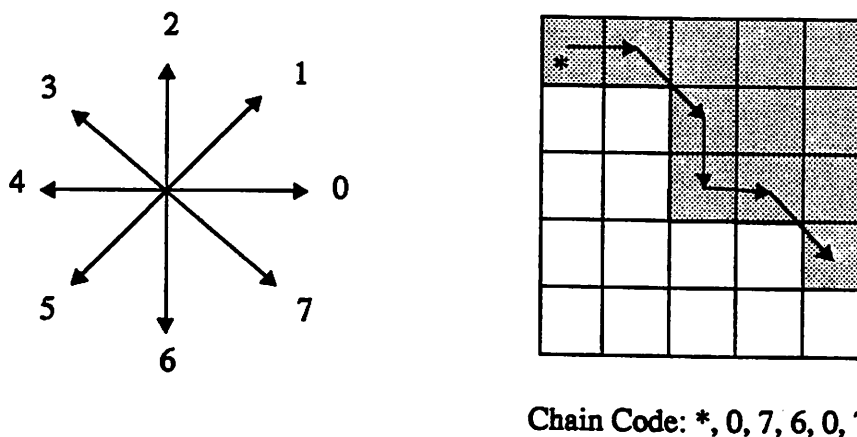


FIGURE 5-13 The 8-connectivity chain code uses 3 bits to represent the pointer from current boundary pixel to the next boundary pixel. The example shown on the right follows a counter-clockwise direction. * stands for the starting point.

geometry based on the chain code sequences within the observed window associated with the current pixel. The longer the chain code sequence we observe, the more accurate the boundary information we obtain. Examples shown in Figure 5-14 use one incoming chain code and one outgoing chain code respectively.

Figure 5-15 illustrates the procedure for generating the new α values based on the chain code pattern in the observed window. The procedure is pixel-wise iterated along the object boundary. In some cases, we need only modify the α value of the current boundary pixel; in other cases, we may need to add a new boundary pixel.

Since there are only a finite number of possible geometrical patterns, the table lookup method is suitable for generating the required α values. The table size depends on the size of the observed window (and therefore the length of the associated chain code

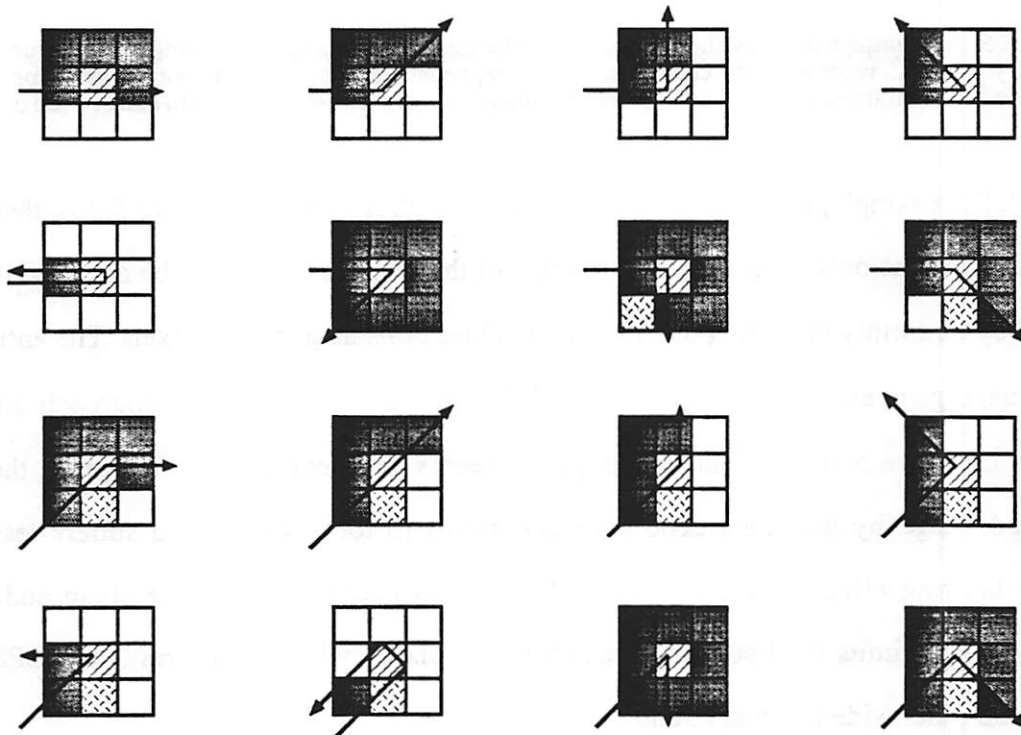


FIGURE 5-14 All possible chain code patterns for a 3 pixels by 3 pixels observed window centered at each boundary pixel. The center pixel represents the current boundary pixel. The chain code runs in the counter-clockwise direction, thus the interior side (the black area) is on the left side. The additional shaded pixel is included as new boundary pixels with fractional α values (from [Takahashi93]).

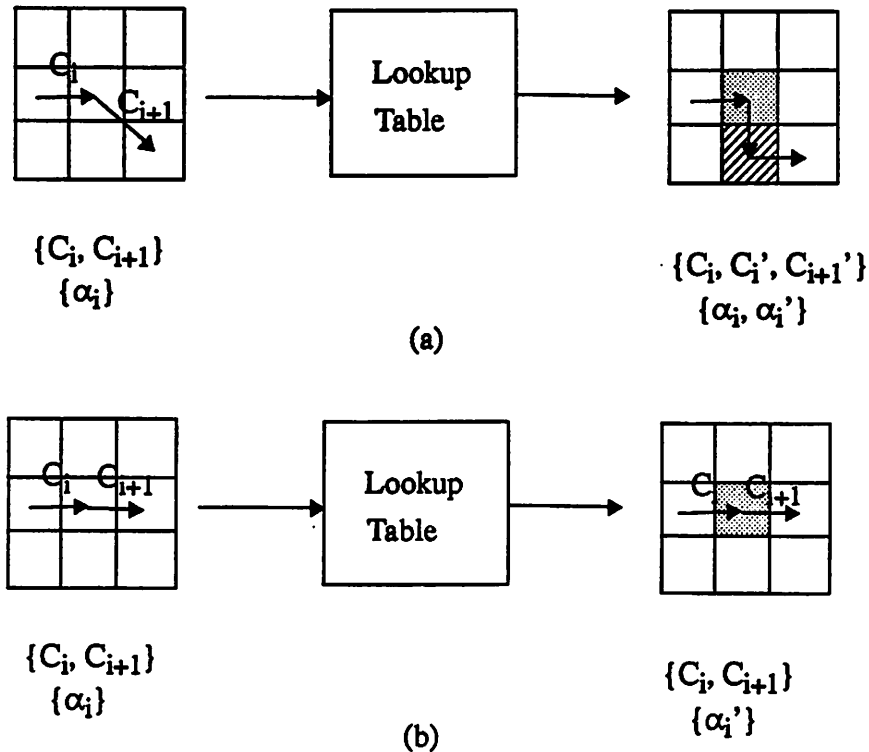


FIGURE 5-15 Examples showing the proposed table lookup technique for generating α values for boundary pixels. C_i represents the chain code and α_i represents the α value. The boundary shape is modified in (a), but not in (b). Shaded pixels highlight changes (either in the chain code or the α value).

sequence). For example, if the observed chain code length is two (as in Figure 5-15), then the lookup table has only 16 entries. The α value of the boundary pixel can be obtained in advance by uniformly averaging the original α values of its neighboring pixels. The anti-aliased image generated by this approach and that by the traditional LPF approach are compared in Figure 5-16. The subjective quality seems to be comparable. However, the anti-aliased image by the chain code approach seems to look sharper and suffers less boundary blurring effect than that by the LPF. This is mainly because the chain code approach always limits the boundary zone to 1 or 2 pixels, while a uniform 3×3 LPF produces a 3-pixel wide boundary zone.

The computational complexity for the chain code anti-aliasing method includes only one table lookup operation per boundary pixel. This has lower computational

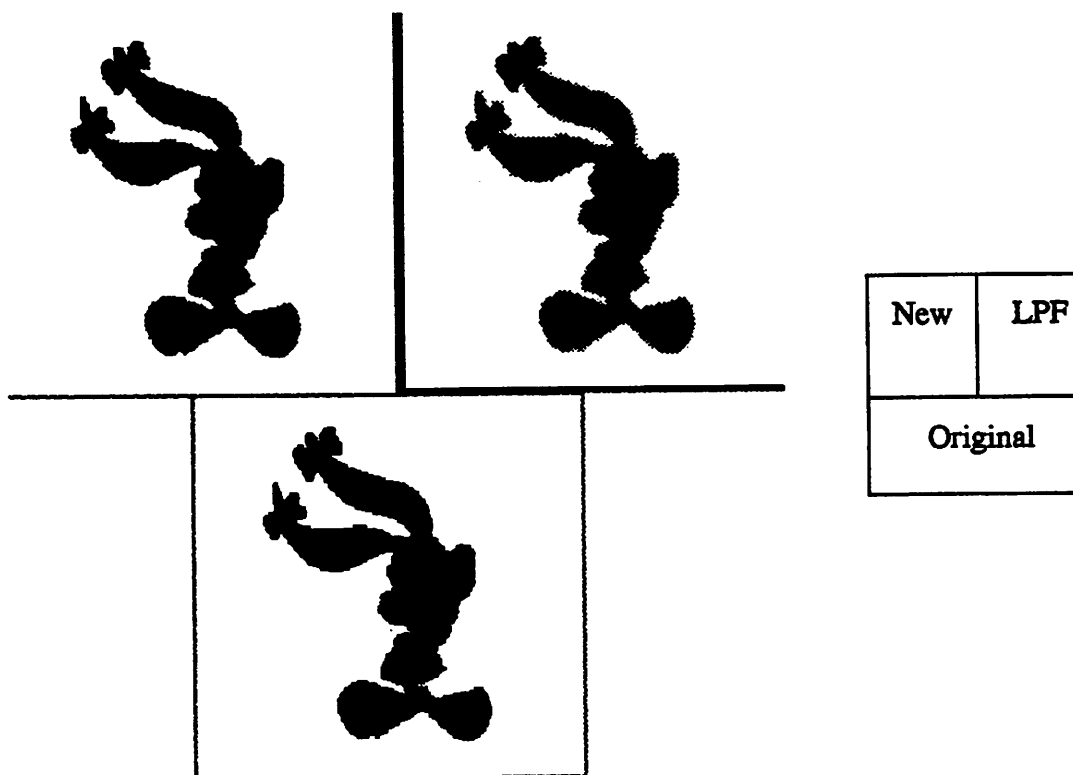


FIGURE 5-16 Example pictures showing the anti-aliasing effect by using the traditional approach (uniform low-pass filtering) vs. our proposed table lookup approach (combination of the use of the chain code and the α value).

complexity than the LPF (9 multiplications plus 8 additions per boundary pixel for a 3×3 uniform LPF). For a test image with a size of 256 pixels by 256 pixels, the chain code approach needs 0.62 seconds for anti-aliasing, compared to 8.87 seconds needed for the LPF. Also, by only storing those fractional α values associated with the boundary pixels, the chain code plus α channel method can achieve a compression rate of about 21:1, compared to the plain pixel map representation.

5.2.3 Iterative Transformations of Arbitrarily-Shaped Video Objects

In situations such as desktop multimedia editing and animation, ASVO's are "manipulated" before they are displayed. By "manipulation" we mean those image transformations such as geometrical transformations, linear filtering, and so on. One question arises when we jointly consider image transformation and anti-aliasing: how do

we treat the α values for anti-aliasing in image transformations? Note that here we focus on processing the α channel of ASVO's. We assume the image pixels are processed by traditional image transformation techniques [Foley90].

Figure 5-17 shows two separate approaches to combining anti-aliasing and image transformations. The first approach keeps the anti-aliased copy of the object for any future image transformations and display. The anti-aliased copy of the video object may be from the initial video capture (point A) or from the results of subsequent manipulations (point B). All image transformations are assumed to have the anti-aliasing function embedded, such as the box area averaging algorithm for scaling and translation [Weiman80]. The advantage of this approach is that the stored copies of AS objects, whether they are original or processed, have been anti-aliased and can be displayed directly without further processing. However, one shortcoming of this approach is that the object boundary becomes more blurred each time the object is processed. For example, Figure 5-18 shows the blurring effect after 10 consecutive subpixel translation operations.

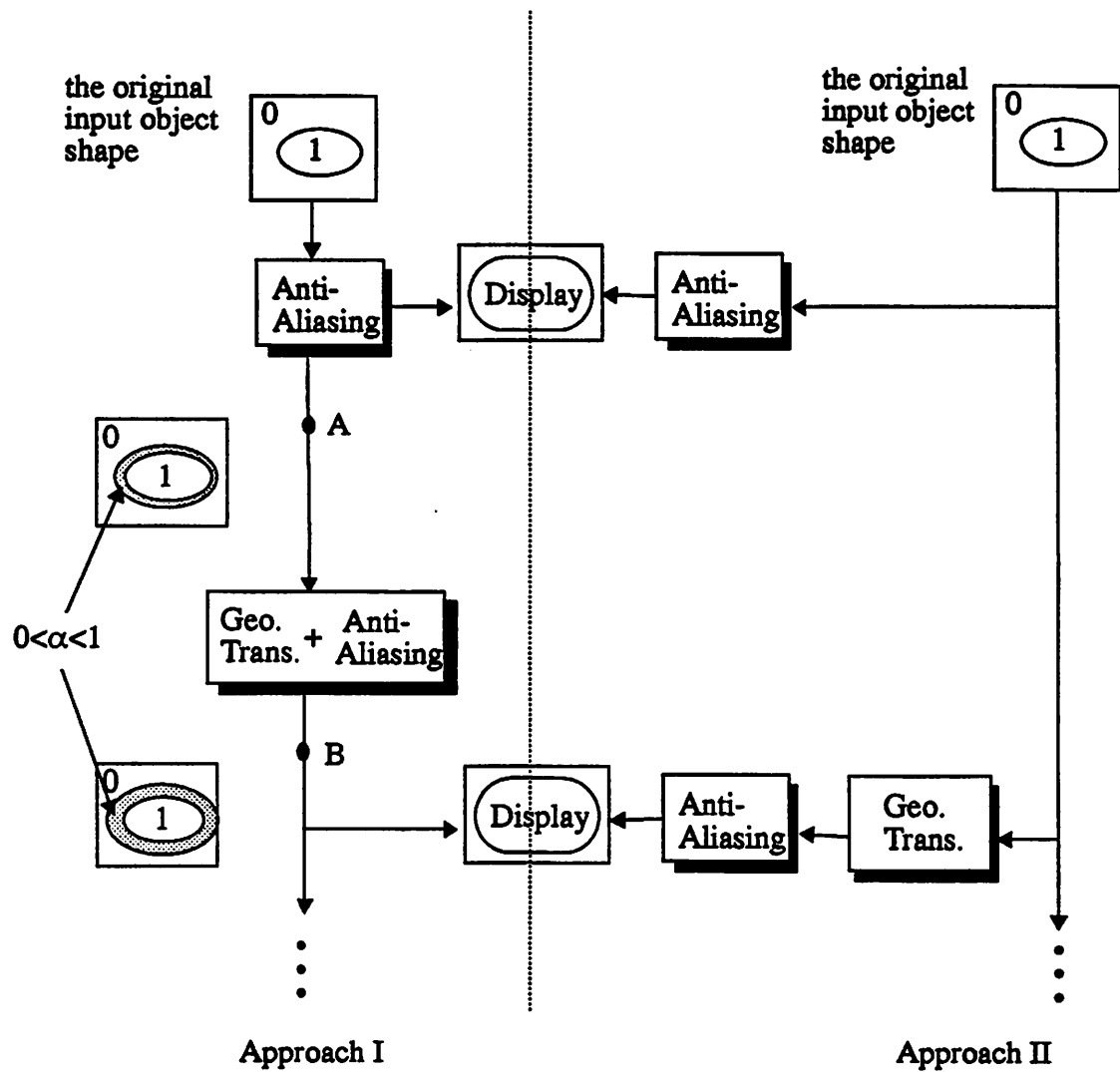


FIGURE 5-17 Two different approaches to combining image transformation and anti-aliasing in multimedia applications such as AS image object animation.

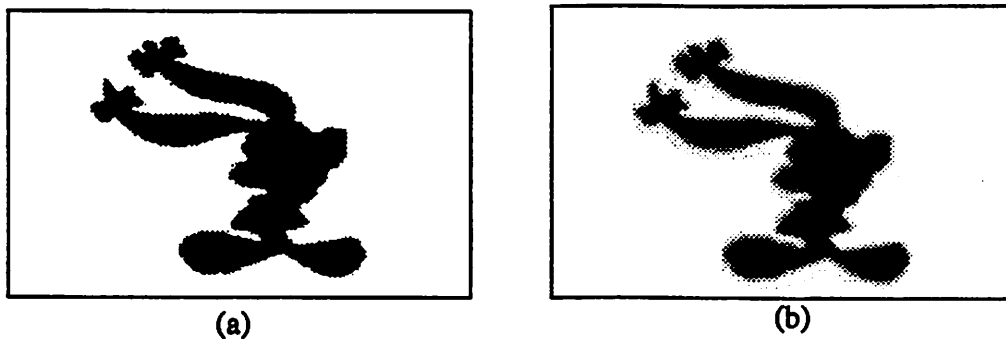


FIGURE 5-18 The blurring effect of iterative image transformations. (a) the original object (b) the result after 10 consecutive subpixel translations, by using Approach I in Figure 5-17. (from [Takahashi93])

As shown in Figure 5-17, another approach to combining image transformation and anti-aliasing is to use the original, “jagged”, copy of the image object as the source image for all future image transformations. The resulting image would need to be anti-aliased before being displayed. By avoiding accumulating the blurring effect in iterative geometrical transformations and anti-aliasing, the final displayed image should be much sharper but with jagged artifacts removed by the final-stage anti-aliasing. One immediate shortcoming of this approach is that the intermediate processed images are not stored. Any future operations need to use the original image object as the source image in order to avoid accumulating the blurring effect. This method can be inefficient in some cases like AS image animation (e.g. translate 1/2 pixel per iteration, or scale up by 1.1 times per iteration) since incremental implementations are not feasible. However, sometimes it may be possible to use a single composite operation to replace a sequence of image transformations. For example, the n -th frame in a 1.1 times-per-iteration up-scaling sequence can be achieved by a $(1.1)^n$ times up-scaling operation. In this case, there is no computational overhead.

Compositing multiple ASVO's based on the α values has been described in Chapter 2. If we use the first approach to manipulate each individual video object, the α

values are already available. In the second approach, because the stored video object is the original “jagged” copy, we need to perform the anti-aliasing process to obtain the α values before performing compositing. After compositing, the resulting composited video object is anti-aliased (i.e., blurred); there is no way to recover the exact boundary information of the original image object. In this case, we are forced to use the first approach, which incorporates anti-aliasing and geometrical transformations into the same iteration loop.

5.3 Summary

We have studied efficient coding schemes for arbitrarily-shaped (AS) image segments, which may appear in applications like multimedia workstations. We investigated coding schemes for both the image pixels and the shape of the AS image segments. For the image pixels, we considered only the intraframe transform coding, which can further be categorized into two different types — *brute-forth* and *shape-adaptive* transform coding. The latter uses the given shape information to improve the coding efficiency by changing the mapping process for calculating transform coefficients (e.g., Kaup and Aach’s successive approximation algorithm) or by changing the transform basis adaptively (e.g., Gilge’s orthogonal transform and our proposed KLT-like transform). Based on our simulations, these algorithms can improve the coding performance by 5-12 dB at the cost of extra computations or memory resources at the receiver.

In relation to the shape information, we propose a new joint approach to encode the shape with the chain code and also use the chain code to generate the α values required in anti-aliasing along the boundaries. This joint approach can improve space efficiency compared to the straightforward bitmap representation of the arbitrary shapes. Because the chain code can provide local geometrical information along the boundaries, it can generate “better” α values (e.g., without unnecessary boundary layers of α values) so that the subjective quality is improved.

In addition to coding of the shape and image pixels, we also discuss two approaches to handle iterative transformations and anti-aliasing of AS image segments.

Keeping the original copy of image segments can avoid accumulation of the blurring effect, but it may not be suitable for interactive applications. The alternative of using the anti-aliased copy in each iteration has the advantage that all intermediate results are displayable and reusable. However, the shortcoming is that the blurring effect could be accumulated. The suitable choice depends on the requirements of specific applications.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Video services on multimedia networks are emerging, as technology progresses in related fields, such as video compression, video transmission, video storage, video compositing/manipulation, and multimedia authoring. This dissertation focuses on video signal compositing and manipulation. Our goals are to thoroughly investigate various issues related to video compositing and manipulation and to provide efficient solutions for low-cost high-performance video compositing systems.

We recognized three different degrees of freedom in implementing video compositing: *feature*, *location*, and *data format*. We also observed strong interactions among these degrees of freedom. In many cases, we can compromise performance in one degree of freedom and gain great improvement in another. It is a non-trivial problem to find the optimal solution for video compositing for different specific applications. We need to explore the advantages of various approaches in each degree of freedom before optimizing the overall system performance.

In Chapter 2, we explored the first degree of freedom—*feature*. We studied the basic operations involved in various video compositing/manipulation functions. The compositing functions can range from simple operations like opaque overlapping to complicated ones like pixel-wise manipulations of arbitrarily-shaped video objects. In order to provide a flexible and efficient platform for representing multi-point distributed video compositing, we also proposed a *structured video* model, in which video objects are defined to represent logical or physical components in a video scene. Compositing functions are used to process each individual video object or to combine multiple video

objects into a single composited scene. Although the definition of compositing functions does not rely on the structured video model, we can represent video compositing functions in an efficient and hierarchical way based on the structured video concept. In particular, we described several possible data structures for representing video compositing functions: *time-varying scripts*, *expressions*, *ordered lists*, and *trees*. The first three structures are independent of final specific implementations and are on a higher abstract level than the last one. The tree structure, however, has closer correspondence with practical implementations.

The structured video model has both advantages and disadvantages. By keeping the video objects in a video scene logically separate along the path from the source to the final destination, various characteristics of user and service requirements can be satisfied better. Also, the structured video model has the potential for more efficient transmission and compression of video signals. However, by treating a video object as an inseparable unit, we must apply the same compositing functions on all pixels in each video object. If we want to apply different compositing functions to different parts of the same video object, we must further divide the video object into separate smaller objects.

In Chapter 3, we proposed the principle of *shared distributed network compositing*. Basically, this concerned with the second degree of freedom for video compositing: *location*. By distributively allocating the compositing hardware throughout the network, we can satisfy various demands of different users and services. For example, broadcasting programs can benefit from compositing at the source while interactive video services can benefit from compositing at the user site. By sharing the same hardwares for implementing the common compositing functions among different users and services, we can reduce the overall cost. In order to find sharable compositing functions and optimal mapping of compositing functions to network hardware resources, we also investigated some restructuring properties of compositing functions based on the representations of

compositing functions proposed in Chapter 2 (such as distributive, commutative, and associative properties of compositing functions). The approach of distributed network compositing also imposed impacts on ATM/BISDN network designs. For example, multi-point multi-connection calls need to be supported. Intelligent routing algorithms are required to ensure the advantages of hierarchical compositing and to reduce redundant traffics in multi-point connections.

The issue of choosing the optimal compositing location is further complicated by the fact that most video signals are compressed when transmitted over networks. Image quality loss could be accumulated in the repetition of compression processes when video signals are composited at the intermediate locations in the network. We quantified this quality loss by simulations in Chapter 3. Also, performing the same compositing function at different locations may result in different recovered image quality. For example, user-site compositing is of preference to other locations for compositing functions requiring low-pass filtering (such as down-scaling).

In Chapter 4, we studied the third degree of freedom: *data format*. Since most video signals are compressed when transmitted or stored, it motivate us to pursue efficient compressed-domain compositing techniques. Compressed-domain compositing provides the potential for a lower computational complexity and thus a lower hardware cost, because the data rate is usually much lower in the compressed domain than in the uncompressed domain and the conversion process between the uncompressed format and the compressed format can be avoided. We derived mathematical formulae for typical compositing functions in the transform domain (such as the DCT domain). We also analytically and numerically compared the computational complexity of the DCT-domain compositing approach and the uncompressed-domain compositing approach. The computational speedup of the DCT-domain approach vs. the uncompressed approach strongly depends on the compression ratio and specific compositing functions. In general,

block-wise operations benefits more than pixel-wise operations. Based on our non-real-time simulations, the computation reduction by using the DCT-domain approach ranges from 60% to 75% for some typical video conferencing scenes.

The lack of linearity and orthogonality in the MC algorithm prevents compositing in the MC-compressed domain. To overcome this obstacle in compositing widely-used MC-DCT compressed video signals, we propose a new decoding algorithm to partially decode the MC-DCT video to the DCT format. This allows us to apply the proposed DCT-domain compositing techniques. Again, the computational speedup strongly depends on the specific compositing functions and the compression characteristics, such as the compression ratio and the non-zero motion vector percentage. For typical video conferencing composited scenes, if block-wise operations are used, the proposed approach can reduce the computation by 10% - 25%.

Besides the above three degrees of freedom for video compositing, we also studied efficient coding schemes of arbitrarily-shaped (AS) image segments, which can be considered as two-dimensional samples of general three-dimensional video objects. In Chapter 5, we used two different problem formulations to derive the optimal transform coding of image pixels of the AS image segments: the *full-block* domain and the *shape-projected* domain. We proposed a mirror-image extension coding method in the full-block domain which could still apply the traditional DCT transform. In the shape-projected domain, we can easily derive existing iterative approximation methods and shape-adaptive transform coding methods in the literature. In addition, we derived a new KLT-like transform bases in the shape-projected domain. Based on our experiments on some AS image segments, the mirror-image extension technique can achieve fair compression performance (in the sense of the rate-distortion relation) without any excess computational overhead. Compared to the straightforward zero-stuffing technique, it can achieve a quality gain of about 3-4 dB. The iterative approximation techniques and the shape-

adaptive techniques can achieve higher performance gains (5-10 dB) at the cost of complicated iterative computations or calculations of new transform bases.

In Chapter 5, we were also concerned with shape coding, anti-aliasing, and iterative manipulations of AS image segments. Anti-aliasing is necessary for removing the “jagged” artifact on the object boundary. Iterative manipulations of AS image segments are encountered in applications such as image segment animation. We proposed a joint approach to shape representation and anti-aliasing. By utilizing the local geometrical information carried by the boundary code, we can efficiently generate the α values for the boundary pixels, which are in turn used to perform the anti-aliasing process. Compared to the traditional approach, which uses the low-pass filtering to produce α values, the proposed approach has advantages of lower computational complexity and sharper object boundaries. As for the iterative manipulations of AS image segments, we observed the tradeoff relationship between the image quality and the computational complexity. The blurring effect on image boundaries will become more and more serious after each iterative transformation on the same image segment. This problem can be solved if we always use the original copy of image segment as the source image in each iteration of image transformation (thus avoiding any accumulation of blurring effect in each iteration). However, in practical applications like animation, this approach could become quite inefficient because the intermediate results are not stored and additional computations are needed for obtaining the desired effect from the original image in each iteration.

In conclusion, we used a *systematic* approach in studying various aspects of network video compositing. By exploring advantages and disadvantages in different degrees of freedom for video compositing, we hope to provide a complete foundation based on which optimal video compositing techniques for each specific network video application can be achieved. Our work distinguishes itself from others in the literature by integrating the explorations in all different degrees of freedom, i.e. feature, location, and

data format, and also by looking at the interactions among these degrees of freedom and the interactions between video compositing/manipulation and other multimedia technologies, in particular video compression.

6.2 Future Work

Our study of network video compositing opens up a number of challenging topics. Some topics are direct extensions of our previous work, while others are based on the implications from our research results. We will discuss four issues in the following.

6.2.1 Optimal Resource Mappings for Distributed Compositing

We advocated the principle of shared distributed network compositing in Chapter 3 in the absence of a concrete algorithm for finding the optimal mapping from the abstract-level representations of video compositing functions to actual network resources. Given the dynamic requirements of users and services, how do we allocate the minimal amount of network resources (e.g. transmission bandwidth, compositing hardware, and video codecs) while completing as many compositing functions as possible? Given the network resource configuration, how do we design the optimal mapping to achieve the highest performance? Also, how do we implement new compositing requests to make the most out of the existing compositing hardware allocations without affecting the quality of existing services?

We have proposed abstract representations for video compositing functions and studied their restructuring properties for finding the optimal mapping. Some researchers have worked on the embodiments of concepts of *resource descriptors* and *resource synthesizers*, which are essential elements for the distributed implementation of compositing functions in heterogeneous network environments. However, there is still an important link missing: a concrete optimization algorithm which takes abstract-level

descriptions of compositing functions and generates optimal mapping to network resources.

Possible strategies are shown in Figure 3-5. We consider the user/service compositing requests, the cost weighting factors of different performance metrics, and the network resource configurations as given inputs. The objective is to find the optimal mappings of all compositing functions to network resources so that the overall cost is minimized. When compositing demands are low, all compositing requests can be completed with the highest performance level. When compositing demands exceed the capacity of the total network resources, compromises need to be made to minimize the overall cost. Sometimes, it might also be necessary to adopt some admission control mechanisms to guarantee a minimal quality level for each compositing request. However, there are some difficult tasks, such as quantification of each specific performance factor and finding the optimal mapping with the lowest cost function, involved in this approach. Hopefully, by having more prototyping experiences and developing some subjective principles, we can transform this problem into a solvable one. Also, the findings about the resulting optimal mapping for each given resource configuration can be fed back to adjust network resource deployment for further cost-performance optimization.

6.2.2 Video Compression and Compositing Co-Design

As we have found through this study, compression algorithms have significant impact on the compositing process. In Chapter 4, we showed that the transform-domain compositing approach is much more efficient than the uncompressed-domain compositing approach for many compositing functions. However, due to the fixed block structure used in the transform coding techniques (such as DCT), pixel-wise compositing functions may not be so efficient as block-wise operations in the compressed domain. We have also shown that, due to the lack of linearity and orthogonality in the MC algorithm, compositing in the MC domain seems to be impossible.

There are compression schemes which support convenient processing and manipulation of images. In Chapter 5, we mentioned that using the hierarchical coding methods (such as quadtree) for arbitrarily-shaped video objects allows us to use efficient processing algorithms (such as scaling, intersection, superposition) directly on the encoded data. In addition, it is known in the field of computer graphics that the Fourier descriptor [Floey90] used for coding arbitrary shapes has an invariant property under general geometrical transformations. However, these methods may not be effective for high-quality colored image coding.

These findings strongly motivate the pursuit of a joint approach to efficient compression and compositing. We may want to compromise some compression performance in order to provide greater flexibility in video compositing and manipulation in the compressed domain. For example, as shown in Chapter 5, if we modify the traditional MC algorithm to limit the percentage of the non-zero motion vector for the MC-DCT compressed video, the computational complexity of later DCT-domain compositing operations can be greatly reduced. In practice, the joint optimization of the compression algorithm and the compositing algorithm should be dependent on the required compression performance, the required video quality, and the desired compositing features.

6.2.3 Model-Based Video Coding

We have proposed a structured video model based on the assumption that all video scenes are composed of video objects, each of which represents a logical or physical object in the video signal. Our goal is to keep component video objects in a video scene logically separate to obtain the flexibility of matching video compositing to the widely-different characteristics of various users, services, and devices.

In the literatures of video compression, a closely related research field is the model-based video coding, also known as analysis-synthesis video coding. By assuming all video sources are associated with some software models for signal generation and display hardwares for image rendition, the analysis part extracts the parameters of the models from the source signals and transmits the necessary model parameters to the synthesis counterpart. The synthesis part uses the received parameters to reconstruct the original video signals based on the same models. In many cases, transmitting the model parameters is much more efficient than transmitting the encoded pixel data for the entire image frame. Therefore, this has strong potentials for very-low-bit-rate video coding applications such as the video telephony. However, obstacles exist in accurate real-time video signal analysis, i.e. source feature extraction. Also, if the video objects become too small, the overall compression advantage will be reduced because of the overheads associated with processing tiny video objects.

After source signal analyses, efficient video object coding techniques are needed. In Chapter 5, we have proposed several algorithms for coding two-dimensional arbitrarily-shaped image segments. It should be a promising approach to extend our techniques to three-dimensional video objects. This is a challenging topic which integrates interests in video/image processing, computer vision, and computer graphics.

6.2.4 Video Format Conversion

There have been a great number of different compression standards proposed for video, still image, and graphics. The choice of optimal compression technique strongly depends on the specific application. For example, intra-frame coding techniques such as JPEG (mainly DCT plus the variable length code) are used for still image compression. Hybrid interframe coding such as H.261 and MPEG are used for visual communications and multimedia applications. Lossless (or eventually lossless) coding techniques are used for graphics data compression.

In many cases, we need to convert video signals from one compressed format to another. An interesting example is video coding in heterogeneous communication networks. Different coding methods are preferred for different parts of the network. Service providers and end users may use different video codecs. Here, the challenging issues include: 1) efficient video format conversions, and 2) designing video compression schemes for compatibility. Our proposed decoding algorithm for converting the MC-DCT compressed video to the DCT format is an example of compression format conversion. It can be extended and applied to conversion from the MPEG format to the JPEG format.

Since heterogeneous compression formats are unavoidable, it would be a significant contribution to provide efficient conversion methods (or the so-called "video transcoding"). One principle may be useful for designing video compression algorithms for compatibility: there should be *strong coupling* between the compression schemes between which the conversions are made. For example, if we know we need to convert from the MC-DCT compressed format to the DCT format, we may want to modify the MC-DCT encoder so that the percentage of non-zero motion vector generated is moderate. On the other hand, if we need to convert the DCT-compressed video signals into the conditional replenishment format, we can use the energy difference of a few primary DCT coefficients of the same image block as the activity measurement instead of using the spatial distortion measurement. By using this strong coupling principle in designing compression algorithms for different parts in a heterogeneous application environment, we hope we can minimize the overheads of video transcoding and optimize the overall performance.

References

- [Adam93] John Adam, "Special Report on Interactive Multimedia — Applications, Implications," *IEEE Computer Magazine*, March 1993, pp. 24-31.
- [Ahmed74] N. Ahmed, T. Natarajan, and K.R. Rao, "Discrete Cosine Transform," *IEEE Trans. on Computers*, Vol. C-23, pp.90-93, Jan. 1974.
- [Ahuja92] Audhir R. Ahuja and J. R. Ensor, "Coordination and Control of Multimedia Conferencing," *IEEE Communications Magazine*, May, 1992, pp.38-43.
- [Bennet84] P.P. Bennet and S.A. Gabriel, "System for Spatially Transforming Images," U.S. Patent 4,472,732, Sep. 18, 1984.
- [Chang92a] S.-F. Chang and D.G. Messerschmitt, "Compositing Motion-compensated video within the Network," *IEEE 4th Workshop on Multimedia Communications*, Monterey, CA, April, 1992.
- [Chang92b] S.-F. Chang, W.-L. Chen and D. G. Messerschmitt, "Video Compositing in the DCT domain," *IEEE Workshop on Visual Signal Processing and Communications*, Raleigh, NC, pp. 138-143, Sep. 1992.
- [Chang93a] S.-F. Chang, W.-L. Chen, and D. G. Messerschmitt, "Method and Apparatus for Compositing Compressed Video Data", U.S. Patent Pending, March, 1993.
- [Chang93b] S.-F. Chang and D. G. Messerschmitt, "A New Approach to Decoding and Compositing Motion Compensated DCT-Based Images," *ICASSP '93*, Minneapolis, Minnesota, pp. V421-V424, April, 1993.
- [Chang93c] S.-F. Chang and D. G. Messerschmitt, "Transform Coding of Arbitrarily-Shaped Image Segments," to appear on *ACM 1st Conference on Multimedia*, Anaheim, CA, Aug. 1993.
- [Chen76] W.-H. Chen and S.C. Fralick, "Image Enhancement Using Cosine Transform Filtering," *Image Sci. Math. Symp.*, Monterey, CA, Nov. 1976.
- [Chen77] W.-H. Chen, C.H. Smith, and S.C. Fralick, "A Fast Algorithm for the Discrete Cosine Transform," *IEEE Trans. on Communications*, Vol. COM-25, No. 9, Sep. 1977, pp. 1004-9.

- [Chen93a] W.-L. Chen, S.-F. Chang, P. Haskell, and D. G. Messerschmitt, "Structured Video Model for Interactive Multimedia Video," in preparation, 1993.
- [Chen93b] W.-L. Chen, P. Haskell, L. Yun, and D. G. Messerschmitt, "Videostation: a Hardware Implementation of Structured Video," in preparation, 1993.
- [Chen93c] W.-L. Chen, P. Haskell, S.-F. Chang, L. Yun, and D.G. Messerschmitt, "VideoStation — Architecture Supporting Multi-Element Compositing Video Displays," U.S. Patent application in progress, 1993.
- [Chiprasert90] B. Chiprasert and K.R. Rao, "Discrete Cosine Transform Filtering," *Signal Processing*, Vol. 19, No. 3, pp. 233-45, Mar. 1990.
- [Clarke85] R.J. Clarke, "Transform Coding of Images," Academic Press, 1985.
- [Cole93] Bernard Cole, "Special Report on Interactive Multimedia — The Technology Framework," *IEEE Computer Magazine*, March 1993, pp. 32-39.
- [Duff85] T. Duff, "Compositing 3-D Rendered Images," *Siggraph*, vol. 19, November 1985, pp. 41-44.
- [Dyre90] Charles R. Dyer, Azriel Rosenfeld, and Hanan Samet, "Region Representation: Boundary Codes from Quadrees," *Communications of the ACM*, Vol. 23, No. 3, pp. 171-9, March 1990.
- [Ferrari90] Domenico Ferrari and Dinesh C. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE JSAC* Vol.8, No.3, April 1990, pp.368-379.
- [Ferrari92] Domenico Ferrari, Anindo Banerjea, and Hui Zhang, "Network Support for Multimedia — A Discussion of the Tenet Approach," Technical Report, TR-92-072, Nov. 1992, International Computer Science Institute, Berkeley, CA 94720.
- [Foley90] J.D. Foley, A. v. Dam, S. K. Feiner and J.F. Hughes, *Computer Graphics: Principles and Practice*, 2nd ed., Addison-Wesley, 1990.
- [Freeman74] H. Freeman, "Computer Processing of Line Drawing Images," *Comput. Surveys*, Vol.6, No.1, 1974, pp.57-97.

- [Gilge89] M. Gilge, T. Engelhardt, and R. Mehlan, "Coding of Arbitrarily Shaped Image Segments Based on A Generalized Orthogonal Transform," *Signal Processing: Image Communication* 1, 1989, pp. 153-180.
- [Golub89] Gene H. Golub and Charles F. Van Loan, *Matrix Computations*, 2nd edition, the John Hopkins University Press, 1989.
- [H26190] CCITT Recommendation H.261, "Video Codec for Audiovisual Services at px64 kbits/s", 1990.
- [Hotter90] M. Hotter, "Object-Oriented Analysis-Synthesis Coding Based on Moving Two-Dimensional Objects," *Signal Processing: Image Communication* 2, 1990, pp.409-428.
- [Hunter79] Gregory M. Hunter, Kenneth Steiglitz, "Operations on Images Using Quad Trees," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-1, No.2, April 1979, pp.145-153.
- [Jain81] J.R. Jain and A.K. Jain, "Displacement Measurement and Its Application in Interframe Image Coding," *IEEE Trans. on Communications*, Vol. COM-29 (12), pp.1799-1808, Dec. 1981.
- [Jain89] A. Jain, "Fundamentals of Digital Image Processing," Prentice-Hall Inc., 1989.
- [Jayant84] N.S. Jayant and Peter Noll, *Digital Coding of Waveforms*, Prentice-Hall, 1984.
- [Jayant92] Nikil Jayant, "Signal Compression: Technology Targets and Research Directions," *IEEE JSAC*, Vol. 10, No. 5, June, 1992, pp. 796-818.
- [JPEG91] Standard Draft, JPEG-9-R7, Feb. 1991
- [Jurgen91] Ronald K. Jurgen, "The Challenges of Digital HDTV," *IEEE Spectrum*, April, 1991, pp.28-30.
- [Karlsson89a] Gunnar Karlsson and Martin Vetterli, "Packet Video and Its Integration into the Network Architecture," *IEEE JSAC*, Vol. 7, No. 5, June 1989, pp.739-751.

- [Karlsson89b] Gunnar Karlsson and Martin Vetterli, "Extension of Finite Length Signals for Sub-band Coding," *Signal Processing*, Vol. 17, No.2, pp.161-168, June, 1989.
- [Katz90] R.H. Katz, G.A. Gibson, and D.A. Patterson, "Disk System Architectures for High Performance Computing," *IEEE Proceedings*, Jan. 1990.
- [Kauffmann88] H.E. Kauffmann, "User's Guide to the Composer," Computer Graphics Group Documentation, Brown University, Providence, RI, May, 1988.
- [Kaup92] A. Kaup and T. Aach, "A New Approach Towards Description of Arbitrarily Shaped Image Segments," *IEEE International Workshop on Intelligent Signal Processing and Communication Systems*, Taipei, Taiwan, March, 1992.
- [Kou91] W. Kou and T. Fjallbrant, "A Direct Computation of DCT Coefficients for a Signal Block from Two Adjacent Blocks," *IEEE Trans. on Signal Processing*, Vol. 39, No. 7, pp. 1692-5, July 1991.
- [Kretz92] Francis Kretz and F. Colaitis, "Standardizing Hypermedia Information Objects," *IEEE Communications Magazine*, May 1992, pp. 60-70.
- [Kunt85] M. Kunt, A. Ikonmopoulos, and M. Kocher, "Second-Generation Image Coding Techniques," *Proceedings of IEEE*, Vol. 73, pp. 549-574, Apr. 1985.
- [Kunt87] Murat Kunt, Michel Benard, and Riccardo Leonardi, "Recent Results in High-Compression Image Coding," *IEEE Trans. on Circuits and Systems*, Vol. CAS-34, No. 11, Nov. 1987, pp. 1306-1336.
- [Lantz84] K. Lantz and W. Nowicki, "Structured Graphics for Distributed Systems," *ACM Transactions on Graphics*, vol.3, #1, January, 1984, pp. 23-51.
- [LeeB84] B.G. Lee, "FCT - A Fast Cosine Transform," *IEEE ICASSP'84*, San Diego, March, 1984, pp.28A.3.1-4.
- [LeeJ92] J.B. Lee and B.G. Lee, "Transform Domain Filtering Based on Pipelining Structure," *IEEE Trans. on Signal Processing*, pp. 2061-4, Vol. 40, No. 8, Aug. 1992.

- [LeeY92] Y.Y. Lee and J.W. Woods, "Video Production with Compressed Images," submitted to the SMPTE journal, 1992.
- [Le Gall91] D. Le Gall, "MPEG: A Video Compression Standard for Multimedia Applications," *Communications of the ACM*, Vol. 34, No.4, April, 1991.
- [Li82] M. Li, W.I. Grosky, and R. Jain, "Normalized Quadrees with Respect to Translations," *Comput. Gr. Image Process.* 20, 1 (Sep.) 72-81.
- [Lin91] H.-D. Lin and D.G. Messerschmitt, "Video Compositing Methods and Their Semantics," *IEEE ICASSP '91*, Ontario, Canada, 1991.
- [Little90a] Thomas D.C. Little and Arif Ghafoor, "Synchronization and Storage Models for Multimedia Objects", *IEEE JSAC*, Vol.8 No.3, pp. 413-427, April 1990.
- [Little90b] T.D.C. Little and A. Ghafoor, "Network Considerations for Distributed Multimedia Object Composition and Communication," *IEEE Network*, Vol. 4, No. 6, Nov. 1990, pp.32-49.
- [Little91a] T.D. Little and A. Ghafoor, "Spatio-Temporal Composition of Distributed Multimedia Objects for Value-Added Networks," *IEEE Computer Mag.*, pp. 42-50, Oct. 1991.
- [Little91b] T.D.C. Little and A. Ghafoor, "Multimedia Synchronization Protocols for Broadband Integrated Services," *IEEE JSAC*, Dec. 1991?
- [Liou91] Ming Liou, "Overview of the p×64 kbit/s Video Coding Standard," *Communications of the ACM*, Vol. 34, No. 4, April 1991.
- [Lukacs92] M. Lukacs, "An Advanced Digital Network Video Bridge for Multipoint with Individual Customer Control," Private Communication, Bell Communications Research, NJ, May 1992.
- [Markey92] Brian D. Markey, "HyTime and MHEG," *IEEE COMPCON*, pp. 25-40, San Francisco, CA, Feb. 1992.
- [Martin91] R.R. Maritn, "Quadrees, Transforms and Image Coding," North-Holland, *Computer Graphics Forum 10* (1991) pp.91-96.
- [McLean92] Patrick McLean, "What's in a Picture? A Structured Approach to Video Coding," *Intl. Symp. on Signals, Systems, and Electronics*, Sept. 1992.

- [Minzer89] Steven Minzer, "Broadband ISDN and Asynchronous Transfer Mode (ATM)," *IEEE Communications Magazine*, Sep. 1989, pp.17-57.
- [MPEG90] Standard Draft, MPEG Video Committee Draft, MPEG 90/ 176 Rev. 2, Dec. 1990.
- [Musmann89] H.G. Musmann, M. Hotter, and J. Ostermann, "Object-Oriented Analysis-Synthesis Coding of Moving Images," *Signal Processing: Image Communication*, pp. 117-138, 1989.
- [Narasimha78] M. Narasimha and A. Peterson, "On the Computation of the Discrete Cosine Transform," *IEEE Trans. on Communications*, Vol. COM-26, No. 6, June 1978, pp.934-6.
- [Netravali79] A.N. Netravali and J.D. Robbins, "Motion-Compensated Television Coding: Part I," *The Bell System Technical Journal*, Vol. 58(3), pp.631-670, Mar. 1979.
- [Netravali88] A.N. Netravali and B.G. Haskell, *Digital Pictures: Representation and Compression*, New York, Plenum Press, 1988.
- [Ngan80] K.N. Ngan and R. J. Clarks, "Lowpass Filtering in the Cosine Transform Domain," *Proc. Intern. Conference on Communications*, pp.31.7.1-31.7.5, Seattle, WA, June 1980.
- [Ngan86] K.N. Ngan, "Two-Dimensional Transform Domain Decimation Techniques," *ICASSP 86, Intern. Confer. on Acoustics, Speech, and Signal Processing*, pp. 1001-1004, Tokyo, Japan, April 1986.
- [Nicolaou90] Cosmos Nicolaou, "An Architecture for Real-Time Multimedia Communication Systems", *IEEE JSAC*, Vol.8, No.3, pp. 391-400, April, 1990.
- [Pasquale93] Joseph Pasquale, George Polyzos, and Vachaspathi Kompella, "Real-Time Dissemination of Continuous Media in Packet-Switched Networks," *IEEE COMPCON*, Feb. 1993, San Francisco, CA.
- [Pavlidis74] T. Pavlidis, "Techniques for Optimal Compaction of Pictures and Maps," *Computer Graphics and Image Processing*, 3, 1974, pp.215-224.
- [Porter84] T. Porter and T. Duff, "Compositing Digital Images," *Computer Graphics*, Vol. 18, pp. 253-259, 1984.

- [Prycker91] Martin de Prycker, "Asynchronous Transfer Mode: Solution for Broadband ISDN," Ellis Horwood Series in Computer Communications and Networking, 1991.
- [Rangan93] P.V. Rangan, H.M. Vin, and S. Ramanathan, "Communication Architectures and Algorithms for Media Mixing in Multimedia Conferences," IEEE/ACM Transactions on Networking, Vol.1, No.1, Feb., 1993.
- [Ripley89] David Ripley, "DVI — Digital Multimedia Technology", Communications of the ACM, Vol. 32, No. 7, pp. 811-822, July 1989.
- [Rosenberg92] Jonathan Rosenberg, Robert K. Kraut, Louis Gomez, and C. Alan Buzard, "Multimedia Communications For Users," IEEE Communications Magazine, May 1992, pp. 20-36.
- [Samet84] Hanan Samet, "The Quadtree and Related Hierarchical Data Structures," Computing Surveys, Vol.16, No.2, June 1984.
- [Samet88] Hanan Samet and Robert E. Webber, "Hierarchical Data Structures and Algorithms for Computer Graphics Part I: Fundamentals," IEEE Computer Graphics and Applications, 1988, pp. 48-68.
- [Schooler92] Eve M. Schooler and Stephen L. Casner, "An Architecture for Multimedia Connection Management," IEEE International Workshop on Multimedia, Monterey, CA, April 1992.
- [Shibata92] Masahiro Shibata, "Proposal for Desk Top Program Production," Proc. 134th SMPTE Tech. Conf., Nov. 1992, Toronto, Canada.
- [Smith87] A.R. Smith, "Planar 2-pass Texture-Mapping and Warping," SIGGRAPH 87, pp. 263-272.
- [Smith92] B.C. Smith and L. Rowe, "A New Family of Algorithms for Manipulating Compressed Images," to appear on IEEE Computer Graphics and Applications, 1993.
- [Soltanian-Zadeh93] Hamid Soltanian-Zadeh and Andrew E. Yagle, "Fast Algorithms for Extrapolation of Discrete Band-Limited Signals," IEEE ICASSP, pp.591-4, April, 1993. Minneapolis, MN.

- [Strobach91] Peter Strobach, "Quadtree-Structured Recursive Plane Decomposition Coding of Images," *IEEE Trans. on Signal Processing*, Vol. 39, No.6, June 1991, pp. 1380-97.
- [Takahashi93] Masahiko Takahashi, S.-F. Chang, and D.G. Messerschmitt, "Joint Shape Representation and Anti-Aliasing for Arbitrarily-Shaped Image Objects," to appear on *IEEE Intern. Workshop on Intelligent Signal Processing and Communication Systems*, Oct. 1993, Sendai, Japan.
- [Tamminen84] Markku Tamminen, "Encoding Pixel Trees," *Computer Vision, Graphics, and Image Processing*, 28, pp.44-57, 1984.
- [Wallace91] G.K. Wallace, "The JPEG Still Picture Compression Standard," *Communications of the ACM*, Vol. 34, No. 4, April 1991.
- [Weiman80] C.F.R. Weiman, "Continuous Anti-Aliased Rotation and Zoom of Raster Images," *SIGGRAPH 80*, 286-293.
- [Wolberg90] G. Wolberg, *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos, CA, 1990.
- [Yagi93] N. Yagi *et al.*, "A Programmable Video Signal Multi-Processor for HDTV Signals," submitted to *IEEE ASAP Workshop 1993*.

Appendix

A.1 Multiplication-Convolution Theorem for the DCT

We derive the multiplication-convolution relationship shown in equation (A-4-7) in this section. First, we start from the one-dimensional case. Suppose $a(n)$ is a signal vector. We extend $a(n)$ to form a new symmetric vector, $s(n)$, and take its $2N$ -point DFT as follows,

$$s(n) = \begin{cases} a(n) & n = 0, \dots, N-1 \\ a(-1-n) & n = -N, \dots, -1 \end{cases}$$

$$S_F(k) = DFT_{2N}\{s(n)\} = \frac{1}{\sqrt{2N}} \sum_{n=-N}^{N-1} s(n) e^{-j\frac{2\pi kn}{N}} \quad k = -N, \dots, N-1$$

(A-1)

The DCT of vector $a(n)$ is defined as,

$$A_c(k) = \sqrt{\frac{2}{N}} C_k \sum_{n=0}^{N-1} a(n) \cos\left(\frac{\pi k(2n+1)}{2N}\right)$$

where $C(k) = \begin{cases} \frac{1}{\sqrt{2}} & k = 0 \\ 1 & k = 1, \dots, N-1 \end{cases}$

(A-2)

It is known that the DCT of a vector is related to its DFT as follows,

$$\hat{A}_c(k) = \exp\left(\frac{-jk\pi}{2N}\right) S_F(k) \quad k = -N, \dots, N-1$$

where $\hat{A}_c(k) = \begin{cases} A_c(k)/C(k) & k = 0, \dots, N-1 \\ A_c(-k)/C(-k) & k = -N+1, \dots, -1 \\ 0 & k = -N \end{cases}$

(A-3)

By using the multiplication-convolution relationship for the DFT, we can easily derive a similar relationship for the DCT,

$$\text{If } y(n) = a(n) \cdot h(n), \quad n = 0, \dots, N-1$$

$$\text{then } \hat{Y}_c(k) = \frac{1}{\sqrt{2N}} \sum_{m=-N}^{N-1} [\hat{A}_c(m) \cdot \hat{H}_c((k-m)_{2N})] \cdot \alpha(k-m)$$

$$k = -N, \dots, N-1$$

$$\text{where } \alpha(i) = \begin{cases} 1 & i \in [-N, N-1] \\ -1 & \text{otherwise} \end{cases}$$

(A-4)

where “ $((n))_{2N}$ ” denotes “ n modulo $2N$ ”. If the $\alpha(k-m)$ variable in the above equation is ignored, the operation is equivalent to a $2N$ -point circular convolution. The reverse relationship, the convolution-multiplication relationship, can be found in [Chiprasert90]. We can easily extend the above 1D results to the 2D case as shown in equation (A-4-7). Note that the circular convolution is of the order of $2N$ points here.

A.2 Transform Domain Filtering

Linear filtering of images can be performed in both the spatial domain and the transform domain [Chang92b, Lee92, Chiprasert90, Ngan80]. For simplicity, we describe the one dimensional (1D) case first. If $x[n]$ represents the 1D input image sequence and $h[n]$ represents the finite-length filter coefficient vector, then the output filtered image sequence, $y[n]$, can be calculated as a linear convolution of $h[n]$ and $x[n]$. That is,

$$y[n] = h[n] \otimes x[n] \quad (\text{A-5})$$

Figure A-1 uses a matrix-vector multiplication to describe the convolution operation. Each element of the output vector, $y[n]$, is the inner product of the filter coefficient vector and a corresponding segment of $x[n]$. If the image sequence is segmented into N -element segments, for example x^k representing the k th N -element segment in the original image sequence, then the above equation can be described as

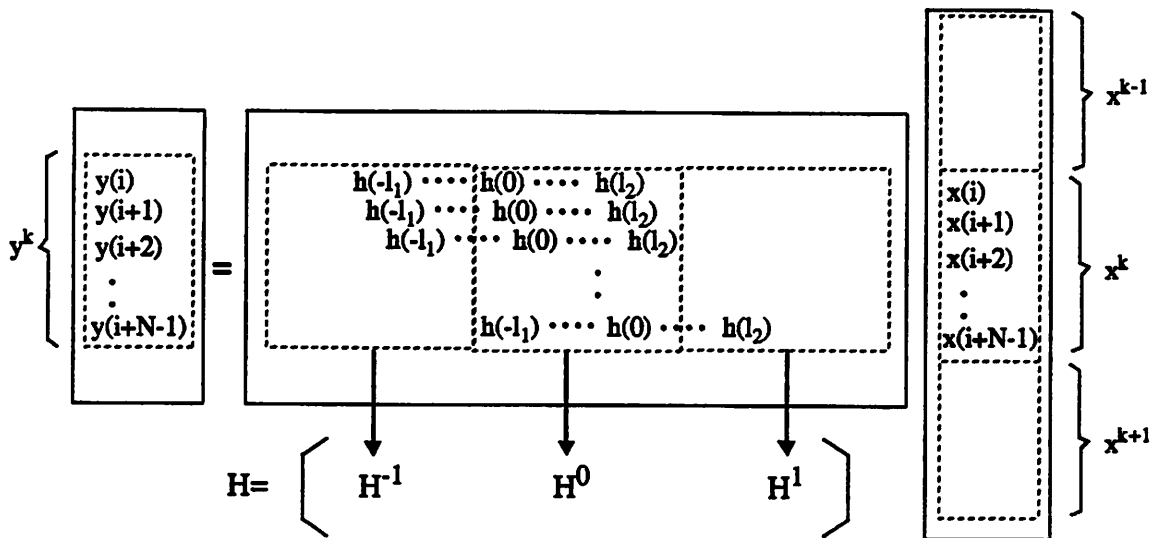


Figure A-1 Using matrix-vector multiplications to describe the convolution operation.

$$y^k = \sum_{j=-s_1}^{s_2-1} H^j x^{k+j} \tag{A-6}$$

where $s_1 = \left\lceil \frac{l_1}{N} \right\rceil$, $s_2 = \left\lceil \frac{l_2}{N} \right\rceil$, and H^j are $N \times N$ matrices, as shown in Figure A-1.

If we concatenate all H^j matrices side by side to form a new matrix, H , then matrix H can be defined as

$$H(i,j) = h(j-s_1 \cdot N-i), \quad \text{if } -l_1 \leq j-s_1 \cdot N-i \leq l_2 \\ = 0, \text{ otherwise} \tag{A-7}$$

Taking the transform operation on both sides of the above equation and inserting an inverse transform/transform pair in front of the input segments, we can easily obtain,

$$T_1 y^k = \sum_{j=-s_1}^{s_2-1} T_1 H^j T_2^{-1} T_2 x^{k+j} \tag{A-8}$$

If we use P^j to represent the matrix product $T_1 H^j T_2^{-1}$ (which can be calculated in advance), X to represent $T_2 x$, and Y to represent $T_1 y$, then the above equation becomes

$$Y^k = \sum_{j=-s_1}^{s_2-1} P^j X^{k+j} \quad (\text{A-9})$$

Note, T_1 and T_2 can be the same transforms such as the DCT.

To extend to two-dimensional (2D) separable filtering, we change the matrix-vector multiplication to matrix multiplication with pre-matrices and post-matrices. Thus, equation (A-6) becomes

$$y^{k,m} = \sum_{i=-s_1}^{s_2-1} \sum_{j=-t_1}^{t_2-1} H^i x^{k+i, m+j} W^j \quad (\text{A-10})$$

where H represents vertical filtering (column-wise) and W represents horizontal filtering (row-wise). The equivalent operation in the transform domain is as follows:

$$T_1 y^{k,m} T_2 = \sum_{i=-s_1}^{s_2-1} \sum_{j=-t_1}^{t_2-1} (T_1 H^i T_3^{-1}) (T_3 x^{k+i, m+j} T_4) (T_4^{-1} W^j T_2) \quad , \quad (\text{A-11})$$

where T_1 can be any transform matrices. For example, in the DCT algorithm, T_2 is the DCT transform matrix and $T_1=T_2^t$. Thus, the transform-domain 2D filtering can be calculated as a summation of several matrix products. Again, the pre-matrices $T_1 H^i T_3^{-1}$ and post-matrices $T_4^{-1} W^j T_2$ can be calculated in advanced. The block structure alignment procedure mentioned in Section 4.4.3 uses similar operations in the transform domain, except that the pre-matrices and post-matrices in block structure alignment are not for linear filtering.

A.3 Computational Complexity of DCT-Domain Compositing Operations

In this section, we discuss the computational complexity of the basic compositing operations in Table 4-2. In these operations, the most important common one is sparse matrix multiplications, such as $A \cdot P$, or $A \cdot P \cdot B$, where P is the image block, A and B are arbitrary matrices. Using regular matrix multiplication, it takes N^3 multiplications and $(N-1) \cdot N^2$ additions to compute $A \cdot P$. But as we mentioned earlier, there are many zeros in the

DCT coefficients of compressed images. We can use the RLC to easily find the locations of these zero DCT coefficients and skip their associated redundant computations. The resulting computational complexity is much less than that by using plain matrix multiplications.

A.3.1 Matrix multiplication

Figure A-2 shows the operations for matrix multiplication of a sparse matrix with regular pre-matrices and post-matrices, $A \cdot P \cdot B$, where P is a sparse matrix such as the DCT coefficients of an image block. In computing $A \cdot P$, each non-zero p_{ij} element needs N multiplications and N additions, which results in $m \cdot N$ multiplications and $m \cdot N$ additions if there are m non-zero elements in P . In the product matrix AP , the number of non-zero columns equals to the number of columns in P which contain non-zero elements. For example, in Figure A-2, there are three non-zero columns in the sparse matrix P , and thus three non-zero columns in the product matrix AP . Each non-zero column in AP needs N^2 multiplications and N^2 additions in computing $(AP) \cdot B$. Therefore, the whole operation, $A \cdot P \cdot B$, needs $(m \cdot N + n \cdot N^2)$ multiplications and $(m \cdot N + n \cdot N^2)$ additions, where n is the number of non-zero columns in the first product matrix, AP .

In terms of compression parameters, the number of non-zero coefficients can be expressed as N^2/β , where β is the compression ratio. Further, the non-zero coefficients are usually located in the low-order corner, i.e., the upper-left corner of P . Thus, an optimistic estimate of the number of non-zero columns is $\sqrt{N^2/\beta}$, namely, $N/\sqrt{\beta}$. The overall complexity of $A \cdot P \cdot B$ becomes $(1/\beta + 1/\sqrt{\beta}) \cdot N^3$ multiplications and $(1/\beta + 1/\sqrt{\beta}) \cdot N^3$ additions.

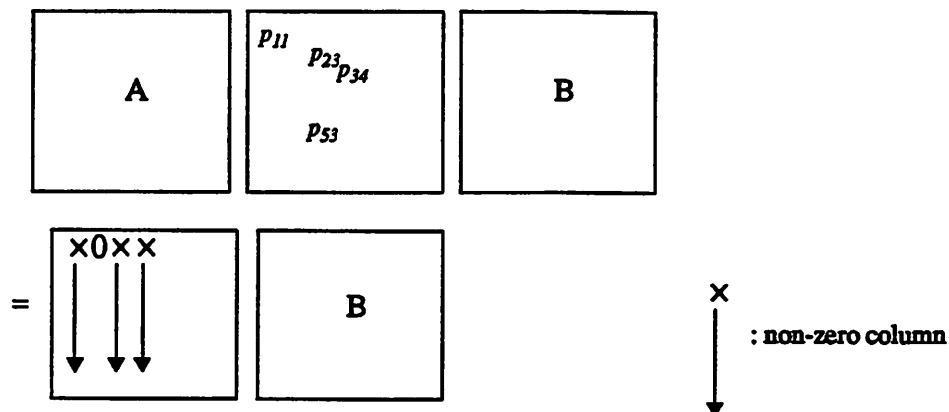


Figure A-2 Matrix multiplication of a sparse matrix with regular pre-matrices and post-matrices.

A.3.2 Scaling

A video conferencing scene usually consists of several video windows of conference participants, which are scaled down from their original size. As shown in Figure 4-21(b) (Scene 2), the input video sequences are scaled down with ratios of $1/2 \times 1/2$ and $1/3 \times 1/3$ respectively. In a $1/2 \times 1/2$ scaling operation, four blocks of original images are reduced to a single block — new block = $H_1B_{11}W_1 + H_2B_{21}W_1 + H_1B_{12}W_2 + H_2B_{22}W_2$, where B_i 's are the original neighboring blocks, H_i 's are the vertical scaling matrices, and W_i 's are the horizontal scaling matrices, as discussed in section 4.4.5. We can reorganize the operations to

$$(H_1B_{11} + H_2B_{21})W_1 + (H_1B_{12} + H_2B_{22})W_2 \quad (\text{A-12})$$

to remove some redundant multiplications. Since these operations are linear, they can be performed in the DCT domain. By using the sparse matrix techniques, we find that its complexity is equal to $(4/\beta + 2/\sqrt{\beta}) \cdot N^3$ multiplications and $(4/\beta + 2/\sqrt{\beta}) \cdot N^3 + 3 \cdot N^2$ additions. After normalized to the original image size, the complexity becomes $(1/\beta + 1/(2 \cdot \sqrt{\beta})) \cdot N$ multiplications and $(1/\beta + 1/(2 \cdot \sqrt{\beta})) \cdot N + 3/4$ additions per pixel, as shown in Table 4-2. Scaling operations with different ratios can be analyzed in a similar way.

A.3.3 MC and Inverse MC in the DCT domain

To perform the (inverse) MC algorithm in the DCT domain (i.e. MCD and MCD^{-1}), we need to handle three different cases, depending on the motion vectors. If both motion vector components, d_x and d_y , are zero or integral multiples of the block width, the DCT coefficients of the prediction errors can be added to the DCT coefficients of the reference block directly. If one of them is non-zero and not integral multiples of the block width, we need to compute the DCT coefficients of the reference block from two neighboring blocks — $DCT(\text{new block}) = DCT(H_1)DCT(B_1) + DCT(H_2)DCT(B_2)$, which requires $2 \cdot N^3/\beta$ multiplications and $2 \cdot N^3/\beta + N^2$ additions based on the sparse matrix techniques.

If both the motion vector components are non-zero and not integral multiples of the block width, we need to use equation (A-4-13) to perform block boundary adjustment in both two directions. Its complexity is $(4/\beta + 2/\sqrt{\beta}) \cdot N$ multiplications and $(4/\beta + 2/\sqrt{\beta}) \cdot N + 3$ additions per pixel, as discussed in the previous section. The overall complexity for the MC algorithm in the DCT domain is the linear combination of the complexity for these three different cases, weighted with the distribution percentage of the motion vectors.

A.3.4 Pixel-Wise Translation

For pixel-wise translation, if the translation distance is not an integral multiple of the block size in both directions, the required computations are exactly the same as those for the MCD algorithm with non-zero motion vectors. But since every product matrix in the parenthesis of equation (A-12) can be shared by two new blocks, the overall computational complexity can be reduced to $(2/\beta + 2/\sqrt{\beta}) \cdot N$ multiplications and $(2/\beta + 2/\sqrt{\beta}) \cdot N + 3$ additions per pixel, as shown in Table 4-2.