國 立 交 通 大 學

電子工程學系 電子研究所碩士班

碩　士　論　文

以物件為基礎之 MPEG-4 視訊之多點視訊會議接收端

A Multipoint Videoconference Receiver for MPEG-4 Object-based Video

研 究 生：簡志凱

指導教授：林大衛 博士

中 華 民 國 九 十 四 年 六 月

以物件為基礎之 MPEG-4 視訊之多點視訊會議接收端

A Multipoint Videoconference Receiver for MPEG-4 Objected-based Video

研　究　生：簡志凱　　　　Student: Chih-Kai Chien

指導教授：林大衛　博士　　　　Advisor: Dr. David W. Lin

國　立　交　通　大　學

電子工程學系　電子研究所碩士班

碩　士　論　文

A Thesis
Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical Engineering and Computer Science
National Chiao Tung University
in Partial Fulfillment of Requirements
for the Degree of
Master of Science
in
Electronics Engineering
June 2005
Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 四 年 六 月

以物件為基礎之 MPEG-4 視訊之多點視訊會議接收端

研究生：簡志凱　　　　　　　　指導教授：林大衛教授

國立交通大學　電子工程系 電子研究所碩士班

摘要

在本篇論文中，我們設計並實現一個在個人電腦上的視訊影像接收端。未來將利用此系統來提供多點視訊會議的服務。

建立此系統必需整合網路、影像解碼、聲音解碼、以及同步與合成單元。首先，我們透過及時傳輸規約（RTP）協定方式與系統發送端連線並接收影像和聲音資料。由不同發送端所接收到的視訊資料將經由各自的 MPEG-4 影像解碼器進行解碼。聲音資料則經由 MPEG-4 音訊解碼器進行解碼。當我們得到解碼後的影像資料，同步與合成單元將會控制各個解碼器使影像同步並合成新的影像。最後將合成影像顯示在螢幕上。

在整體圖形應用及控制介面的實現上，我們採用了 Microsoft Windows SDK 來建構整個介面。在 AMD 2.1-GHz CPU 及 512-MB RAM 的個人電腦上，當只有一人連接到此接收端上時，系統的執行速度是每秒約可顯示 12 張影像。若有兩人時，則是每秒約 6 張。如果增加到三人時，系統的執行速度將降到每秒 3.5 張。

A Multipoint Videoconference Receiver for MPEG-4 Object-based Video

Student: Chih-Kai Chien Advisor: Dr. David W. Lin

Department of Electronics Engineering & Institute of Electronics
National Chiao Tung University

Abstract

We consider the design and implementation of the videoconference receiver on a personal computer (PC). The intended application is PC-based multipoint videoconference system.

To implement this system, we need to integrate network, video decoder, audio decoder, and the unit of the synchronization and composition. First, through RTP (Real-time Transport Protocol) the receiver can be connected to the different transmitters and receive the video and audio streams at the same time. Next, the video streams from different transmitters would be decoded by MPEG-4 visual decoders and the audio streams would decoded by MPEG-4 AAC decoder. After decoding the video stream, the unit of the synchronization and composition controls the each visual decoder to make these decoders synchronously and composes these video streams. Finally, the composite video would be display on the desktop.

In real-time PC-based software implementation of the receiver, we employ a graphical user interface established using the Windows SDK. With an AMD 2.1-GHz CPU and 512-MB RAM, the current un-optimized implementation yields a speed of about 12 frames per second and for CIF (352x288) video when only one user connects to the receiver. In the presence of two users, the processing time is about 6 frames per second. If we increase to three client user, the processing time goes down to 3.5 frames per second.

誌謝

本篇論文的產生，最感謝的是我的指導教授－林大衛老師。無論是課業或研究上，老師均給予我多方面的協助與鼓勵，讓我在研究學習中能夠順利進行，進而完成本篇論文。老師除給了我完善的專業知識訓練外，在論文實作部份，老師更培養了我認真踏實的的研究態度，讓我獲益良多。

實驗室完善的資源，讓我可以順利的克服許多實作上所遇的的困難。感謝岳賢及夢遠學長給予我在研究上的指導，使青澀的我能找到研究的方向，感謝昱昇、景中、汝芩、家揚學長、崑健學長等實驗室所有同伴，以及我最好的伙伴—鎮宇，陪我度過兩年的研究生涯。

特別感謝我的爸媽、姊姊、弟弟以及關心我的家人，對於家庭的細心照顧，讓我可在無後顧之憂下全力進行研究學習及工作，並給予我精神上的支持與鼓勵，陪伴我度過所有的艱難。在此，我要將我的論文獻給所有關心與幫助我的人。

簡志凱

民國九十四年六月於新竹

Contents

List of Figures

Chapter 1

Introduction

We consider the design and implementation of multi-point videoconference system on a personal computer (PC). Figure 1.1 shows the overall system structure. The transmitter captures video and audio in real-time. Then it encodes and sends out the encoded video and audio. The receiver would receive video and audio streams from different transmitters, decode them, compose the decoded signals into one stream, and display the composite signal on the desktop. In this thesis, we consider the design and implementation of the receiver. It consists of four parts: Real-time transport protocol (RTP) network interface, visual object decoder, audio object decoder, and synchronization and composition, as shown in Fig. 1.2.

The RTP provides end-to-end delivery services for data with real-time characteristics which is suitable for our live application. We use the jrtplib 3.1.0 software [15] developed at the Expertise Centre for Digital Media (EDM) to realize the RTP network interface. The video and audio data streams each has it own RTP session. The RTP packetizing and the insertion of header time information as per the RTP standard mechanisms is handled by the server. The visual and audio object decoders are based on MPEG-4 standard. The MPEG-4 standard is originally intended for very high compression coding of audio-visual information at very low bit-rate to help us decrease the bandwidth requirement of the network. To achieve the multi-point conferencing capability, we use the multi-threading technique to manage the video and the audio decoders. The multi-threading technique could make the operating system handle each user's own decoders almost simultaneously.

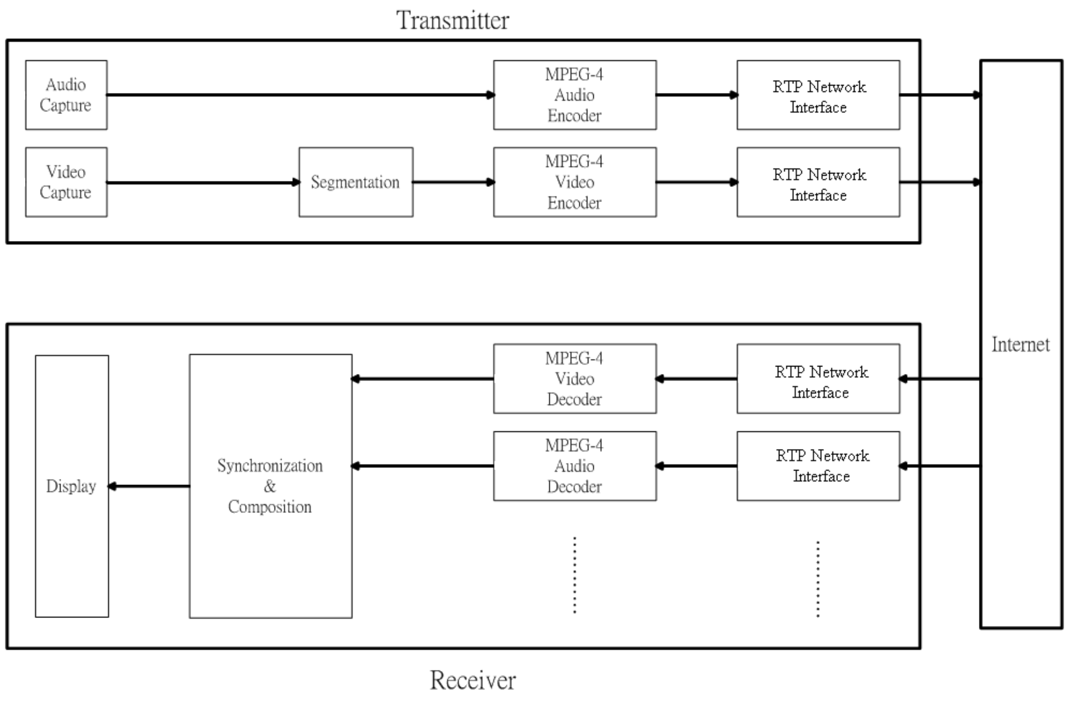


Figure 1.1: Overall videoconference system.

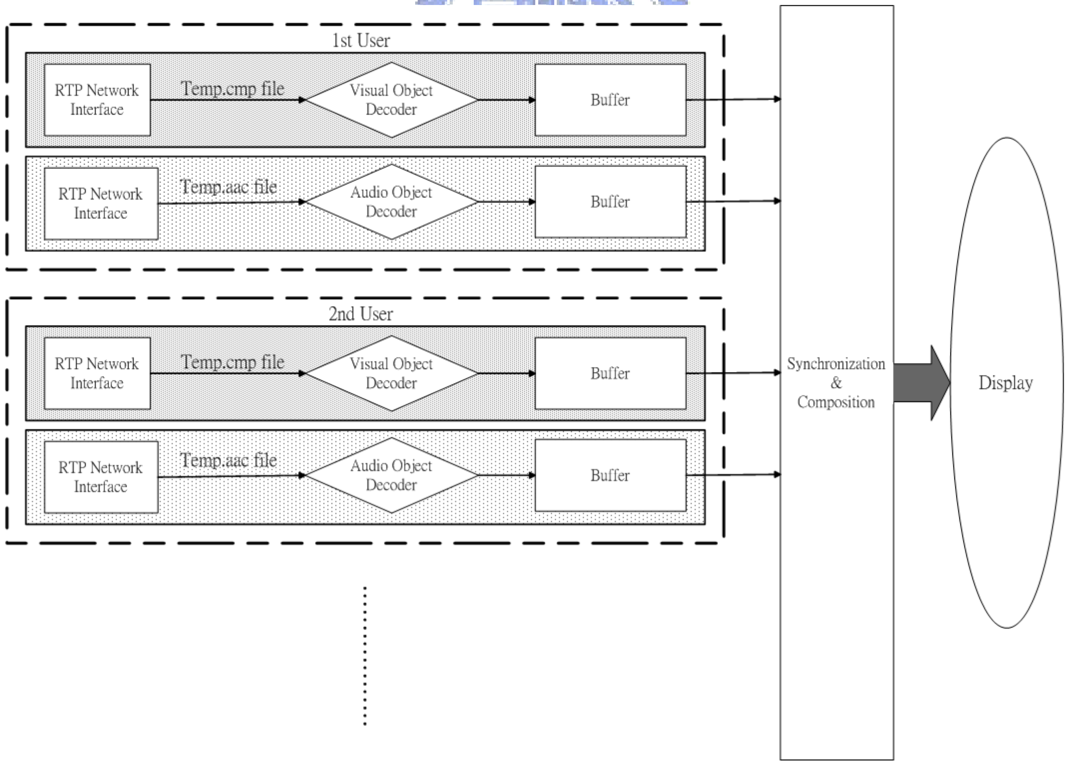


Figure 1.2: Block diagram of the videoconference receiver.

However, the synchronization among the multi-threads becomes an important problem. We add some control mechanism to resolve this issue. Finally we compose the video objects from different transmitters and display the result, with each video object placed at a user-specified position on the desktop. The audio would also be decoded, composed, and played in this system.

This thesis is organized as follows. Chapter 2 is an overview of the MPEG-4 standard. Chapter 3 describes the RTP technology. Chapter 4 discusses the detail of the receiver's architecture for multi-point videoconference. The experimental results of the receiver are described in Chapter 5. Finally, Chapter 6 contains the conclusion.

Chapter 2

Overview of the MPEG-4 Standard

MPEG-4 is an ISO standard (ISO/IEC international standard 14496) developed by MPEG (Moving Picture Experts Group), the committee that also developed the Emmy Award winning standards known as MPEG-1 and MPEG-2. These standards made interactive video on CD-ROM, DVD and Digital Television possible. The first version of MPEG-4 was finalized in October 1998 and became an International Standard in the first months of 1999. The fully backward compatible extensions under the title of MPEG-4 Version 2 were frozen at the end of 1999, to acquire the formal International Standard Status early in 2000 [2].

MPEG-4 defines the deployment of non-proprietary multimedia content independently of platform or transmission medium. It has relied on and taken from a number of existing technologies while at the same time adding a number of innovative tools and concepts. It is also more than just another highly efficient standard for delivering video and audio sequences. It integrates a number of key technologies into a solid and uniform platform by avoiding the use of proprietary and non inter-working formats, while promoting and delivering user interactivity to the wider consumer market, beyond the desktop computer [2].

MPEG-4 builds on the proven success of three fields [2]:

- digital television,
- interactive graphics applications (synthetic content), and

- interactive multimedia (World Wide Web, distribution of and access to content).

2.1 Organization of the MPEG-4 Standard

The MPEG-4 standard addresses the generic coding of audio-visual objects, as illustrated in Fig. 2.1. It consists of the following basic parts. The following description of the different parts are mainly taken from [1] and [2]:

1. ISO/IEC 14496-1: Systems

   The MPEG-4 Systems specification defines architecture and tools to create audio-visual scenes from individual objects. A major tool for MPEG-4 systems is scene description. The MPEG-4 scene description, a totally new component in the MPEG specifications, is based on VRML (virtual reality modeling language) and specifies the spatial-temporal composition of objects in a scene. The scene description is at the core of the systems specification, and allows easy creation of compelling audio-visual content.

2. ISO/IEC 14496-2: Visual

   The MPEG-4 visual specification defines the main video codec. It consists of natural, arbitrary shape and synthetic video coding.

   For natural video coding, the main video coding tools are still texture coding, similarly to MPEG-1 and MPEG-2. For intra coding, the MPEG-4 visual specification uses DCT, IDCT, intra prediction, quantization and de-quantization to reduce spatial redundancy. For inter coding, the MPEG-4 visual specification uses motion estimation and motion compensation to reduce temporal redundancy. In visual coding, the major difference from MPEG-1 and MPEG-2 is object coding. In MPEG-4, each picture is considered as consisting of objects, since some MPEG-4 functionalities require access not only to entire pictures but also to objects.

   For synthetic video coding, in MPEG-4, mesh-based representation is useful. MPEG-4 includes a tool for triangular mesh-based representation of general objects.

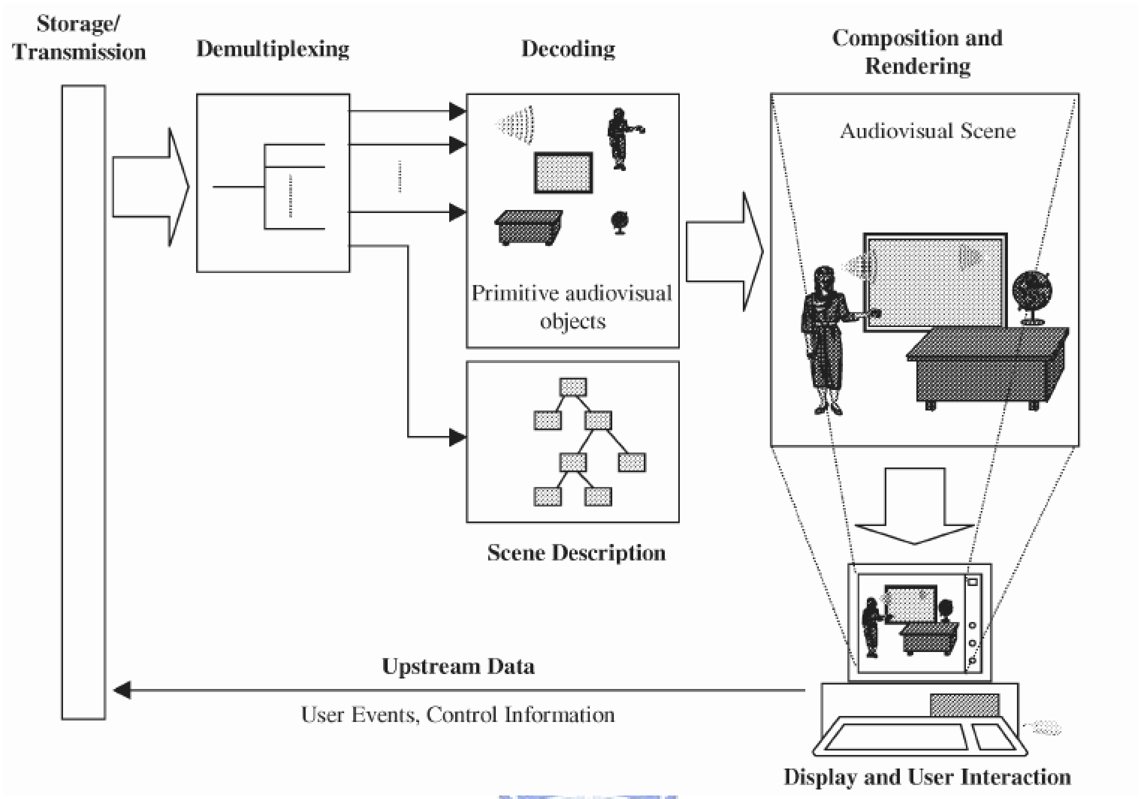

Storage/ Transmission
Demultiplexing
Decoding
Composition and Rendering
Audiovisual Scene
Primitive audiovisual objects
Scene Description
Upstream Data
User Events, Control Information
Display and User Interaction

Figure 2.1: A high level view of an MPEG-4 terminal (from[5]).

3. ISO/IEC 14496-3: Audio

   ISO/IEC 14496-3 (MPEG-4 Audio) is a standard that integrates many different types of audio coding: natural sound with synthetic sound, low bit-rate delivery with high-quality delivery, speech with music, complex sound tracks with simple ones, and traditional content with interactive and virtual-reality content. MPEG-4, unlike previous audio standards created by ISO/IEC and other groups, does not target at a single application such as real-time telephony or high-quality audio compression. MPEG-4 Audio is a rather generic standard that applies to applications requiring the use of advanced sound compression, synthesis, manipulation, or playback. The subparts specify state-of-the-art coding tools in several domains. However, MPEG-4 Audio is more than just the sum of its parts. As the tools described are integrated with the rest of the MPEG-4 standard, new possibilities for object-based audio coding, interactive presentation, dynamic sound tracks, and other sorts of new media, are enabled.

4. ISO/IEC 14496-4: Conformance Testing

   This part of ISO/IEC 14496 specifies how tests can be designed to verify whether bitstreams and decoders meet requirements specified in parts 1, 2, and 3 of ISO/IEC 14496. In this part of ISO/IEC 14496, encoders are not addressed specifically. An encoder may be said to be an ISO/IEC 14496 encoder if it generates bitstreams compliant with the syntactic and semantic bitstreams requirements specified in parts 1, 2 and 3 of ISO/IEC 14496.

5. ISO/IEC 14496-5: Reference Software

   Reference software is normative in the sense that any conforming implementation of the software, taking the same conforming bitstreams, using the same output file format, will output the same file. Complying ISO/IEC 14496 implementations are not expected to follow the algorithms or the programming techniques used by the reference software. Although the decoding software is considered normative, it should not add anything to the technical description included in parts 1, 2, 3 and 6 of ISO/IEC 14496.

6. ISO/IEC 14496-6: DMIF

   DMIF, or Delivery Multi-media Integration Framework, is an interface between the application and the transport, which enables the MPEG-4 application developer to stop worrying about the transport. A single application can run on different transport layers when supported by the right DMIF instantiation.

   MPEG-4 DMIF supports the following functionalities:

   - A transparent MPEG-4 DMIF-application interface irrespective of whether the peer is a remote interactive peer, broadcast or local storage media.
   - Control of the establishment of FlexMux channels.
   - Use of homogeneous networks between interactive peers: IP, ATM, mobile, PSTN, Narrowband ISDN.

- Support for mobile networks, developed together with ITU-T.
- User commands with acknowledgment messages.
- Management of MPEG-4 Sync Layer information.

However, the available software has not been developed well. Interest in DMIF has waned in recent MPEG activities and the DMIF software has suffered a maintenance problem [9].

2.2 MPEG-4 Visual Overview

Contents in this section are mainly taken from [3].

The target of MPEG-4 video is providing standardized core technologies allowing efficient storage, transmission and manipulation of video data in multimedia environments. It provides technologies to view, access and manipulate objects rather than pixels, with great error robustness at a large range of bit-rates. In order to achieve this broad goal, video activities in MPEG-4 aim at providing solutions in the form of tools and algorithms enabling functionalities such as efficient compression, object scalability, spatial and temporal scalability, error resilience, and fine granularity scalability. The standardized MPEG-4 video provides a toolbox containing tools and algorithms bringing solutions to the above mentioned functionalities and more.

2.2.1 Structure of MPEG-4 Video Data

An input video sequence can be defined as a sequence of related snapshots or pictures, separated in time. Many of MPEG-4 functionalities require access not only to entire sequence of pictures, but to an entire object, and further, not only to individual pictures, but also to temporal instances of these objects within a picture.

The concept of Video Objects (VOs) and their temporal instances, Video Object Planes (VOPs) is central to MPEG-4 video. A VOP can be fully described by a set of luminance and chrominance values and shape representation. Figure 2.2 shows the decomposition of a picture into a number of separate VOPs.

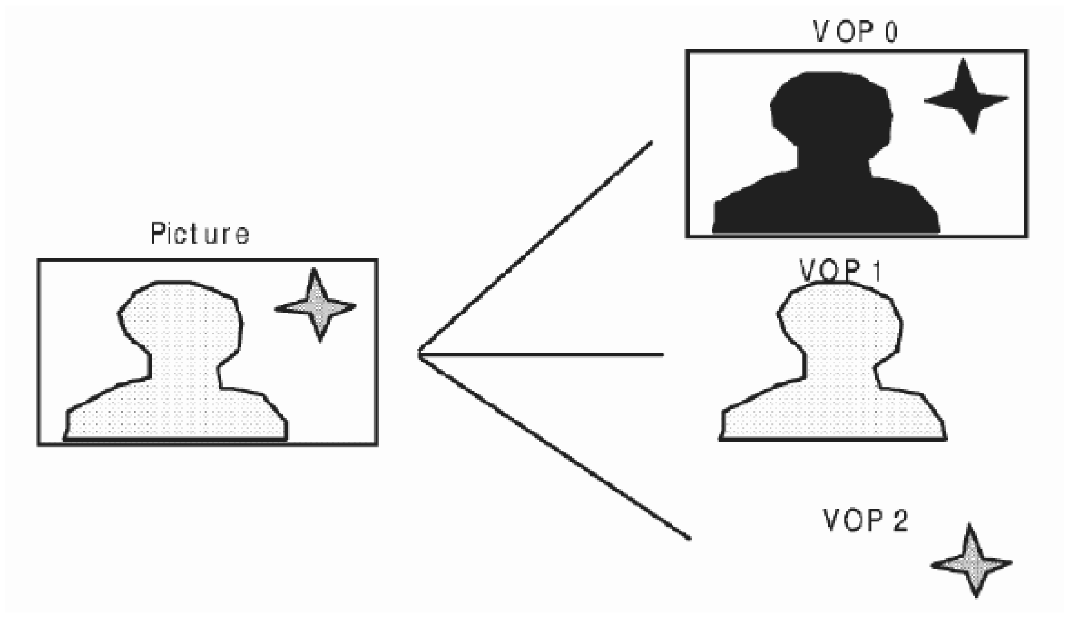


Figure 2.2: Segmentation of a picture into VOPs (from [5]).

Each VO is encoded separately and multiplexed to form a bitstream that users can access and manipulate. The encoder sends, together with VOs, information about scene composition to indicate where and when VOPs of a VO are to be displayed. Figure 2.3 shows the organization of coded MPEG-4 Video in a top-down hierarchical structure.

- VideoSession (VS): A Video session is the highest syntactic structure of the coded visual bitstream and simply consists of an ordered collection of video objects. The complete MPEG-4 scene which may contain any 2-D or 3-D natural or synthetic objects.

- VideoObject (VO): A Video object (2D + time) represents a complete scene or a portion of a scene with a semantic. In the simplest case this can be a rectangular frame, or it can be an arbitrarily shaped object corresponding to a physical object or background of the scene.

- VideoObjectLayer (VOL): Each video object can be encoded in scalable (multi-layer) or non-scalable form (single layer), depending on the application, represented by VOL. The VOL provides support for scalable coding. A video object can be

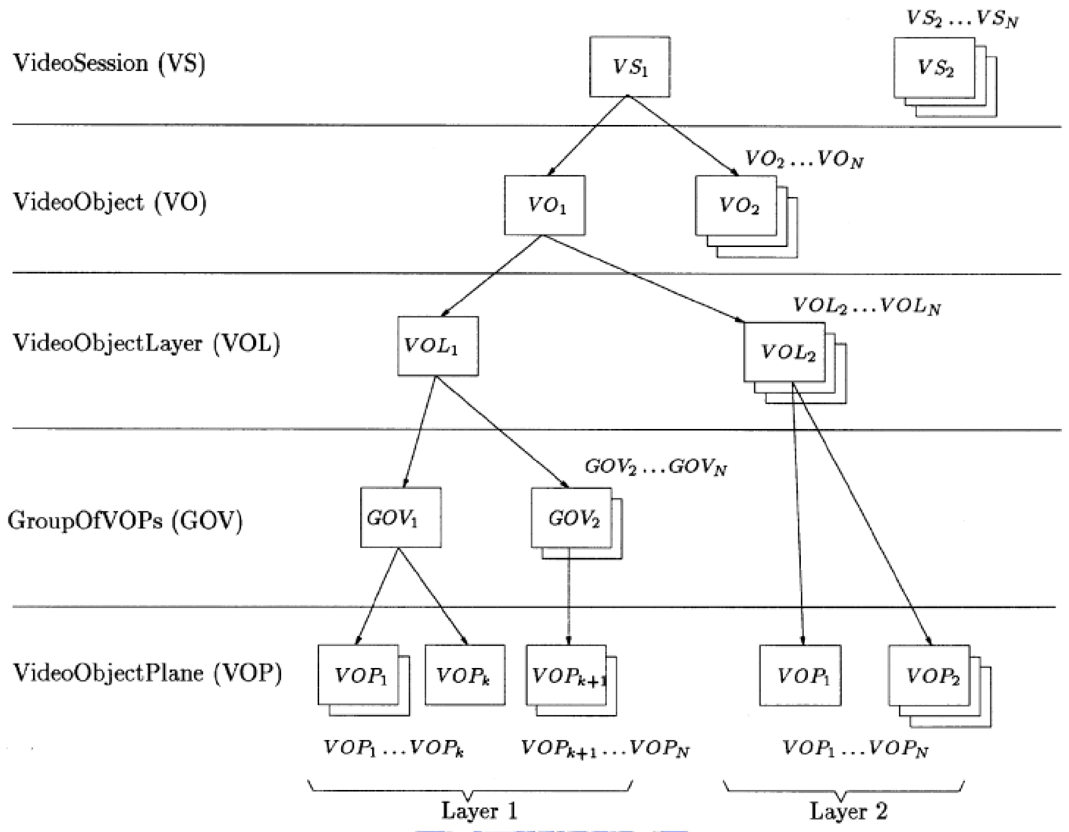


Figure 2.3: Structure of coded video data (from [7]).

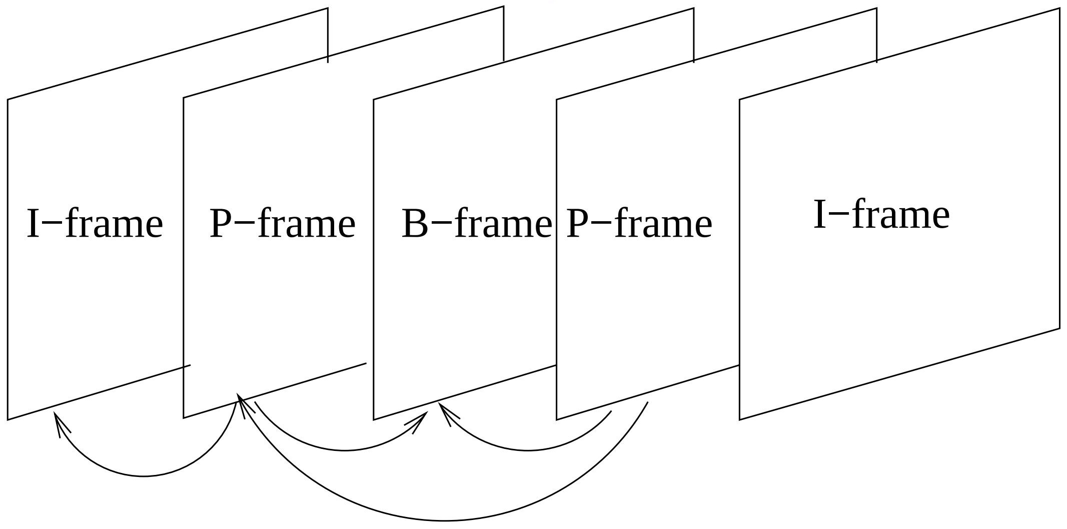


Figure 2.4: Types of VOP (from [8]).

encoded using spatial or temporal scalability, going from coarse to fine resolution.

- GroupOfVideoObjectPlanes (GOV): Group of video object planes are optional entities. The GOV groups together video object planes. GOVs can provide points in the bitstream where video object planes are encoded independently from each other, and can thus provide random access points into the bitstream.
- VideoObjectPlane (VOP): A VOP is a time sample of a video object. Figure 2.4 shows three of the four types of VOP that use different coding methods.

1. An Intra-coded (I) VOP is coded using information only from itself.
2. A Predictive-coded (P) VOP is a VOP which is coded using motion compensated prediction from a past reference VOP.
3. A Bidirectionally predictive-coded (B) VOP is a VOP which is coded using motion compensated prediction from a past and/or future reference VOP(s).
4. A Sprite (S) VOP is a VOP for a sprite object or a VOP which is coded using prediction based on global motion compensation from a past reference VOP.

The macroblock (MB) is a basic coding structure constructing VOP. In the MPEG-4 standard, a macroblock contains a section of the luminance component and the sub-sampled chrominance components in 4:2:0 format. In this format, there are 4 luminance blocks and 2 chrominance blocks in a macroblock. The luminance and chrominance samples are positioned as shown in Fig. 2.5.

2.3 MPEG-4 Video Decoding

Contents in this section are mainly taken from [5], [6] and [7]

Figure 2.6 is a diagram of the video decoding process without any scalability feature. The diagram is simplified for clarity. The same decoding scheme is applied when decoding all the VOPs of a session. The decoder is mainly composed of three parts: shape decoder, motion decoder and texture decoder. The reconstructed VOP is obtained by

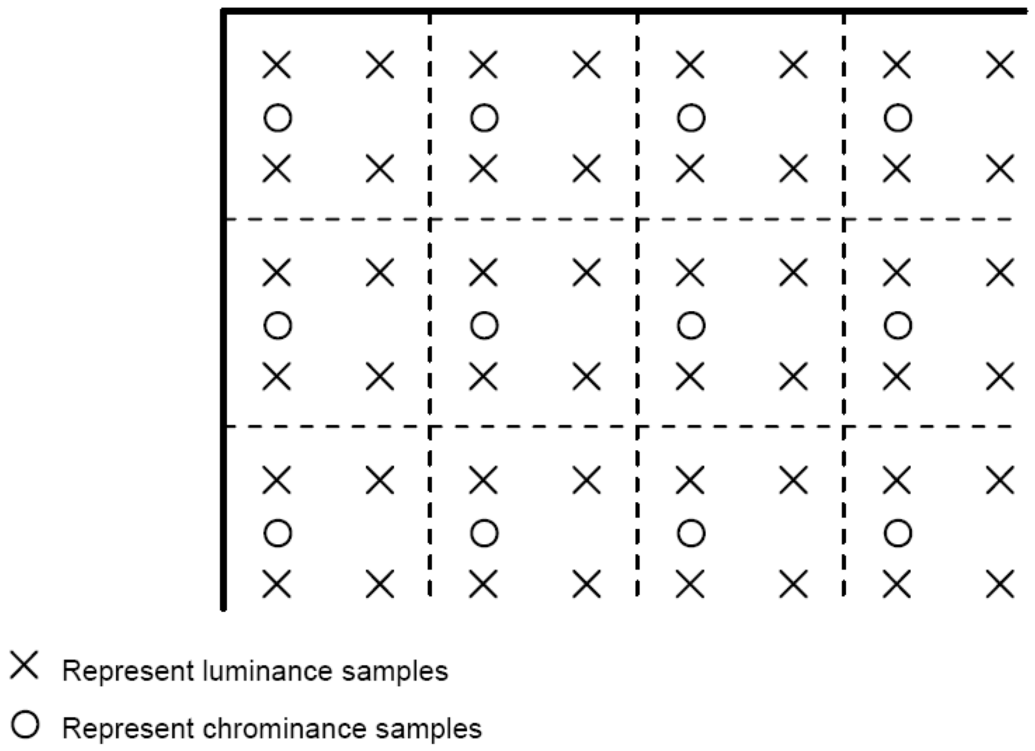


Figure 2.5: Positions of luminance and chrominance samples in 4:2:0 data (from [3]).

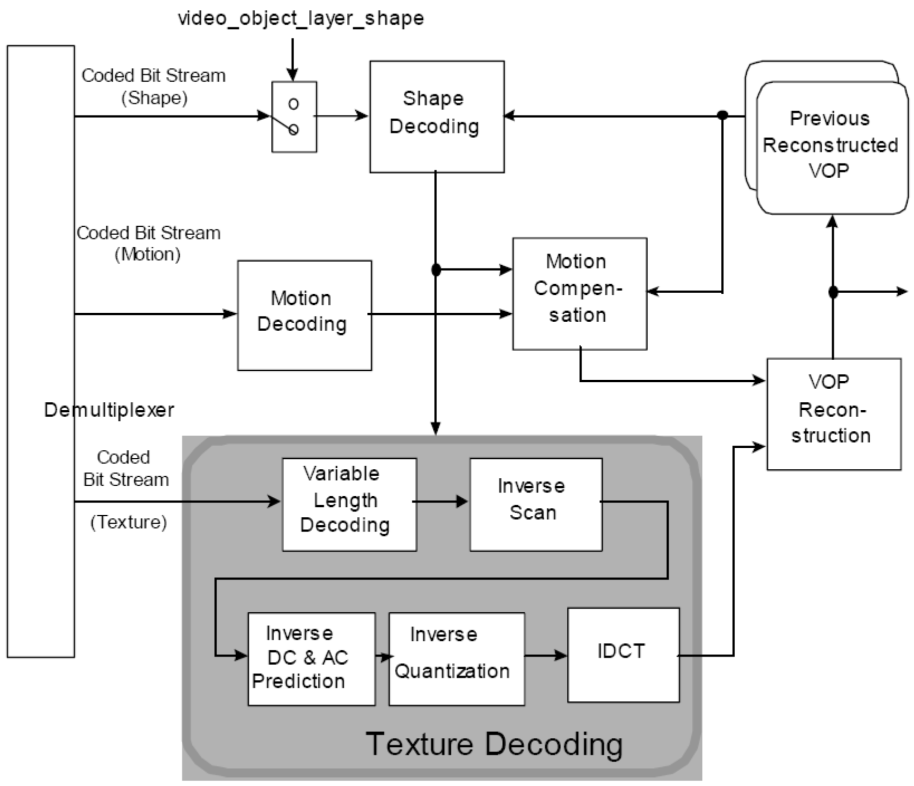


Figure 2.6: Detailed structure of the video decoder (from [3]).

combining the decoded shape, texture and motion information. With the exception of the inverse discrete cosine transform (IDCT), it is defined such that all decoders shall produce numerically identical results. Any decoding process that produces identical results to the process described here complies with ISO/IEC 14496-2.

2.3.1 Shape Decoding

The ability to represent arbitrary shapes is an important capability of the MPEG-4 video standard. In general, shape representation can be either implicit (based on chroma-key and texture coding) or explicit (based on shape information from texture coding). Implicit shape representation offers less encoding flexibility. However, it can result in quite usable shapes while being relatively simple and computationally inexpensive. Explicit shape representation is more complex and expensive but offers flexible encoding and bet-

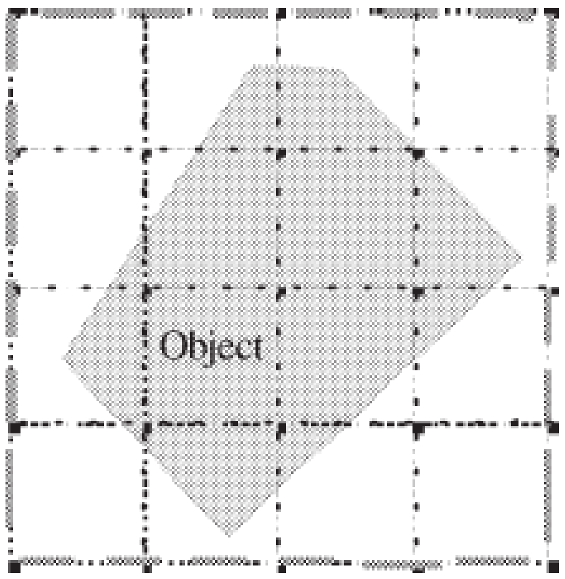


Figure 2.7: A VOP in a bounding box (from [7]).

ter quality shapes. Therefore explicit shape representation was chosen in MPEG-4 video.

For each VO given as a sequence of VOPs of arbitrary shapes, the corresponding sequence of alpha planes is known (generated via segmentation or via chroma-key). Binary alpha planes are encoded by modified context-based binary arithmetic encoding (CAE) and gray scale alpha planes are encoded by motion compensated DCT similar to texture coding. For the binary alpha plane, an extended rectangular bounding box enclose the shape to be coded. The bounded alpha plane is partitioned into blocks of 16×16 samples called alpha blocks. In fact, each alpha block is co-located with each texture macroblock. It is important to minimize the number of alpha blocks in the bounding box for efficient coding. The pixels on the boundaries or inside the object are assigned a value of 255 and are considered opaque while the other pixels are assigned a value of 0 and considered transparent. Figure 2.7 shows an example of an arbitrary shape VOP with bounding box and the 16×16 block structure.

Binary shape decoding

Binary shape coding is based on a block-based representation. The basic coding methods are CAE and block-based motion compensation. The primary data structure used is denoted as the binary alpha block (BAB). Each BAB can be coded in intra mode or in inter mode. In intra mode, no explicit prediction is performed. In the inter mode, shape information is differenced with respect to the prediction obtained using a motion vector. The motion vector is coded differentially and included in the bitstream. Therefore each BAB can be assigned a mode from among the following choices:

1. The block is all transparent. In this case no coding is necessary. Texture information is not coded for such blocks either.

2. The block is all opaque. Again, shape coding is not necessary for such blocks, but texture information needs to be coded.

3. The block is coded using IntraCAE without use of past information.

4. Motion vector difference (MVD) is zero but the block is not updated.

5. MVD is non-zero, but the block is not updated.

6. MVD is zero and the block is updated. InterCAE is used for coding the block update.

7. MVD is non-zero, and the block is coded by InterCAE.

CAE encoding is used to code each binary pixel of the BAB. Prior to coding the first pixel, the arithmetic encoder is initialized. Each binary pixel is then encoded in raster order. The process for encoding a given pixel is as follows:

1. Compute a context number.

2. Index a probability table using the context number.

3. Use the indexed probability to drive an arithmetic encoder.

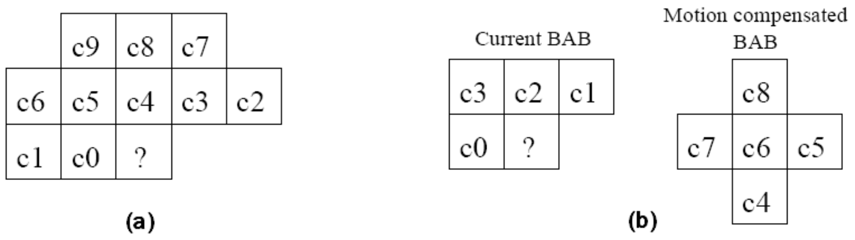


Figure 2.8: Pixel templates used for (a) intra and (b) inter context determination of a BAB. Pixel to be coded is marked with “?”. (From [3])

When the final pixel has been processed, the arithmetic code is terminated. Figure 2.8 shows the computation of the contexts for INTRA and INTER modes.

The decoding of binary alpha block follows the inverse sequence of operations expect for encoder specific tasks such as motion estimation, subsampling factor determination, mode decision, etc.

Gray scale shape decoding

The gray scale shape coding has a structure similar to that of binary shape with the difference that each pixel can take on a range of values (usually 0 to 255) representing the degree of the transparency of that pixel. The gray scale shape corresponds to the notion of alpha plane used in computer graphics, in which 0 corresponds to a completely transparent pixel and 255 to a completely opaque pixel. Intermediate values of the pixel correspond to intermediate degrees of transparencies of that pixel.

Gray level alpha plane is encoded as its support function and the alpha values on the support. The support function is encoded by binary shape coding as described previously and the alpha values are encoded using a block based motion compensated DCT similar to that of texture coding. Figure 2.9 shows the block diagram of gray shape coding.

2.3.2 Motion Coder

The motion coder consists of a motion estimator, motion compensator, previous/next VOPs store and motion vector predictor and coder. The motion coding is necessary only

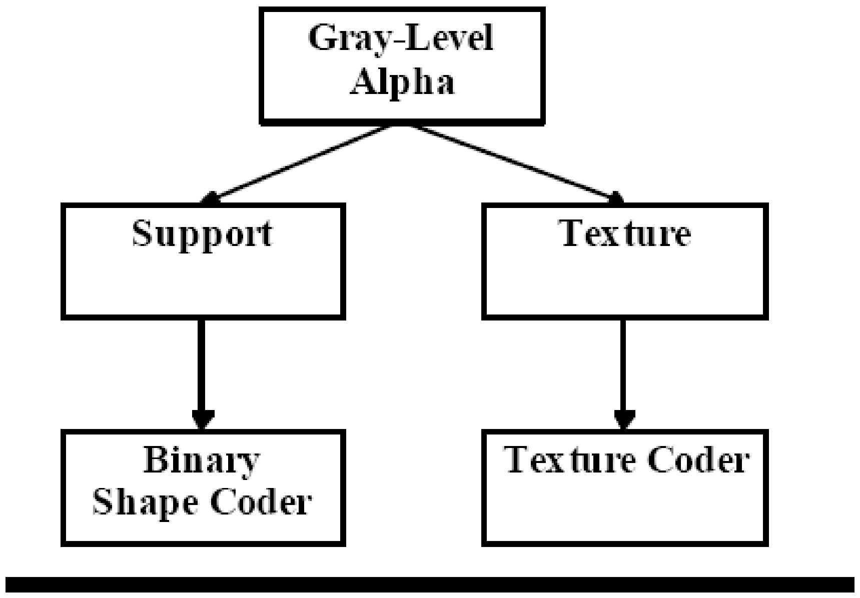


Figure 2.9: Gray shape coding (from [3]).

for P-VOP and B-VOP to reduce temporal redundancy. For P-VOP, motion estimator computes motion vectors using the current VOP and temporally previous reconstructed VOP from the previous reconstructed VOP store. In case of B-VOP, motion estimator computes motion vectors using the current VOP and temporally previous reconstructed VOP from the previous reconstructed VOP store, as well as the current VOP and temporally next VOP from the next constructed VOP store. The motion compensator uses these MVs to compute motion compensated prediction using the reference VOP which is the temporally previous reconstructed version of the same VOP. The MV predictor and coder generates prediction for the MV to be coded. Furthermore, a special padding technique is required for the reference VOP.

However, MPEG-4 video also supports an H.263 based mode for motion compensation, referred to as the direct mode. In direct mode, the MV in a B-VOP is obtained by scaling of the P-VOP MV, and further correcting it by a small MV. In addition, MPEG-4 supports efficient coding of interlaced video. It combines the macroblock based frame/field motion compensation of MPEG-2 with the normal motion compensation of MPEG-4.

It allows motion compensation of arbitrary shaped VOPs of interlaced video whereas MPEG-2 only supports rectangular pictures of intelaced video.

2.3.3 Texture Decoding

The texture information of a video object plane is presented in the luminance Y and two chrominance components Cb and Cr of the video signal. The texture coder codes the signal forming macroblocks within a VOP. Two types of macroblocks exist, those that lie inside the VOP and those that lie on the boundary of the VOP. The blocks that lie inside the VOP are coded using DCT like that used in H.263 but optimized in MPEG-4. The blocks that lie on the VOP boundary are first padded and then coded similiar to the blocks that lie inside the VOP. The general operations in the texture encoder are: DCT on original or prediction error blocks, quantization of 8×8 block DCT coefficients, scanning of quantized coefficients and variable length coding of quantized coefficients.

Similar to MPEG-1/2 and H.263, the block DCT codes blocks of size 8×8 samples with the difference that since VOP shapes can be arbitrary, the blocks on the VOP boundary require padding prior to texture coding. The encoder does forward transform before quantization and inverse transform after inverse quantization in the loop to obtain reconstructed image for the next temporal frame. MPEG-4 video supports two techniques of quantization. One refers to as the H.263 quantization method, and the orther is MPEG quantization method. Figure 2.10 shows the quantizer characteristics in H.263. It has uniform quantization for intra DC coefficients and nearly uniform quantization with dead zone for the inter DC and all AC coefficients. The step size Q can be changed in increment of 2 from 2 to 62 depending on rate controllor. In the MPEG quantization method, all coefficients are quantized by the uniform quantizer with different step size Q. The step size Q is decided by quantizer matrix and the default quantizer matrix is defined as shown in Fig. 2.11. This matrix can be modified by the rate controllor.

After quantization, the DC coefficients of intra blocks are coded by intra prediction as shown in Fig. 2.12. The intra prediction is a new operation in the MPEG-4 standard to reduce the spatial redundancy between 8×8 blocks. For instance, the DC coefficient of block X is predicted from the DC coefficient of blocks A, B and C. Not only the

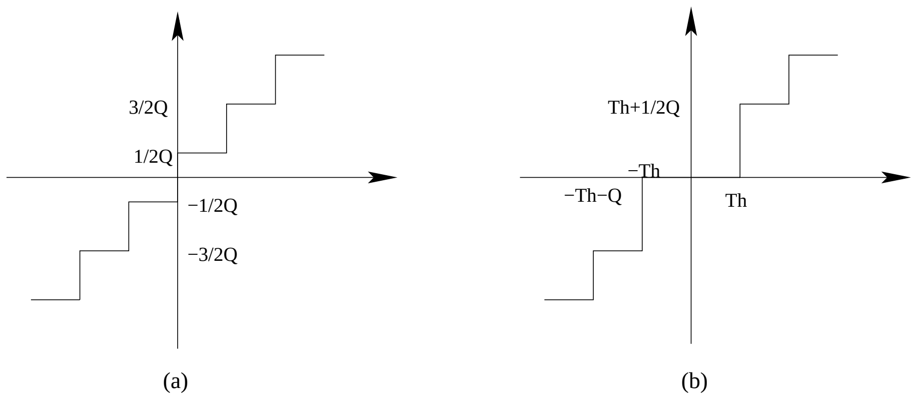
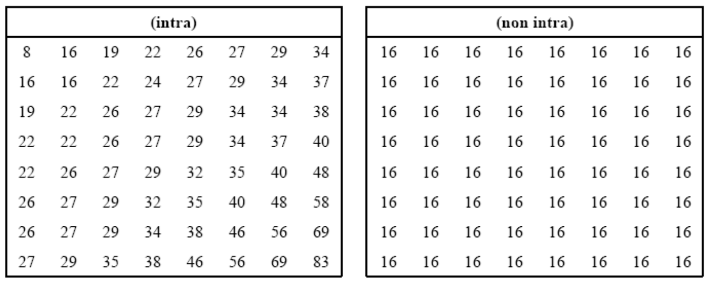


Figure 2.10: Quantizers in H.263. (a) Intra DC coefficients only. (b) Inter DC and all AC coefficients (from [8]).



(intra)							
8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

(non intra)							
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16

Figure 2.11: Default quantization matrix Q (from [3]).

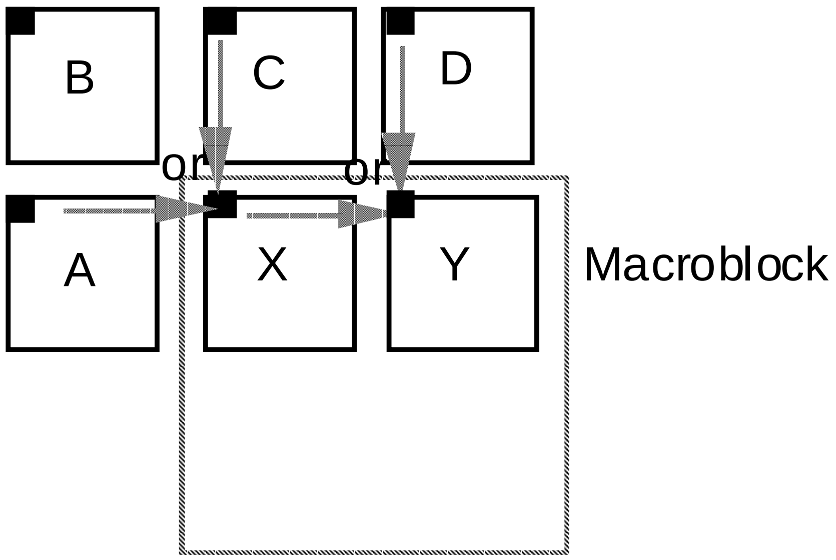


Figure 2.12: Prediction of DC coefficients of blocks in an intra macroblock (from [3]).

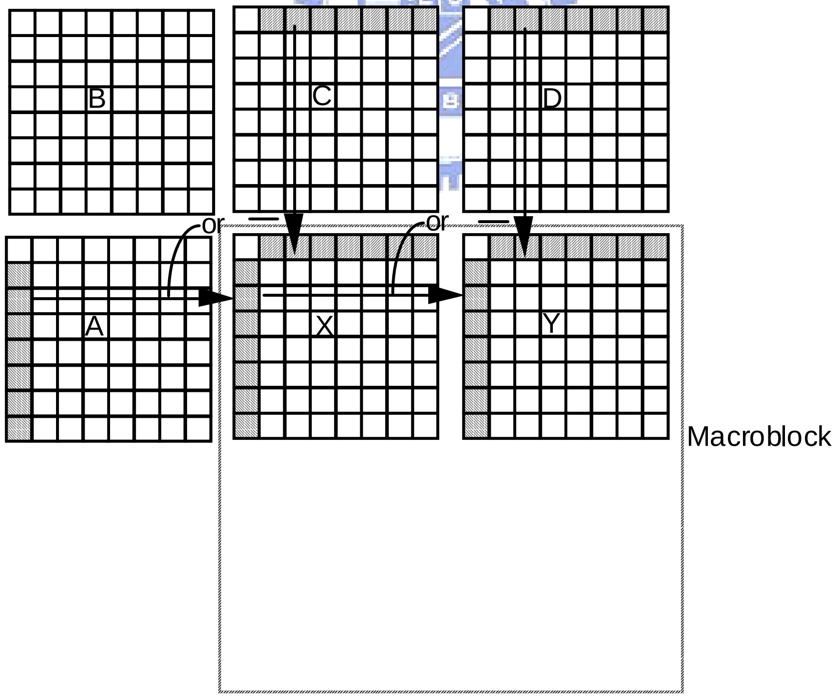
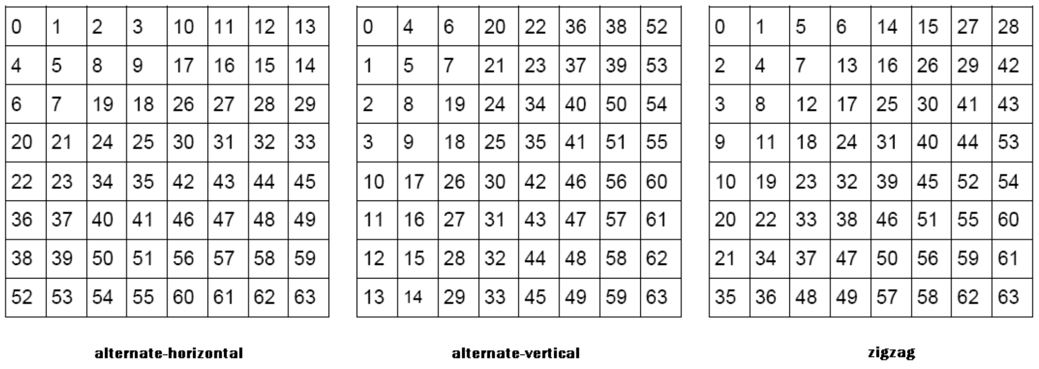


Figure 2.13: Prediction of AC coefficients of blocks in an intra macroblock (from [3]).



0	1	2	3	10	11	12	13
4	5	8	9	17	16	15	14
6	7	19	18	26	27	28	29
20	21	24	25	30	31	32	33
22	23	34	35	42	43	44	45
36	37	40	41	46	47	48	49
38	39	50	51	56	57	58	59
52	53	54	55	60	61	62	63

alternate-horizontal

0	4	6	20	22	36	38	52
1	5	7	21	23	37	39	53
2	8	19	24	34	40	50	54
3	9	18	25	35	41	51	55
10	17	26	30	42	46	56	60
11	16	27	31	43	47	57	61
12	15	28	32	44	48	58	62
13	14	29	33	45	49	59	63

alternate-vertical

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

zigzag

Figure 2.14: Scans for 8×8 block (from [3]).

DC coefficients of intra blocks are predicted and coded differentially, so are some of the AC coefficients. The AC prediction depends on DC prediction and is given in Fig 2.13. The AC coefficients in the first row or in the first column are predicted with three previous decoded AC coefficients. The predicted DC and AC coefficients as well as the unpredicted AC coefficients of DCT blocks in intra macroblacks are scanned by one of three scans: alternate-horizontal, alternate-vertical, and zigzag. These processes are shown in Fig. 2.14 and they change the the two dimensional image to one dimensional image. The actual scan used depends on the coefficient predictions used. For example, if AC prediction is disabled, all blocks in that macroblock are zigzag-scanned. If DC prediction is enabled and DC prediction is selected from the vertically adjacent block, alternate-horizontal scan is used. Likewise, if AC prediction is enabled and DC prediction is selected from the horizontally adjacent block, alternate-vertical scan is used.

After the scan, the coefficients usually become a data sequence with many zeros at the end. This situation is good for run-length coding. A three dimensional variable length code (VLC) is used to encode the DCT events of intra blocks. An event is a combination of three items: last, run and level. The "last" means a last non-zero coefficients indication. The "run" indicates number of zero coefficients preceding the current nonzero coefficient. The "level" is the amplitude of the quantized coefficient. In inter blocks, the scanned coefficients using the zigzag scan are also coded by three dimensional VLC table like intra VLC but with code entries optimized for inter statistics. The decoding process follows

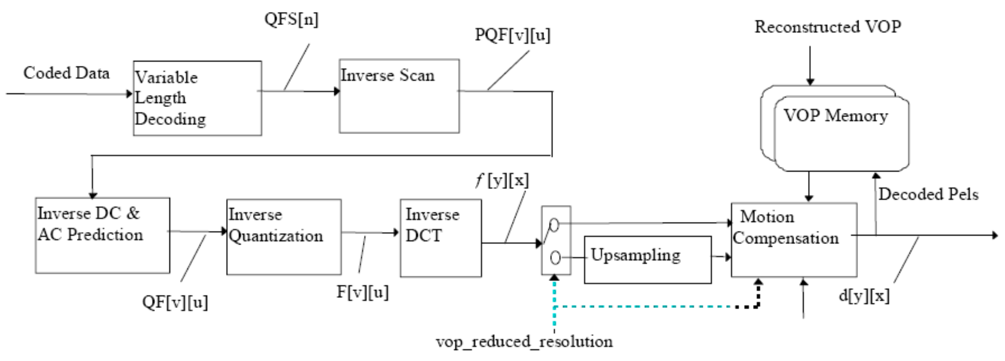


Figure 2.15: Structure of the texture decoder (from [3]).

the inverse sequence of the encoder and is described in Fig. 2.15.

2.4 MPEG-4 Audio Overview

Contents in this section are taken from [10] and [11].

MPEG-4 is based on the notion that the audio part of the audiovisual scene presented at the receiver is composed of one or more so-called audio objects. Different audio compression tools (codecs) are available to enable an efficiently coded representation of the audio objects in a scene. Natural audio objects, such as recorded speech and music, can be coded at bitrates typically ranging from 2 kb/s (for narrowband speech) to 64 kb/s/ch (for CD quality music) using parametric speech coding (HVXC), CELP-based speech coding, parametric audio coding (HILN) or transform-based general audio coding (AAC, TwinVQ). The natural audio and speech coding tools support bitrate scalability, also known as embedded coding. In addition, the parametric coding tools also provide speed and pitch change functionality in the decoder. Synthetic audio objects can be represented using a Text-To-Speech Interface (TTSI) or the Structured Audio (SA) synthesis tools. Other uses of the SA tools are adding effects, like reverberation, and mixing different audio objects to compose the final “audio scene” that is presented to the listener.

Figure 2.16 shows the block diagram of a complete MPEG-4 Audio decoder. It includes the decoding tools for the audio objects defined in Part 3 (Audio) of the MPEG-4

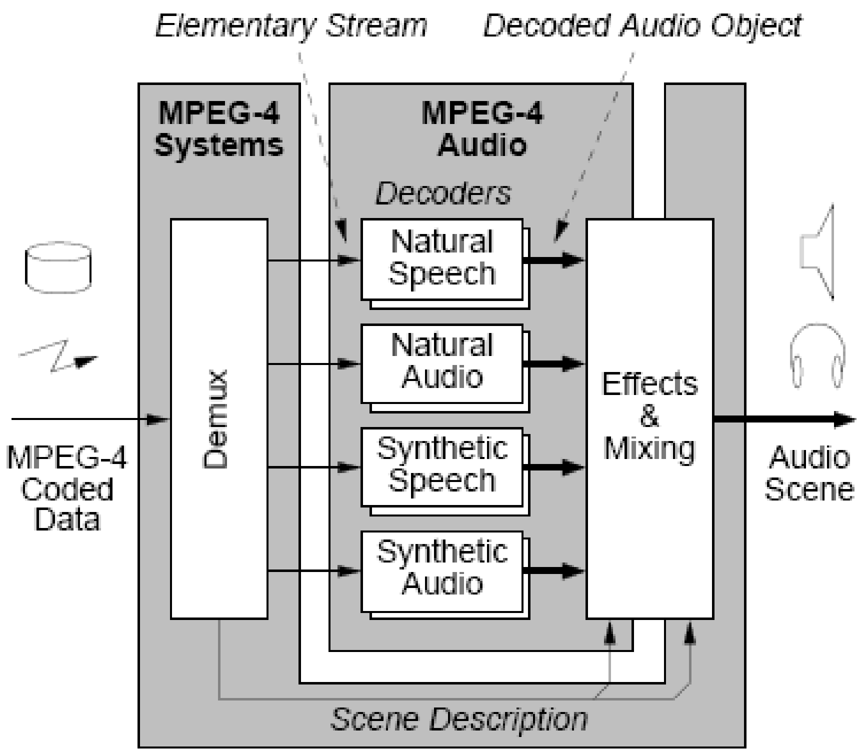


Figure 2.16: Block diagram of a complete MPEG-4 audio decoder (from [11]).

Standard as well as bitstream demultiplexing and scene composition defined in the Systems part.

The information needed to decode an audio object at the decoder is conveyed by means of a so-called elementary stream (ES), as shown in Fig. 2.17. In case of bitrate scalable configurations, a base-layer ES and one or more enhancement-layers ES(s) are used. The initial ES Descriptor contains an AudioSpecifcConfig element, which carries the audio object type (AOT) and further information required to instantiate the required audio decoder. Using the ES ID, the ES Descriptor points to the stream of the actual compressed audio data, which is a sequence of so-called access units (AU), i.e., decodable bitstream frames. A common audio object type is AAC LC, which indicates that MPEG-4 Advanced Audio Coding (AAC) in its low-complexity (LC) version is used. This is the primary general audio coder in MPEG-4, which is a compatible extension of the MPEG-2 AAC transform-based audio coder.

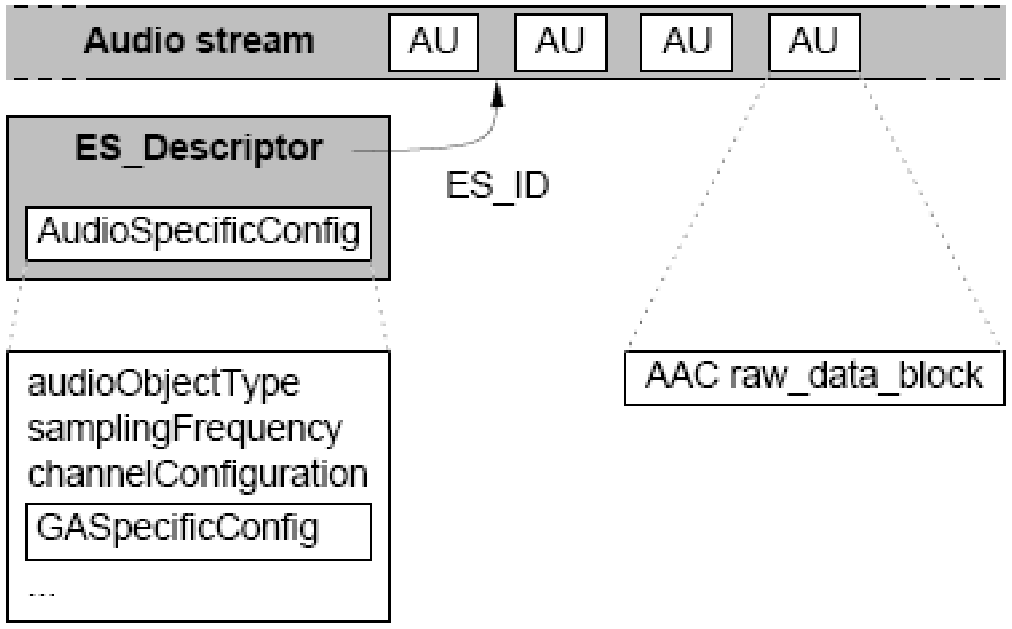


Figure 2.17: MPEG-4 elementary stream conveying an audio object (from [11]).

2.5 Profiles and Levels

Contents in this section are mainly taken from [3] and [10].

Although there are many tools in the MPEG-4 standard, not every MPEG-4 decoder will have to implement all of them. Similar to MPEG-2, profiles and levels are defined as sub-sets of the entire bitstreams syntax of all the tools. The purpose of defining conformance points in the form of profiles and levels is to facilitate interchange of bitstreams among different applications.

There are eight profiles defined in the visual part of MPEG-4 standard: simple, core, main, simple scalable, animated & mesh, basic animated texture, still scalable texture profile, and simple face. The detailed definitions are given in Fig 2.18. The simple profile of MPEG-4 is very similar to the coding method in H.263. The difference is that the simple profile has error resilience but does not have B-frame coding. The simple scalable profile is the same as simple profile, but with the rectangular scalability added. The core profile is the profile with all tools of the simple profile, temporal scalability, B-VOP coding and binary shape coding. The main profile is the profile with all tools in core profile, gray shape coding, interlace and sprite coding. The other profiles are for particular purposes, such as 2D dynamic mesh coding and facial animation coding.

MPEG-4 Audio provides several profiles to allow the optimal use of MPEG-4 in different applications. At the same time the number of profiles is kept as low as possible in order to maintain maximum interoperability. There are four profiles:

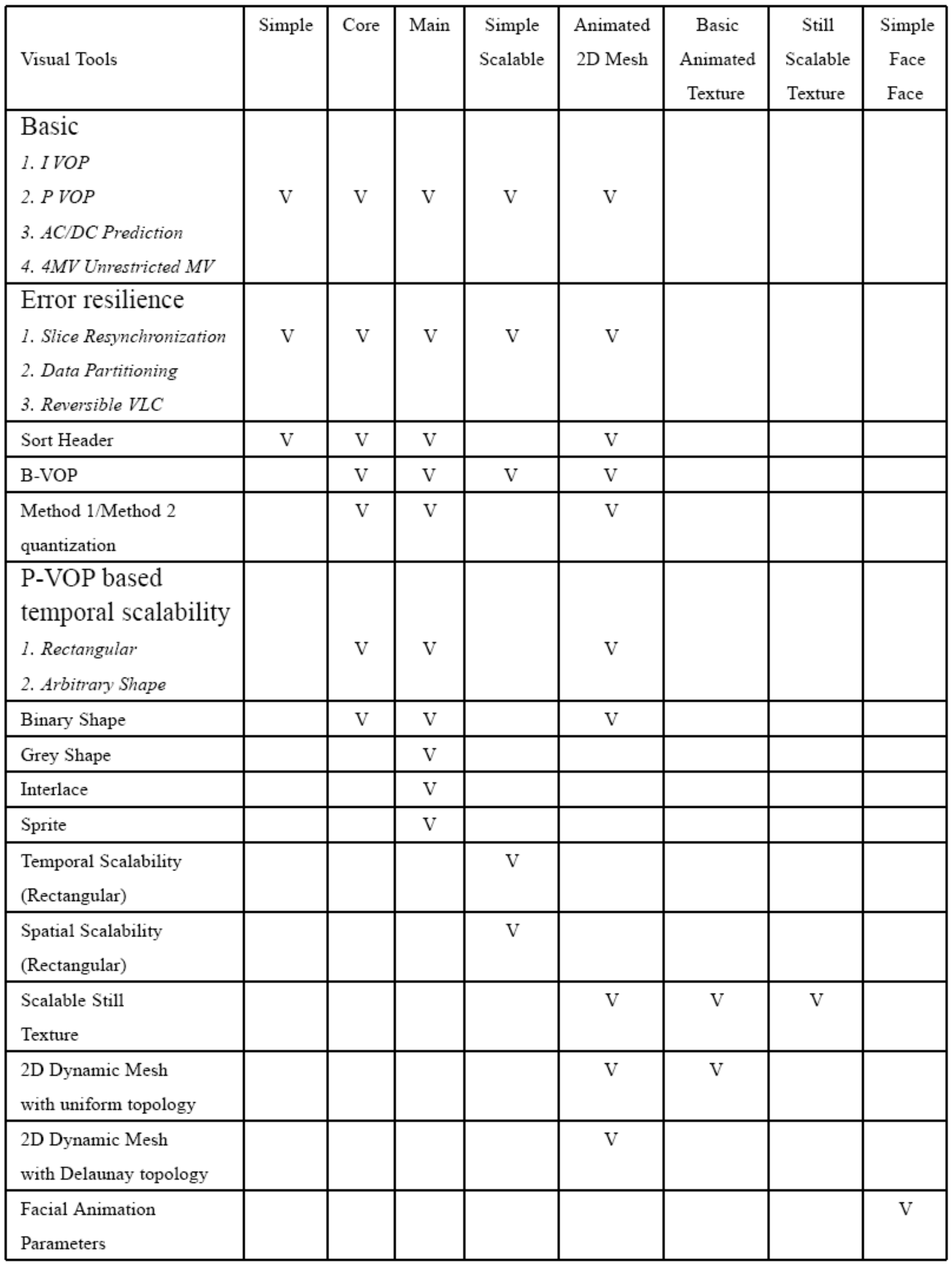
- The Speech Audio Profile provides a parametric speech coder, a CELP speech coder and a Text-To-Speech interface.

- The Synthesis Audio Profile provides the capability to generate sound and speech at very low bitrate.

- The Scalable Audio Profile, a superset of the Speech Profile, is suitable for scalable coding of speech and music and for different transmission methods, such as Internet and Digital Broadcasting.

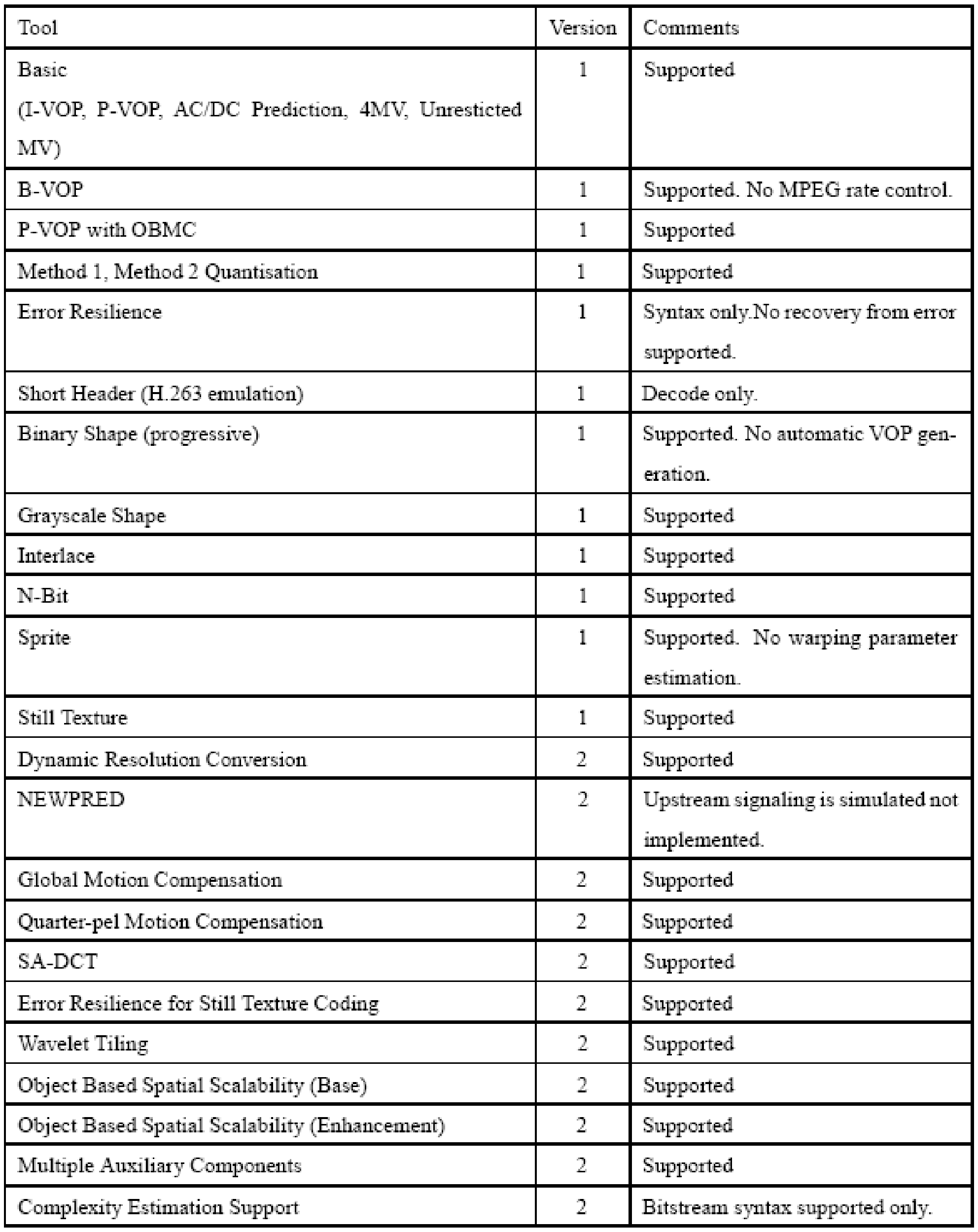
- The Main Audio Profile is a rich superset of the three previous profiles (scalable, speech, synthesis) containing tools for both natural and synthetic audio.

In our implement, the visual object decoder is the Microsoft MPEG-4 Visual Reference Software ([12]). Figure 2.19 indicates which tools are supported in this reference software. The functionalities defined by the reference software conforms to main and simple scalable profiles of MPEG-4. System layer and 3D/SNHC part are not included. For our application, we make use of the binary shape object coding of the reference software, belonging to main profile of MPEG-4. The audio object decoder is the Freeware Advance Audio Decoder (FAAD2) which include HE, LC, MAIN and LTP profile, MPEG-2 and MPEG-4 AAC decoder. We only use the MAIN profile for our audio streams.



Visual Tools	Simple	Core	Main	Simple Scalable	Animated 2D Mesh	Basic Animated Texture	Still Scalable Texture	Simple Face Face
Basic *1. I VOP* *2. P VOP* *3. AC/DC Prediction* *4. 4MV Unrestricted MV*	V	V	V	V	V			
Error resilience *1. Slice Resynchronization* *2. Data Partitioning* *3. Reversible VLC*	V	V	V	V	V			
Sort Header	V	V	V		V			
B-VOP		V	V	V	V			
Method 1/Method 2 quantization		V	V		V			
P-VOP based temporal scalability *1. Rectangular* *2. Arbitrary Shape*		V	V		V			
Binary Shape		V	V		V			
Grey Shape			V					
Interlace			V					
Sprite			V					
Temporal Scalability (Rectangular)				V				
Spatial Scalability (Rectangular)				V				
Scalable Still Texture					V	V	V	
2D Dynamic Mesh with uniform topology					V	V		
2D Dynamic Mesh with Delaunay topology					V			
Facial Animation Parameters								V

Figure 2.18: Video profiles and tools defined in MPEG-4 (from [3]).



Tool	Version	Comments
Basic (I-VOP, P-VOP, AC/DC Prediction, 4MV, Unresticted MV)	1	Supported
B-VOP	1	Supported. No MPEG rate control.
P-VOP with OBMC	1	Supported
Method 1, Method 2 Quantisation	1	Supported
Error Resilience	1	Syntax only.No recovery from error supported.
Short Header (H.263 emulation)	1	Decode only.
Binary Shape (progressive)	1	Supported. No automatic VOP generation.
Grayscale Shape	1	Supported
Interlace	1	Supported
N-Bit	1	Supported
Sprite	1	Supported. No warping parameter estimation.
Still Texture	1	Supported
Dynamic Resolution Conversion	2	Supported
NEWPRED	2	Upstream signaling is simulated not implemented.
Global Motion Compensation	2	Supported
Quarter-pel Motion Compensation	2	Supported
SA-DCT	2	Supported
Error Resilience for Still Texture Coding	2	Supported
Wavelet Tiling	2	Supported
Object Based Spatial Scalability (Base)	2	Supported
Object Based Spatial Scalability (Enhancement)	2	Supported
Multiple Auxiliary Components	2	Supported
Complexity Estimation Support	2	Bitstream syntax supported only.

Figure 2.19: Functionalities of the Microsoft MPEG-4 Video Reference Software (from [12]).

Chapter 3

Overview of the RTP

3.1 Introduction

Originally, we intended to use the DMIF to implement the network interface in our system. Unfortunately, the available DMIF software [20] is very difficult to use. Moreover interest in DMIF has waned in recent MPEG activities and the DMIF software has suffered a maintenance problem. Therefore we decide to use RTP (Real-time Transport Protocol) for our network interface.

The RTP provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video. These services include payload type identification, sequence numbering, timestamping and delivery monitoring. Applications typically run RTP on top of UDP to make use of its multiplexing and checksum services; both protocols contribute parts of the transport protocol functionality. RTP supports data transfer to multiple destinations using multicast distribution if provided by the underlying network. RTP itself does not provide any mechanism to ensure timely delivery or provide other quality-of-service guarantees, but relies on lower-layer services to do so. It does not guarantee delivery or prevent out-of-order delivery, nor does it assume that the underlying network is reliable and delivers packets in sequence. The sequence numbers included in RTP allow the receiver to reconstruct the sender's packet sequence, but sequence numbers might also be used to determine the proper location of a packet, for example in video decoding, without necessarily decoding packets in sequence. Although RTP is primarily designed to satisfy the needs of multiparticipant multimedia conferences, it is not limited to that particular application. Other applications may also find RTP applicable. The following text in this chapter is mainly taken from [13] and [14].

3.2 Definitions Used in the RTP

- RTP payload: The data transported by RTP in a packet, for example audio samples or compressed video data.
- RTP packet: A data packet consisting of the fixed RTP header, a possibly empty list of contributing sources, and the payload data. Some underlying protocols may require an encapsulation of the RTP packet to be defined. Typically one packet of the underlying protocol contains a single RTP packet, but several RTP packets may be contained if permitted by the encapsulation method.
- RTCP packet: A control packet consisting of a fixed header part similar to that of RTP data packets, followed by structured elements that vary depending upon the RTCP packet type. Typically, multiple RTCP packets are sent together as a compound RTCP packet in a single packet of the underlying protocol; this is enabled by the length field in the fixed header of each RTCP packet.
- RTP session: The association among a set of participants communicating with RTP. For each participant, the session is defined by a particular pair of destination transport addresses. The destination transport address pair may be common for all participants, as in the case of IP multicast, or may be different for each, as in the case of individual unicast network addresses plus a common port pair. In a multimedia session, each medium is carried in a separate RTP session with its own RTCP packets. The multiple RTP sessions are distinguished by different port number pairs and/or different multicast addresses.
- Synchronization source (SSRC): The source of a stream of RTP packets, identified by a 32-bit numeric SSRC identifier carried in the RTP header so as not to be dependent upon the network address. All packets from a synchronization source form

part of the same timing and sequence number space, so a receiver groups packets by synchronization source for playback. Examples of synchronization sources include the sender of a stream of packets derived from a signal source such as a microphone or a camera, or an RTP mixer. A synchronization source may change its data format, e.g., audio encoding, over time. The SSRC identifier is a randomly chosen value meant to be globally unique within a particular RTP session. A participant need not use the same SSRC identifier for all the RTP sessions in a multimedia session; the binding of the SSRC identifiers is provided through RTCP. If a participant generates multiple streams in one RTP session, for example from separate video cameras, each must be identified as a different SSRC.

- Contributing source (CSRC): A source of a stream of RTP packets that has contributed to the combined stream produced by an RTP mixer. The mixer inserts a list of the SSRC identifiers of the sources that contributed to the generation of a particular packet into the RTP header of that packet. This list is called the CSRC list. An example application is audio conferencing where a mixer indicates all the talkers whose speech was combined to produce the outgoing packet, allowing the receiver to indicate the current talker, even though all the audio packets contain the same SSRC identifier.

- Mixer: An intermediate system that receives RTP packets from one or more sources, possibly changes the data format, combines the packets in some manner and then forwards a new RTP packet. Since the timing among multiple input sources will not generally be synchronized, the mixer will make timing adjustments among the streams and generate its own timing for the combined stream. Thus, all data packets originating from a mixer will be identified as having the mixer as their synchronization source.

- Translator: An intermediate system that forwards RTP packets with their synchronization source identifier intact. Examples of translators include devices that convert encodings without mixing, replicators from multicast to unicast, and application level filters in firewalls.

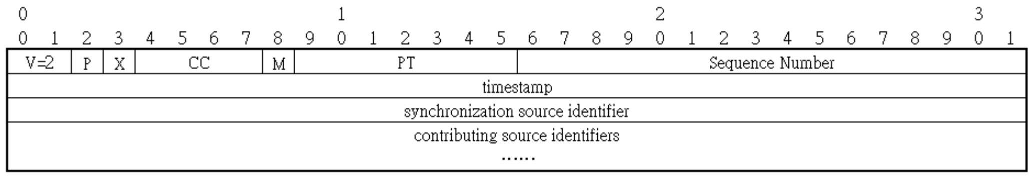


Figure 3.1: Format of the RTP header (from [13]).

- Monitor: An application that receives RTCP packets sent by participants in an RTP session, in particular the reception reports, and estimates the current quality of service for distribution monitoring, fault diagnosis and long-term statistics. The monitor function is likely to be built into the applications participating in the session, but may also be a separate application that does not otherwise participate and does not send or receive the RTP data packets. These are called third party monitors.

- Non-RTP means: Protocols and mechanisms that may be needed in addition to RTP to provide a usable service. In particular, for multimedia conferences, a conference control application may distribute multicast addresses and keys for encryption, negotiate the encryption algorithm to be used, and define dynamic mappings between RTP payload type values and the payload formats they represent for formats that do not have a predefined payload type value. For simple applications, electronic mail or a conference database may also be used.

3.3 RTP Data Transfer Protocol

Figure 3.1 shows the format of RTP header. The first twelve octets are present in every RTP packet, while the list of CSRC identifiers is present only when inserted by a mixer. These fields have the following meaning:

- Version (V): This field identifies the version of RTP.

- Padding (P): If the padding bit is set, the packet contains one or more additional padding octets at the end which are not part of the payload. The last octet of the

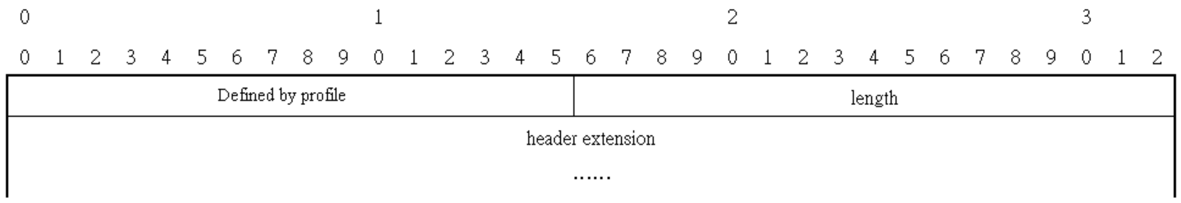


Figure 3.2: The RTP header extension (from [13]).

padding contains a count of how many padding octets should be ignored. Padding may be needed by some encryption algorithms with fixed block sizes or for carrying several RTP packets in a lower-layer protocol data unit.

- Extension (X): An extension mechanism provides to allow individual implementations to experiment with new payload-format-independent functions that require additional information to be carried in the RTP data packet header. If the X bit in the RTP header is one, a variable-length header extension shown in Fig. 3.2 is appended to the RTP header, following the CSRC list if present. The header extension contains a 16-bit length field that counts the number of 32-bit words in the extension, excluding the four-octet extension header. To allow multiple interoperating implementations to each experiment independently with different header extensions, or to allow a particular implementation to experiment with more than one type of header extension, the first 16 bits of the header extension are left open for distinguishing identifiers or parameters.

- CSRC count (CC): The CSRC count contains the number of CSRC identifiers that follow the fixed header.

- Marker (M): The interpretation of the marker is defined by a profile. It is intended to allow significant events such as frame boundaries to be marked in the packet stream. A profile may define additional marker bits or specify that there is no marker bit by changing the number of bits in the payload type field.

- Payload type (PT): This field identifies the format of the RTP payload and determines its interpretation by the application. A profile specifies a default static mapping of payload type codes to payload formats. Additional payload type codes may be defined dynamically through non-RTP means. An initial set of default mappings for audio and video is specified in the companion profile Internet-Draft draft-ietf-avt-profile. An RTP sender emits a single RTP payload type at any given time; this field is not intended for multiplexing separate media streams.

- Sequence number: The sequence number increments by one for each RTP data packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence. The initial value of the sequence number is random (unpredictable) to make known-plaintext attacks on encryption more difficult, even if the source itself does not encrypt, because the packets may flow through a translator that does.

- Timestamp: The timestamp reflects the sampling instant of the first octet in the RTP data packet. The sampling instant must be derived from a clock that increments monotonically and linearly in time to allow synchronization and jitter calculations. The resolution of the clock must be sufficient for the desired synchronization accuracy and for measuring packet arrival jitter. The clock frequency is dependent on the format of data carried as payload and is specified statically in the profile or payload format specification that defines the format, or may be specified dynamically for payload formats defined through non-RTP means. If RTP packets are generated periodically, the nominal sampling instant as determined from the sampling clock is to be used, not a reading of the system clock.

- SSRC: The SSRC field identifies the synchronization source. This identifier is chosen randomly, with the intent that no two synchronization sources within the same RTP session will have the same SSRC identifier. Although the probability of multiple sources choosing the same identifier is low, all RTP implementations must be prepared to detect and resolve collisions.

- CSRC list: The CSRC list identifies the contributing sources for the payload contained in this packet. The number of identifiers is given by the CC field. If there are more than 15 contributing sources, only 15 may be identified. CSRC identifiers are inserted by mixers, using the SSRC identifiers of contributing sources. For example, for audio packets the SSRC identifiers of all sources that were mixed together to create a packet are listed, allowing correct talker indication at the receiver.

3.4 RTCP: RTP Control Protocol

The RTP control protocol (RTCP) is based on the periodic transmission of control packets to all participants in the session, using the same distribution mechanism as the data packets. The underlying protocol must provide multiplexing of the data and control packets, for example using separate port numbers with UDP. RTCP performs four functions:

1. The primary function is to provide feedback on the quality of the data distribution. This is an integral part of the RTP's role as a transport protocol and is related to the flow and congestion control functions of other transport protocols. The feedback may be directly useful for control of adaptive encodings, but experiments with IP multicasting have shown that it is also critical to get feedback from the receivers to diagnose faults in the distribution. With a distribution mechanism like IP multicast, it is also possible for an entity such as a network service provider who is not otherwise involved in the session to receive the feedback information and act as a third-party monitor to diagnose network problems. This feedback function is performed by the RTCP sender and receiver reports.

2. RTCP carries a persistent transport-level identifier for an RTP source called the canonical name or CNAME. Since the SSRC identifier may change if a conflict is discovered or a program is restarted, receivers require the CNAME to keep track of each participant. Receivers also require the CNAME to associate multiple data streams from a given participant in a set of related RTP sessions, for example to synchronize audio and video.

3. The first two functions require that all participants send RTCP packets, and therefore the rate must be controlled in order to scale up to a large number of participants

for RTP. By having each participant send its control packets to all the others, each can independently observe the number of participants. This number is used to calculate the rate at which the packets are sent.

4. A fourth, optional function is to convey minimal session control information, for example participant identification to be displayed in the user interface. This is most likely to be useful in "loosely controlled" sessions where participants enter and leave without membership control or parameter negotiation. RTCP serves as a convenient channel to reach all the participants, but it is not necessarily expected to support all the control communication requirements of an application. Sometimes a higher-level session control protocol may be needed.

Functions 1-3 are mandatory when RTP is used in the IP multicast environment, and are recommended for all environments. RTP application designers are advised to avoid mechanisms that can only work in unicast mode and will not scale to larger numbers.

3.5 RTP Translators and Mixers

A RTP translator/mixer connects two or more transport-level "clouds." Typically, each cloud is defined by a common network and transport protocol, multicast address or pair of unicast addresses, and transport level destination port. Network-level protocol translators, such as IP version 4 to IP version 6, may be present within a cloud invisibly to RTP. One system may serve as a translator or mixer for a number of RTP sessions, but each is considered a logically separate entity.

In order to avoid creating a loop when a translator or mixer is installed, the following rules must be observed:

- Each of the clouds connected by translators and mixers participating in one RTP session either must be distinct from all the others in at least one of these parameters (protocol, address, port), or must be isolated at the network level from the others.

- A derivative of the first rule is that there must not be multiple translators or mixers

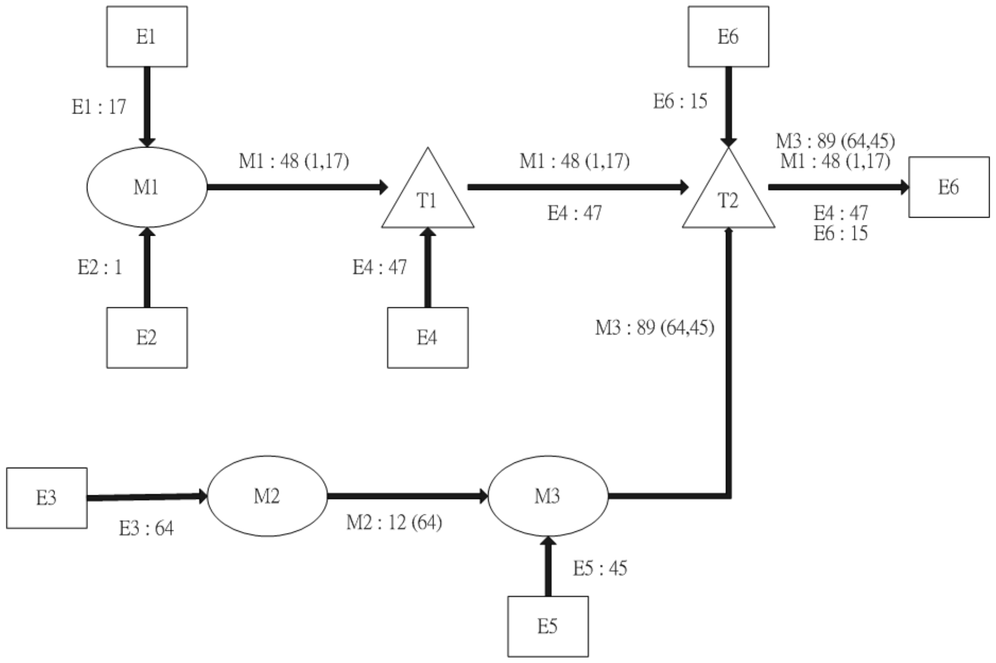


Figure 3.3: Sample RTP network with end systems, mixers and translators (from [13]).

connected in parallel unless by some arrangement they partition the set of sources to be forwarded.

Similarly, all RTP end systems that can communicate through one or more RTP translators or mixers share the same SSRC space, that is, the SSRC identifiers must be unique among all these end systems. There may be many varieties of translators and mixers designed for different purposes and applications. For some applications, it may be acceptable for a mixer not to identify sources in the CSRC list. However, this introduces the danger that loops involving those sources could not be detected.

A collection of mixers and translators is shown in Fig. 3.3 to illustrate their effect on SSRC and CSRC identifiers. In the figure, end systems are shown as rectangles (named E), translators as triangles (named T) and mixers as ovals (named M). The notation “M1: 48(1,17)” designates a packet originating at mixer M1, identified with M1’s (random) SSRC value of 48 and two CSRC identifiers, 1 and 17, copied from the SSRC identifiers of packets from E1 and E2.

3.6 Security

Lower layer protocols may eventually provide all the security services that may be desired for applications of RTP, including authentication, integrity, and confidentiality. These services have recently been specified for IP. Since the need for a confidentiality service is well established in the initial audio and video applications that are expected to use RTP, a confidentiality service is defined for use with RTP and RTCP until lower layer services are available. The overhead on the protocol for this service is low, so the penalty will be minimal if this service is obsoleted by lower layer services in the future.

Alternatively, other services, other implementations of services and other algorithms may be defined for RTP in the future if warranted. A profile may specify which services and algorithms should be offered by applications, and may provide guidance as to their appropriate use.

Confidentiality means that only the intended receivers can decode the received packets; for others, the packet contains no useful information. Confidentiality of the content is achieved by encryption. When encryption of RTP or RTCP is desired, all the octets that will be encapsulated for transmission in a single lower-layer packet are encrypted as a unit. For RTCP, a 32-bit random number is prepended to the unit before encryption to deter known plaintext attacks. For RTP, no prefix is required because the sequence number and timestamp fields are initialized with random offsets.

The default encryption algorithm is the data encryption standard (DES) algorithm in cipher block chaining (CBC) mode, except that padding to a multiple of 8 octets is indicated for the P bit. Implementations that support encryption should always support the DES algorithm in CBC mode as the default to maximize interoperability. This method is chosen because it has been demonstrated to be easy and practical to use in experimental audio and video tools in operation on the Internet. Other encryption algorithms may be specified dynamically for a session by non-RTP means.

3.7 JRTPLIB

The jrtplib software [15] developed at the Expertise Centre for Digital Media (EDM) is an object-oriented library written in C++ to help developers using RTP. The latest version of the library is 3.1.0 (October 2004) and the program is totally free. The 3.x.x series is a complete rewrite of the library and is meant to be compliant with RFC 3550 [14]. The library makes it possible for the user to send and receive data using RTP, without worrying about SSRC collisions, scheduling and transmitting RTCP data, etc. The user only needs to provide the library with the payload data to be sent and the library gives the user access to incoming RTP and RTCP data.

The RTPSession class in this library produces the necessary functions for sending RTP data and handles the RTCP part internally. In addition, the code which is specific for the underlying protocol by which RTP packets are transported, is bundled in a class which inherits its interface from a class called RTPTransmitter. This makes it easy for different underlying protocols to be supported. There is support for UDP over IPv4 and UDP over IPv6 in the latest version. For applications such as a mixer or translator using the RTPSession class will not be a good solution. Other components can be used for this purpose: a transmission component, an SSRC table, an RTCP scheduler, etc. Using these, it should be much easier to build all kinds of applications.

We use the RTPSession class to receive data and use the TransmissionParams class to handle the transmission component of the UDP-over-IPv4. The RTPPacket class would get the data from the RTPSession class. The three major classes make it easy to establish the RTP network interface for our application.

Chapter 4

Integration of the MPEG-4 Video and Audio Decoders with RTP

4.1 Introduction

The MPEG-4 video and audio standards and the RTP protocol have been introduced above. Now we discuss how we integrate the decoders and play the decoded video and audio streams on PC, with the decoder input fed by a RTP receiver. Recall Fig. 1.2 which shows the structure of the receiver. The visual object decoder is from the Microsoft MPEG-4 Video Reference Software [12] which is a public source for encoding and decoding video stream using the MPEG-4 compression format. The encoder of this reference software has been optimized previously [8] but the decoder is not. The audio object decoder is the Freeware Audio Decoder (FAAD2) [16] written by M. Bakker. The jrtplib 3.1.0 software [15] is used to establish the RTP network interface.

There are two modes of application programming in Windows OS [18]: Console mode which uses the file I/O and GUI (Graphical User Interface) mode. The GUI mode is suitable for our application and it can be implemented using the Windows SDK (Software Development Kit) which is developed by Microsoft for high-level computer languages for ease of implementation in the GUI mode. The GUI mode can make the use of our system easy.

4.2 System Overview

Figure 4.1 shows the flow of our program. The GUI block creates the window and the user can input the different ports and the different positions of the different clients through the GUI. After getting these pieces of information, the GUI would create a number of decoder threads and get the information indicating which decoder is working currently. This information help the synchronization block to decide which one can handle the other decoders. For instance, if only the first decoder and the third decoder are working now, the third decoder would handle the the first decoder. At the same time, the visual decoders and the audio decoders begin to decode the data received through the RTP network interface. The next step is to compose the video and the audio. Finally the composite video would be displayed on the window created by the GUI and the composite audio played.

4.3 RTP Network Interface

To use RTP, first we have to create an RTPSession object. The default setting of the RTPSession class is such that the UDP-over-IPv4 transmission component will be used unless specified otherwise. In order to actually create the session, we must call the Create member function which takes two arguments. The first one is RTPSessionParams and specifies the general options for the session. One parameter of this class must be set explicitly. This parameter is the timestamp unit of the data and can be calculated by dividing a certain time interval (in seconds) by the number of sampling rate. Another argument of the Create member function is RTPTransmissionParams which describes the parameters for the transmission component. There can be serveral transmission components, so we choose RTPUDPv4TransmissionParams class which inherits RTPTransmissionParams for the UDP over IPv4 component. In addition, we must set our portbase to create the channels. Since each transmitter sends two streams (video and audio), we create two channels to receive video and audio streams. First we set a special port which is the same as the transmitter setting selected by the user and this special port is used for video. For audio, we add 100 to this special port number. Then we set the timestamp parameter to be 1 section per second for the video stream and set another timestamp parameter to be

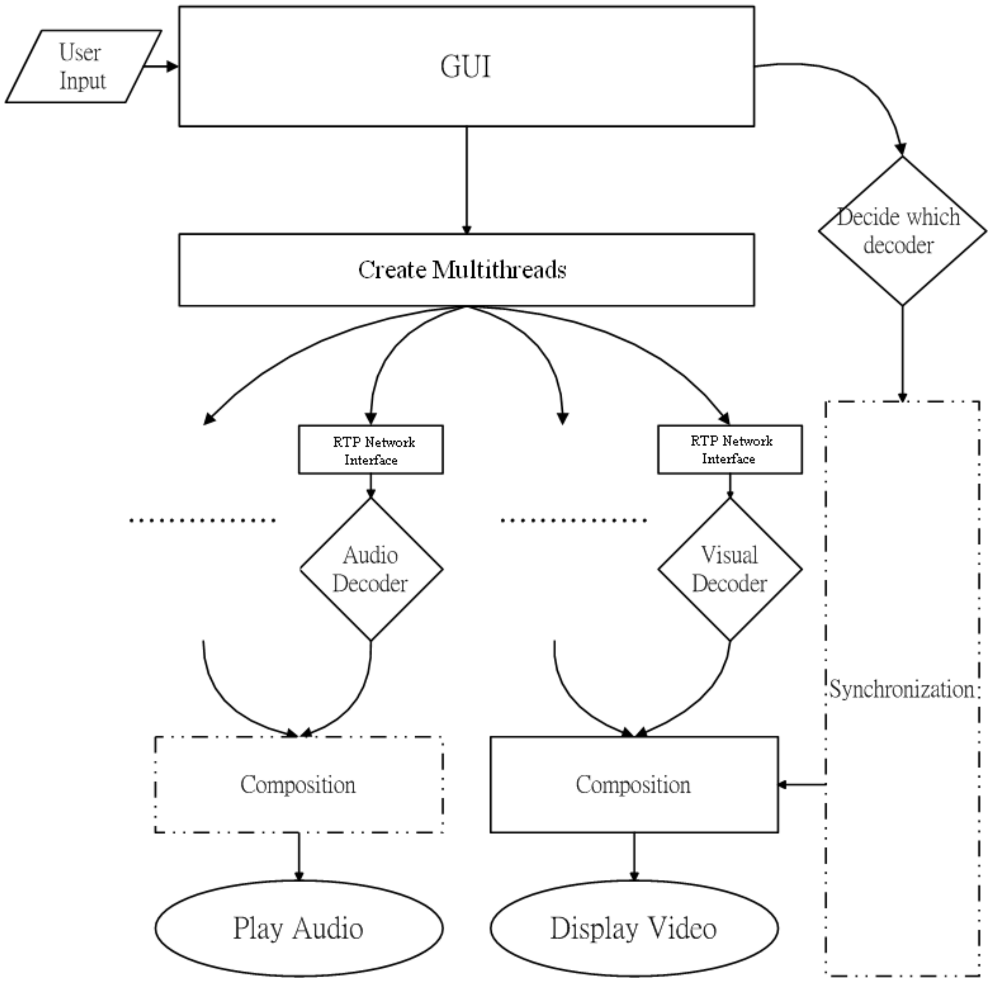
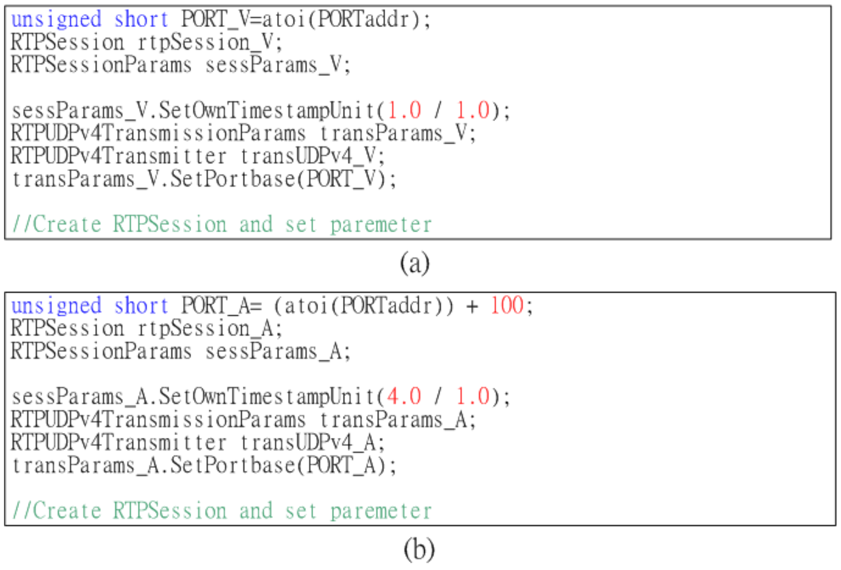


Figure 4.1: The flow diagram of the receiver.



```
unsigned short PORT_V=atoi(PORTaddr);
RTPSession rtpSession_V;
RTPSessionParams sessParams_V;

sessParams_V.SetOwnTimestampUnit(1.0 / 1.0);
RTPUDPv4TransmissionParams transParams_V;
RTPUDPv4Transmitter transUDPv4_V;
transParams_V.SetPortbase(PORT_V);

//Create RTPSession and set paremeter
```

(a)

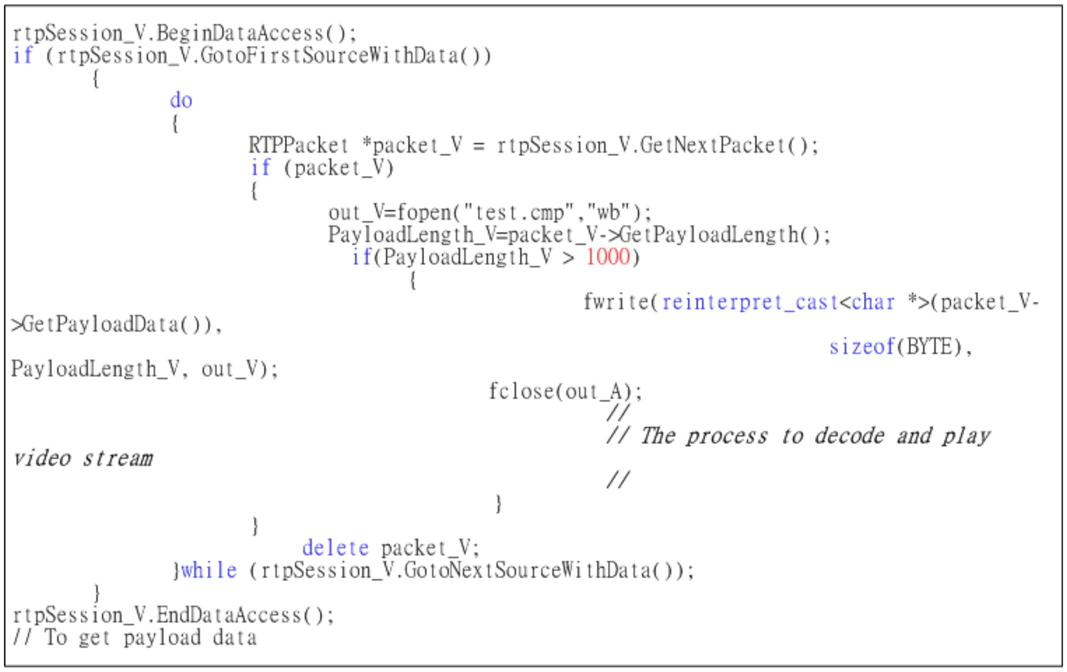
```
unsigned short PORT_A= (atoi(PORTaddr)) + 100;
RTPSession rtpSession_A;
RTPSessionParams sessParams_A;

sessParams_A.SetOwnTimestampUnit(4.0 / 1.0);
RTPUDPv4TransmissionParams transParams_A;
RTPUDPv4Transmitter transUDPv4_A;
transParams_A.SetPortbase(PORT_A);

//Create RTPSession and set paremeter
```

(b)

Figure 4.2: Related code of the RTPSession. (a) For video. (b) For audio.

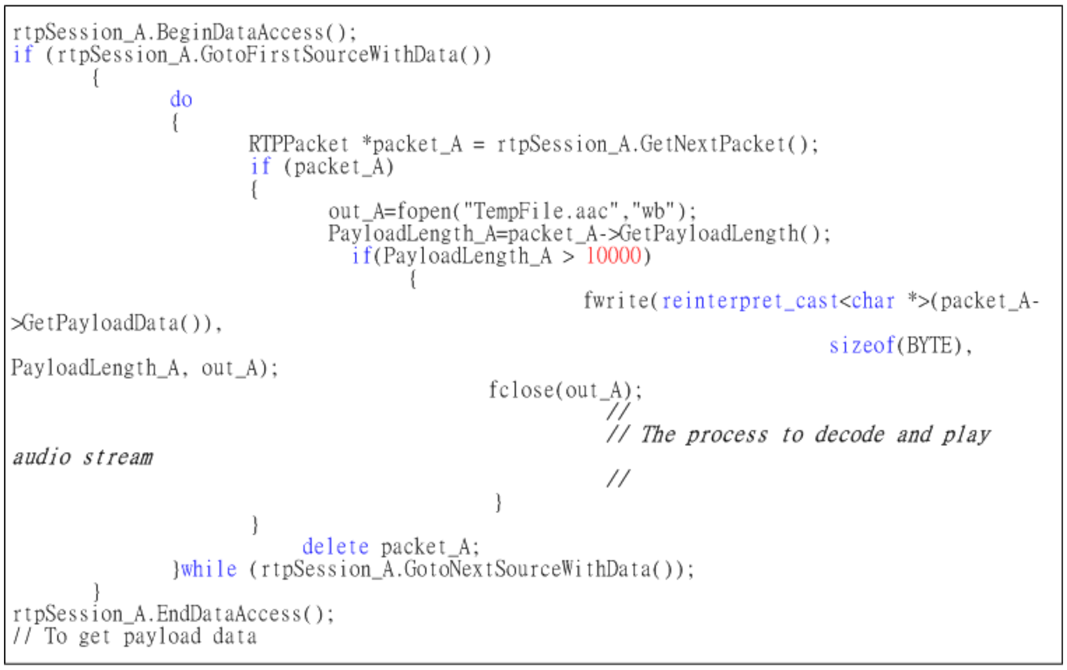
1 section per 4 seconds for the audio stream. The related code is shown in the Fig. 4.2.

After creating the sessions successfully, we add a main loop shown in Fig. 4.3. There are two functions called StartRTP and ResetRTP and related code is shown in Fig. 4.4. We can control the main loop through the two functions. For instance, if StartRTP function is called, the main loop is enabled. Then RTP network interface begins to receive the data. If ResetRTP function is called, the main loop is disabled that the RTP network interface stops. Then we can input new ports or new positions through the GUI. In this main loop, information about participants in the session, packet retrieval, etc., has to be done between calls to the RTPSession member functions BeginDataAccess and EndDataAccess. This ensures that the background thread does not try to change the same data that we are trying to access. Then we iterate over the participants using the GotoFirstSourceWithData and GotoNextSourceWithData member functions. Packets from the currently selected participant can be retrieved using the GetNextPacket member function which returns a pointer to an instance of the RTPPacket class. If the packet information is obtained with success, we can use the member functions of the RTPPacket class to receive the RTP payload and write the stream of payload into a temporary file. In this library, the network interface would lose the first two packets and receive the third packet successfully.



```
rtpSession_V.BeginDataAccess();
if (rtpSession_V.GotoFirstSourceWithData())
        {
                do
                {
                        RTPPacket *packet_V = rtpSession_V.GetNextPacket();
                        if (packet_V)
                        {
                                out_V=fopen("test.cmp","wb");
                                PayloadLength_V=packet_V->GetPayloadLength();
                                  if(PayloadLength_V > 1000)
                                        {
                                                        fwrite(reinterpret_cast<char *>(packet_V-
>GetPayloadData()),
                                                                                sizeof(BYTE),
PayloadLength_V, out_V);
                                                fclose(out_A);
                                                            //
                                                            // The process to decode and play
video stream
                                                            //
                                                }
                        }
                             delete packet_V;
                }while (rtpSession_V.GotoNextSourceWithData());
        }
rtpSession_V.EndDataAccess();
// To get payload data
```

(a)



```
rtpSession_A.BeginDataAccess();
if (rtpSession_A.GotoFirstSourceWithData())
        {
                do
                {
                        RTPPacket *packet_A = rtpSession_A.GetNextPacket();
                        if (packet_A)
                        {
                                out_A=fopen("TempFile.aac","wb");
                                PayloadLength_A=packet_A->GetPayloadLength();
                                  if(PayloadLength_A > 10000)
                                        {
                                                        fwrite(reinterpret_cast<char *>(packet_A-
>GetPayloadData()),
                                                                                sizeof(BYTE),
PayloadLength_A, out_A);
                                                fclose(out_A);
                                                            //
                                                            // The process to decode and play
audio stream
                                                            //
                                                }
                        }
                             delete packet_A;
                }while (rtpSession_A.GotoNextSourceWithData());
        }
rtpSession_A.EndDataAccess();
// To get payload data
```

(b)

Figure 4.3: Related code of the RTPClient. (a) For video. (b) For audio.

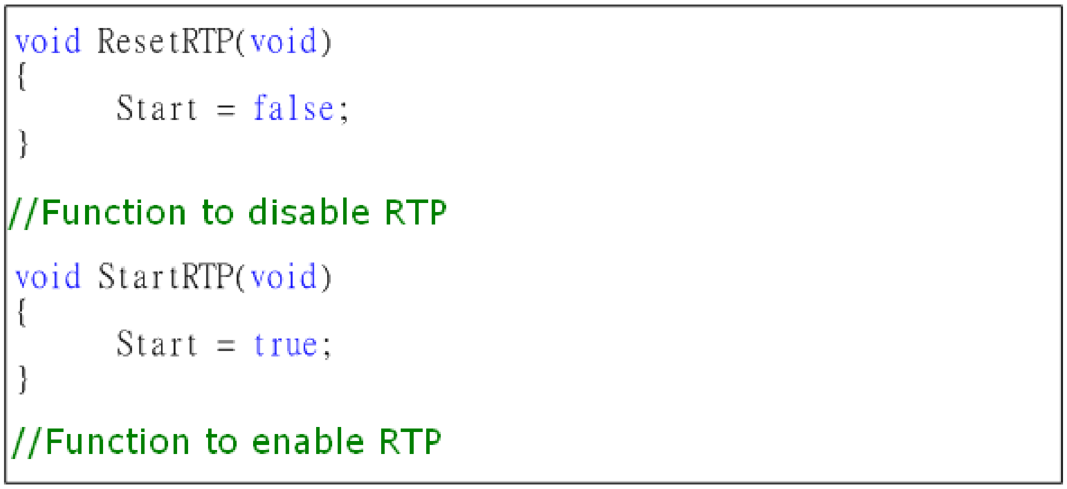

```
void ResetRTP(void)
{
        Start = false;
}

//Function to disable RTP

void StartRTP(void)
{
        Start = true;
}

//Function to enable RTP
```

Figure 4.4: Related code of the control functions of the RTP.

4.4 Video Decoding and Composition

4.4.1 Video decoding

Firstly, we use VTune to track the Microsoft reference decoder and the complexity analysis is shown in Fig. 4.5. As we can see, the major complexity rests in the dumpWithMask function since the function writes decoded data pixel by pixel into the buffer. The other major parts are functions doing motion compensation and prediction. Besides, the calculation of the binary shape form and DCT transform are also the more computation-intensive portions of the decoder. Although the decoder is fast enough to decode the video data in real time, if the number of decoders increases, the performance of the system will degrade.

In order to integrate the video decoder into our system, we modify the original program into a function called MPEG4VDecoder and give it two parameters. The first parameter is the handle of the window created by us. This handle can give the decoder some information to control the window and display the video stream. The second parameter is the index of the thread.

Another major function is dumpFrame function. This function writes the video stream which has been decoded by MPEG-4 decoder into a YUV file. Figure 4.6 shows the decla-

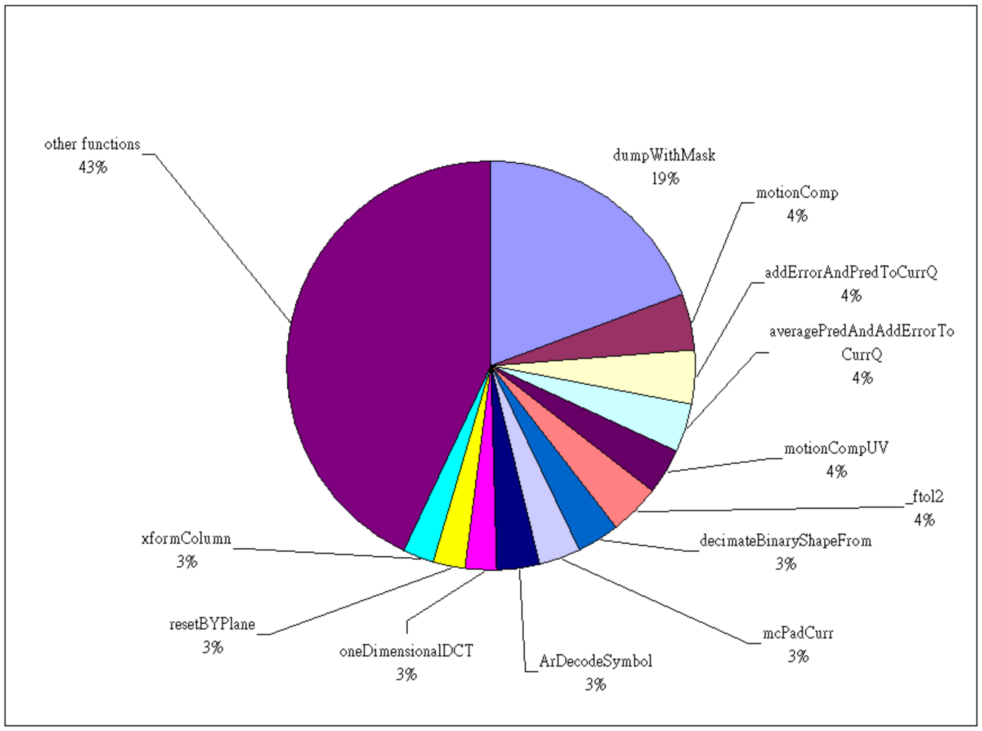
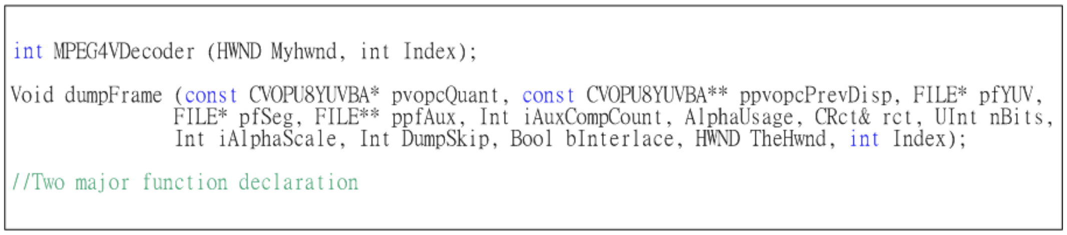


Figure 4.5: Complexity breakdown of the original decoder.



```
int MPEG4VDecoder (HWND Myhwnd, int Index);

Void dumpFrame (const CVOPU8YUVBA* pvopcQuant, const CVOPU8YUVBA** ppvopcPrevDisp, FILE* pfYUV,
                FILE* pfSeg, FILE** ppfAux, Int iAuxCompCount, AlphaUsage, CRct& rct, UInt nBits,
                Int iAlphaScale, Int DumpSkip, Bool bInterlace, HWND TheHwnd, int Index);

//Two major function declaration
```

Figure 4.6: The declaration of the two main functions.

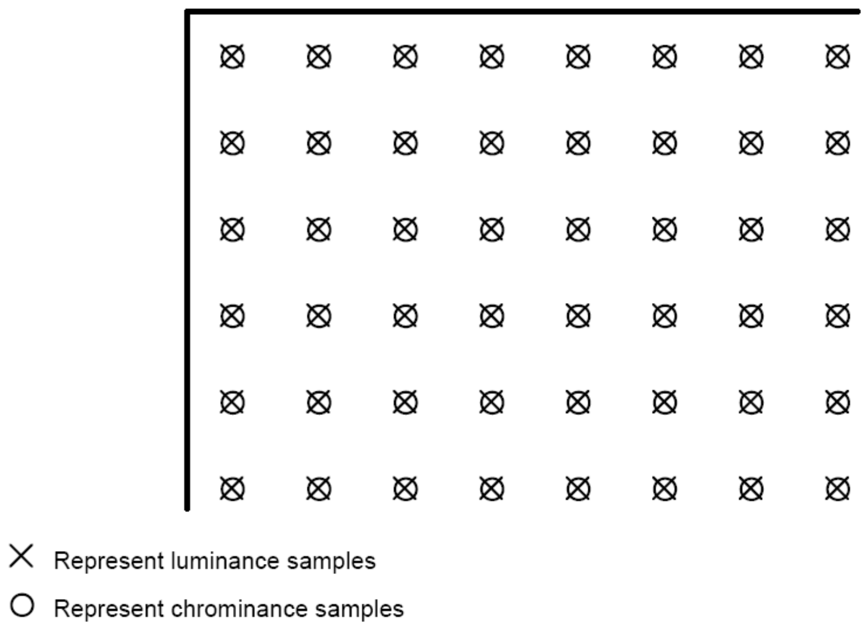


Figure 4.7: Positions of the luminance and chrominance samples in 4:4:4 format (from [3]).

ration of the two main functions. Firstly, ModifiedYdump or ModifiedYdumpWithMask member functions saves the luminance component into the tmpY buffer. ModifiedDump or ModifiedDumpWithMask member function saves the two chrominance components into tmpU and tmpV buffers. Since MPEG-4 standard use 4:2:0 format, we change this format to 4:4:4 format for display purpose. Figure 4.7 shows the position of the luminance and chrominance components in 4:4:4 format. To change the format, the ModifiedDump or ModifiedDumpWithMask member functions up-sample the U and V components and Fig. 4.8 shows the modified code.

After getting the luminance and chrominance components, we calculate the RGB values of each pixel and save these values into the rgbValue buffers. There are two possible situations before display of the rgbValue. The first is that only one rgbValue buffer is not empty. The other is that more than one decoder work at the same time and it would be discussed in the next subsection. The first situation means that only one user connects to the system and we just display this video stream. The ShowImageN function can implement

```
        for (Int i = rctRegionOfInterest.top; i < rctRegionOfInterest.bottom; i++)
        {
                for(x = 0; x<rctRegionOfInterest.width; x++)
                        if(ppxlcMask[x] == 0)
                        {
                                Bf[2*(i*352+x)]=(int (pxlZero))-128;
                                Bf[2*(i*352+x)+1]=(int (pxlZero))-128;
                                Bf[(2*i+1)*352+2*x]=(int (pxlZero))-128;
                                Bf[(2*i+1)*352+2*x+1]=(int (pxlZero))-128;
                        }
                        else if(iScale==256)
                        {
                                Bf[2*(i*352+x)]=(int (ppxlc[x]))-128;
                                Bf[2*(i*352+x)+1]=(int (ppxlc[x]))-128;
                                Bf[(2*i+1)*352+2*x]=(int (ppxlc[x]))-128;
                                Bf[(2*i+1)*352+2*x+1]=(int (ppxlc[x]))-128;
                        }
                        else
                        {
                                pxlTmp = (ppxlc[x] * iScale)>>8;
                                Bf[2*(i*352+x)]=(int (pxlTmp))-128;
                                Bf[2*(i*352+x)+1]=(int (pxlTmp))-128;
                                Bf[(2*i+1)*352+2*x]=(int (pxlTmp))-128;
                                Bf[(2*i+1)*352+2*x+1]=(int (pxlTmp))-128;
                        }
        }
//Up-sample chrominance components
```

Figure 4.8: Code for up-sampling of chrominance components.

```
void ShowImageN(HWND MyHwnd, int Index)
{
        int i, j;
        HDC hDC = GetDC(MyHwnd);
        if(Index == 0)
        {
                for(i=0;i<288;i++)
                        for(j=0;j<352;j++)
                        {
                                SetPixelV(hDC, j + PosX1, i + PosY1, rgbValue1[i * 352 + j]);
                        }
        }
        else
        {
                for(i=0;i<288;i++)
                        for(j=0;j<352;j++)
                        {
                                SetPixelV(hDC, j + PosX2, i + PosY2, rgbValue2[i * 352 + j]);
                        }
        }
        ReleaseDC(MyHwnd, hDC);
}
//Show video stream
```

Figure 4.9: Related code of showing one video stream.

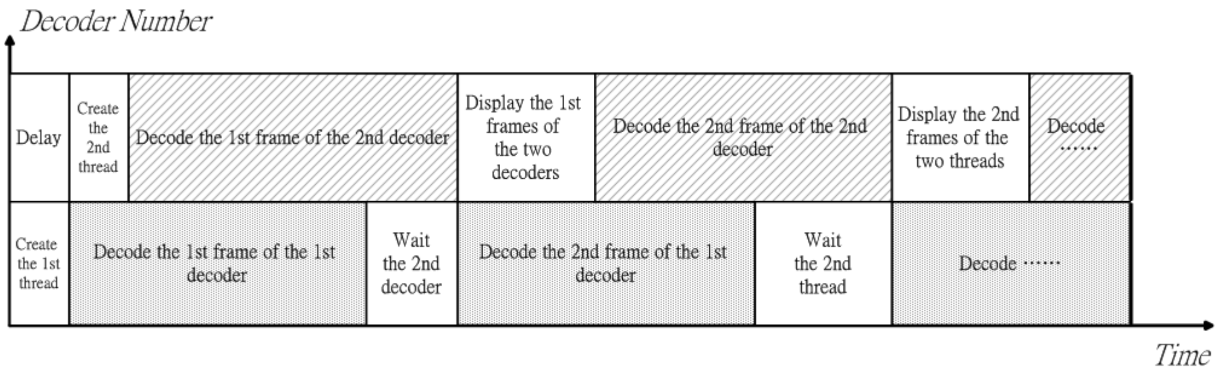
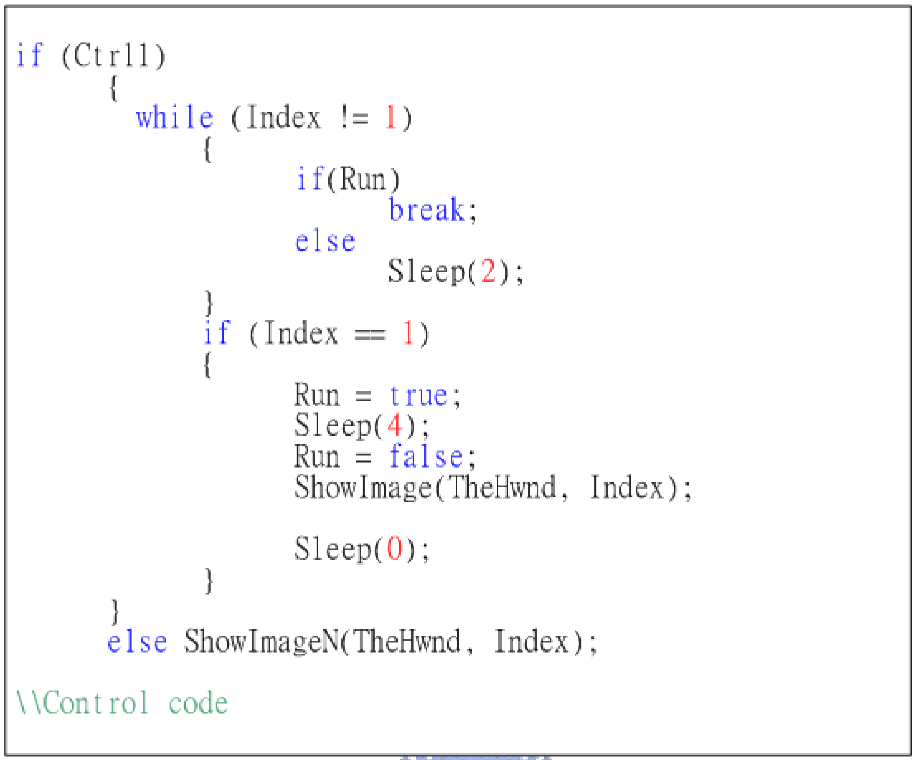

Decoder Number
Delay
Create the 2nd thread
Decode the 1st frame of the 2nd decoder
Display the 1st frames of the two decoders
Decode the 2nd frame of the 2nd decoder
Display the 2nd frames of the two threads
Decode ……
Create the 1st thread
Decode the 1st frame of the 1st decoder
Wait the 2nd decoder
Decode the 2nd frame of the 1st decoder
Wait the 2nd thread
Decode ……
Time

Figure 4.10: Relation between the two decoders in time domain.

this case as shown in Fig. 4.9. As we can see, the GetDC function gets the device context that defines a set of graphic objects and their associated attributes from the window handle. Then index parameter helps us to judge which thread is working and to play this video stream. The SetPixelV function provided by the Windows SDK library displays the RGB value pixel by pixel. When we do not need the device context anymore, it has to be released.

4.4.2 Synchronization and Composition

Now we discuss how to deal with the situation where there are two video inputs and two decoders work at the same time. In this situation, synchronization is an important problem since the two decoders may receive the data at different times. The solution is that the first decoder waits until the second decoder is ready. Figure 4.10 shows the relation between the two decoders and Fig. 4.11 shows the waiting function. As we can see, the Index parameter indicate which decoder, “0” means the first decoder and “1” means the second decoder. The two decoders would be sequentially created and the Ctrl0 for first decoder and Ctrl1 for second decoder would also be set to “TRUE.” The first decoder finishes decoding the first frame earlier than the second decoder and waits for some time. When the second decoder also finishes, it will send a message “Run” and display the two images in the meantime. The first decoder would break the waiting loop and continue to decode the next frame after getting the message.



```
if (Ctrl1)
        {
          while (Index != 1)
                {
                        if(Run)
                                break;
                        else
                                Sleep(2);
                }
                if (Index == 1)
                {
                        Run = true;
                        Sleep(4);
                        Run = false;
                        ShowImage(TheHwnd, Index);

                        Sleep(0);
                }
        }
        else ShowImageN(TheHwnd, Index);

\\Control code
```

Figure 4.11: Related code for waiting the other decoder to finish.

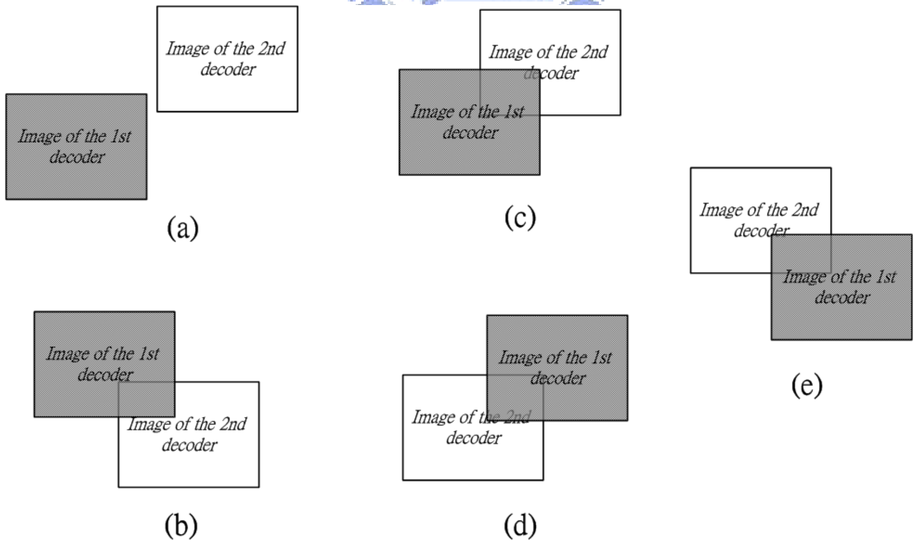
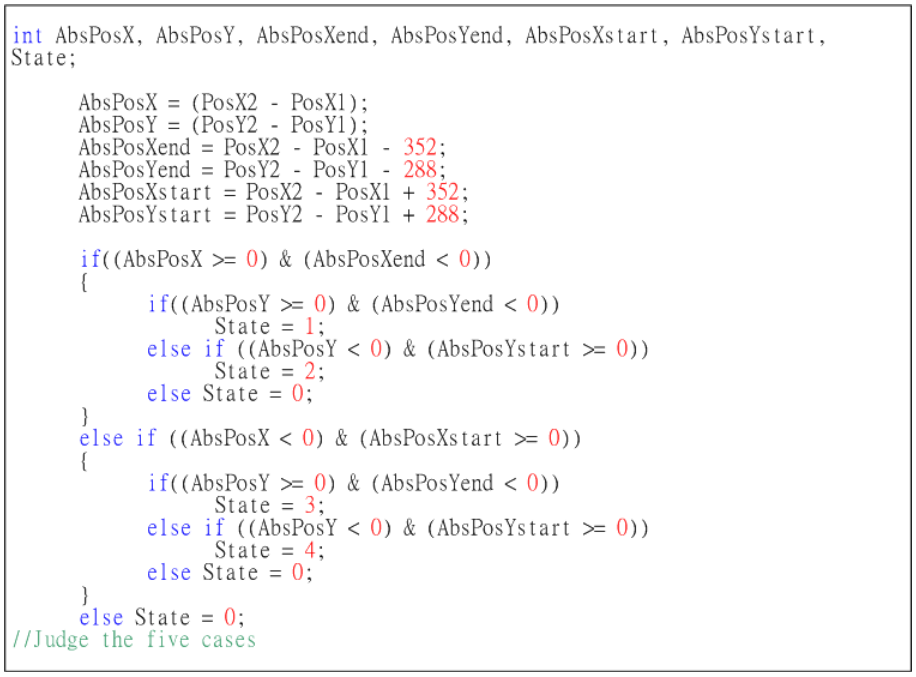


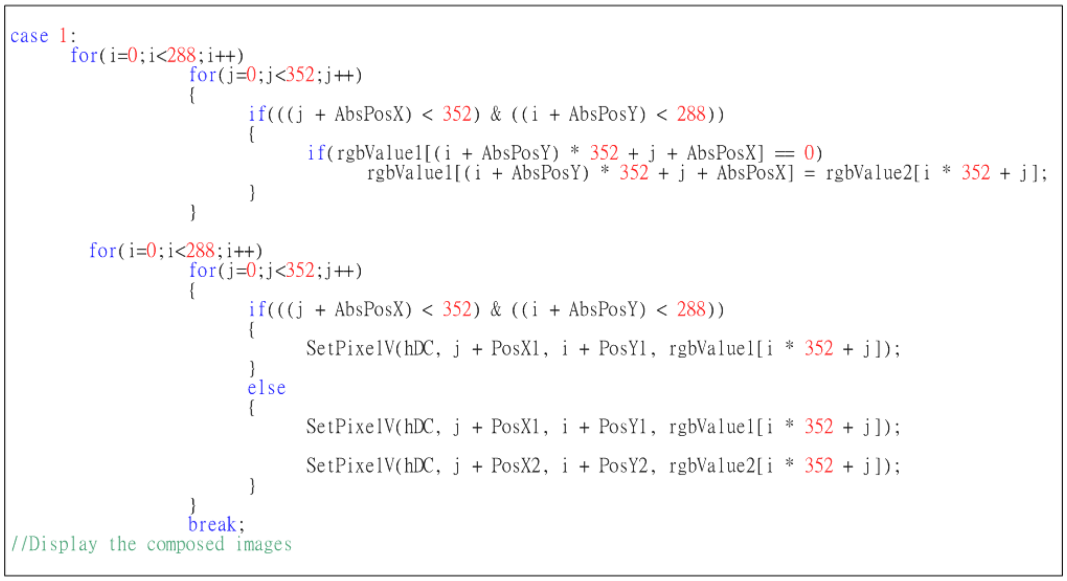
Figure 4.12: Different compositions of the two images. (a) Case 0, no overlap. (b) Case 1, with overlap. (c) Case 2, with overlap. (d) Case 3, with overlap. (e) Case 4, with overlap.



```
int AbsPosX, AbsPosY, AbsPosXend, AbsPosYend, AbsPosXstart, AbsPosYstart,
State;

      AbsPosX = (PosX2 - PosX1);
      AbsPosY = (PosY2 - PosY1);
      AbsPosXend = PosX2 - PosX1 - 352;
      AbsPosYend = PosY2 - PosY1 - 288;
      AbsPosXstart = PosX2 - PosX1 + 352;
      AbsPosYstart = PosY2 - PosY1 + 288;

      if((AbsPosX >= 0) & (AbsPosXend < 0))
      {
            if((AbsPosY >= 0) & (AbsPosYend < 0))
                  State = 1;
            else if ((AbsPosY < 0) & (AbsPosYstart >= 0))
                  State = 2;
            else State = 0;
      }
      else if ((AbsPosX < 0) & (AbsPosXstart >= 0))
      {
            if((AbsPosY >= 0) & (AbsPosYend < 0))
                  State = 3;
            else if ((AbsPosY < 0) & (AbsPosYstart >= 0))
                  State = 4;
            else State = 0;
      }
      else State = 0;
//Judge the five cases
```

Figure 4.13: Code to decide the five composition situations.

The last step is to compose the two images decoded by the two decoders. There are relative spatial relations between the images as shown in Fig. 4.12. Case 0 is where there is no overlap between the two images. Cases 1–4 are where there are some overlap. Figure 4.13 shows how the ImageShow function decides the five composition cases and Fig. 4.14 shows an example of the code included by the ImageShow function to display the composed images. We declare six kinds of the variables to determine the five cases. The AbsPosX variable decides that the image decoded by the second decoder is to the left or the right side of the image decoded by the first decoder. The AbsPosY variable decides that the image decoded by the second decoder is higher or lower than the image decoded by the first decoder. The last four variables help us to judge that the two images overlap or not. If there is some overlap between the two images, we replace the first image data with zero value by the second image data in the overlap zone. It means there is no object on the position of the first image data and can be replaced by the second data. Then the function displays the first image data completely whole displays the second image excluding the overlapping part.



```
case 1:
        for(i=0;i<288;i++)
                        for(j=0;j<352;j++)
                        {
                                if(((j + AbsPosX) < 352) & ((i + AbsPosY) < 288))
                                {
                                        if(rgbValue1[(i + AbsPosY) * 352 + j + AbsPosX] == 0)
                                                rgbValue1[(i + AbsPosY) * 352 + j + AbsPosX] = rgbValue2[i * 352 + j];
                                }
                        }

          for(i=0;i<288;i++)
                        for(j=0;j<352;j++)
                        {
                                if(((j + AbsPosX) < 352) & ((i + AbsPosY) < 288))
                                {
                                        SetPixelV(hDC, j + PosX1, i + PosY1, rgbValue1[i * 352 + j]);
                                }
                                else
                                {
                                        SetPixelV(hDC, j + PosX1, i + PosY1, rgbValue1[i * 352 + j]);

                                        SetPixelV(hDC, j + PosX2, i + PosY2, rgbValue2[i * 352 + j]);
                                }
                        }
                        break;
//Display the composed images
```

Figure 4.14: An example of the code to display the composed image.

In the previous paragraph, we only consider the case of two videos. Now we allow three or four users. Figure 4.15 shows the diagram explaining the method. As we can see, the images decoded by the first decoder and the second decoder are composed and padded to become a new composite stream. The images decoded by the third decoder and the fourth decoder would also be composed the same way. The two new composite images would be composed and padded again. Each composition employs the method for two users described in the previous paragraph and shown in Fig. 4.12. Furthermore, we add a new step called padding and Fig. 4.16 shows the related code. Since the original composite image would not become a frame with the form of a box, the padding process is needed to make the new composite image into the form of a box. We also make the two new composite images have the same size. In addition, the image in the node of the left side would cover the image in the node of the right side but the decoder in the node of the right side would control the decoder in the node of the left side. For instance, if the images decoded by the third decoder and the fourth decoder overlap, the overlapped region of the image decoded by the fourth decoder would be covered with the image decoded by the third decoder. However, the third decoder must wait until the fourth decoder decodes a frame completely. This means that the fourth decoder can control the other three decoders

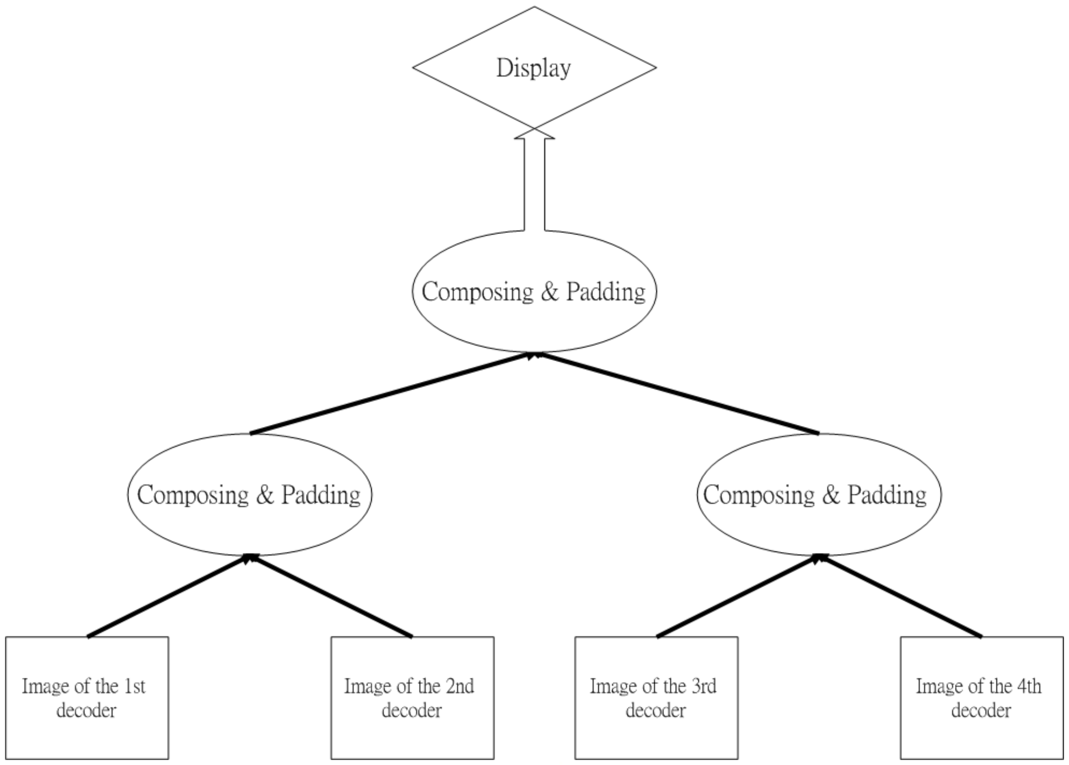


Figure 4.15: Video composition for three or four users.

and the relation among the four decoders is shown in Fig. 4.17. Finally the completely composed image would be displayed on the desktop.

We observed that the SetPixelV function is very slow and the time consumed by the SetPixelV function is about 70% of all time. Hence, we must reduce the use of the SetPixelV function. The method is to record the pixels of the last video object and only update the pixels of the current video object and the last video object. The PreBiV buffer could record the object's pixels of the preceding frame. The original code is shown in Fig. 4.18 and the modified code is shown in Fig. 4.19.

If more than four client users connect the receiver, we just create more threads for all decoders and repeat the padding and composition process. When we repeat these process one time, the system can deal with the data from two client users. For reason of execution speed, we only allow four client users presently.

```
case 1:
     for(i = 0; i < PaddingHeight; i++)
          for(j = 0; j < PaddingWidth; j++)
          {
               if ((j < OldWidth) & (i < OldLength))
               {
                    if((j >= Frame.x) & (i >= Frame.y))
                    {
                         if(rgbValue1[i * OldWidth + j] == 0)
                              rgbV1[i * PaddingWidth + j] =
                               rgbValue2[(i - Frame.y) * OldWidth + j - Frame.x];

                         else rgbV1[ i * PaddingWidth + j ] =
                                rgbValue1[ i * OldWidth + j];
                    }
                    else rgbV1[ i * PaddingWidth + j ] = rgbValue1[ i * OldWidth + j];
               }
               else if ((j >= Frame.x) & (i >= Frame.y) & (j < OldWidth + Frame.x)
                                        & (i < OldLength + Frame.y))

                    rgbV1[ i * PaddingWidth + j ] =
                     rgbValue2[ (i - Frame.y) * OldWidth + j - Frame.x];

               else rgbV1[ i * PaddingWidth + j ] = 0;
          }
     break;

//Related code of the padding process
```

Figure 4.16: A part of the related code of the padding process.

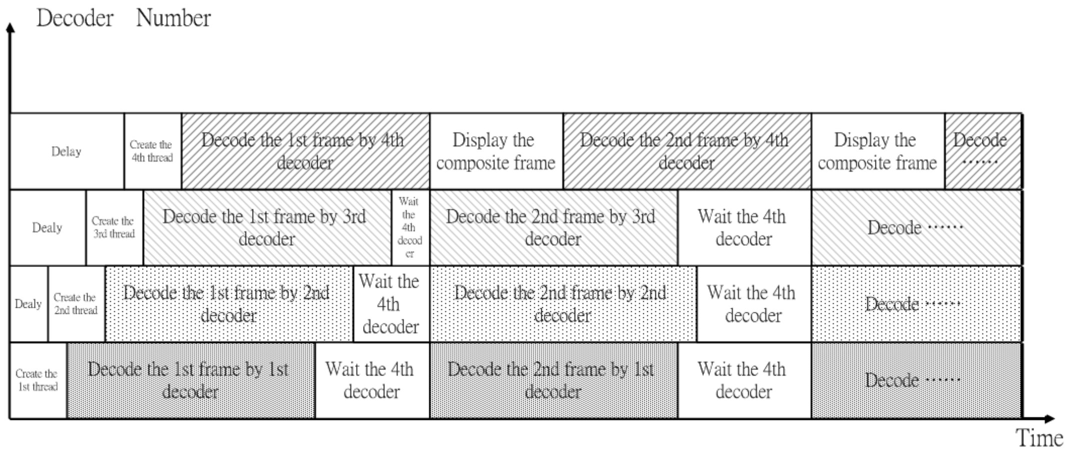
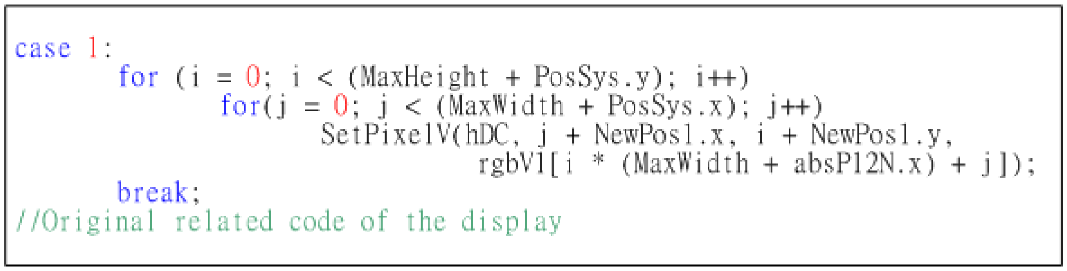
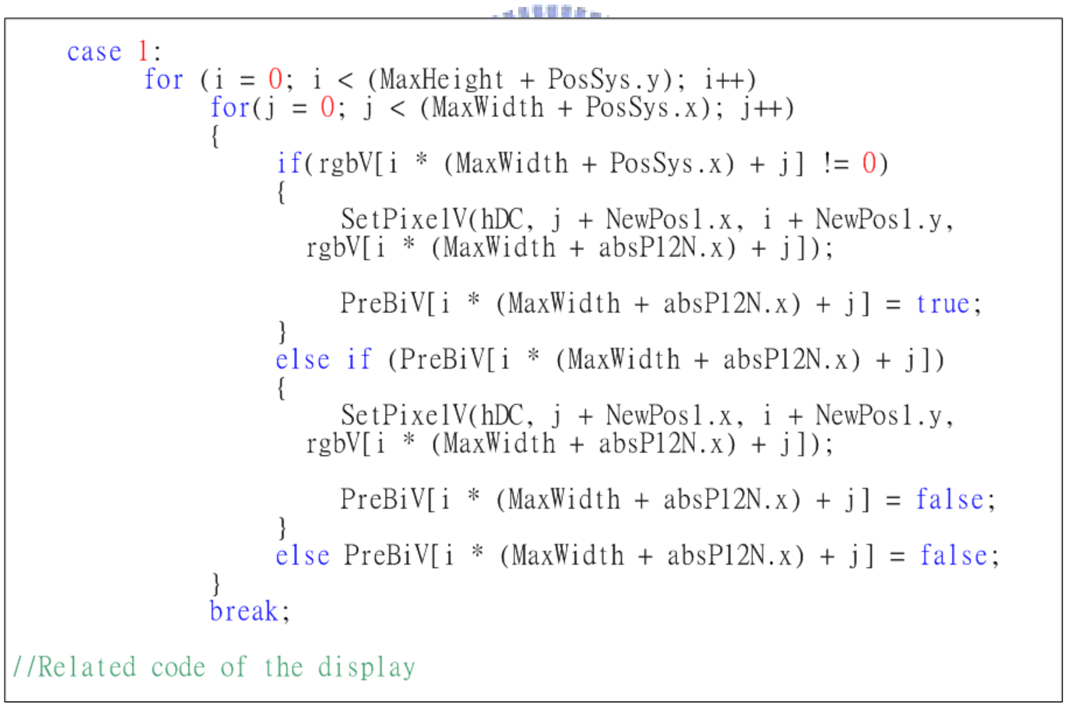

Decoder Number
Delay
Create the 4th thread
Decode the 1st frame by 4th decoder
Display the composite frame
Decode the 2nd frame by 4th decoder
Display the composite frame
Decode ……
Dealy
Create the 3rd thread
Decode the 1st frame by 3rd decoder
Wait the 4th decoder
Decode the 2nd frame by 3rd decoder
Wait the 4th decoder
Decode ……
Dealy
Create the 2nd thread
Decode the 1st frame by 2nd decoder
Wait the 4th decoder
Decode the 2nd frame by 2nd decoder
Wait the 4th decoder
Decode ……
Create the 1st thread
Decode the 1st frame by 1st decoder
Wait the 4th decoder
Decode the 2nd frame by 1st decoder
Wait the 4th decoder
Decode ……
Time

Figure 4.17: Relation among the four decoders in time domain.



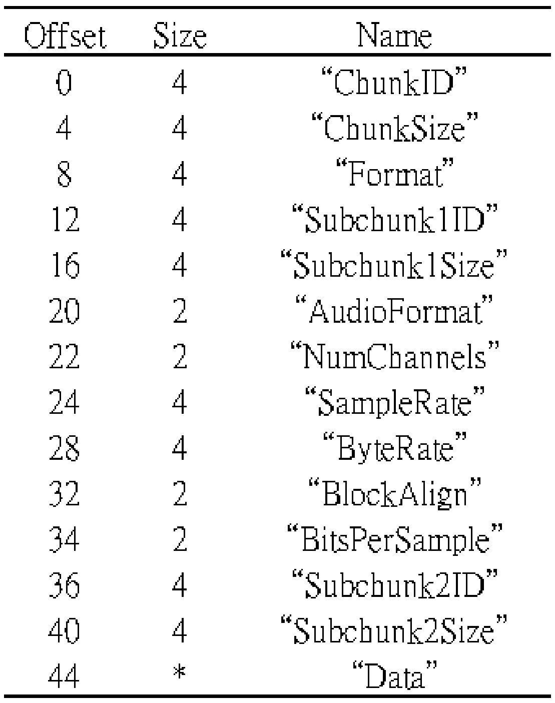
```
case 1:
        for (i = 0; i < (MaxHeight + PosSys.y); i++)
                for(j = 0; j < (MaxWidth + PosSys.x); j++)
                        SetPixelV(hDC, j + NewPos1.x, i + NewPos1.y,
                                    rgbV1[i * (MaxWidth + absP12N.x) + j]);
        break;
//Original related code of the display
```

Figure 4.18: A part of the code of the display process without reducing the use of the SetPixelV function.



```
    case 1:
          for (i = 0; i < (MaxHeight + PosSys.y); i++)
               for(j = 0; j < (MaxWidth + PosSys.x); j++)
               {
                    if(rgbV[i * (MaxWidth + PosSys.x) + j] != 0)
                    {
                         SetPixelV(hDC, j + NewPos1.x, i + NewPos1.y,
                      rgbV[i * (MaxWidth + absP12N.x) + j]);

                         PreBiV[i * (MaxWidth + absP12N.x) + j] = true;
                    }
                    else if (PreBiV[i * (MaxWidth + absP12N.x) + j])
                    {
                         SetPixelV(hDC, j + NewPos1.x, i + NewPos1.y,
                      rgbV[i * (MaxWidth + absP12N.x) + j]);

                         PreBiV[i * (MaxWidth + absP12N.x) + j] = false;
                    }
                    else PreBiV[i * (MaxWidth + absP12N.x) + j] = false;
               }
               break;

//Related code of the display
```

Figure 4.19: A part of the code of the display process which reduces the use of the Set-PixelV function.



Offset	Size	Name
0	4	“ChunkID”
4	4	“ChunkSize”
8	4	“Format”
12	4	“Subchunk1ID”
16	4	“Subchunk1Size”
20	2	“AudioFormat”
22	2	“NumChannels”
24	4	“SampleRate”
28	4	“ByteRate”
32	2	“BlockAlign”
34	2	“BitsPerSample”
36	4	“Subchunk2ID”
40	4	“Subchunk2Size”
44	*	“Data”

Figure 4.20: WAV file format (from [9]).

4.5 Audio Decoding and Composition

4.5.1 WAV Format [17]

WAV format is a common format used in audio systems. The Media Control Interface (MCI), a high level open interface, provides two ways to play the audio data. The two methods would play the audio data through the WAV file. Therefore we introduce the WAV format as shown in Fig. 4.20.

- ChunkID: Contains the letters “RIFF” in ASCII form.

- ChunkSize: $36 + Subchunk2Size$, or more precisely: $4 + (8 + Subchunk1Size) + (8 + Subchunk2Size)$. This is the size of the rest of the chunk following this number and the size of the entire file in the bytes minus 8 bytes for the two field not included in these counts: ChunkID and ChunkSize.

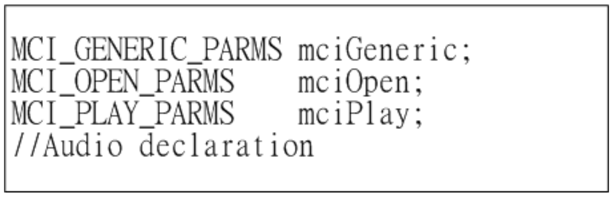
- Format: Contains the letters “WAVE” in ASCII form. The “WAVE” format consists of two subchunks: “fmt” and “data.” The “fmt” subchunk describes the sound data’s format.

- Subchunk1ID: Contains the letters “fmt” in the ASCII form.
- Subchunk1Size: 16 for pulse code modulation (PCM). This is the size of the rest of the subchunk which follows this number.
- AudioFormat: PCM $= 1$ (i.e. Linear quantization). Values other than 1 indicate some form of compression.
- NumChannels: Mono $= 1$, Stereo $= 2$, etc.
- SampleRate: 8000, 44100, etc.
- ByteRate: ByteRate $= SampleRate \times NumChannels \times BitsPerSample/8$.
- BlockAlign: BlockAlign $= NumChannels \times BitsPerSample/8$.
- BitsPerSample: 8 bits $=$ 8, 16 bits $=$ 16, etc. The “data” subchunk contains the size of the data and the actual sound.
- Subchunk2ID: Contains the letters “data” in ASCII form.
- Subchunk2Size: Subchunk2Size $= NumSample \times NumChannels \times BitsPerSample/8$.
- Data: The actual sound data.

4.5.2 Audio Player

Figure 4.21 shows the declaration of audio MCI class. Figure 4.22 shows the opening of the audio data and Fig. 4.23 shows the playing of the audio data.

The most important MCI function is mciSenCommand. The prototype of mciSendCommand is “MCIERROR mciSendCommand (MCIDEVICEID IDDEVICE, UINT uMsg, DWORD fdwCommand, DWORD_PTR dwParam).” The following paragraph are mainly taken from [19].



```
MCI_GENERIC_PARMS mciGeneric;
MCI_OPEN_PARMS    mciOpen;
MCI_PLAY_PARMS    mciPlay;
//Audio declaration
```

Figure 4.21: Related code of the audio declaration.

The first parameter, IDDevice, is the device identifier of the MCI device that is to receive the command message. The second parameter, uMsg, is the command message. The third parameter, fdwCommand, is the flag of the command message. The last parameter, dwParam, is a pointer to a structure that contain parameters for the command message. This function would return zero if successful or an error otherwise. The low-order word of the returned DWORD value contains the error return value. If the error is device-specific, the high-order word of the return value is the driver identifier; otherwise, the high-order word is zero.

After FAAD2 audio decoder decodes the audio stream received from the RTP network interface, it would be saved as a temporal audio file with WAV format. As we can see, we use the pointer, MCI_OPEN_PARMS, to set the device opening the audio WAV file. The pointer, MCI_PLAY_PARMS, lets the mciSendCommand function send a message to device to play the audio file. Then the program would wait for four seconds and the device would play in the time. Finally, the pointer, MCI_GENERIC_PARMS, would stop and close the device.

For audio composition, we directly add the decoded audio streams from different decoders but the effect is not good. The voice of the one decoder may become the noise for the another decoder. In addition, the overflow value also influence the tone quality. Hence we now only allow the audio stream from the first user. There may be a better method to solve this problem.

```
                    mciOpen.dwCallback       = 0;
                    mciOpen.wDeviceID        = 0;
                    mciOpen.lpstrDeviceType  = NULL;
                    mciOpen.lpstrElementName = szFileName;
                    mciOpen.lpstrAlias       = NULL;

                    dwError = mciSendCommand (0, MCI_OPEN, MCI_WAIT |
                     MCI_OPEN_ELEMENT, (DWORD) (LPMCI_OPEN_PARMS) &mciOpen);

                    if(dwError != 0)
                    {
                          ShowError (hwnd, dwError);
                    }
//Open waveform audio
```

Figure 4.22: Related code for audio data opening.

```
                    wDeviceID = mciOpen.wDeviceID;

                    mciPlay.dwCallback = (DWORD) hwnd;
                    mciPlay.dwFrom     = 0;
                    mciPlay.dwTo       = 0;

                    mciSendCommand (wDeviceID, MCI_PLAY, MCI_NOTIFY, (DWORD)
                    (LPMCI_PLAY_PARMS) &mciPlay);

                    bPlaying = TRUE;

                    Sleep(4000);

                    mciGeneric.dwCallback = 0;

                    mciSendCommand (wDeviceID, MCI_STOP, MCI_WAIT, (DWORD)
                    (LPMCI_GENERIC_PARMS) &mciGeneric);

                    mciSendCommand (wDeviceID, MCI_CLOSE, MCI_WAIT, (DWORD)
                    (LPMCI_GENERIC_PARMS) &mciGeneric);

                    bPlaying = FALSE;

//Play waveform audio
```

Figure 4.23: Related code for audio playing.

Chapter 5

Experimental Results

In the last chapter, we discussed our system how to compose multiple videos and audios and display the result on the desktop. In this chapter, we present some experimental results on the speed and outcome in different situations. The test sequence is brea_cif.cmp file which is a commonly used video example with binary shape information.

Figure 5.1 shows that the speed of the original decoder software with different number of users. The initial values in all case are almost the same because the program starts the decoder sequentially. If only one video decoder is present, the frame rate can get up to 53 fps (frames per second). If two decoders are at work, the frame rate would go down to 29 fps. The frame rate is 19 fps for three decoders and 14 fps for four decoders. The decreasing rate is not proportion to the number of the video decoders. This may be because the hardware and the operating system are more efficient when there are multiple decoder threads.

Since we need the 4:4:4 format, we must pad the data and calculate the RGB values. As shown in Fig. 5.2, the frame rate decreases by about 5–10 fps if we add the padding function. The efficiency is shown in Fig. 5.3 and it decreases by about 20% compared with the original decoder.

Then, we must compose these videos and display the composite video. Figure 5.4 shows four composition cases and Fig. 5.5 shows the execution performance. According to our observation the frame rates are down to 5 fps in the situation with only one decoder and 2.5 fps in the situation with the two decoders. The decoder becomes too slow. This is

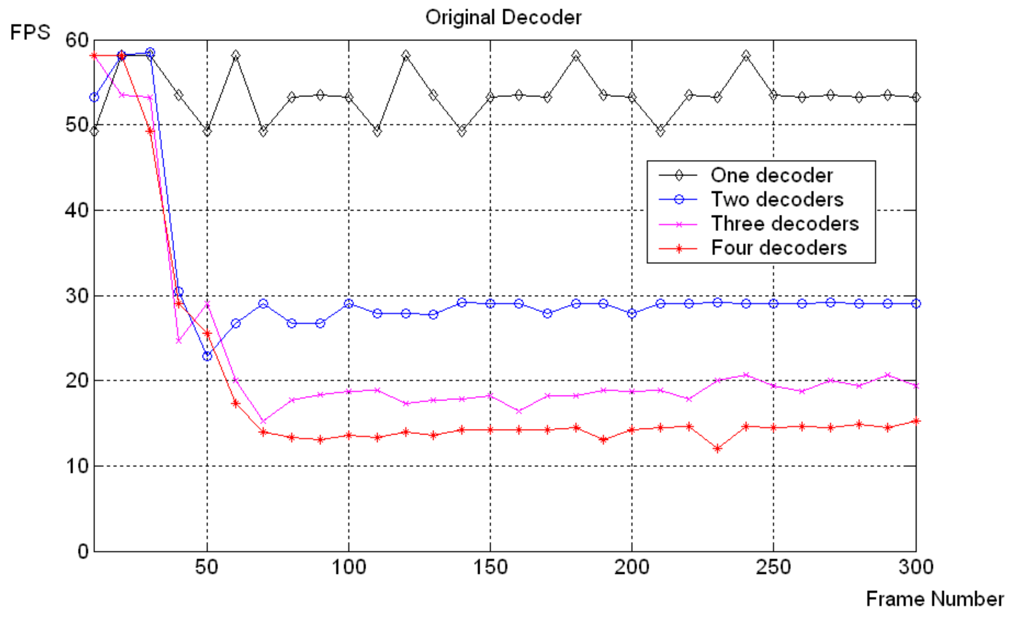


Figure 5.1: Performance of the original decoder.

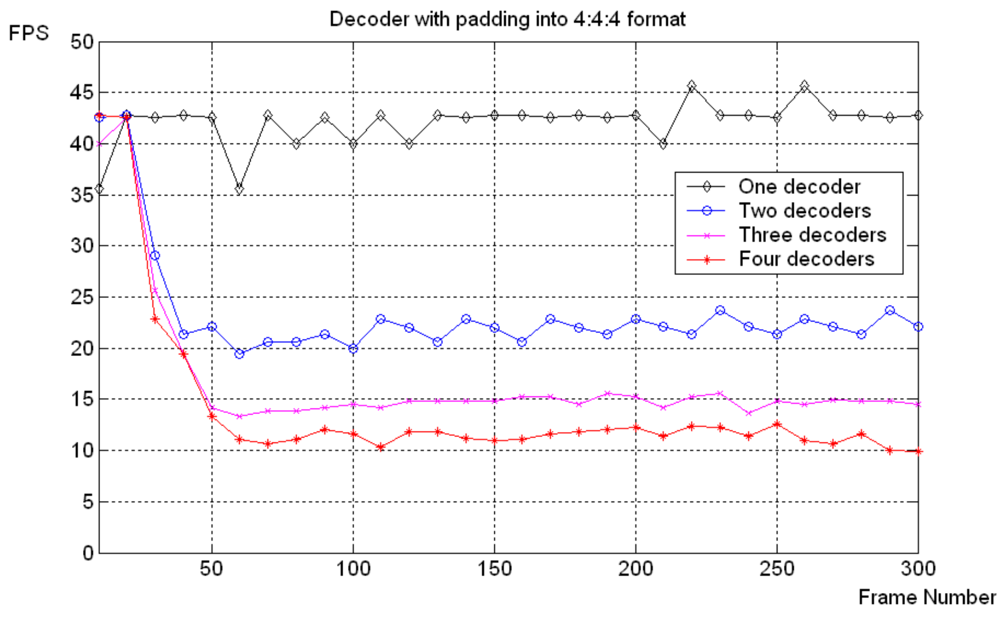
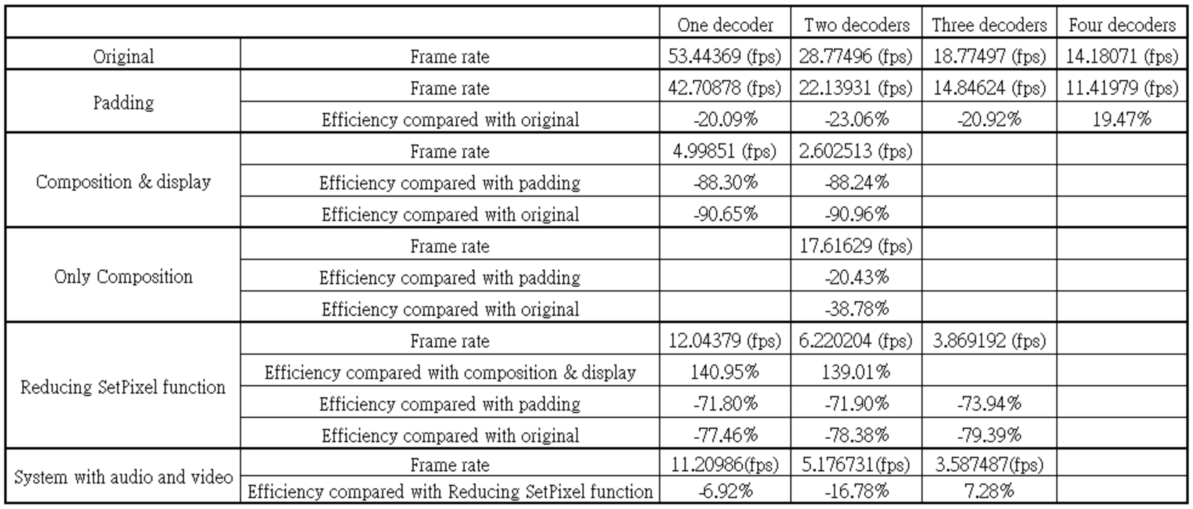
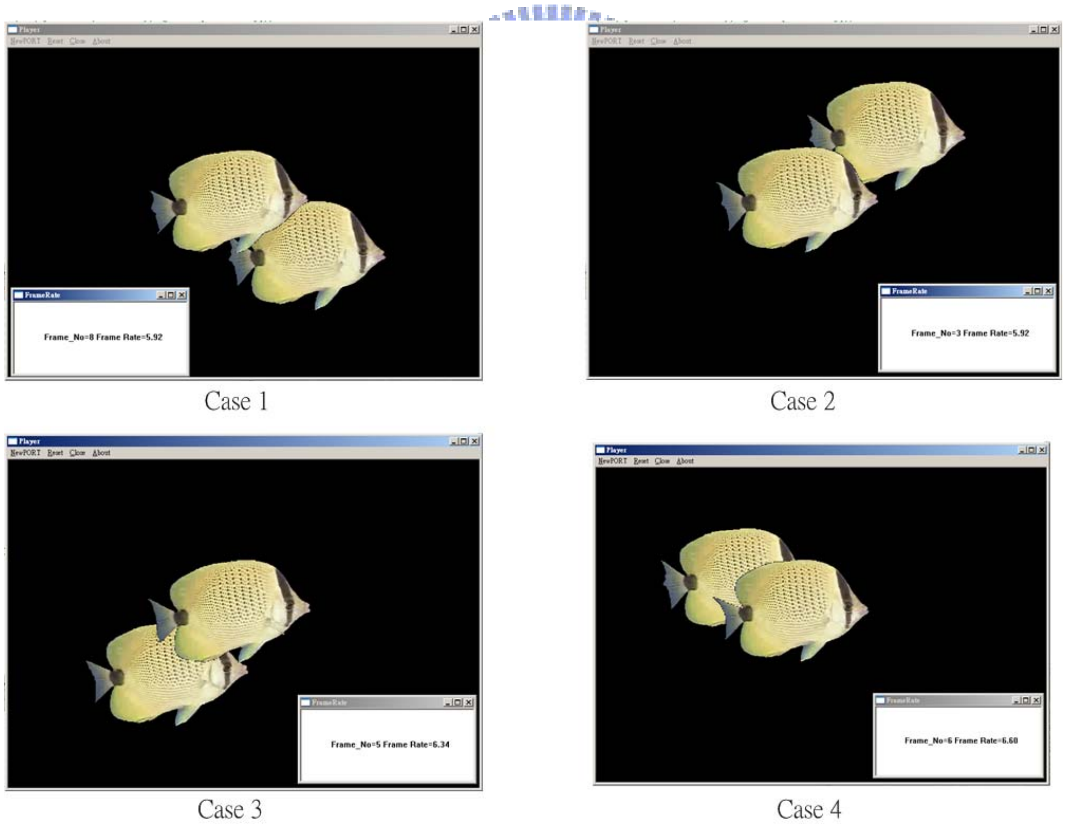


Figure 5.2: Performance of decoder with padding.



		One decoder	Two decoders	Three decoders	Four decoders
Original	Frame rate	53.44369 (fps)	28.77496 (fps)	18.77497 (fps)	14.18071 (fps)
Padding	Frame rate	42.70878 (fps)	22.13931 (fps)	14.84624 (fps)	11.41979 (fps)
	Efficiency compared with original	-20.09%	-23.06%	-20.92%	19.47%
Composition & display	Frame rate	4.99851 (fps)	2.602513 (fps)		
	Efficiency compared with padding	-88.30%	-88.24%		
	Efficiency compared with original	-90.65%	-90.96%		
Only Composition	Frame rate		17.61629 (fps)		
	Efficiency compared with padding		-20.43%		
	Efficiency compared with original		-38.78%		
Reducing SetPixel function	Frame rate	12.04379 (fps)	6.220204 (fps)	3.869192 (fps)	
	Efficiency compared with composition & display	140.95%	139.01%		
	Efficiency compared with padding	-71.80%	-71.90%	-73.94%	
	Efficiency compared with original	-77.46%	-78.38%	-79.39%	
System with audio and video	Frame rate	11.20986(fps)	5.176731(fps)	3.587487(fps)	
	Efficiency compared with Reducing SetPixel function	-6.92%	-16.78%	7.28%	

Figure 5.3: Relative efficiency.



Frame_No=8 Frame Rate=5.92

Frame_No=3 Frame Rate=5.92

Frame_No=5 Frame Rate=6.34

Frame_No=6 Frame Rate=6.60

Figure 5.4: Four composition situations.

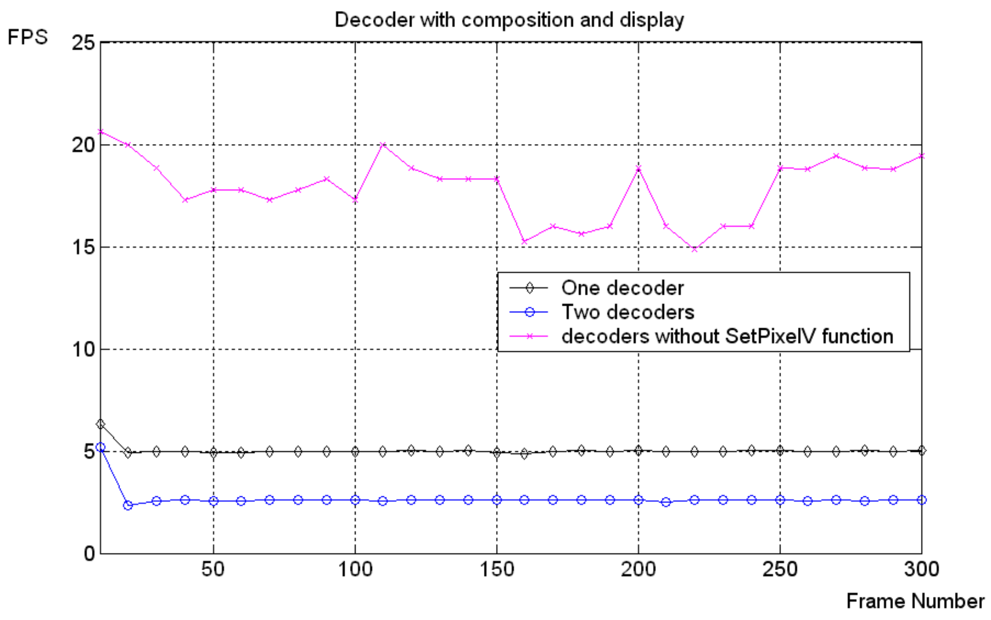


Figure 5.5: Performance of the decoder with composition and display.

because the SetPixelV function used to display the video on the desktop is a slow function. We can see the decoding without display took down at the case with only composition and without SetpixelV function. The frame rate can be up to 18 fps and the SetPixelV function dominates the efficiency of the decoder. Figure 5.6 shows that the execution time for image display is about 70% of the overall time.

In order to decrease the effect of the SetPixelV function, we reduce the number of the SetPixelV function calls. The result is shown in Fig. 5.7. As we can see, we increase the frame rate into 12 fps and 6 fps in the two situations. The efficiency compared with the previous case without reducing the number of the SetPixelV function increases by about 140%. However, according to the Fig. 5.3, the efficiency compared with the original system still decreases by about 72%.

Finally, we add the audio decoder into our system. Figure 5.8 shows the effect with the audio decoder with different numbers of video decoder. We can see that the frame rates decrease about 1 fps compared with the system with only video decoders reducing SetPixelV function calls. This means that the effect of the audio decoder is small, almost negligible.

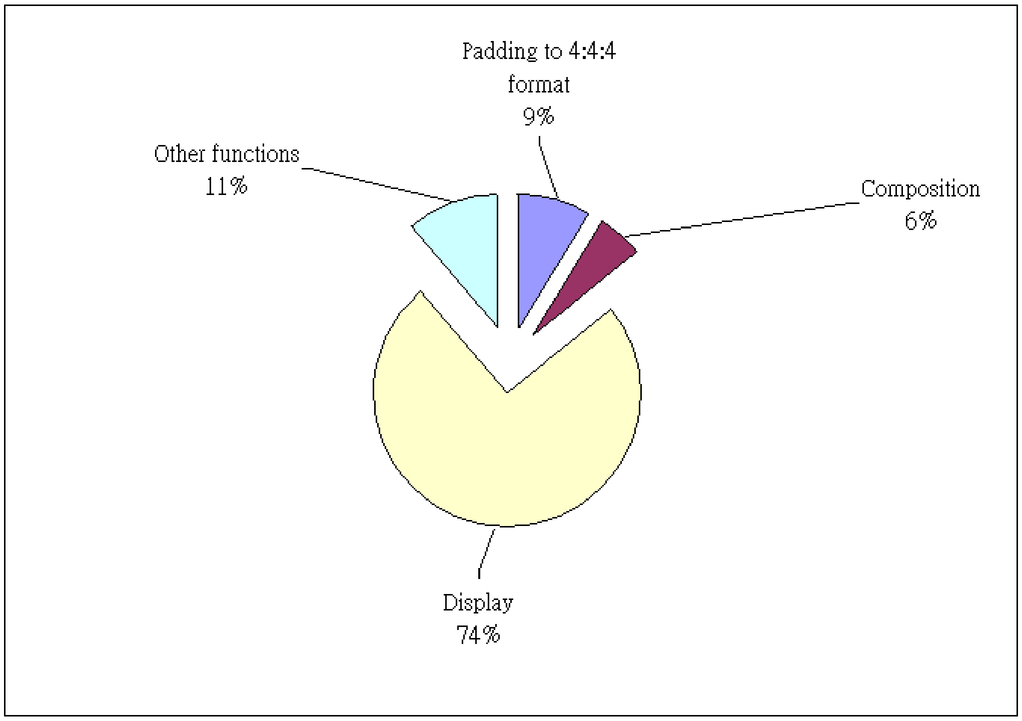


Figure 5.6: Time analysis of the decoder.

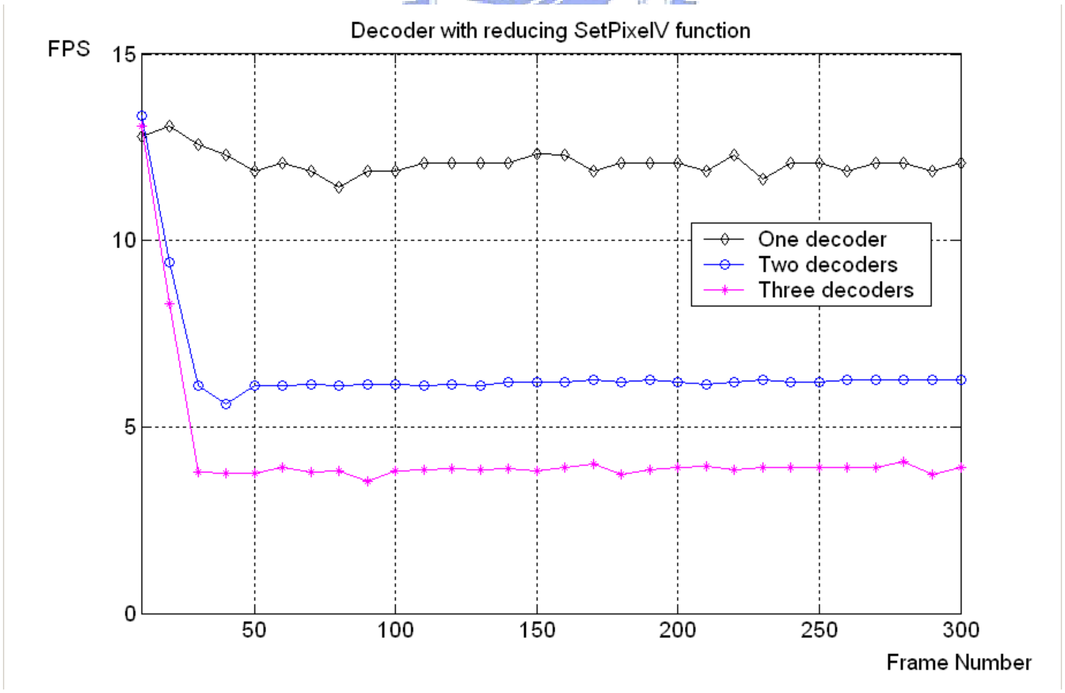


Figure 5.7: Performance of the decoder after reducing the SetPixelV function.

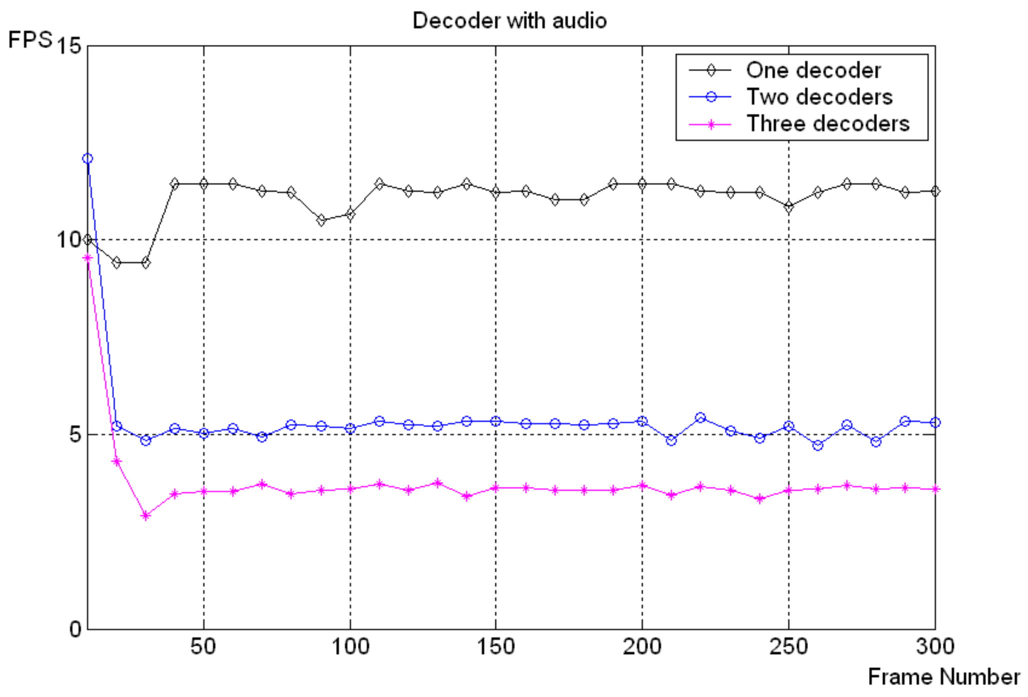


Figure 5.8: Performance after adding audio decoder.

Chapter 6

Conclusion and Future Work

We developed and implemented the receiver for multi-point video conference on personal computer. The system combined three public programs: JRTPlib, Microsoft MPEG-4 video software, and FAAD2. The relative simulations in previous chapters showed the system allowed four client users and displayed them on the special position of the desktop. Furthermore the four video could be overlapped and the other user's video would be cover with the previous user's video in this system. The speed of the system was about 6 fps for two client users and 3.5 fps for three client users.

For quality improvement we can do some improvements for the main project in the future.

1. Compose audio streams.

   In the multi-point system, the users may speak at the same time. At this situation, we may get a lot of audio streams in the meantime but the system only play one audio stream. Hence we could combine these audio streams become one stream to resolve this problem.

2. Design more module for visual composition.

   We only provide moving the video object to any position of the desktop. We can also provide more functions as rotation, scale, etc.. The user can use these functions to embellish the desktop. Maybe we also can add some virtual objects on the desktop.

3. More optimization.

   The current processing time is not fast enough and a better optimization is need to improve the efficiency. We may focus on the SetPixelV function or modifying the MPEG-4 decoder.

Bibliography

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

[2] MPEG-4 Video Group, "MPEG-4 overview — (V.21 Jeju Version)," doc. no. ISO/IEC JTC1/SC29/WG11 N4668, Mar. 2002.

[3] ISO/IEC 14496-2:2001, *Information Technology — Coding of Audio-Visual Objects — Part 2: Visual*. July 2001.

[4] T. Sikora, "The MPEG-4 video standard verification model," *IEEE Trans. Circuits Systems Video Tech.*, vol. 7, no. 1, pp. 19–31, Feb. 1997.

[5] A. Puri and A. Eleftheriadis, "MPEG-4: an object-based multimedia coding standard supporting mobile applications mobile networks and applications," *Mobile Networks Applic.* vol. 3, pp. 5–32, 1998.

[6] MPEG-4 Video Group, "MPEG-4 video verification model version 18.0," doc. no. ISO/IEC JTC1/SC29/WG11 N3908, Pisa, Jan. 2001.

[7] A. Ebrahimi and C. Horne, "MPEG-4 natural video coding – an overview," *Signal Processing Image Commun.* vol. 15, pp. 365–385, 2000.

[8] M.-Y. Liu, "Real-time implementation of MPEG-4 video encoder using SIMD-enhanced Intel processor," M.S. thesis, Department of Electronics Engineering, National Chaio Tung University, Hsinchu, Taiwan, R.O.C., July 2004.

[9] C.-Y. Tsai, “Integration of videoconference transmitter with MPEG-4 object-based video encoding,” M.S. thesis, Department of Electronics Engineering, National Chaio Tung University, Hsinchu, Taiwan, R.O.C., June 2005.

[10] ISO/IEC 14496-3:1999, *Information Technology — Coding of Audio-Visual Objects — Part 3: Audio.* Dec. 1999.

[11] Martin Wolters, Kristofer kjorling, Daniel Homm and Heiko Purnhagen, “A closer look into MPEG-4 High Efficiency AAC,” presented at the 115th Convention., NY, USA, Oct. 2003.

[12] Microsoft, ISO/IEC 14496 (MPEG-4) Video Reference Software User Manual, Oct. 2004.

[13] H. Schulzrine, S. Casner, R. Frederick, and V. Jacobson, “RTP: a transport protocol for real-time application,” RFC 1889, Network Working Group, Jan 1996.

[14] H. Schulzrine, S. Casner, R. Frederick, and V. Jacobson, “RTP: a transport protocol for real-time application,” RFC 3550, Network Working Group, July 2003.

[15] “JRTPLIB,” http://research.edm.luc.ac.be/jori/page.html.

[16] “AudioCoding.com,” http://www.audiocoding.com/.

[17] “Microsoft RIFF,” http://netghost.narod.ru/gff/graphics/summary/micriff.htm.

[18] “Microsoft MSDN Library,” http://msdn.microsoft.com/library/default.asp.

[19] “Microsoft MSDN Library for Windows Multimedia,” http://msdn.microsoft.com/library/default.asp?url=/library/ en-us/multimed/htm/_win32_mcisendcommand.asp.

[20] “DMIF software,” http://lan.ece.ubc.ca/memberza.html.