BI & TRI DIMENSIONAL SCENE DESCRIPTION AND COMPOSITION IN THE MPEG-4 STANDARD

By

Edward Cooke, BSc

A thesis submitted for the degree of

Masters in Electronic Engineering

Dublin City University

Supervisor Dr Thomas Curran

School of Electronic Engineering

March 1998

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Masters in Electronic Engineering is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work

Signed _____ ID No 95971351

Candidate

Date 27/4/98

TABLE OF CONTENTS

ABSTRACT

BI AND TRI DIMENSIONAL SCENE DESCRIPTION AND COMPOSITION IN THE MPEG-4 STANDARD

By Edward Cooke

MPEG-4 is a new ISO/IEC standard being developed by MPEG (Moving Picture Experts Group) The standard is to be released in November 1998 and version 1 will be an International Standard in January 1999 The MPEG-4 standard addresses the new demands that arise in a world in which more and more audio-visual material is exchanged in digital form MPEG-4 addresses the coding of objects of various types Not only traditional video and audio frames, but also natural video and audio objects as well as textures, text, 2- and 3-dimensional graphic primitives, and synthetic music and sound effects

Using MPEG-4 to reconstruct an audio-visual scene at a terminal, it is hence no longer sufficient to encode the raw audio-visual data and transmit it, as MPEG-2 does in order to synchronize video and audio In MPEG-4, all objects are multiplexed together at the encoder and transported to the terminal Once de-multiplexed, these objects are composed at the terminal to construct and present to the end user a meaningful audio-visual scene The placement of these elementary audio-visual objects in space and time is described in the scene description of a scene While the action of putting these objects together in the same representation space is the composition of audio-visual objects

My research was concerned with the scene description and composition of the audio-visual objects that are defined in an audio-visual scene Scene descriptions are coded independently from streams related to primitive audio-visual objects The set of parameters belonging to the scene description are differentiated from the parameters that are used to improve the coding efficiency of an object While the independent coding of different objects may achieve a higher compression rate, it also brings the ability to manipulate content at the terminal This allows the modification of the scene description parameters without having to decode the primitive audio-visual objects themselves This approach allows the development of a syntax that describes the spatio-temporal relationships of audio-visual scene objects The behaviours of objects and their response to user inputs can thus also be represented in the scene description, allowing richer audio-visual content to be delivered as an MPEG-4 stream

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank my supervisor, Dr Thomas Curran, and all my colleagues in the Video Coding Group as well as the MoMuSys and MPEG-4 consortiums for their continued support and technical help during the last two years

I also want to thank the people who have helped me morally during my research, these are the group of people who would stoop low enough to call themselves my friends, and of course my family who, unfortunately for them, had no real say in their relationship to me

Thanks

Chapter

1 INTRODUCTION

1 1 Introduction

After setting the MPEG-1 and MPEG-2 standards, MPEG (Moving Pictures Experts Group, ISO/IEC Joint Technical Committee 1, Sub Committee 29, Work Group 11) is now working on a new audio-visual standard, called MPEG-4 While the initial objective of MPEG-4 was to achieve very low bit-rates, MPEG has adapted the work plan to changes in the audio-visual environment and modified its targets considerably [4] The MPEG standard under development now addresses the new demands that arise in a world in which more and more audio-visual material is exchanged in digital form

The first two sets of MPEG standards (MPEG-1 and MPEG-2) are well known to people involved in digital communication They are widely adopted in commercial products, such as CD-interactive, digital audio broadcasting, digital television and many video-on-demand trials MPEG-1 and -2 deal with 'frame-based video' and audio Although these standards provide a large improvement, in randomly accessing content, over standards that existed before, the granularity of the interaction is limited to the video frame, with its associated audio In this sense, the functionality could be compared with that of audio and video cassette players, albeit with non-linear controls Their most important goal is to make storage and transmission more efficient, by compressing the material The new MPEG-4 standard does not only aim to achieve efficient storage and transmission, but also to satisfy other needs of future image communication users To reach this goal, MPEG-4 will be fundamentally different in nature from its predecessors, as it makes the move towards representing the scene as a composition of (potentially meaningful) objects, rather than 'just' the pixels

The most important innovation that MPEG-4 brings is it defines an audio-visual scene as a coded representation of 'audio-visual objects' that have certain relations in space and time, rather than 'video frames with associated audio' Depending on the application, the scene can be composed of 2D or 3D time varying objects 3D scenes may be composed of 3D, 2D, synthetic and natural objects Such an object could be a video object a car, a dog, or the complete background It could also be an audio object one instrument in an orchestra, the barking of the dog, a voice When an audio and a video object are associated, audio-visual object results the image of a running dog together with the sound it makes This new approach to information representation allows for much more interactivity, for versatile re-use of data, and for intelligent schemes to manage bandwidth, processing resources (e g memory, computing power) and error protection It also eases the integration of natural and synthetic audio and video material, as well as other data types, such as text overlays and graphics

1 2 Research Objectives

The subject of my research was concerned with the scene description and composition of the audio-visual objects that are defined in an audio-visual scene In MPEG-4, all audio-visual objects are multiplexed together at the encoder and transported to the terminal Once de-multiplexed, these objects are composed at the terminal to construct and present to the end user a meaningful audio-visual scene The placement of these elementary audio-visual objects in space and time is described in the Scene Description The action of putting these objects together in the same representation space is the Composition of audio-visual objects

Scene descriptions are coded independently from streams related to primitive audio-visual objects Special care is devoted to the identification of the parameters belonging to the scene description This is done by differentiating parameters that are used to improve the coding efficiency of an object (e g motion vectors in video coding algorithm), from those used as modifiers of an object's characteristics within the scene

(e g position of the object in the global scene) The idea was to standardise a syntax that describes the spatio-temporal relationships of the audio-visual scene objects

The compositor uses this spatio-temporal information to reconstruct the complete scene Composition information is thus used to synchronise different objects in time, and to give them the right position in space

During the course of my research I analysed the fundamental principles of bi and tri dimensional scene description and graphic composition with an interest in how these principles could be developed to aid in the creation of the MPEG-4 standard My research also involved the development of an MPEG-4 terminal, which would utilise the MPEG-4 scene description language and composite the audio-visual objects described in an audio-visual scene

1 3 Structure of Thesis

A general overview of the MPEG-4 standard is given in chapter 2 A brief introduction is given of how the need to establish a universal, efficient coding standard for different forms of audio-visual data arose, and the scope and features the standard offers to authors, service providers and end users This is followed by a technical description of the various layers i e , Video, Audio, Systems etc, which combine to form MPEG-4 An overview of the new concepts that have been developed within these layers to create the standard is given This chapter is designed to give an explanation of how the final standard is designed to function

The fundamental principles of bi and tri dimensional graphic composition and rendering are discussed in chapter 3 The chapter is designed to explain how geometrical transformations based on matrix mathematics can be used to simplify the composition of 2D and 3D scenes It is through these transformations that graphics applications can

create 2D renditions of 3D objects The chapter also introduces and explains the notion of an objects' local co-ordinate system and how a global co-ordinate system, the scene, can be created by combining different objects co-ordinate systems together Finally viewing projections of 2D and 3D objects are explained

Chapter 4 introduces the notion of a scene description in the MPEG-4 standard Scene descriptions are coded independently from streams related to primitive audio-visual objects Special care is devoted to the identification of the parameters belonging to the scene description This is done by differentiating parameters that are used to improve the coding efficiency of an object from those used as modifiers of an object's characteristics within the scene In keeping with MPEG-4's objective to allow the modification of this latter set of parameters without having to decode the primitive audio-visual objects themselves, these parameters form part of the scene description and are not part of the primitive audio-visual objects The idea was to standardise a syntax that describes the spatio-temporal relationships of Scene Objects This chapter is a detailed analysis of how scene description languages function and how the functionality of the MPEG-4 scene description language has been developed since its conception We see how the JAVA language was initially used for a flexible form of scene description language and the development of this language This is followed by an explanation of the overheads involved in developing a real-time implementation of an MPEG-4 terminal and how the JAVA environment was too heavy for such development Finally VRML is introduced as a possible scene description language

The development of the MPEG-4 scene description language into a Binary Format for Scene Description (BIFS) and how bi and tri dimensional composition is achieved using this description is explained in chapter 5 We see the disadvantages of the VRML scene

description language for MPEG-4 as well as how VRML was used as a building block for BIFS The use of the OpenGL API for 2D and 3D composition is also described A detailed explanation of a developed MPEG-4 player is also given

Chapter 6 gives an account of how the upcoming MPEG-4 standard has been divided into two versions An overview of the currently developing version, version 1 which went to a Committee Draft document in November 1997, is given as well as a timetable for the future development planned for MPEG-4 versions 1 and 2 A description is given of how the VRML and MPEG-4 consortiums are converging and the future prospects for BIFS An explanation of the new dynamic scene description language, Adaptive Audio-Visual Session format (AAVS), is also given The chapter finishes with an analysis of several different types of applications that the MPEG-4 standard will enable developers to create

Chapter

2 OVERVIEW OF THE MPEG-4 STANDARD

2 1 Introduction

In this chapter the MPEG-4 standard is introduced The initial section explains how the demand for MPEG-4 arose, this is followed by an explanation of what MPEG-4 offers as a new technology Then a technical description of the various layers i e , Video, Audio, Systems etc, which combine to form MPEG-4 are introduced and the new concepts that have been developed within these layers to create the standard are explained

As the MPEG-4 project description [4] states, a number of concurrent evolution's have created the need for new ways to represent, integrate, and exchange pieces of audio-visual information

- the deployment of diverse new two-way delivery systems such as fixed broadband and mobile narrowband,
- the progress of micro-electronic technology that is providing extremely powerful and programmable processors, and
- the change of the audio-visual information production and consumption paradigm, because of the increased role of synthetic information and higher degrees of interactivity

The MPEG-4 project aims to establish universal, efficient coding of different forms of audio-visual data, called audio-visual objects These objects can be of natural or synthetic origin

2 2 Scope and features of the MPEG-4 standard

The MPEG-4 standard under development will provide a set of technologies to satisfy the needs of authors, service providers and end users alike [21] [22]

- To *authors*, MPEG-4 will enable the production of content that is more reusable, has greater flexibility, and can be better protected than possible today with individual technologies such as digital television, animated graphics, World Wide Web (WWW) pages and their extensions

- To *network service providers,* MPEG-4 will offer content transportation mechanisms that match the Quality of Service (QoS) required by the individual media,

- To *end users*, MPEG-4 will allow higher levels of interaction with content, within the limits set by the author, avoiding the risk of proprietary formats and players

MPEG-4 achieves these goals by providing standardised ways to

- Represent units of aural, visual or audio-visual content, called "audio-visual objects" or AVOs (The very basic unit is more precisely called a "primitive AVO"),

- Compose these objects together, to create compound audio-visual objects (e g an audio-visual scene),

- Multiplex and synchronise the data associated with AVOs, so that they can be transported over networks providing a QoS appropriate for the nature of the specific AVOs,

- Interact with the audio-visual scene generated at the receiver's end

The next sections illustrate the described functionalities of MPEG-4, using the audio-visual scene depicted in Figure 1

2 3 Representation of primitive Audio-Visual Objects

This audio-visual scene is composed of several AVOs, organised in a hierarchical fashion At the leaves of the hierarchy, we find primitive AVOs, such as

- a 2-dimensional fixed background,
- the picture of a talking person (without the background)
- the voice associated with that person,

MPEG-4 standardises a number of such primitive AVOs, capable of representing both natural and synthetic content types, which can be either 2- or 3-dimensional In addition to the AVOs mentioned above and shown in Figure 1, MPEG-4 defines the coded representation of objects like

- talking heads and associated text to be used at the receiver's end to synthesise the speech and animate the head,
- animated human bodies,
- subtitles of a scene containing text and graphics

In their coded form, these objects are represented as efficiently as possible This means that not more information is spent on coding these objects than necessary for supporting the desired functionalities Such functionality may be error robustness, or allowing extraction and editing of the object, or having the object available in a scaleable form It is important to note that the coded representation is able to represent the object (aural or visual) independently, that is, without surroundings or background

2 4 Composition of Audio-Visual Objects

Figure 1 gives an example that highlights the way in which an audio-visual scene in MPEG-4 is composed of individual objects The figure contains compound AVOs that group arbitrary AVOs together For example, the visual object corresponding to the talking person and the corresponding voice are tied together to form a new compound AVO Such grouping allows authors to construct complex scenes, and enables consumers to manipulate meaningful (sets of) objects [26]

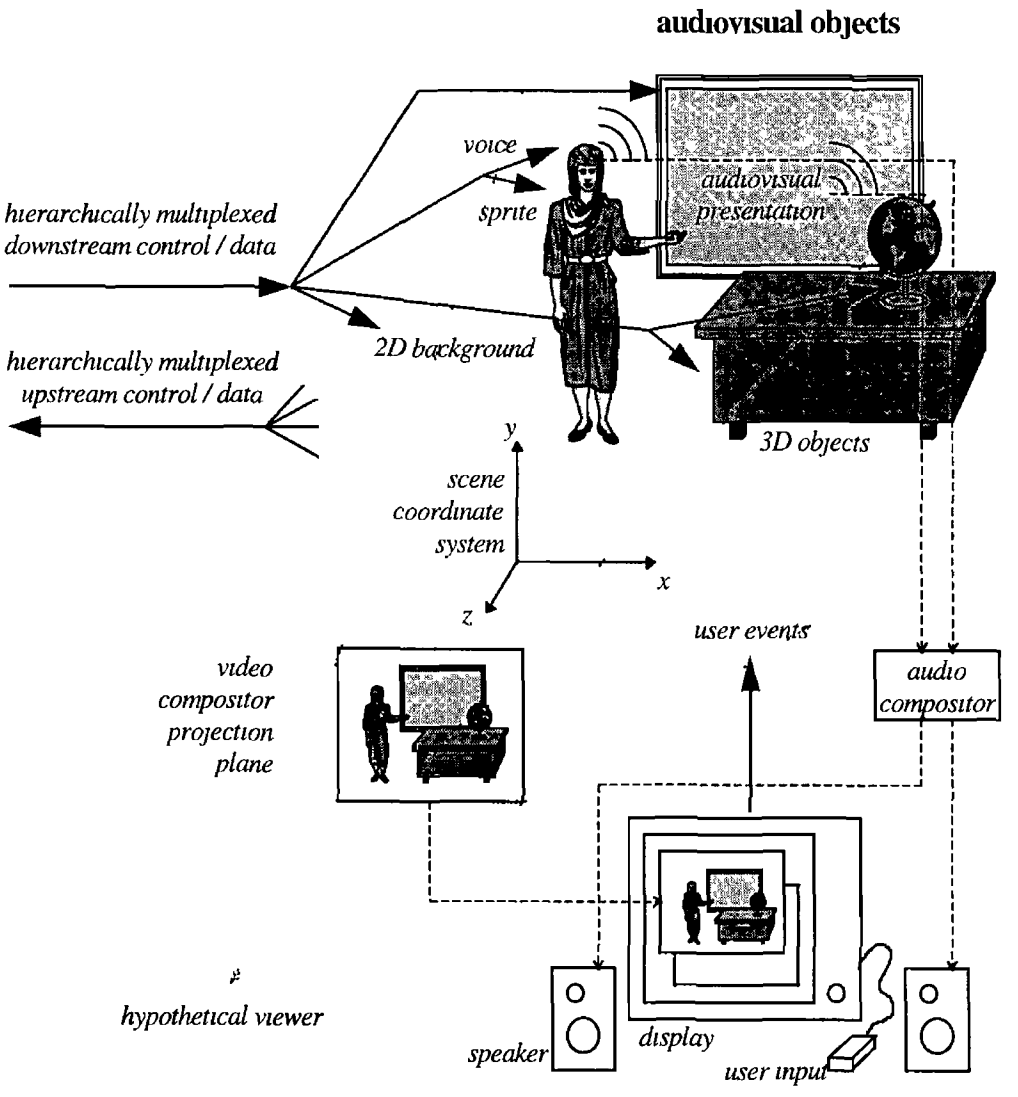


Figure 1 An example of an MPEG-4 audio-visual scene

More generally, MPEG-4 provides a standardised way to compose a scene, allowing for example to

- place AVOs anywhere in a given co-ordinate system,
- group primitive AVOs in order to form compound AVOs,
- apply streamed data to AVOs, in order to modify their attributes (e g moving texture belonging to an object, animating a moving head by sending animation parameters),
- change, interactively, the user's viewing or hearing points anywhere in the scene

2 5 Multiplex and Synchronisation of Audio-Visual Objects

AVO data is conveyed in one or more Elementary Streams The streams are characterised by the QoS they request for transmission (e g , maximum bit rate, bit error rate, etc), as well as other parameters, including stream type information to determine the required decoder resources and the precision for encoding timing information How such streaming information is transported in a synchronised manner from source to destination, exploiting different QoS as available from the network, is specified in terms of an Access Unit Layer and a conceptual two-layer multiplexer, as depicted in Figure 2

The Access Unit Layer allows identification of Access Units (e g , video or audio frames, scene description commands) in Elementary Streams, recovery of the AVO's or scene description's time base and enables synchronisation among them The Access Unit header can be configured in a large number of ways, allowing use in a broad spectrum of systems

The "FlexMux" (Flexible Multiplexing) Layer is fully specified by MPEG It contains a multiplexing tool that allows grouping of Elementary Streams (ESs) with a low multiplexing overhead This may be used, for example, to group ES with similar QoS requirements

The "TransMux" (Transport Multiplexing) layer in Figure 2 models the layer that offers transport services matching the requested QoS Only the interface to this layer is specified by MPEG-4 Any suitable existing transport protocol stack such as (RTP)/UDP/IP, (AAL5)/ATM, or MPEG-2's Transport Stream over a suitable link layer may become a specific TransMux instance The choice is left to the end user/service provider, and allows MPEG-4 to be used in a wide variety of operation environments

Use of the FlexMux multiplexing tool is optional and, as shown in Figure 2, this layer may be bypassed if the underlying TransMux instance provides equivalent functionality The Access Unit Layer, however, is always present

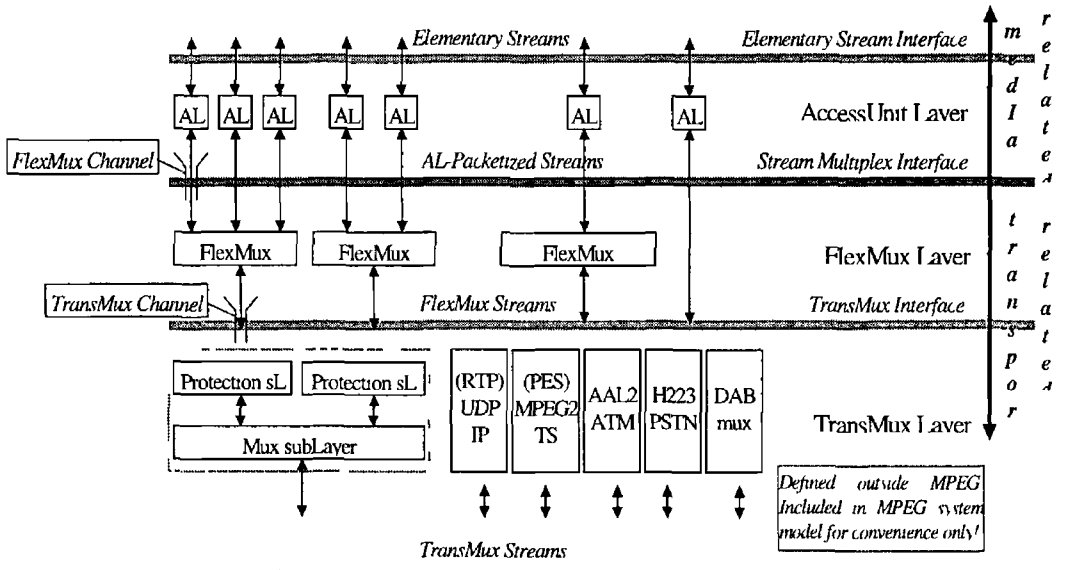


Figure 2 The MPEG-4 System Layer Model

With regard to Figure 2, it will be possible to

- identify access units, transport timestamps and clock reference information and identify data loss

- optionally interleave data from different ESs into FlexMux Streams
- convey control information to
  - indicate the required QoS for each Elementary Stream and FlexMux stream,
  - translate such QoS requirements into actual network resources,
  - convey the mapping of ESs, associated to AVOs, to FlexMux and TransMux channels

Part of the control functionalities will be available only in conjunction with a transport control entity like the DMIF framework

2 6 Interaction with Audio-Visual Objects

In general, the user observes a scene composed following the design of some author Depending on the degree of freedom allowed by the author the user has the possibility to interact with the scene Operations a user may be allowed to perform include

- changing the viewing/hearing point of the scene (e g by navigation through a scene),
- dragging objects in the scene to a different position, deleting objects from a scene,
- but also more complex kinds of behaviour can be triggered (e g a virtual phone rings, the user answers and a communication link is established)

2 7 Technical description of the MPEG-4 standard

The remaining sections in this chapter provide a technical description of the major components of the MPEG-4 standard A more detailed description can be found in [1] As shown in Figure 3, streams coming from the network (or a storage device) as TransMux Streams are demultiplexed into FlexMux Streams and passed to appropriate

FlexMux demultiplexers that retrieve Elementary Streams This is described in Section 2 9 The ESs are parsed and passed to the appropriate decoders Decoding recovers the data in an AVO from its encoded form and performs the necessary operations to reconstruct the original AVO ready for rendering on the appropriate device Audio and visual objects are represented in their coded form, which is described in sections 2 11 and 2 12 respectively The reconstructed AVO is made available to the composition layer for potential use during scene rendering Decoded AVOs, along with scene description information, are used to compose the scene as described by the author Scene description and Composition are explained in Section 2 13 The user can, to the extent allowed by the author, interact with the scene that is eventually rendered and presented Section 2 14 describes this interaction

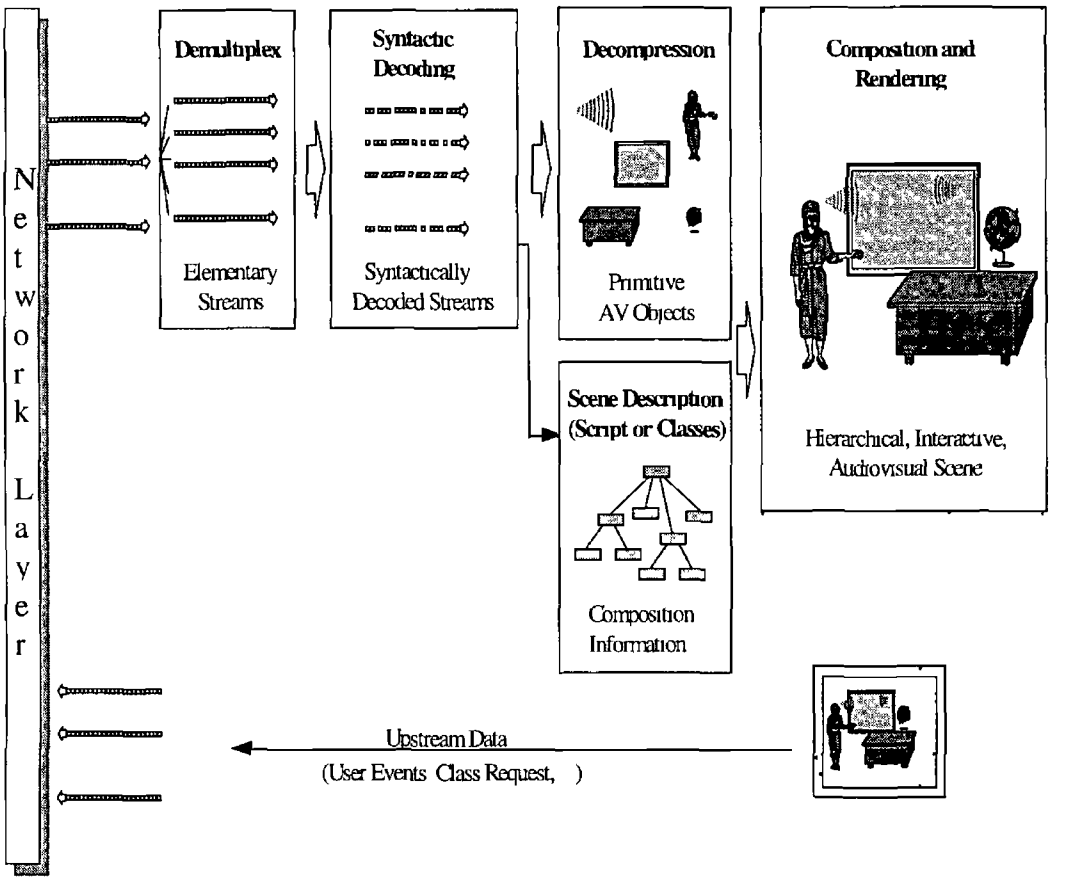


Figure 3 Major components of an MPEG-4 terminal (receiver side)

2 8 DMIF

The Delivery Multimedia Integration Framework (DMIF) addresses the operation of multimedia applications over interactive networks, in broadcast environments and from disks The DMIF architecture is such that applications, which rely on DMIF for communications, do not have to be concerned with the underlying communications method The implementation of DMIF takes care of the network details, presenting the application with a simple interface DMIF is located between the MPEG-4 application and the transport network as shown in Figure 4 below

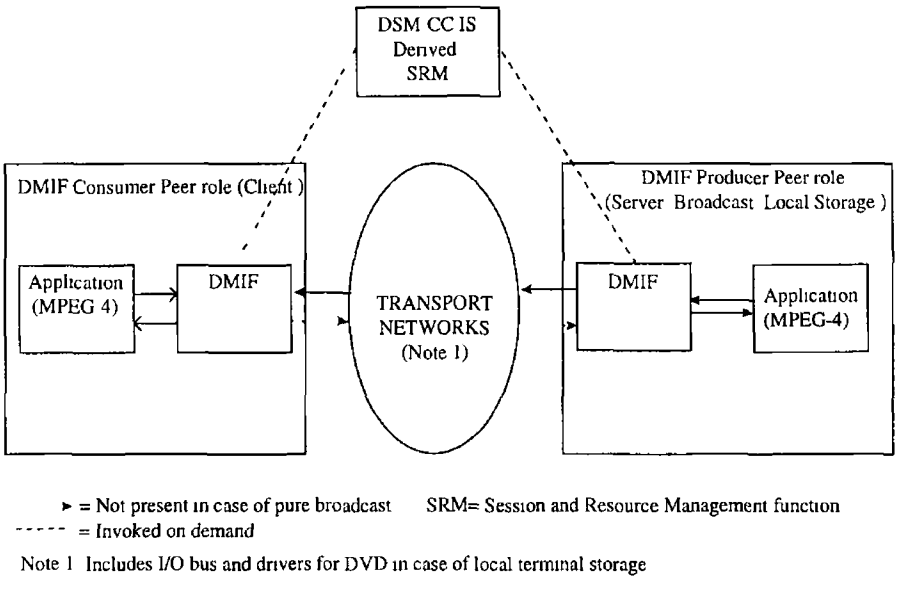


Figure 4 The DMIF Architecture

To the application, DMIF presents a consistent interface irrespective of whether MPEG-4 streams are received by interacting with a remote interactive DMIF peer over networks and/or by interacting with broadcast or storage media An interactive DMIF peer as shown in Figure 4, is an end-system on a network that can originate a session with a target peer A target peer can be an interactive peer, a set of broadcast MPEG-4 streams or a set of stored MPEG-4 files

An MPEG-4 application through the DMIF interface can establish a multiple peer application session Each peer is identified by a unique address A peer may be a remote interactive peer over a network or can be pre-cast (over broadcast or storage media) An interactive peer irrespective of whether it initiated the session can select a service, obtain a scene description and request specific streams for AVOs from the scene to be transmitted with the appropriate QoS

The MPEG-4 application can request from DMIF the establishment of channels with specific QoSs and bandwidths for each elementary stream DMIF ensures the timely

establishment of the channels with the specified bandwidths while preserving the QoSs over a variety of intervening networks between the interactive peers DMIF allows each peer to maintain its own view of the network, thus reducing the number of stacks supported at each terminal

Control of DMIF spans both the FlexMux and the TransMux layers shown in Figure 2 In the case of FlexMux, DMIF provides control of the establishment of FlexMux channels In the case of TransMux, DMIF uses an open interface, which accommodates existing and future networks through templates called connection resource descriptors MPEG-4 will offer a transparent interface with signalling primitive semantics These MPEG-4 semantics at the interface to DMIF are interpreted and translated into the appropriate native signalling messages of each network, with the help of relevant standards bodies having the appropriate jurisdiction In the area of QoS, MPEG-4 provides a first step towards defining a generic QoS parameter set for media at the DMIF interface The exact mapping for these translations are beyond the scope of MPEG-4 and are left to be defined by network providers

The DMIF SRM functionality in Figure 4 encompasses the MPEG-2 DSM-CC SRM functionality However, unlike DSM-CC, DMIF allows the choice whether or not to invoke SRM DMIF provides a globally unique network session identifier, which can be used to tag the resources and log their usage for subsequent billing

In a typical operation an end-user may access AVOs distributed over a number of remote interactive peers, broadcast and storage systems The initial network connection to an interactive peer may consist of a best effort connection over a ubiquitous network If the content warrants it, the end-user may seamlessly scale up the quality by adding enhanced AVO streams over connection resources with guaranteed QoS

2 9 Demultiplexing, buffer management and time identification

Individual Elementary Streams have to be retrieved from incoming data from some network connection or a storage device Each network connection or file is homogeneously considered a TransMux Channel in the MPEG-4 system model The demultiplexing is partially or completely done by layers outside the scope of MPEG-4, depending on the application For the purpose of integrating MPEG-4 in system environments, the Stream Multiplex Interface (see Figure 2) is the reference point Adaptation Layer-packetized Streams are delivered at this interface The FlexMux Layer specifies the optional FlexMux tool The TransMux Interface specifies how either AL-packetized Streams (no FlexMux used) or FlexMux Streams are to be retrieved from the TransMux Layer This is the interface to the transport functionalities not defined by MPEG The data part of the interfaces is considered here while the control part is dealt with by DMIF

In the same way that MPEG-1 and MPEG-2 described the behaviour of an idealised decoding device along with the bitstream syntax and semantics, MPEG-4 defines a System Decoder Model This allows the precise definition of the terminal's operation without making unnecessary assumptions about implementation details This is essential in order to give implementers the freedom to design real MPEG-4 terminals and decoding devices in a variety of ways These devices range from television receivers, which have no ability to communicate with the sender, to computers, which are fully enabled with bi-directional communication Some devices will receive MPEG-4 streams over isochronous networks while others will use non-isochronous means (e g , the Internet) to exchange MPEG-4 information The System Decoder Model provides a common model on which all implementations of MPEG-4 terminals can be based

The specification of a buffer and timing models is essential to encoding devices which may not know ahead of time what the terminal device is or how it will receive the encoded stream Though the MPEG-4 specification will enable the encoding device to

inform the decoding device of resource requirements, it may not be possible, as indicated earlier, for that device to respond to the sender It is also possible that an MPEG-4 session is received simultaneously by widely different devices, it will, however, be properly rendered according to the capability of each device

2 9 1 Demultiplexing

The retrieval of incoming data streams from network connections or storage media consists of two tasks First, the channels must be located and opened This requires a transport control entity, e g , DMIF Second, the incoming streams must be properly demultiplexed to recover the Elementary Streams from downstream channels (incoming at the receiving terminal) In interactive applications, a corresponding multiplexing stage will multiplex upstream data in upstream channels (outgoing from the receiving terminal) These elementary streams carry either AVO data, scene description information, or control information related to AVOs or to system management

The MPEG-4 demultiplexing stage is specified in terms of a conceptual two-layer multiplexer consisting of a TransMux Layer and a FlexMux Layer as well as an Access Unit Layer that conveys synchronisation information

The generic term 'TransMux Layer' is used to abstract any underlying multiplex functionality – existing or future – that is suitable to transport MPEG-4 data streams Note that this layer is not defined in the context of MPEG-4 Examples are MPEG-2 Transport Stream, H 223, ATM AAL 2, IP/UDP The TransMux Layer is modelled as consisting of a protection sublayer and a multiplexing sublayer indicating that this layer is responsible for offering a specific QoS Protection sublayer functionality includes error protection and error detection tools suitable for the given network or storage medium In some TransMux instances, it may not be possible to separately identify these sublayers

In any concrete application scenario one or more specific TransMux Instances will be used Each TransMux demultiplexer gives access to TransMux Channels The requirements on the data interface to access a TransMux Channel are the same for all TransMux Instances They include the need for reliable error detection, delivery, if possible, of erroneous data with a suitable error indication and framing of the payload, which may consist of either AL-packetized streams or FlexMux streams These requirements are summarised in an informative way in [5]

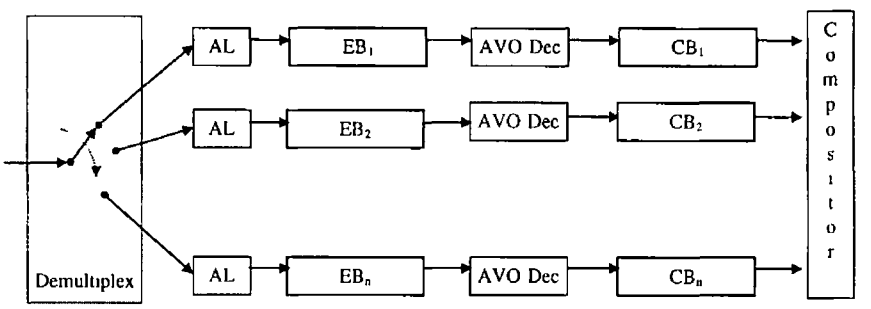
The FlexMux layer, on the other hand, is completely specified by MPEG It provides a flexible, low overhead, low delay tool for interleaving data that may optionally be used and is especially useful when the packet size or overhead of the underlying TransMux instance is large The FlexMux is not itself robust to errors and can either be used on TransMux Channels with a high QoS or to bundle Elementary Streams that are equally error tolerant The FlexMux requires reliable error detection and sufficient framing of FlexMux packets (for random access and error recovery) from the underlying layer These requirements are summarised in the Stream Multiplex Interface, which defines the data access to individual transport channels The FlexMux demultiplexer retrieves AL-packetized streams from FlexMux Streams

The Access Unit Layer has a minimum set of tools for consistency checking, and padding to convey time base information and to carry time stamped Access Units of an Elementary Stream Each packet consists of one Access Unit or a fragment of an Access Unit These time stamped Access Units form the only semantic structure of Elementary Streams that is visible on this layer The AU Layer requires reliable error detection and framing of each individual packet from the underlying layer, which can be accomplished, e g , by using the FlexMux How the compression layer can access data is summarised in [5] The AU Layer retrieves Elementary Streams from AL-packetized Streams

To be able to relate Elementary Streams to AVOs within a scene, Object Descriptors and StreamMapTables are used Object Descriptors convey information about the number and properties of Elementary Streams that are associated to particular AVOs The StreamMapTable links each stream to a ChannelAssociationTag that serves as a handle to the channel that carries this stream Resolving ChannelAssociationTags to the actual transport channel as well as the management of the sessions and channels is addressed by the DMIF part of the MPEG-4 standard

2 9 2 Buffer Management

To predict how the decoder will behave when it decodes the various elementary data streams that form an MPEG-4 session, the Systems Decoder Model enables the encoder to specify and monitor the minimum buffer resources that are needed to decode a session The required buffer resources are conveyed to the decoder within Object Descriptors during the set-up of the MPEG-4 session, so that the decoder can decide whether it is capable of handling this session



AL Access Unit Layer EB Elementary Stream Buffer AVO Dec AVO Decoder CB Composition Buffer

Figure 5 Buffer architecture of the System Decoder Model

By managing the finite amount of buffer space the model allows a sender, for example, to transfer non real-time data ahead of time, if sufficient space is available at the receiver side to store it The pre-stored data can then be accessed when needed, allowing at that time real-time information to use a larger amount of the channel's capacity if so desired

2 9 3 Time Identification

For real time operation, a timing model is assumed in which the end-to-end delay from the signal output from an encoder to the signal input to a decoder is constant Furthermore, the transmitted data streams must contain implicit or explicit timing information There are two types of timing information The first is used to convey the speed of the encoder clock, or time base, to the decoder The second, consisting of time stamps attached to portions of the encoded AV data, contains the desired decoding time for Access Units or composition and expiration time for Composition Units This information is conveyed in AL-PDU Headers generated in the Access Unit Layer With this timing information, the inter-picture interval and audio sample rate can be adjusted at the decoder to match the encoder's inter-picture interval and audio sample rate for synchronised operation

Different AVOs may be encoded by encoders with different time bases, with the accompanying slightly different speed It is always possible to map these time bases to the time base of the receiving terminal In this case, however, no real implementation of a receiving terminal can avoid the occasional repetition or drop of AV data, due to temporal aliasing (relative reduction or extension of their time scale)

Although systems operation without any timing information is allowed, defining a buffering model is not possible

2 10 Syntactic decoding

MPEG-4 defines a *syntactic description language* to describe the exact binary syntax of an AVO's bitstream representation as well as that of the scene description information This language is an extension of C^{++}, and is used to describe the syntactic representation of objects and the overall AVO class definitions and scene description information in an integrated way A more detailed description can be found in [6]

2 11 Coding of Audio Objects

MPEG-4 coding of audio objects provides tools for representing natural sounds (such as speech and music) and for synthesising sounds based on structured descriptions The representations provide compression and other functionalities, such as scalability or playing back at different speeds The representation for synthesised sound can be formed by text or instrument descriptions and by coding parameters to provide effects such as reverberation and spatialization [30]

2 11 1 Natural Sound

MPEG-4 standardises natural audio coding at bit-rates ranging from 2 kbit/s up to 64 kbit/s The presence of the MPEG-2 AAC standard within the MPEG-4 tool set will provide for general compression of audio in the upper bit rate range In order to achieve the highest audio quality within the full range of bit-rates and at the same time provide the extra functionalities, three types of coder have been defined The lowest bit-rate range is covered by parametric coding techniques Speech coding uses Code Excited Linear Predictive (CELP) For bit-rates starting below 16 kbit/s, time to frequency (T/F) coding techniques, namely the TwinVQ and AAC codecs, are applied This is illustrated in Figure 6

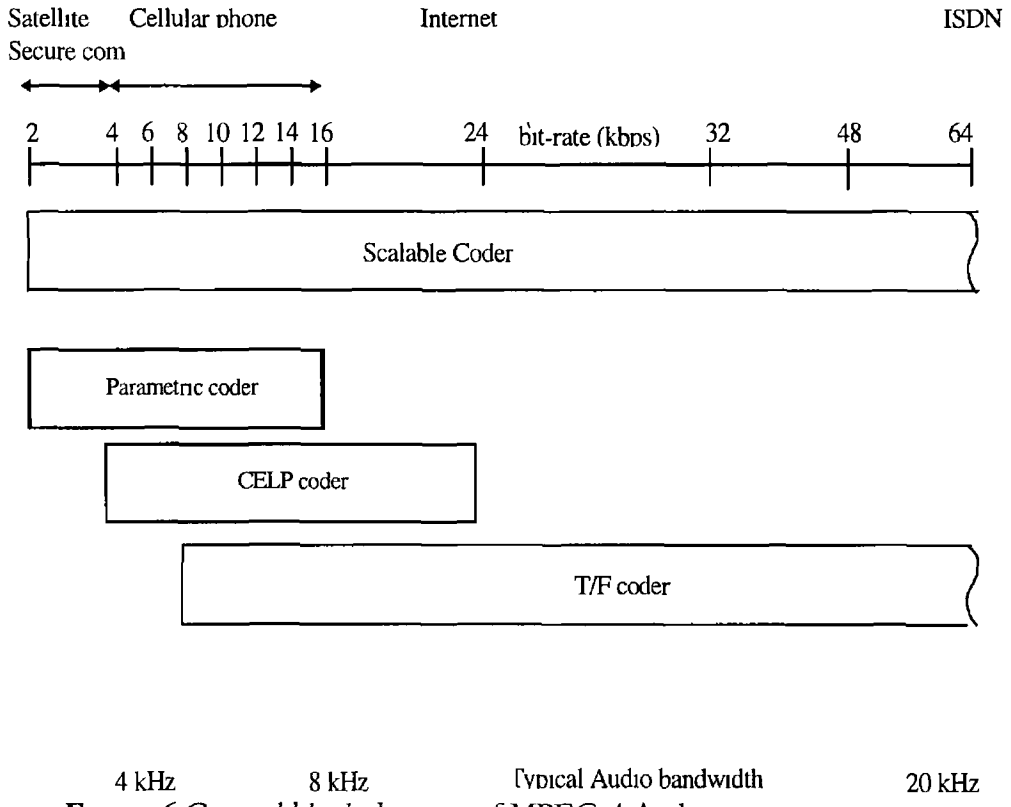


Figure 6 General block diagram of MPEG-4 Audio

2 1 1 2 Synthesised Sound

Decoders are also available for generating sound based on structured inputs Text input is converted to speech in the Text-To-Speech (TTS) decoder, while more general sounds including music may be normatively synthesised Synthetic music may be delivered at extremely low bit-rates while still describing an exact sound signal

Text To Speech TTS allows a text or a text with prosodic parameters (pitch contour, phoneme duration, and so on) as its inputs to generate intelligible synthetic speech

Score Driven Synthesis The Structured Audio Decoder decodes input data and produces output sounds This decoding is driven by a special synthesis language called SAOL (Structured Audio Orchestra Language) standardised as part of MPEG-4

MPEG-4 does not standardise "a method" of synthesis, but rather a method of describing synthesis A more detailed description of the coding of audio objects can be found in [7]

2 11 3 Effects

As well as being used for defining instruments, the SAOL language is used to describe special processing effects for use in the MPEG-4 Systems Binary Format for Scene Description The Audio BIFS system processes decoded audio data to provide an output data stream that has been manipulated for special effects with timing accuracy consistent with the effect and the audio sampling rate

2 12 Coding of Visual Objects

Visual objects can be either of natural or of synthetic origin

2 12 1 Natural Textures, Images and Video

The tools for representing natural video in the MPEG-4 visual standard aim at providing standardised core technologies allowing efficient storage, transmission and manipulation of textures, images and video data for multimedia environments These tools will allow the decoding and representation of atomic units of image and video content, called "video objects" (VOs) An example of a VO could be a talking person (without background) which can then be composed with other AVOs to create a scene Conventional rectangular imagery is handled as a special case of such objects [24] [25]

In order to achieve this broad goal the MPEG-4 standard provides solutions in the form of tools and algorithms for

- efficient compression of images and video
- efficient compression of textures for texture mapping on 2D and 3D meshes
- efficient compression of implicit 2D meshes

- efficient compression of time-varying geometry streams that animate meshes
- efficient random access to all types of visual objects
- extended manipulation functionality for images and video sequences
- content-based coding of images and video
- content-based scalability of textures, images and video
- spatial, temporal and quality scalability
- error robustness and resilience in error prone environments

The visual part of the MPEG-4 standard will provide a toolbox containing tools and algorithms bringing solutions to the above mentioned functionalities and more

2 12 2 Synthetic Objects

Synthetic objects form a subset of the larger class of computer graphics, as an initial focus the following visual synthetic objects will be described [23] [27]

- Parametric descriptions of
  - a synthetic description of human face and body
  - animation streams of the face and body
- Static and Dynamic Mesh Coding with texture mapping
- Texture Coding for View Dependent applications

2 12 2 1 facial animation

The shape, texture and expressions of the face are generally controlled by the bitstream containing instances of Facial Definition Parameter (FDP) sets and/or Facial Animation Parameter (FAP) sets Initially the Face object contains a generic face with a neutral expression If FDPs are received, they are used to transform the generic face into a particular face determined by its shape and (optionally) texture Optionally, a complete face model can be downloaded via the FDP set as a scene graph for insertion in the face node The Face object can also receive local controls that can be used to modify the look or behaviour of the face locally by a program or by the user

2 12 2 2 body animation

The Body object is capable of producing virtual body models and animations in the form of a set of 3D polygon meshes ready for rendering Two sets of parameters are defined for the body Body Definition Parameter (BDP) set, and Body Animation Parameter (BAP) set The BDP set defines the set of parameters to transform the default body to a customised body with its body surface, body dimensions, and (optionally) texture The Body Animation Parameters (BAPs) will produce body posture and animation on different body models No assumption is made and no limitation is imposed on the range of motion of joints

2 12 2 3 2D animated meshes

A 2D *mesh* is a tessellation (or partition) of a 2D planar region into polygonal patches The vertices of the polygonal patches are referred to as the *node points* of the mesh MPEG4 considers only triangular meshes where the patches are triangles Triangular meshes have long been used for efficient 3D object shape (geometry) modelling and rendering in computer graphics 2D-mesh modelling may be considered as projection of such 3D triangular meshes onto the image plane An example of a 2D mesh is depicted in Figure 7

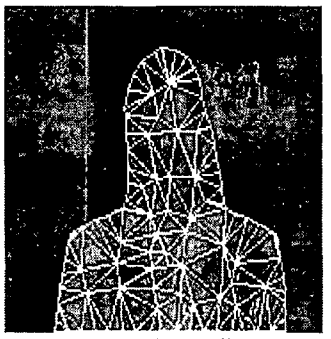


Figure 7 2D mesh modelling of the "Akiyo" video object

The attractiveness of 2D mesh modelling is that it is able to model the shape (polygonal approximation of the object contour) and motion of a VOP in a unified framework, which is also extensible to the 3D object modelling when data to construct such models is available In particular, the 2D-mesh representation of video objects enables the following functionalities

- Video Object Manipulation
- Video Object Compression
- Content-Based Video Indexing

2 12 2 4 Generic 3D meshes

The MPEG-4 visual standard will support generic meshes to represent synthetic 3D objects The toolbox will provide algorithms for

- Efficient compression of generic meshes
- (Level Of Detail) scalability of 3D meshes
- Spatial scalability

2 12 2 5 view dependent scalability

The view-dependent scalability enables scalability of stream texture maps that are used in realistic virtual environments It consists in taking into account the viewing position in the 3D virtual world in order to transmit only the most visible information

2 12 3 Structure of the tools for representing Natural Video

The MPEG-4 image and video coding algorithms will give an efficient representation of visual objects of arbitrary shape, with the goal to support so-called content-based functionalities It will also support MPEG-1 and MPEG-2

A basic classification of the bit rates and functionalities currently provided by the MPEG-4 visual standard for natural images and video is depicted in Figure 8 below, with the attempt to cluster bit-rate levels versus sets of functionalities

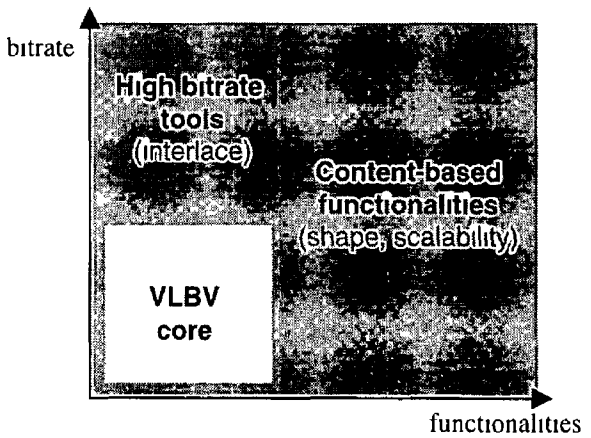


Figure 8 Classification of the MPEG-4 Image and Video Coding Algorithms and Tools

At the bottom end a "VLBV Core" (VLBV Very Low Bit-rate Video) provides algorithms and tools for applications operating at bit-rates typically between 5 64 kbits/s The basic applications specific functionalities supported by the VLBV Core include

- VLBV coding of conventional rectangular size image sequences with high coding efficiency and high error robustness/resilience, low latency and low complexity for real-time multimedia communications applications, and

- provisions for "random access" and "fast forward" and "fast reverse" operations for VLB multimedia database storage and access applications

The same basic functionalities outlined above are also supported at higher bit-rates

Content-based functionalities support the separate encoding and decoding of content This provides the most elementary mechanism for interactivity, flexible representation and manipulation with/of VO content of images or video in the compressed domain, without the need for further segmentation or transcoding at the receiver

2 12 4 Support for Conventional and Content-Based Functionalities

The MPEG-4 Video standard will support the decoding of conventional rectangular images and video as well as the decoding of images and video of arbitrary shape As in Figure 9 below

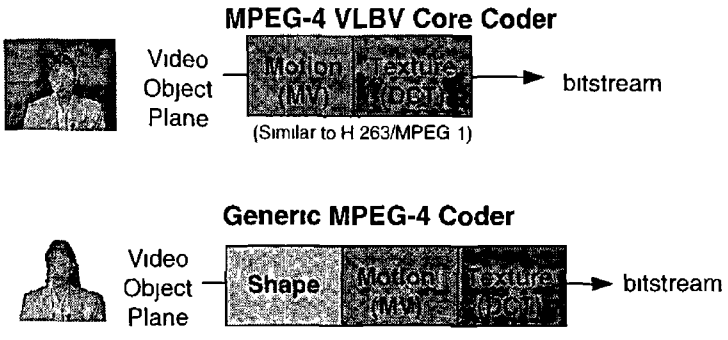


Figure 9 VLBV Core and the Generic MPEG-4 Coder

The coding of conventional images and video is achieved similar to conventional MPEG-1/2 coding and involves motion prediction/compensation followed by texture

coding For the content-based functionalities, where the image sequence input may be of arbitrary shape and location, this approach is extended by also coding shape and transparency information

2 12 5 Robustness in Error Prone Environments

MPEG-4 provides error robustness and resilience to allow accessing image or video information over a wide range of storage and transmission media In particular, due to the rapid growth of mobile communications, it is extremely important that access is available to audio and video information via wireless networks This implies a need for the useful operation of audio and video compression algorithms in error-prone environments at low bit-rates (i e , less than 64 Kbps)

A more detailed description of the coding of visual objects can be found in [8]

2 13 Scene description

In addition to providing support for coding individual objects, MPEG-4 also provides facilities to compose a set of such objects into a scene The necessary composition information forms the scene description, which is coded and transmitted together with the AVOs

In order to facilitate the development of authoring, manipulation, and interaction tools, scene descriptions are coded independently from streams related to primitive AVOs Special care is devoted to the identification of the parameters belonging to the scene description This is done by differentiating parameters that are used to improve the coding efficiency of an object (e g , motion vectors in video coding algorithms), and the ones that are used as modifiers of an object (e g , the position of the object in the scene) Since MPEG-4 should allow the modification of this latter set of parameters without having to decode the primitive AVOs themselves, these parameters are placed in the scene description and not in primitive AVOs

The following list gives some examples of the information described in a scene description

How objects are grouped together An MPEG-4 scene follows a hierarchical structure, which can be represented as a directed acyclic graph Each node of the graph is an AVO, as illustrated in Figure 10 (note that this tree refers back to Figure 1) The tree structure is not necessarily static, node attributes (e g , positioning parameters) can be changed while nodes can be added, replaced, or removed

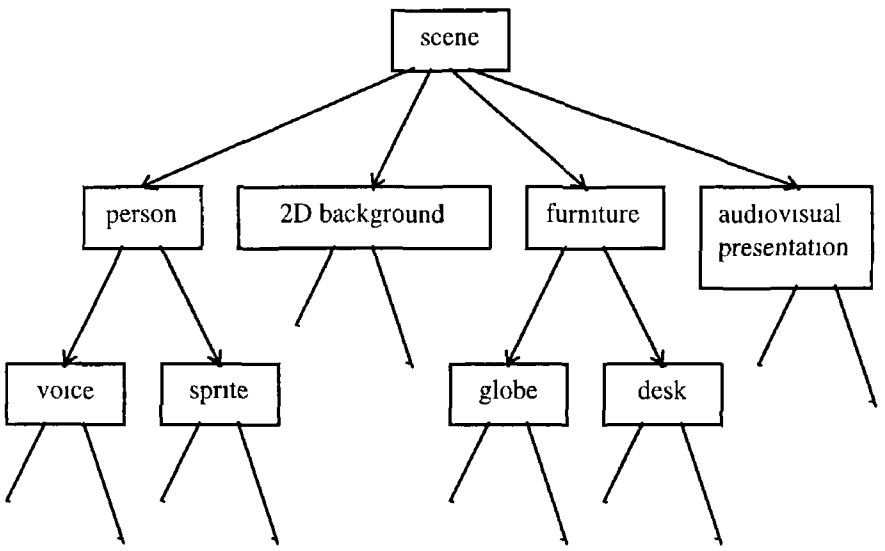


Figure 10 Logical structure of a scene

How objects are positioned in space and time In the MPEG-4 model, audio-visual objects have both a spatial and a temporal extent Each AVO has a local co-ordinate system A *local co-ordinate system* for an object is one in which the object has a fixed spatio-temporal location and scale The local co-ordinate system serves as a handle for manipulating the AVO in space and time AVOs are positioned in a scene by specifying a co-ordinate transformation from the object's local co-ordinate system into a global co-ordinate system defined by one or more parent scene description nodes in the tree

Attribute Value Selection Individual AVOs and scene description nodes expose a set of parameters to the composition layer through which part of their behaviour can be controlled Examples include the pitch of a sound, the colour for a synthetic object, activation or deactivation of enhancement information for scaleable coding, etc

Other transforms on AVOs The scene description structure and node semantics are heavily influenced by VRML, including its event model This provides MPEG-4 with a very rich set of scene construction operators, including graphics primitives that can be used to construct sophisticated 2D and 3D scenes

2 14 User interaction

MPEG-4 allows for user interaction with the presented content This interaction can be separated into two major categories client-side interaction and server-side interaction Client-side interaction involves content manipulation, which is handled locally at the end-user's terminal, and can take several forms In particular, the modification of an attribute of a scene description node, e g , changing the position of an object, making it visible or invisible, changing the font size of a synthetic text node, etc , can be implemented by translating user events (e g , mouse clicks or keyboard commands) to scene description updates The MPEG-4 terminal can process the commands in exactly the same way as if they originated from the original content source As a result, this type of interaction does not require standardisation

Other forms of client-side interaction require support from the scene description syntax, and are specified by the standard The use of the VRML event structure provides a rich model on which content developers can create compelling interactive content

Server-side interaction involves content manipulation that occurs at the transmitting end, initiated by a user action This, of course, requires that a back channel is available

Chapter

3 COMPOSITION & RENDERING OF BI & TRI DIMENSIONAL OBJECTS

3 1 Introduction

An MPEG-4 scene contains coded objects of a 2 and 3 dimensional nature In addition to providing support for the coding of the individual objects, the composition of such objects into a scene has been considered This scene description information has been coded independently from the coding of the objects in order to allow the modification of this former set of parameters without having to decode the primitive AVOs themselves

This chapter introduces the fundamental principles of bi and tri dimensional graphic composition and rendering The chapter is designed to explain how geometrical transformations based on matrix mathematics can be used to simplify the composition of 2D and 3D scenes It is through these transformations that graphics applications can create 2D renditions of 3D objects The chapter also introduces and explains the notion of an objects' local co-ordinate system and how a global co-ordinate system, the scene, can be created by combining different objects co-ordinate systems together Finally viewing projections of 2D and 3D objects are explained

3 2 Geometrical Transformations

3 2 1 2D Transformations

We can *translate* points in the (x,y) plane to new positions by adding translation amounts to the co-ordinates of the points For each point P(x,y) to be moved by d_x units parallel to the x axis and by d_y units parallel to the y axis to the new point P'(x',y'), we can write

$$x' = x + d_x, \qquad y' = y + d_y \tag{3 1}$$

If we define the column vectors

$$P = \begin{bmatrix} x \\ y \end{bmatrix}, P' = \begin{bmatrix} x' \\ y' \end{bmatrix}, T = \begin{bmatrix} d_x \\ d_y \end{bmatrix} \qquad (3\ 2)$$

then (3 1) can be expressed more concisely as

$$P' = P + T \qquad (3\ 3)$$

We could translate an object by applying Eq (3 1) to every point of the object Because each line in an object is made up of an infinite number of points, however, this process would take an infinitely long time Fortunately, we can translate all the points on a line by translating only the line's endpoints and by drawing a new line between the translated endpoints, this is also true of scaling and rotation

Points can be *scaled* by s_x along the x axis and by s_y along the y axis into new points by the multiplications

$$x' = s_x \ x, \qquad y' = s_y \ y \qquad (3\ 4)$$

In matrix form, this is

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad or \quad P' = S \ P \qquad (3\ 5)$$

where S is the matrix in Eq (3 5)

Points can be *rotated* through an angle θ about the origin A rotation is defined mathematically by

$$x' = x \ \cos\theta - y \ \sin\theta, \qquad y' = x \ \sin\theta + y \ \cos\theta \qquad (3\ 6)$$

In matrix form, we have

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad or \quad P' = R\ P \qquad (3\,7)$$

where R is the rotation matrix in Eq (3 7) Both the scaling and rotation matrices work about the origin Positive angles are measured counterclockwise from x toward y For negative (clockwise) angles, the identities cos(-θ) = cosθ and sin(-θ) = -sinθ can be used to modify Eqs (3 6) and (3 7)

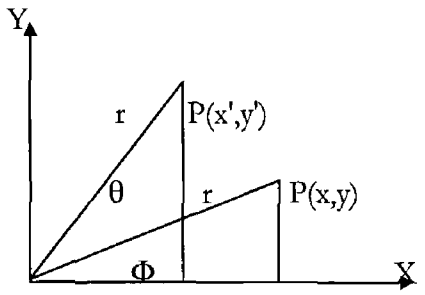


Figure 11 Derivation of the rotation equation

Equation (3 5) is easily derived from Figure 11, in which a rotation by θ transforms P(x,y) into P'(x',y') Because the rotation is about the origin, the distances from the origin to P and to P', labelled r in Figure 11, are equal By simple trigonometry, we find that

$$x = r\ \cos\phi, \qquad y = r\ \sin\phi \qquad (3\,8)$$

and

$$\begin{aligned} x' &= r\ \cos(\theta+\phi) = r\ \cos\phi\ \cos\theta - r\ \sin\phi\ \sin\theta, \\ y' &= r\ \sin(\theta+\phi) = r\ \cos\phi\ \sin\theta + r\ \sin\phi\ \cos\theta \end{aligned} \qquad (3\,9)$$

Substituting Eq (3 8) into Eq (3 9) yields Eq(3 6)

3 2 2 Homogeneous co-ordinates and matrix representation of 2D transformations

Unfortunately, translation is treated differently (as an addition) from scaling and rotation (as multiplications) If points are expressed in *homogeneous co-ordinates*, all three transformations can be treated as multiplications In homogeneous co-ordinates, we add a third co-ordinate to a point Instead of being represented by a pair of numbers (x,y), each point is represented by a triple (x,y,W) At the same time, we say that two sets of homogeneous co-ordinates (x,y,W) and (x',y',W') represent the same point if and only if one is a multiple of the other Thus (2,3,6) and (4,6,12) are the same points represented by different co-ordinate triples That is, each point has many different homogeneous co-ordinates representations Also, at least one of the homogeneous co-ordinates must be nonzero (0,0,0) is not allowed If the W co-ordinate is nonzero, we can divide through by it (x,y,W) represents the same point as (x/W, y/W,1) When W is nonzero, we normally do this division, and the numbers x/W and y/W are called the Cartesian co-ordinates of the homogeneous point The points with W = 0 are called the points at infinity

Triples of co-ordinates typically represent points in 3-space, but here we use them to represent points in 2-space The connection is this If we take all the triples representing the same point - that is, all triples of the form (tx, ty, tW), with $t \neq 0$ - we get a line in 3-space Thus, each homogeneous point represents a line in 3-space If we *homogenise* the point (divide by W), we get a point of the form (x,y,1) Thus, the homogenised points form the plane defined by the equation W =1 in (x,y,W) -space

Because points are now three-element row vectors, transformation matrices, which multiply a point vector to produce another vector, must be 3 X 3 In the 3 X 3 matrix form for homogeneous co-ordinates, the translation equations Eq (3 1) are

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad (3\ 10)$$

The scaling equations Eq (3 4) are represented in matrix form as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad (3\ 11)$$

The rotation equations Eq (3 6) can be represented as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad (3\ 12)$$

3 2 3 Composition of 2D transformations

The basic purpose of composing transformations is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformations, one after the other

Consider the rotation of an object about some arbitrary point P_1 Because we know how to rotate only about the origin, we convert our original problem into three separate problems Thus, to rotate about P_1, we need a sequence of three fundamental transformations

- Translate such that P_1 is at the origin
- Rotate
- Translate such that the point at the origin returns to P_1

The first translation is by (-x1, -y1), whereas the later translation is by the inverse (x1, y1) The net transformation is

$$T(x_1,y_1)\ R(\theta)\ T(-x_1,-y_1)=\begin{bmatrix}1&0&x_1\\0&1&y_1\\0&0&1\end{bmatrix}\begin{bmatrix}\cos\theta&-\sin\theta&0\\\sin\theta&\cos\theta&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&-x_1\\0&1&-y_1\\0&0&1\end{bmatrix}$$
$$=\begin{bmatrix}\cos\theta&-\sin\theta&x_1(1-\cos\theta)+y_1\sin\theta\\\sin\theta&\cos\theta&y_1(1-\cos\theta)-x_1\sin\theta\\0&0&1\end{bmatrix} \quad (3\ 13)$$

A similar approach would be used to scale an object about an arbitrary point P1 First, translate such that P1 goes to the origin, then scale, then translate back to P1

$$T(x_1,y_1)\ S(s_x,s_y)\ T(-x_1,-y_1)=\begin{bmatrix}1&0&x_1\\0&1&y_1\\0&0&1\end{bmatrix}\begin{bmatrix}s_x&0&0\\0&s_y&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&-x_1\\0&1&-y_1\\0&0&1\end{bmatrix}$$
$$=\begin{bmatrix}s_x&0&x_1(1-s_x)\\0&s_y&y_1(1-s_y)\\0&0&1\end{bmatrix} \quad (3\ 14)$$

3 2 4 The window-to-viewport transformation

Some graphics packages allow the programmer to specify output primitive co-ordinates in a floating-point world co-ordinate system, using whatever units are meaningful to the application program The term *world* is used because the application program is representing a world that is being interactively created or displayed to the user

Given that output primitives are specified in world co-ordinates, the graphics subroutine package must be told how to map world co-ordinates onto screen co-ordinates This is done by specifying a rectangular region in world co-ordinates, called the *world co-ordinate window*, and a corresponding rectangular region in screen co-ordinates, called the *viewport*, into which the world co-ordinate window is to be mapped

The transformation that maps the window into the viewport is applied to all of the output primitives in world co-ordinates, thus mapping them into screen co-ordinates If the window and viewport do not have the same height-to-width ratio, a non-uniform scaling occurs If the application program changes the window or viewport, then new output primitives drawn onto the screen will be affected by the change

Given a window and a viewport the transformation matrix that maps the window from world co-ordinates into the viewport in screen co-ordinates is as follows The window, specified by its lower-left and upper-right corners, is first translated to the origin of world co-ordinates Next, the size of the window is scaled to be equal to the size of the viewport Finally, a translation is used to position the viewport The overall matrix M_{wv} is

$$M_{wv} = T(u_{min}, v_{min})\ S\left(\frac{u_{max}-u_{min}}{x_{max}-x_{min}}, \frac{v_{max}-v_{min}}{y_{max}-y_{min}}\right)\ T(-x_{min}-y_{min})$$

$$= \begin{bmatrix} 1 & 0 & u_{min} \\ 0 & 1 & v_{min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{u_{max}-u_{min}}{x_{max}-x_{min}} & 0 & 0 \\ 0 & \frac{v_{max}-v_{min}}{y_{max}-y_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_{min} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{u_{max}-u_{min}}{x_{max}-x_{min}} & 0 & -x\ \frac{u_{max}-u_{min}}{x_{max}-x_{min}} + u_{min} \\ 0 & \frac{v_{max}-v_{min}}{y_{max}-y_{min}} & -y\ \frac{v_{max}-v_{min}}{y_{max}-y_{min}} + v_{min} \\ 0 & 0 & 1 \end{bmatrix} \qquad (3\ 15)$$

Multiplying $P=M_{wv}[x\ y\ 1]^T$ gives the expected result

$$P = \left[(x-x_{min})\ \frac{u_{max}-u_{min}}{x_{max}-x_{min}} + u_{min} \qquad (y-y_{min})\ \frac{v_{max}-v_{min}}{y_{max}-y_{min}} + v_{min} \qquad 1 \right] \qquad (3\ 16)$$

3 2 5 Matrix representation of 3D transformations

Just as 2D transformations can be represented by 3 X 3 matrices using homogeneous co-ordinates, so 3D transformations can be represented by 4 X 4 matrices, providing we use homogeneous co-ordinate representations of points in 3-space as well Thus, instead of representing a point as (x,y,z), we represent it as (x,y,z,W), where two of these quadruples represent the same point if one is a nonzero multiple of the other, the quadruple (0,0,0,0) is not allowed As in 2D, a standard representation of a point (x,y,z,W) with $W \neq 0$ is given by (x/W,y/W,z/W,1) Transforming the point to this form is called homogenising Also, points whose W co-ordinate is zero are called points at infinity There is a geometrical interpretation as well Each point in 3-space is being represented by a line through the origin in 4-space, and the homogenised representations of these points form a 3D subspace of 4-space which is defined by the single equation W = 1 The 3D co-ordinate system is right-handed hence positive rotations are such that, when looking from a positive axis toward the origin, a 90° counterclockwise rotation will transform one positive axis into the other

Translation in 3D is a simple extension from that in 2D

$$T(d_x,d_y,d_z)=\begin{bmatrix}1 & 0 & 0 & d_x\\ 0 & 1 & 0 & d_y\\ 0 & 0 & 1 & d_z\\ 0 & 0 & 0 & 1\end{bmatrix} \qquad (3\ 17)$$

Scaling is similarly extended

$$S(s_x,s_y,s_z)=\begin{bmatrix}s_x & 0 & 0 & 0\\ 0 & s_y & 0 & 0\\ 0 & 0 & s_z & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \qquad (3\ 18)$$

Rotations for z,x, and y axis's are respectively

$$R_z(\theta)=\begin{bmatrix}\cos\theta & -\sin\theta & 0 & 0\\ \sin\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \qquad R_x(\theta)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\theta & -\sin\theta & 0\\ 0 & \sin\theta & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$R_y(\theta)=\begin{bmatrix}\cos\theta & 0 & \sin\theta & 0\\ 0 & 1 & 0 & 0\\ -\sin\theta & 0 & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \qquad (3\ 19)$$

3 2 6 Transformations as a change in co-ordinate system

When we have multiple objects, each defined in its own local co-ordinate system, and we want to combine them so as to express these objects' co-ordinates in a single, global co-ordinate system it is useful to think of transformations as changes in co-ordinate systems

If we define $M_{i\leftarrow j}$ as the transformation that converts the representation of a point in co-ordinate system j into its representation in co-ordinate system i We define $P^{(i)}$ as the representation of a point in co-ordinate system i, $P^{(j)}$ as the representation of a point in co-ordinate system j, and $P^{(k)}$ as the representation of a point in co-ordinate system k, then,

$$P^{(i)} = M_{i\leftarrow j}\ P^{(j)} \qquad and \qquad P^{(j)} = M_{j\leftarrow k}\ P^{(k)} \qquad (3\ 20)$$

Substituting,

$$P^{(i)} = M_{i\leftarrow j}\ P^{(j)} = M_{i\leftarrow j}\ M_{j\leftarrow k}\ P^{(k)} = M_{i\leftarrow k}\ P^{(k)} \qquad (3\ 21)$$

So

$$M_{i \leftarrow k} = M_{i \leftarrow j} \; M_{j \leftarrow k} \qquad (3\ 22)$$

So we can think of each object as being defined in its own co-ordinate system and then being scaled, rotated, and translated by redefinition of its co-ordinates in the new world-co-ordinate system

3 3 Viewing in 3D

The 3D viewing process is inherently more complex than the 2D viewing process In 2D, we simply specify a window on the 2D world and a viewport on the 2D view surface Conceptually, objects in the world are clipped against the window and are then transformed into the viewport for display The extra complexity of 3D viewing is caused in part by the added dimension and in part by the fact that display devices are only 2D The solution to the mismatch between 3D objects and 2D displays is accomplished by introducing *projections*, which transform 3D objects onto a 2D projection plane

In 3D viewing, we specify a *view volume* in the world, a projection onto a projection plane, and a viewport on the view surface Conceptually, objects in the 3D world are clipped against the 3D view volume and are then projected The contents of the projection of the view volume onto the projection plane, called the window, are then transformed into the viewport for display Figure 12 shows this conceptual model of the 3D viewing process

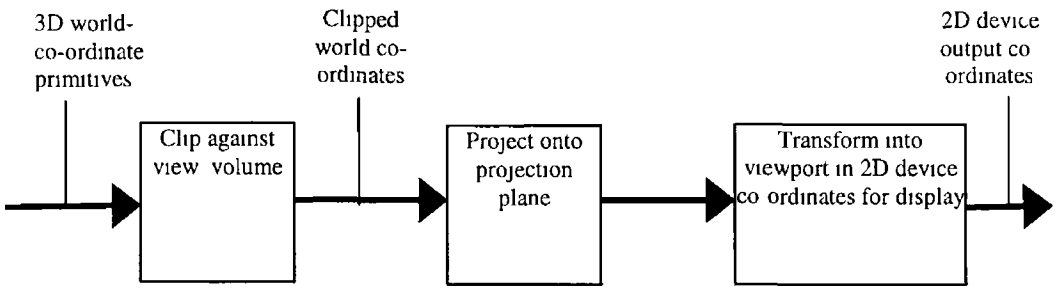


Figure 12 Conceptual model of the 3D viewing process

3 3 1 Projections

In general, projections transform points in a co-ordinate system of dimension *n* into points in a co-ordinate system of dimension less than *n* The projection of a 3D object is defined by straight projection rays emanating from a *centre of projection*, passing through each point of the object, and intersecting a *projection plane* to form the projection Figure 13 shows two different projections of the same line

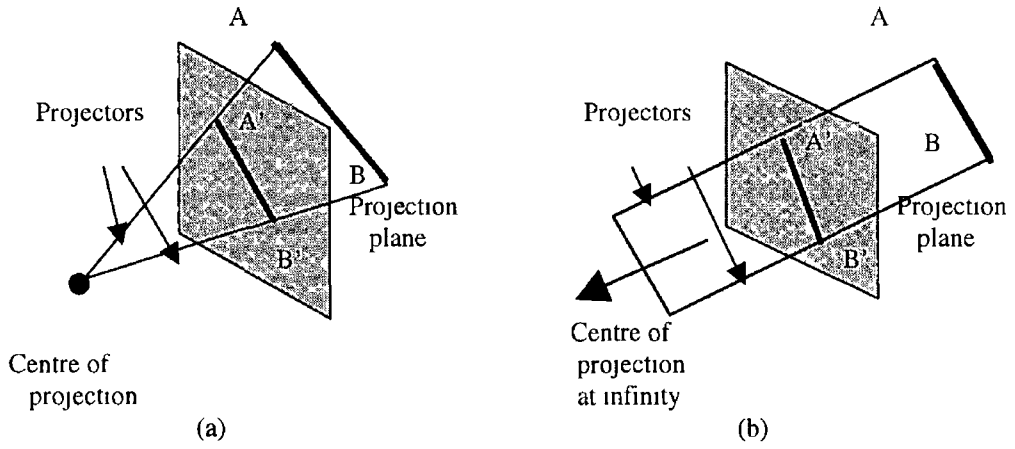


Figure 13 (a) Line AB and its perspective projection A'B' **(b)** Line AB and its parallel projection A'B' Projectors AA' and BB' are parallel

Projections can be divided into two basic classes perspective and parallel The distinction is in the relation of the centre of projection to the projection plane If the distance from one to the other is finite, then the projection is perspective if the distance is infinite, the projection is parallel

When defining a perspective projection, we explicitly specify its *centre of projection*, for a parallel projection, we give its *direction of projection* The centre of projection, being a

point, has homogenous co-ordinates of the form (x,y,z,1) Since the direction of projection is a vector (i e , a difference between points), it can be computed by subtracting two points d = (x,y,z,1) - (x',y',z',1) = (a,b,c,0) Thus, *directions* and *points at infinity* correspond in a natural way A perspective projection whose centre is a point at infinity becomes a parallel projection

The visual effect of a perspective projection is similar to that of photographic systems and of the human visual system, and is known as *perspective foreshortening* The size of the perspective projection of an object varies inversely with the distance of that object from the centre of projection Thus, although the perspective projection of objects tend to look realistic, it is not particularly useful for recording the exact shape and measurements of the objects, distances cannot be taken from the projection, angles are preserved only on those faces of the object parallel to the projection plane, and parallel lines do not in general project as parallel lines

The parallel projection is a less realistic view because perspective foreshortening is lacking, although there can be different constant foreshortening along each axis The projection can be used for exact measurements and parallel lines do remain parallel As with the perspective projection, angles are preserved only on faces of the object parallel to the projection plane

3 3 2 Perspective Projections

The perspective projections of any set of parallel lines that are not paiallel to the projection plane converge to a *vanishing point* In 3D, the parallel lines meet only at infinity, so the vanishing point can be thought of as the projection of a point at infinity There is of course an infinity of vanishing points, one for each of the infinity of directions in which a line can be oriented

If the set of lines is parallel to one of the three principal axes, the vanishing point is called an *axis vanishing point* There are at most three such points, corresponding to the

number of principal axes cut by the projection plane Perspective projections are categorised by their number of principal vanishing points and therefore by the number of axes the projection plane cuts

3 3 3 Parallel Projections

Parallel projections are categorised into two types, depending on the relation between the direction of projection and the normal to the projection plane In *orthographic* parallel projections, these directions are the same, so the direction of projection is normal to the projection plane

Axonometric orthographic projections use projection planes that are not normal to the principal axis and therefore show several faces of an object at once They differ from perspective projections in that the foreshortening is uniform rather than being related to the distance from the centre of projection Parallelism of lines is preserved but angles are not *Oblique projections*, the second class of parallel projections, differ from orthographic projections in that the projection-plane normal and the direction of projection differ The projection plane is normal to a principal axis, so the projection of the face of the object parallel to this plane allows measurement of angles and distances Other faces of the object project also, allowing distances along principal axes, but not angles, to be measured

3 3 4 Specifying an arbitrary 3D view

3D viewing involve not just a projection but also a view volume against which the 3D world is clipped The projection and view volume together provide all the information needed to clip and project into 2D space Then, the 2D transformation into physical device co-ordinates is straightforward The projection plane or *view plane* is defined by a point on the plane called the *view reference point* (VRP) and a normal to the plane called the *view-plane normal* (VPN)

Given the view plane, a window on the view plane is needed The window's role is similar to that of a 2D window its contents are mapped into the viewport, and any part of the 2D world that projects onto the view plane outside of the window is not displayed

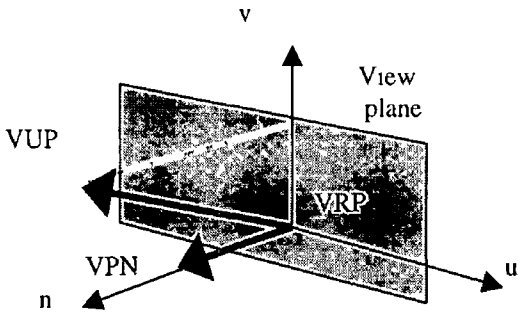


Figure 14 The view plane is defined by VPN and VRP, the v axis is defined by the projection of VUP along VPN onto the view plane The u axis forms the right-handed viewing reference-co-ordinate system with VPN and v

To define a window on the view plane, we need some means of specifying minimum and maximum window co-ordinates along two orthogonal axes These axes are part of the 3D *viewing-reference co-ordinate* (VRC) system The origin of the VRC system is the VRP One axis of the VRC is VPN, this axis is called the *n* axis A second axis of the VRC is found from the *view up vector* (VUP), which determines the *v*-axis direction on the view plane The v-axis is defined such that the projection of VUP parallel to VPN onto the view plane is coincident with the v axis The *u*-axis direction is defined such that u, v, and n form a right-handed co-ordinate system, as in Figure 14 The VRP and the two direction vectors VPN and VUP are specified in the right-handed world-co-ordinate system

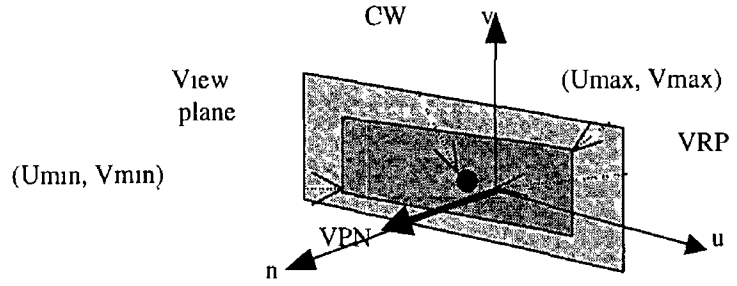


Figure 15 The view reference-co-ordinate system (VRC) is a right-handed system made up of the u, v, and n axes The n axis is always the VPN CW is the centre of the window

With the VRC system defined, the window's minimum and maximum u and v values can be defined as in Figure 15 The centre of projection and direction of projection (DOP) are defined by a *projection reference point* (PRP) plus an indicator of the projection type If the projection type is perspective, then PRP is the centre of projection If the projection type is parallel, then the DOP is from the PRP to CW The CW is in general not the VRP, which need not even be within the window bounds

The PRP is specified in the VRC system, not in the world-co-ordinate system, thus, the position of the PRP relative to the VRP does not change as VUP or VRP are moved The advantage of this is that the programmer can specify the direction of projection required and then change VPN and VUP (hence changing VRC), without having to recalculate the PRP needed to maintain the desired projection On the other hand, moving the PRP about to get different views of an object may be more difficult

The view volume bounds that portion of the world that is to be clipped out and projected onto the view plane For a perspective projection, the view volume is the semi-infinite pyramid with apex at the PRP and edges passing through the corners of the window

Figure 16 shows a perspective-projection view volume Positions behind the centre of projection are not included in the view volume and thus are not projected For parallel projections, the view volume is an infinite parallelepiped with sides parallel to the direction of projection, which is the direction from the PRP to the centre of the window

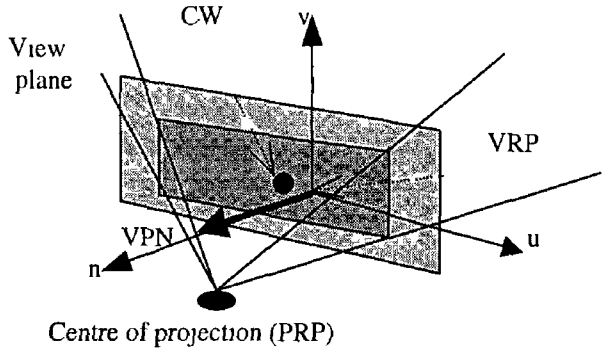


Figure 16 The semi-infinite pyramid view volume for perspective projection CW is the centre of the window

In order to limit the number of output primitives projected onto the view plane we need the view volume to be finite This is done with a front clipping plane and back clipping plane which are parallel to the view plane, their normal is the VPN The planes are specified by the signed quantities front distance (F) and back distance (B) relative to the view reference point and along the VPN, with positive distances in the direction of the VPN For the view volume to be positive, the front distance must be algerbraically greater than the back distance Dynamic modification of either the front or rear distances can give the viewer a good sense of the spatial relationships between different parts of the object as these appear and disappear from view

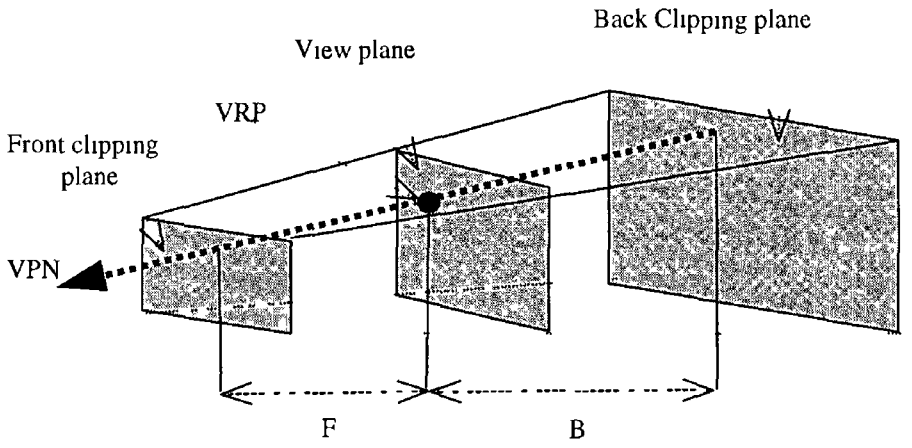


Figure 17 Truncated view volume

Chapter

4 EVOLUTION OF THE SCENE DESCRIPTION IN MPEG-4

4 1 Introduction

MPEG-4 addresses the coding of objects of various types Not only traditional video and audio frames, but also natural video and audio objects as well as textures, text, 2- and 3-dimensional graphic primitives, and synthetic music and sound effects To reconstruct a multimedia scene at the terminal, it is hence no longer sufficient to encode the raw audio-visual data and transmit it, as MPEG-2 does, in order to convey a video and a synchronised audio channel In MPEG-4, all objects are multiplexed together at the encoder and transported to the terminal Once de-multiplexed, these objects are composed at the terminal to construct and present to the end user a meaningful multimedia scene The placement of these elementary AVOs in space and time is described in what is called the Scene Description layer The action of putting these objects together in the same representation space is called the Composition of AVOs While the action of transforming these AVOs from a common representation space to a specific rendering device (speakers and a viewing window for instance) is called Rendering

The independent coding of different objects may achieve a higher compression rate, but also brings the ability to manipulate content at the terminal The behaviours of objects and their response to user inputs can thus also be represented in the Scene Description layer, allowing richer multimedia content to be delivered as an MPEG-4 stream This chapter is a detailed analysis of how scene description languages function and how the functionality of the MPEG-4 scene description language has been developed since its conception

4 2 Scene Description

In addition to providing support for coding individual objects, MPEG-4 also provides facilities to compose a set of such objects into a scene The scene description information is composed of the composition details of the various AVOs in the scene Scene descriptions are coded independently from streams related to primitive AVOs Special care is devoted to the identification of the parameters belonging to the scene description This is done by differentiating parameters that are used to improve the coding efficiency of an object (e g motion vectors in video coding algorithm), from those used as modifiers of an object's characteristics within the scene (e g position of the object in the global scene) In keeping with MPEG-4's objective to allow the modification of this latter set of parameters without having to decode the primitive AVOs themselves, these parameters form part of the scene description and are not part of the primitive AVOs The idea was to standardise a syntax that describes the spatio-temporal relationships of Scene Objects

4 3 Initial 2D Scene Description

Initially two ways were identified to describe the composition for 2D scenes The first, fixed scene description was mainly aimed at describing the composition parameters for the 2D video objects described in the initial verification model (VM) There was no notion of a hierarchical scene structure The AVOs were video object planes (VOPs) which were positioned with respect to the 2D frame in which they are composited The second case dealt with more complex 2D scene structures The two proposals are now described

4 3 1 2D Fixed Scene Description

At any given time, a scene is composed of a collection of objects By default, the objects are displayed as specified in the object stream (video object stream for instance) Additional warping transforms can be applied, by sending motion parameters in a composition stream These parameters are timestamped to indicate at what time the

decoded object should be transformed and presented, they are also related to an object by the video object id According to the timestamp, objects are requested in order to be transformed according to the motion parameters sent in the composition stream

4 3 2 2D Flexible Scene Description

In this implementation the 2D scene structure was transmitted as a program Each AVO is sent as a class The methods of the class formed the scene description

In order for Fixed and Flexible scene descriptions to be implemented the notion of composition flexibility was developed

4 4 Composition Flexibility

In its previous standards, MPEG defined rigid a priori known templates for transmitted information What composition flexibility was defined to do was create a representation of these templates that could be transmitted to configure the receiving system MPEG-4 initially defined two types of profiles for receiver programmability the fixed profiles and the flexible profiles

4 4 1 Fixed Profiles

In the fixed profiles, programmability is achieved through the use of switches or selectors in the binary stream The switches or selectors are *n*-ary elements that select which of *n* pre-defined templates will be used for the incoming information This allows, for example, the choice of a pre-defined standardised configuration This kind of programmability has the nice feature of being simple, practical and bit efficient which is a major requirement of prospective users of the MPEG-4 standard

4 4 2 Flexible Profiles

MPEG-4 defined enhanced profiles on receiver programmability, the flexible profiles These profiles allow the communication of information templates To represent these

templates, a possibility is to rely on classes (in the object-oriented sense of the word) or to send scripts to reconfigure the application

The flexible profiles require

- a definition of a set of standardised APIs (Application Program Interfaces),
- a definition of a standardised format to download templates If this format is executable, it has to be processor independent,
- a standardised protocol for downloading and installing templates in the above format in the flexible terminal

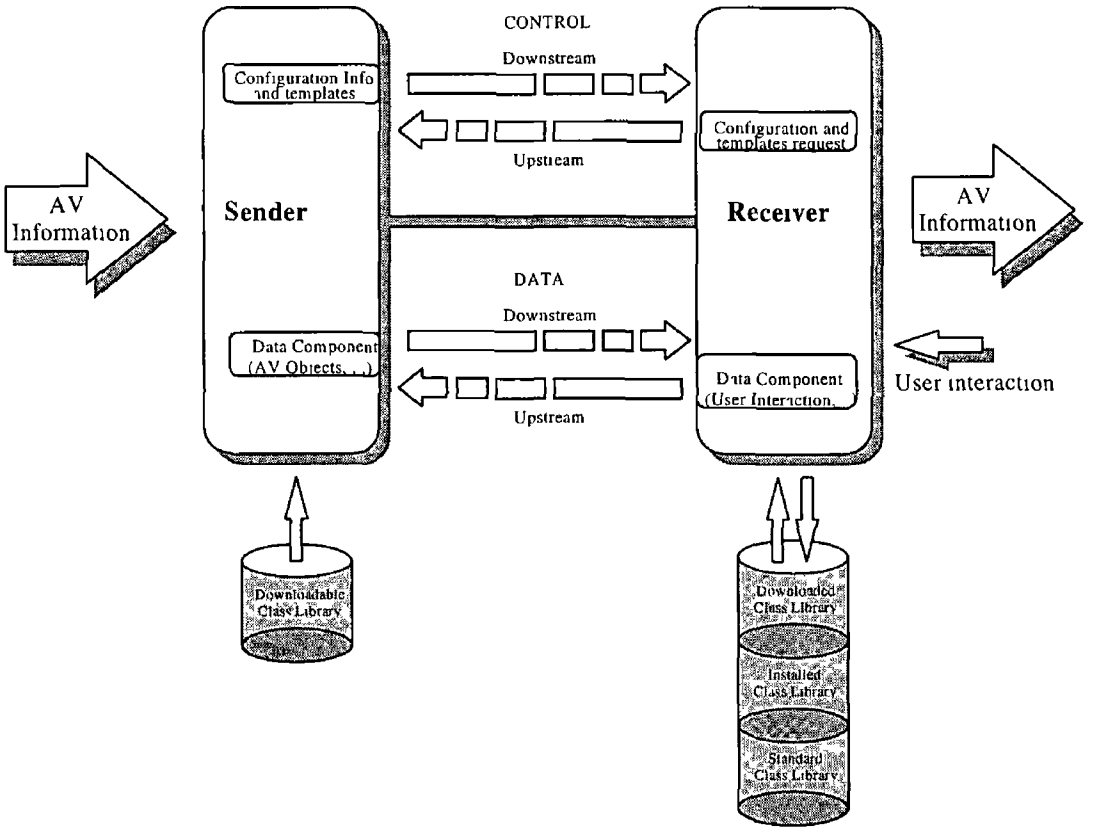


Figure 18 Flexible Configuration

The communication of AVOs within the context of flexible profiles is as follows Before the AVOs are transmitted, the sender and receiver exchange configuration information The sender determines which algorithms, tools, and other objects are needed by the receiver to process the AVOs The definitions of missing audio-visual information are downloaded to the receiver, where they supplement or override existing definitions, whether installed or pre-defined

As the receiver runs, new templates may be needed In such a case, the receiver can request the download of specific additional information templates The additional templates may be downloaded in parallel with the transmitted data The above aspects are illustrated in Figure 18

4 5 Scene Description of the initial MPEG-4 Verification Model

The MPEG-4 Verification Model was designed as a testbed for emerging ideas during the evolution and development of the MPEG-4 standard It is seen as an implementation of the standard The idea being that several different implementations of functionalities can exist but they must comply with the standard, which is set about in the verification model [28]

During the development of the initial verification model one of the major concerns was how we could implement the different levels of composition flexibility What was required was not only a standardised format to download templates but also a standardised protocol for downloading and installing the templates If this format was to be executable it had to be processor independent This seemed a huge task, but at the time we started to develop our verification model a new programming language called JAVA from Sun Microsystems had just been launched which seemed to overcome the above problems

4 5 1 The JAVA Development Environment

In [9] Sun describes JAVA as follows

JAVA A simple, object oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language

What made JAVA interesting as a possible development environment for the verification model was the fact that it is distributed, interpreted, robust, secure, architecture neutral, portable, and dynamic

4 5 1 1 Distributed

JAVA has an extensive library of routines for coping easily with TCP/IP protocols like HTTP and FTP This makes creating network connections much easier than in C or C++ JAVA applications can open and access objects across the net via URLs with the same ease that programmers are used to when accessing a local file system

4 5 1 2 Interpreted

The JAVA compiler generates byte-codes, rather than native machine code JAVA bytecodes provide an architecture neutral object file format, the code is designed to transport programs efficiently to multiple platforms

4 5 1 3 Robust

JAVA is intended for developing software that must be robust, highly reliable, and secure, in a variety of ways There's strong emphasis on early checking for possible problems, as well as later dynamic (run-time) checking, to eliminate error-prone situations

4 5 1 4 Architecture Neutral

The JAVA compiler doesn't generate "machine code" in the sense of native hardware instructions--rather, it generates bytecodes a high-level, machine-independent code for a hypothetical machine that is implemented by the JAVA interpreter and run-time system, the JAVA virtual machine Which means that if the JAVA run-time system is made available on a given hardware and software platform, an application written in

JAVA can then execute on that platform without the need to perform any special porting work for that application

4 5 1 5 Secure

The JAVA language compiler and run-time system implement several layers of defence against potentially incorrect code The environment starts with the assumption that nothing is to be trusted, and proceeds accordingly

- Memory layout is deferred to run time, and will potentially differ depending on the characteristics of the hardware and software platforms on which the JAVA system executes

- Compiled code references memory via symbolic "handles" that are resolved to real memory addresses at run time by the JAVA interpreter, hence programmers can't forge pointers to memory

- Very late binding of structures to memory means that programmers can't infer the physical memory layout of a class by looking at its declaration

- The JAVA run-time system doesn't trust the incoming code, but subjects it to bytecode verification The bytecode verifier traverses the bytecodes, constructs the type state information, and verifies the types of the parameters to all the bytecode instructions

4 5 1 6 Portable

JAVA defines a standard behaviour that will apply to the data types across all platforms and specifies the sizes of all its primitive data types and the behaviour of arithmetic on them

4 5 1 7 Dynamic

The JAVA language's portable and interpreted nature produces a highly dynamic and dynamically-extensible system The JAVA language was designed to adapt to evolving environments Classes are linked in as required and can be downloaded from across networks Incoming code is verified before being passed to the interpreter for execution

So our approach was to use JAVA to define templates for both the fixed and flexible profiles These templates are designed to be audio visual objects, which form the tree structure of the scene Since JAVA allows us to create classes, which can be run on any JAVA Virtual Machine, i e are processor independent, and can be easily downloaded across a number of network protocols, it was decided that the initial verification model would be developed using JAVA JAVA classes would be used to create the various MPEG-4 templates

4 6 Development of the MPEG-4 Class Library

Initially what was required was to identify the templates, i e classes, which were going to be needed to implement an MPEG-4 compositor within the verification model The following section is a description of the various JAVA classes that were implemented

4 6 1 Class Library

This section defines the set of classes called the MPEG-4 Standard Class Library The Standard Class Library is the minimal set of classes that an MPEG-4 terminal must implement in order to support every MPEG-4 application that uses flexibility (Individual profiles may require implementation of only a subset of the Standard Class Library) Each class in the Standard Class Library corresponds to an MPEG-4 tool or algorithm, and has a specified interface through which commands and data are passed These interfaces collectively constitute an application program interface (API) for

MPEG-4 The MPEG-4 Standard Class Library is defined in terms of this API, along with a description of the intended relationships between the MPEG-4 tools, algorithms, downloaded AVOs, and execution environment

The classes in the Standard Class Library naturally fall into categories according to layers in the MPEG-4 decoder architecture We are only concerned about the scene description and composition layers These are made up of the AVObject layer, the Composition layer, and the Presentation layer

4 6 2 AVObject Layer Classes

- AVObject

AVObject is a base class that inherits from the MPEG4Object class, the parent MPEG-4 class, and from which all audio and visual objects derive

- VideoObject (extends AVObject)

VideoObject is an AVO that uses one of the standard video decoding process objects to decode its input elementary stream A header in the elementary stream specifies which of the decoding process objects to use i e MPEG-4, MPEG-2, H263 etc

- Image (extends VideoObject)

An Image is a primitive AVO that represents a rectangular array of pixels In general, the image may have multiple colour components, multiple fields (e g , for interlaced displays), and multiple channels (e g , for stereo displays) The colour components, if any, may lie in various colour spaces, and may be subsampled with respect to each other Colour components that are not specified in the Image default to values in the current Properties sheet , Composition Layer class, in the Compositor when the Image is rendered

4 6 3 Composition Layer Classes

- Compositor

A compositor is a tightly coupled video and audio compositor A compositor contains references to the following objects

- an output video frame,
- an output audio frame,
- a viewpoint,
- a transform stack for co-ordinate transformations,
- a properties stack for object rendering properties,
- an input elementary stream,
- an output elementary stream
- an environment containing a list of (attribute, value) pairs for use in the passing of generic messages between AVOs

The primary functions of a compositor are methods for rendering AVOs onto the current video and audio frames, using the current properties (as needed), the current transform, and the viewpoint Stream references exist in order to assist compound AVOs in passing sub-streams to sub-objects

- Transform

A Transform object is, semantically, a 5x5 homogeneous co-ordinate transformation matrix T

$$\begin{matrix} a & b & c & 0 & d \\ e & f & g & 0 & h \\ i & j & k & 0 & l \\ 0 & 0 & 0 & m & n \\ 0 & 0 & 0 & 0 & 1 \end{matrix}$$

It is used for performing geometrical transformations on AVOs

- Properties

A Properties sheet is primarily a list of the current default properties for any primitive AVOs that are rendered If a primitive AVO, when rendered through a compositor, does not specify a needed property, then that property will be taken from the current Properties sheet in the compositor

- Viewpoint

A Viewpoint is an object in a local co-ordinate system that can be rendered like any AVO, at a given location, orientation, and scale within a scene Rendering a viewpoint causes its scene-to-local co-ordinate transformation to be stored in the Compositor's viewpoint object, for use in subsequent rendering

4 6 4 Presentation Layer Classes

- Presenter

A presenter is the subsystem responsible for displaying the scene reconstructed by the Compositor It is also the subsystem responsible for handling events generated by the user interacting with the presentation

4 7 Implementation of an initial MPEG-4 compliant viewer

When we started to build the MPEG-4 compliant viewer we did so in order to validate and test the standardised set of APIs and classes What we built was a JAVA based viewer, a primer on the viewer can be found in [10] This involved developing a viewer package Each package is a group of classes, which have some common functionality We developed an MPEG-4 package, which contained all the standardised classes, and a viewer package, which was not part of the MPEG standard but was developed as a verification model for the standard It consisted of the following classes

Viewer Simply wraps an interface around an Executive

Executive The Executive presents the frames to the user, and also periodically funnels user feedback to the Scene object

Presenter Subsystem responsible for presentation of video and audio frames to the user and for the collection of user input events

4 7 1 MoMuSys Viewer

The MoMuSys viewer was an MPEG-4 viewer that was developed under the auspices of the European ACTS Projects The following is a description of how the viewer functioned The Viewer class is the first instantiated class This class instantiates all the other needed objects for the interface of the application, and MPEG-4 audio-visual objects

4 7 1 1 Viewer

Viewer owns a Compositor, an Executive, a Presenter object, and the "main" AVO which is the top AVO of the scene All these objects are gathered in one class to be accessible everywhere in the viewer

4 7 1 2 Presenter

The display and event handling of the scenes takes place in an X11 window as the viewer was developed on an SGI machine using the X11 windowing system The Presenter class is the link between the JAVA part of the viewer and native code for the X11 window To handle an event, the Presenter passes an event object to the native code Events that occur in the X11 window are converted and copied into the MPEG4Event object

4 7 1 3 Executive

The Executive object is a thread It can be seen as the operating system of the display of a scene Its *run* method is an infinite loop and performs the following steps

- Initialisation of the inputstream and outputstream of the compositor, and instantiation of the top AVO scene (the "main" AVO of the viewer)

- Beginning of the following infinite loop

  - Clear the compositor's frame

  - Rendering of the top AVO of the scene During this phase, all the objects of the scenes are mapped onto the compositor's frame, according to the transformation matrix This frame is then ready to be displayed in the X11 window

  - Display of the frame of the compositor in the X11 window During this phase, the X11 window is refreshed with the new image of the scene It is a kind of double buffering At the same time, the Presenter looks at the X11 event stack and takes the oldest event according to the predefined X11 event mask This event is converted and copied into the MPEG4Event object

  - Handling of the Presenter's MPEG4Event object by the scene, by calling the handle method of the top AVO of the scene

4 7 2 How the MoMuSys Viewer functions?

4 7 2 1 Overview of how the MPEG-4 Viewer displays audio-visual coded information

The decoder receives a class definition for a main AVO, which is inherited from the MPEG-4 package class AVObject This class is instantiated as the root of the hierarchical scene graph and its render method is called once for each audio-visual frame that the decoder wishes to present The root render method invokes other methods and other objects, for example

- calls to render methods of related AVObjects,
- calls to methods of decoding process objects to recover image and audio objects from encoded data streams,
- calls to parsing or entropy decoding methods to extract syntactic decoded data streams from elementary streams,
- calls to demultiplexing methods to extract elementary data streams from logical input channels

4 7 2 2 What a Compositor does and how it is instantiated?

The Compositor class composes, renders, and blends audio-visual objects onto output audio and video frames The compositor maintains one audio frame, a finite sequence of audio samples, or one video frame, a rectangular array of pixels, for each output channel Audio-visual effects are produced frame by frame The compositor controls the spatio-temporal mapping of the scene, the default audio-visual rendering properties, the projection and clipping planes, the acoustic sink points, the input data stream, the output data stream, and the passing of data between audio-visual objects

4 7 2 3 How AVObjects are rendered?

The difficult work done by a Compositor is rendering audio-visual objects The purpose of an audio-visual object's *render* method is to render the object onto the specified compositor The compositor doesn't know intrinsically how to render encoder-defined audio-visual objects, but the objects know how to render themselves So when the render method of the compositor is called upon it calls the object's render method, with itself as the argument

4 7 2 4 How events are handled by AVObjects?

The *handle* method of an audio-visual object is designed to deal with synchronously generated script, or asynchronously generated user input In an encoder-defined audio-visual object the body of the routine consists of a script that describes step by step how to handle events, which at the very least examines the event structure and passes the events to the objects sub-objects

4 7 2 5 How decoding of AVObjects works?

ProcessObjects are the decoding tools used by AVObjects to decode themselves In addition to the render and handle methods of AVObjects many may have a *decode* method The decode method of an AVObject decodes the attributes of the object itself It builds and instantiates from a coded representation all the attributes of the AVObject

4 7 2 6 Presenting the AVObject

The *presenter* class is the subsystem responsible for presentation of video and audio frames to the user, and for the collection of user input events

4 8 Expanding the MPEG-4 Class Library to handle VOPs

4 8 1 VOP Definition

As defined in [11] Video Objects (VOs) correspond to entities in the bitstream that the user can access and manipulate (cut, paste) Instances of Video Object in given time are called Video Object Planes (VOPs) A VOP can be a semantic object in the scene it is made of Y, U, V components plus shape information The encoder sends together with the VOP, composition information (using the composition layer syntax) to indicate where and when each VOP is to be displayed At the decoder side the user may be allowed to change the composition of the scene displayed by interacting with the composition information

4 8 2 Creating a VOP Class

What was required was to create a class or template, which could be used to read in luminance and chorminance (YUV) VOPs and display them in the verification model

The VOP sequence was to be based on a QCIF sequence of frames In the QCIF file format each frame is 176 * 144 pixels, width * height The sequence was sampled at 25Hz Each frame is made up of YUV values stored in the 4 2 2 format This implies that 4 bytes of luminance, Y, and 2 bytes each of chrominance, UV, go to make up each pixel displayed Each VOP frame is stored as a chain of Y, U, V data without gaps The frame is stored from the 1st line, 1st pixel, from left to right, top to bottom, down to the last line, last pixel Associated with each YUV frame is an alpha plane This is a binary mask representing the shape of an object within the frame A value of 0 is used to indicate a pixel outside of the object and the value 255 is used to indicate a pixel inside the object The mask is used to composite the YUV pixel values of the object, while those pixels outside the object are not composited

So what was required for each frame was read in the YUV sequence and then decide using the alpha plane what the dimensions of the image to be displayed were A cut down version of the VM was developed as a test bed It consisted of a JAVA

compositor, which was a scaled down version of the official MPEG one An interface to read in both the YUV and segmentation mask files was developed The development process was to create an extension of the MPEG-4 Image class to composite the YUV VOP In order to composite the VOP conversion from YUV to an RGB format was necessary

The process now involved reading in a YUV frame, mask out the unnecessary pixels and convert to RGB The first two stages were straight forward and a mathematical formula for conversion from YUV to RGB was easily developed When tested this conversion functioned correctly, however under time critical conditions i e *running* the sequence at 25Hz the process proved to be too slow The reason for this centred around a combination of the JAVA bytecode, this machine independent code must go through the process of conversion to machine code in order to run, and the JAVA pointer system, pointer manipulation in JAVA is very restrictive in order to ensure network security

The JAVA language provides for this restrictive nature by supplying both just-in-time compilers and native methods Native methods allow JAVA to call methods in other languages This allowed the implementation of a C^{++} DLL, which had a quicker implementation of the conversion In order to speed up the process even further the conversion process was enveloped into an independent thread running as a unique process

4 8 3 Integration of VOP class in MPEG-4 Verification Model

A YUV implementation of the MPEG4 Image class was developed This contained an implementation of a bounding box on the segmented image The bounding box information was used to allow user interaction Based on mouse clicks and returned pixel values in relation to bounding box values it was established whether a VOP had been selected If so the user was allowed to change the composition of the scene by transforming the VOP to the position of the mouse release

4 9 2D & 3D Scene Description and Composition in the Verification Model

At this stage in MPEG-4's evolution it was clear from other MPEG-4 groups that both 2D & 3D composition would be required in an MPEG-4 terminal There was, at this point in the development, no specification proposed for a complete 3D scene description, and the abilities of JAVA to provide a complete overall 2D scene description were being questioned

There were two proposals for a 2D and 3D solution for scene description

- Use a fixed composition This means transmitting composition parameters with a fixed syntax These parameters include positioning, reference to the object, time stamp, and order of composition

The first scenario is already specified in [12] AVOs are decoded as specified in the object stream, for example the video object stream Sending parameters in the composition stream specifies the scene description information However, this implies a lot of restrictions The scene graph is flat, there is no way to describe complex trajectories without transmitting the positions at each frame, and there are no dependencies between objects This kind of composition is suited for broadcast applications with a low level of interactivity

A better approach would be to have a hierarchical type scene structure The reason for the hierarchical scene structure is to efficiently allow property nodes to affect geometry nodes that are after them in the graph So the scene graph minimises the storage requirements by having nodes share these state variables and as the application renders the scene it sets the current state and then draws all affected geometry This gives scene graph creators the ability to create very efficient scenes by organising similar geometry nodes in the graph to minimise state changes If there is no hierarchical structure then

all the information must be encapsulated within the node in order for it to be rendered properly, this would be rather inefficient

- Use a scene described with JAVA classes, as was currently done in the VM

The second scenario, the 'flexible' scene description based on JAVA classes, has been described in previous sections, a full description is available in [12] However there are also some limitations because of the structure of the scene graph that is hard coded in the AVOs, it is impossible to have this scene graph evolving in time It is also impossible for two AVOs to communicate and exchange information

4 10 Limitations imposed by the initial verification model

As described in the previous section the initial software implementation of the Systems VM [12], developed co-operatively by several institutions active in MPEG-4, was based on a mixed approach using JAVA code for the high level part of the system (user interface, allocation of the components of the system, management of thread(s) associated to the application) and using C code for the low level and computation intensive parts of the system, namely the elementary decoders for video, composition of elementary video objects through geometric transformations and alpha blending, and presentation on the machine specific windowing system

However after a couple of months of development several issues arose about the development environment

- the effectiveness of the threads scheduling provided by the JAVA platform for an application where real-time performance of audio/video decoders is critical,

- the coherency of the overall architecture to accommodate the fixed (parametric) and the flexible (bytecode description) of the Scene Graph,

- the efficiency of the implementation, related to the speed of the JAVA code execution, and the communication between the JAVA code and the C code

The real problem was that 'JAVA', itself, was made up of several different components

- a programming language JAVA as a language is a "simplified" version of C^{++}, with most of the features of the Object Oriented Paradigm, but with some restrictions to avoid typical sources of trouble within C^{++} programming

- a bytecode, i e an intermediate version of the code, produced by a compiler and used by an interpreter to run the application (the pro of this approach is platform independence, the con is lower efficiency than native executable code for a specific machine)

- a run-time system, i e a porting on a specific architecture (hardware plus operating system) of the execution environment

The programming language forces the use of strict programming rules The net result of the compiled code being less efficient than compiled C^{++} code, since some of the features provided by C^{++} for hand-made code optimisation (e g playing with pointer arithmetic) are forbidden by the JAVA compilers The trade-off here is less efficiency for more reliability

Bytecode (or intermediate object code) provides a level of abstraction from the specific native executable code of a machine Introducing this level of abstraction results in lower performance in executing an application (even when the intermediate code is re-compiled to native code, e g by means of a Just-In-Time (JIT) compiler translating to native code just before execution) Better performance can always be achieved mixing JAVA code and C code Using JIT compilers still preserves code portability (no need to change any part of the source code), while mixing C code requires some extra work

when porting to different platforms The trade-off is less efficiency for more portability

The run-time system provides the environment required to run a JAVA application on a specific hardware/OS platform This environment requires not only an interpreter for the bytecode but it must also support the Standard Class Library of the JAVA platform The standard library provides an abstraction for many OS services support for multithreading (a platform independent implementation of Threads, Monitors, Scheduler), support for automatic garbage collection, support for a platform independent windowing system, for a platform independent networking system (TCP and UDP) Most of these abstractions simply imply a trade-off of lower performance for higher portability (seamless execution of the same application on different platforms) But the standardised support for multithreading and garbage collection cause significant troubles to developers working on applications that require deterministic control on execution of the individual threads, especially when meeting processing dead-lines is essential The main reason for this is that even for a single threaded application, the run-time system is running its own "system threads" (e g a thread for garbage collection, a thread for updating the screen) on which the application developer has no control

More details on the problems to be solved to use the JAVA platform for real-time applications can be found at [13] So it was decided that while JAVA was a promising development environment, it was still a very immature one Development of the MPEG-4 verification model split into two parts, one continuing along the flexible scene description path using JAVA and another which focused on the fixed scene description and which will be described in the following sections

4 11 VRML and Scene Description in the Verification Model

The Virtual Reality Modelling Language (VRML) allows the description of 2D & 3D objects and to combine them into scenes and worlds, a detailed description can be

found in [14] It's a modelling language, which means it is used to describe 2D & 3D scenes It's more complex than HTML, but less complex than a programming language

The scope of the standard incorporates the following

- a mechanism for storing and transporting two-dimensional and three-dimensional data
- elements for representing two-dimensional and three-dimensional primitive information
- elements for defining characteristics of such primitives
- elements for viewing and modelling two-dimensional and three-dimensional information
- a container mechanism for incorporating data from other metafile formats
- mechanisms for defining new elements which extend the capabilities of the metafile to support additional types and forms of information

VRML gives a hierarchical description of a 3D scene as a tree of "nodes" Nodes can represent geometrical objects, light or sound sources, objects appearance properties and so on Moreover VRML allows the programmer to put code in a scene description and to extend the standard set of nodes by means of the "PROTO" nodes

4 12 Analysis of an MPEG-4 & VRML Combined Browser

It was decided to develop an application that could test the validity of an MPEG-4 browser, which would use VRML as a scene description language The following sections develop the proposals of [15] and [16] on how 2D and 3D scenes could be described and composited within MPEG-4

4 12 1 Proposed Architecture

The approach to integrating 3D and 2D rendered scenes was to develop an MPEG-4 Browser based on existing VRML browsers Figure 19 shows the proposed system

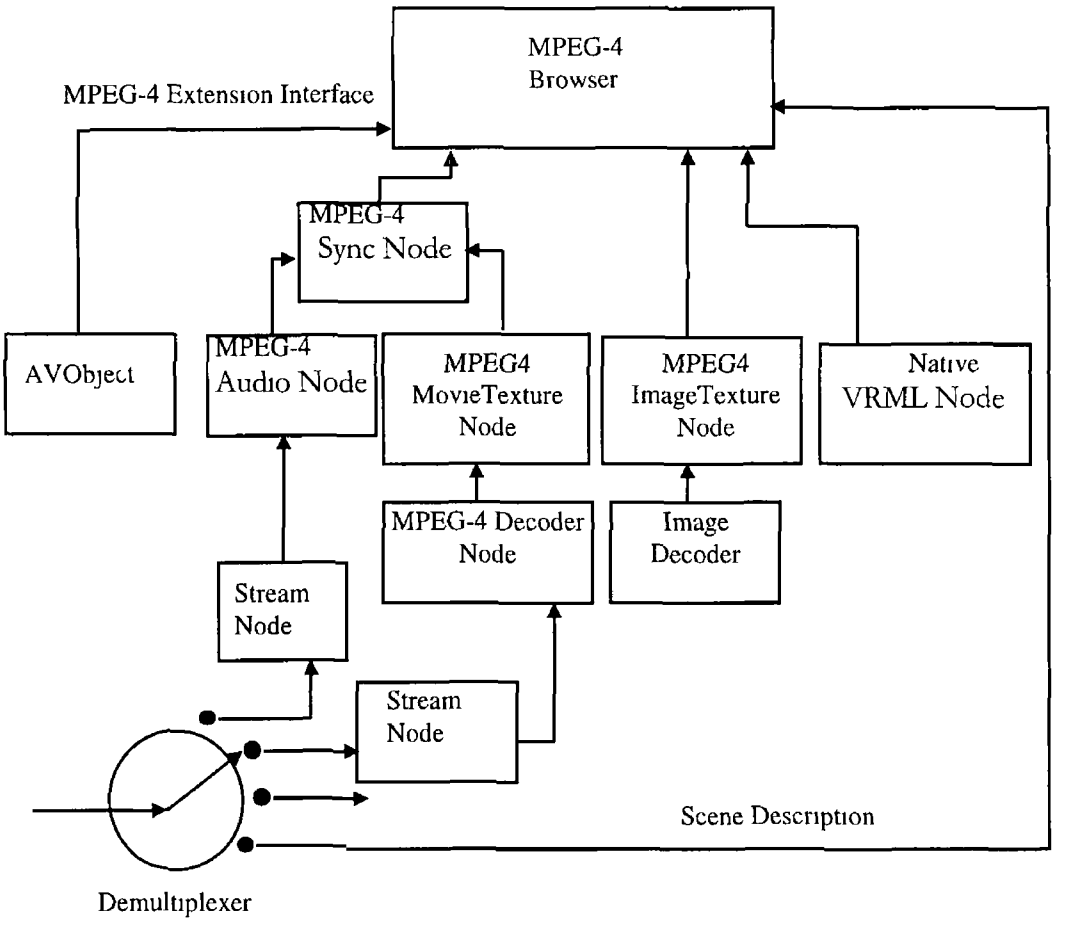


Figure 19 Proposed 3D Architecture

The VRML 2 0 support can be provided using a VRML Browser API, that is the API on which the browser code is built (Open Inventor, Cosmo3D,) This was deemed necessary for speedy development These APIs offer a very flexible environment for interactive 3D graphics New MPEG-4 nodes can be added and the browser designed with timing and synchronisation mechanisms at the core of it

An MPEG-4 Extension Interface was to be specified to allow the AVOs from the existing 2D VM to be supported

4 12 2 Scene Composition with 2D and 3D Objects

Figure 20 shows a typical scene composition The MPEG-4 browser is displaying a 2D VOP on a 3D Billboard node, taken directly from VRML The latter is partially hidden by a 3D sphere and is hiding some of a 3D cone As one moves through the scene going towards the VOP we pass by the sphere and the whole VOP will appear and as we continue on we will walk through the VOP to see the whole cone If we change our viewpoint we can use the Billboard node to define what exactly shall happen If we want we can ensure the view is consistent for all viewpoints or let the VOP be warped by our position It is up to us to define the behaviour

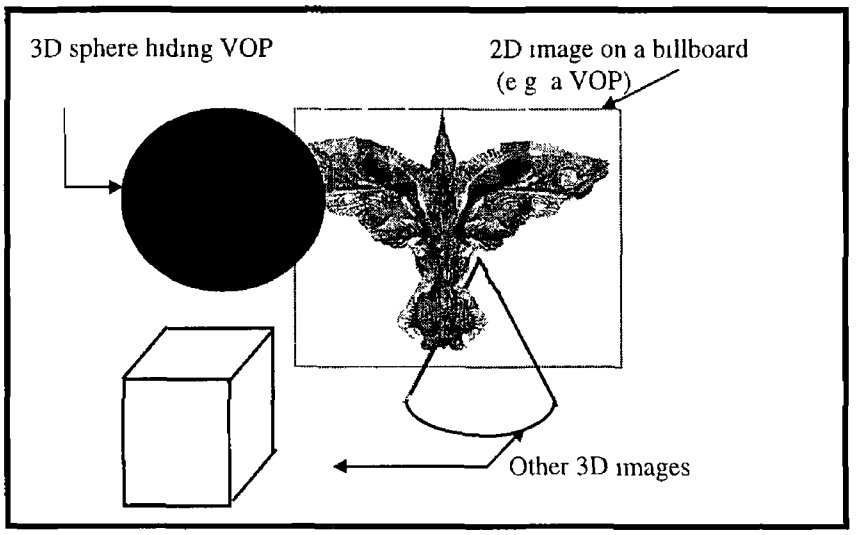


Figure 20 2D & 3D Composited Scene

An MPEG-4 Browser which combines the cornerstones of MPEG (decoding, synchronisation, demultiplexing, streaming data,) with direct support for VRML 2 0 can be used to implement a full 2D and 3D (fixed and flexible) compositor

4 13 Implementation of a 3D Verification Model

It was generally accepted that the VRML 3D modelling language had a lot to offer MPEG-4 VRML offered ways of allowing AVOs to communicate via routes, to be extended via scripting, and to dynamically modify the scene graph While we had developed a verification model based on JAVA to handle 2D AVOs we had no concrete plans on how to integrate 3D AVOs into our verification model Moreover, a lot of 3D complete API's existed, so the need to create a completely new 3D API for our verification model was questionable It seemed the best scenario would be to use VRML as our 3D modelling language and encapsulate its nodes into the standard Where appropriate, they would be modified to better meet MPEG-4's requirements Other nodes and concepts would be introduced to meet any remaining requirements that cannot be met by simple modifications to VRML 2 0

The implementation commenced using one of the existing 3D packages, Liquid Reality from DimensionX [17] The idea was to implement a 3D viewer which could also composite our existing 2D AVOs, a full explanation of the work carried out is described in [18], the following sections give an overview of the implementation

4 13 1 Analysis of a 3D verification model

A 3D MPEG-4 Viewer/Browser will be required to support 2D AV nodes in addition to the 3D nodes It is highly desirable that 2D nodes, AVOs, implemented in the 2D VM can be easily integrated into the 3D VM In the short term, during the development of the 3D VM, this is useful in order to re-use code from the 2D VM In the longer term, this will be a valuable feature to users who want to create a new 2D node They will be able to "plug-and-play" their 2D node into a 3D MPEG-4 Viewer/Browser

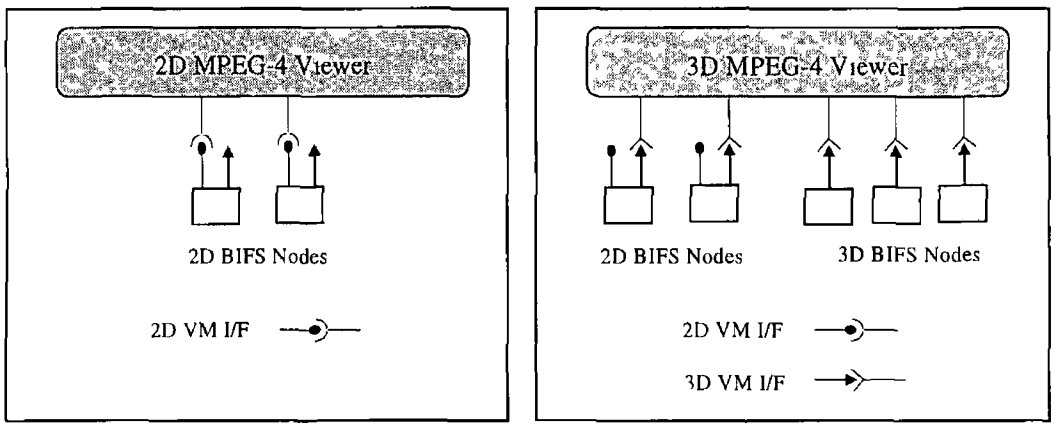


Figure 21 2D and 3D interfaces for AV nodes

Figure 21 shows a symbolic representation of a solution for the problem Each 3D node conforms to the 3D interface (currently the interface to the Liquid Reality browser) A 2D node conforms to the 2D interface (currently that of the 2D VM implementation) To allow itself to be used in 3D scenes, a 2D node also implements an output that conforms to the 3D interface Internally, the viewer/browser indicates to the node implementation which interface it should use This may require a small change in the 2D VM nodes

4 13 2 Implementing a Liquid Reality Extension Node from a 2D AVO

Liquid Reality (LR) implements the VRML 2 0 specification using a set of JAVA classes VRML 2 can be extended by writing LR extension nodes, these are sub-classes of LR's node classes This allows us to incorporate MPEG-4 specific nodes into VRML 2 worlds The class's location is described by an EXTERNPROTO description in the VRML world file and will be automatically loaded by the JAVA interpreter when needed by LR

LR documentation is available at [19], while a detailed example of how to create an extension node is given at [20]

4 1 3 3 Implementing a GifJpegDecoder Extension Node

A node that was implemented for the 2D VM, GifJpegDecoder, was taken as an example of a node that we would like to include in the 3D VM This node can take an image file, decide whether it is GIF or JPEG and decode it

A VRML world containing a GifJpegDecoder was implemented This required taking the current scene class GifJpegSequence and turning it into an extension node In this class we override the createNodeDefinition and initFields methods to define the GifJpegSequence node Here the GifJpegDecoder is instantiated once a TimeSensor eventIn is received from the VRML world The dnx lr Node method *handleEvent* is used to define what should happen on the eventIn In this case after the TimeSensor has started it sends an eventIn to the GifJpegSequence Node and this causes the GifJpegDecoder to decode a GIF file and display it on a cube To add the node definition to the VRML world file in a manner that will make it understandable to the Liquid Reality browser we use the EXTERNPROTO declaration This indicates to the browser where to go to find the created JAVA class and instantiate it

4 1 3 4 Implementing a Plug-and-Play Interface

A plug-and-play interface implies that a scene class can be seamlessly rendered on either a 2D VM or 3D VM This is now possible by implementing two render methods, one based on the 2D VM compositor and one on the 3D one Hence our GifJpegSequence can be rendered on either the 2D VM, Figure 22, or the implementation of a 3D VM, Figure 23

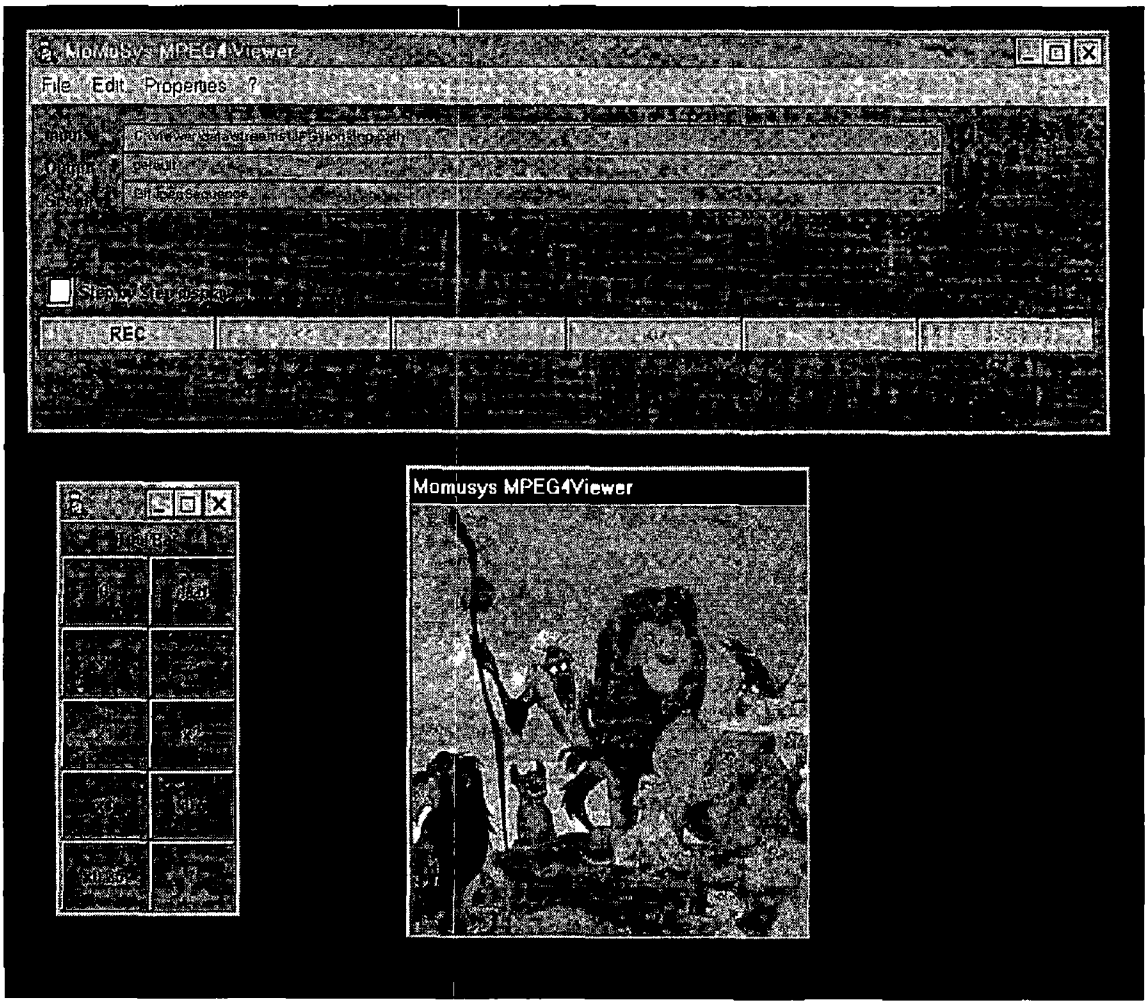


Figure 22 GifJpegSequence rendered on 2D VM

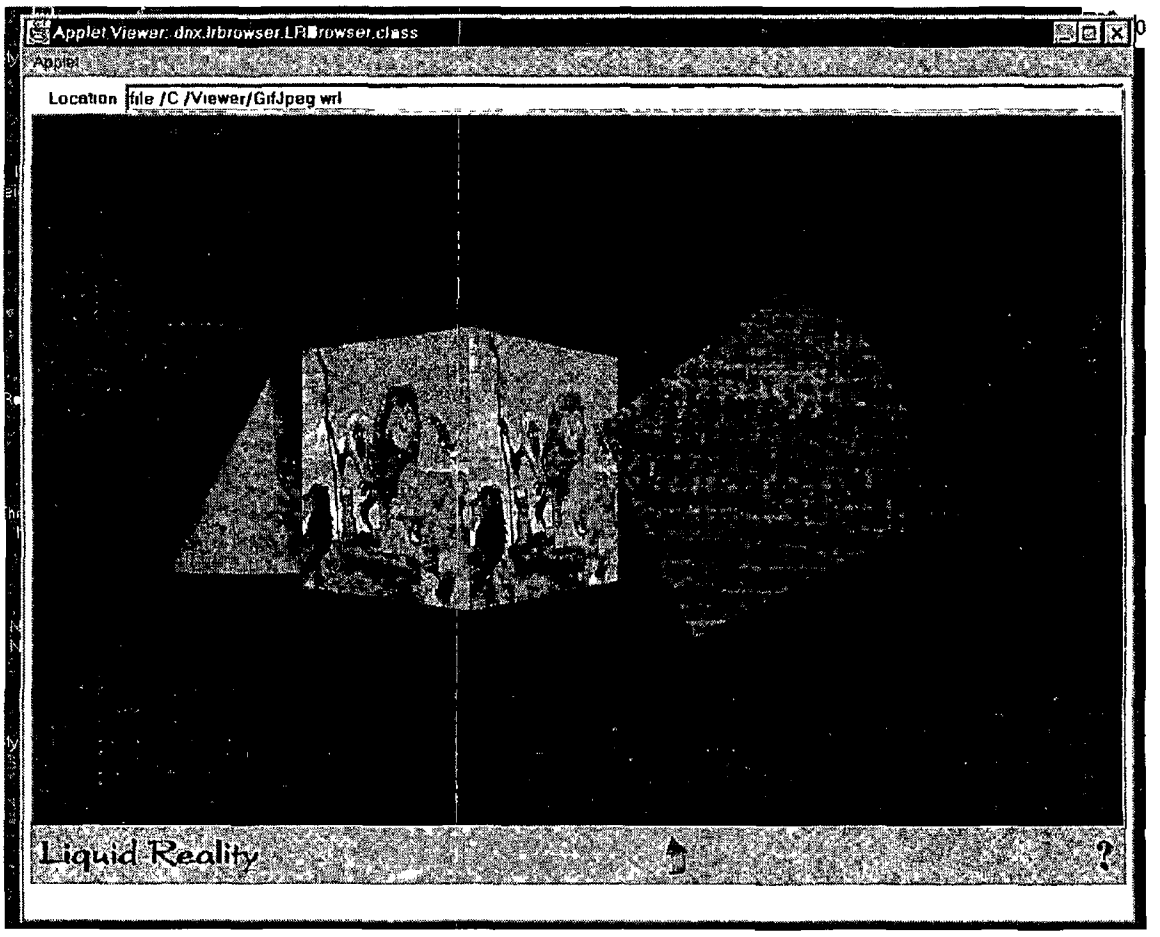


Figure 23 GifJpegSequence rendered on 3D VM

Chapter

5 BIFS AND BI & TRI DIMENSIONAL COMPOSITION

5 1 Introduction

As described in the previous chapter, VRML seemed suitable for MPEG4 purposes, in fact, as was shown an MPEG4 scene can be well described with the VRML mechanism But, the use of a commercial VRML browser enhanced for managing streaming data is not very promising, in fact the mechanism for writing native PROTO nodes (i e, developed in languages such as JAVA, C, and C^{++}) as needed for MPEG4 real-time constraints is not yet standardised and therefore these mechanisms are not yet (or only partially) supported by the available VRML browsers Additionally there is no chance to directly modify the embedded timing system and those available are very loose and not suited for the MPEG4 requirements

This highlighted the need for a "clean room" implementation of a VRML based MPEG-4 player which could be based on the basic, already standardised, VRML nodes and enhanced with the MPEG-4 peculiarities such as audio, video, and graphics synchronisation, links with streaming data, a binary scene description format, and enhanced interactivity

It was decided that, while VRML was a useful scene description language, it was lacking qualities that were vital for use in MPEG-4 MPEG-4 would develop its own scene description language called BIFS

5 2 Binary Format for Scene Description (BIFS)

BIFS scene description is the tool in MPEG-4 that enables us to describe interactive 2D and 3D scenes made up of several MPEG-4 so called Media Objects MPEG-4 has used as a basis for its scene description tool VRML 2 0, [14]

5 3 VRML/BIFS relationships

5 3 1 What VRML offers?

The Virtual Reality Modeling Language is basically a 3D interchange format aimed at including 3D objects and worlds in the World Wide Web VRML defines a set of nodes that describe the following elements

- The structure of the scene, so called Scene Graph The scene graph defines the spatial hierarchical relationships between VRML geometric and media elements in the 3D space

- 3D geometric components, such as geometric primitives, material and texture bindings, lighting effects, etc,

- VRML enables the use of media streams through the URL mechanism In particular, videos and audio streams can be pointed at by VRML descriptions However, VRML does not define any transport or global synchronization mechanism

- VRML defines behaviors of objets using routes and interpolators

- VRML defines the user interaction with the content using sensors and routes

Additionally to these elements, VRML further defines

- Scripts, which define an API which allows simple executable code to be inserted inside VRML scene descriptions

- A way to create reusable components made of several existing components, known as PROTOs and EXTERNPROTOs

- An API (not yet included in [14]) known as the External Authoring Interface (EAI), which enables interaction with the VRML scene from outside the world

- A binary format (not yet included in [14]), essentially based on an IBM proposal on mesh coding with topological surgery

A VRML file is an ASCII file instantiating several of the above described nodes The typical model for using a VRML file is first to load the entire "world" and let the user interact with it The EAI enables interaction between a HTML file and or a JAVA Applet and a VRML browser

5 3 2 What is BIFS?

The Binary Format for Scenes tool is essentially a binary format for representing 2D and 3D scenes made up of several streaming objects defined by the various MPEG-4 sub groups As with other MPEG-4 tools, BIFS can be used both in a pull and push scenario From the beginning, BIFS has adopted VRML as a basis and extended it in various ways

- BIFS has defined a set of new nodes to accommodate MPEG-4 specific needs

    - Definition of 2D nodes for representing 2D scenes, including images and videos as well as graphic and text primitives, and specific behaviors and interaction primitives

    - Definition of nodes to interface with Face and Body animation tools

    - Definition of nodes to interface with new synthetic and natural sound mixing capabilities

- Definition of nodes to mix 2D and 3D content in the same MPEG-4 presentation

- BIFS has defined a complete compression scheme for all these nodes When using only VRML nodes without meshes, the first tests show 3 to 10 times better compression results When using 3D meshes, the result are roughly 10% better in BIFS encoding, using the current status of the 3D mesh encoding SNHC tool This tool will be released in MPEG-4 version 2 (see chapter 6)

- BIFS has defined the BIFS Update protocol, which enables a command stream to continuously modify BIFS scenes Commands include the capabilities to add and remove objects, to modify scene properties, or to replace the whole scene

- BIFS has defined the BIFS-Anim protocol, which enables us to continuously animate some properties of the scene, such as faces, meshes, object positions or colors

There are still a few nodes and VRML concepts that have not been adopted in MPEG-4 In particular, the extension capabilities provided by Script, PROTOs and EXTERNPROTOs are not yet considered in the current BIFS specification, although many MPEG experts have recognized their usefulness

A very important point is that the BIFS tool is designed so that it works well with the rest of the MPEG-4 tools, the Multiplex, the System Decoder Model, the Object Descriptors, and all the streams defined in the Video, SNHC and Audio groups, as well as the control by DMIF of the Session

5 3 3 Using VRML content in the MPEG-4 context

With the current BIFS specification it is possible to use VRML content, compress it, and carry it over MPEG-4 streams If we look at a complete MPEG-4 scenario with a

scene containing a 3D scene, 2D graphic components, an MPEG-4 video and audio, as well as update and animation streams, the following components would typically be used

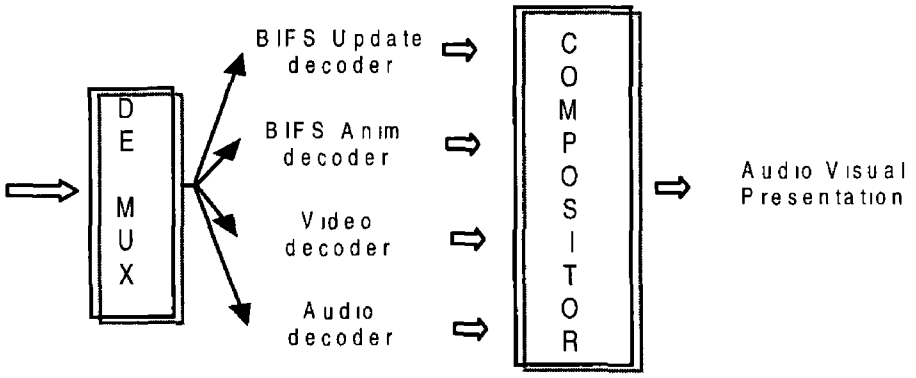


Figure 24 A typical MPEG-4 terminal architecture

In this scenario, the output of the demultiplexer is 4 distinct elementary streams

- The BIFS Update stream, that carries a command and a set of nodes, including VRML and new MPEG-4 nodes compressed with the BIFS algorithm
- The BIFS Anim Stream, that carries continuous changes of a set of properties of the scene
- The Video Stream
- The Audio Stream

One of the key elements in the MPEG-4 Systems architecture is the respect of time events and the System Decoder Model, which ensures the synchronization of media streams In that case, changes represented in the BIFS-Update and BIFS-Anim streams must be synchronized with the audio and video streams

5 3 4 Using BIFS content in the VRML context

Some of the functionalities supported in BIFS can be represented in a standard VRML environment

- The BIFS-Udpate and BIFS-Anim decoders can use the EAI to modify the scene
- Script nodes can be used to instantiate Video and Audio decoders
- For non standard VRML nodes, a library of PROTOs and EXTERNPROTOs can be used

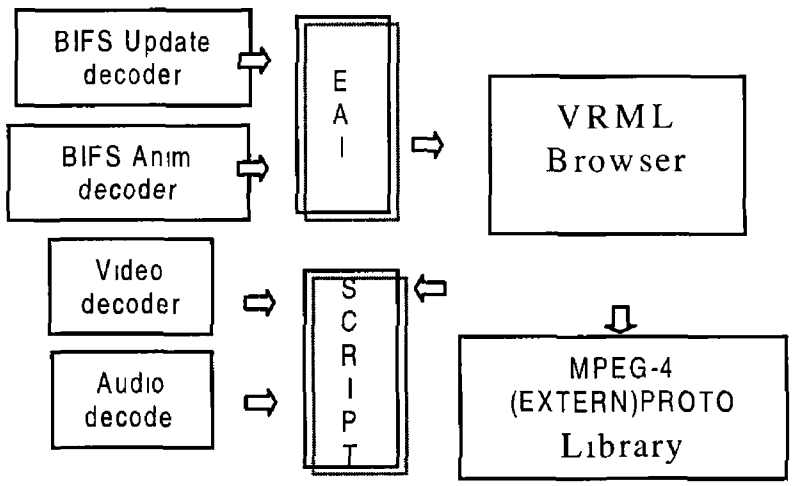


Figure 25 BIFS capabilities in a standard VRML environment

However, there are several limitations and constraints imposed by this architecture

- Some of the nodes cannot be represented in standard VRML In particular, the MPEG-4 specific audio capabilities (mixing of natural and synthetic sources), and the mixing of 2D and 3D scenes is not achievable in a standard VRML browser

- The fact that 2D scenes would have to be implemented as a special case of a 3D implementation imposes some constraints that are not acceptable to MPEG-4 profiles that only need 2D primitives, and may not provide an optimal implementation for 2D In any case, in the MPEG-4 context, 2D interfaces should be offered to allow implementers to develop their terminal using specific 2D or 3D based implementations of 2D primitives

- This architecture does not provide a precise enough control of time, which will lead to non synchronized media streams

- Since Scripts and PROTOs need to be used to represent additional MPEG-4 functionalities, the content would be less compact than in the case of the MPEG-4 terminal that considers these extensions as native extensions

- Scripts and the EAI impose more components to be included in any MPEG-4 terminal than in the case of the MPEG-4 terminal of Figure 24

5 4 Implementation of BIFS and 2D & 3D Composition

Currently BIFS and 2 and 3 dimensional composition are being developed and implemented in a real-time MPEG-4 player through work with the adhoc group on Systems Software Implementation in MPEG-4 and work within the ACTS project MoMuSys The software is being developed using the C^{++} language and the OpenGL API for composition It is freeware and can be downloaded from

http //televr fou telenor no/~karlo/compositor/

The idea is to produce a verification model for the functionalities being developed in MPEG-4 The following section is an overview of how 2 and 3 dimensional scenes are described using BIFS, it is not intended as a detailed explanation of how the player

functions, such information can be found in the documentation repository at the software site

5 4 1 The Components of the MPEG-4 Player

Figure 26 associates each component with the class objects it consists of

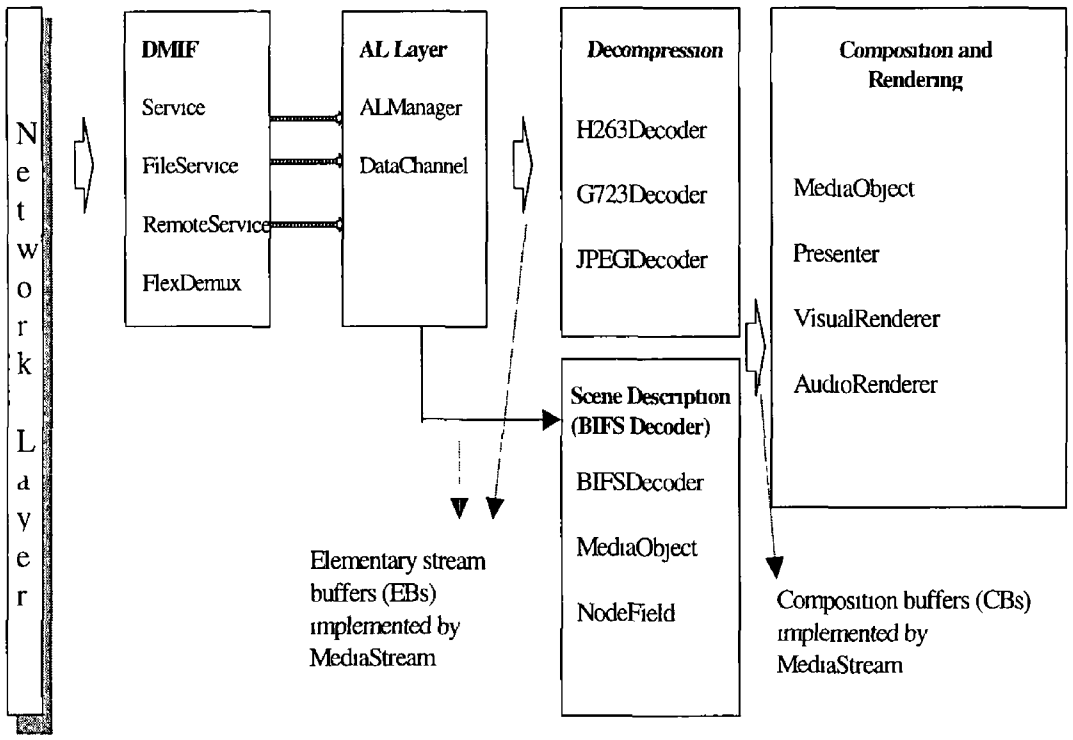


Figure 26 Implementation of major components of MPEG-4 Player

5 4 2 MediaObjects

Within the organisation of the classes that were created to implement an MPEG-4 scene the MediaObject class is the most fundamental It is the base class for all nodes defined by BIFS

A media object is an object that exists in the 3D space defined by the compositor Media objects are arranged hierarchically in the scene graph, which is basically a media object tree The root object or node identifies the scene The nodes which media

objects define are varied, some nodes are objects like Box, Spotlight etc , other nodes are used as containers to hold related nodes A shape node, for example, contains a geometry node and an appearance node A full list of the available BIFS nodes can be found in [5] These nodes can, in turn, contain other nodes In addition media objects that consume streams, like video and audio clips have been defined These are associated with media streams that are used to fetch stream units

MediaObjects have the following properties

- A MediaObject has zero or more "fields", each defined as either an object of a class derived from NodeField, or in the case of *eventIn*, as an event-handling member function

- A MediaObject can be a parent to zero or more other media objects All the child objects share the attributes of the parent object A position of a child object is relative to its parent object

5 4 3 MediaStreams

This is the object that handles the buffering and the transfer of data streams It consists of a memory buffer, and a FIFO mechanism to store/fetch access units into/out of the buffer The object also incorporates timing control, i e , stored access units may have a time stamp attached to them, and the fetch procedure will fetch only matured units

Delivering data over a MediaStream is performed as following·

- Before the originating object produces an access unit, it allocates space on the stream's buffer It asks for the amount of space it needs or, in cases when this size is not known before the data is actually produced, for the space it thinks would be usually sufficient Then it uses the allocated space to store the data it produces

- In case it turns out that the allocated space was not enough the object expands the allocated block

- When done, the actual size of the unit, as well as its presentation time is stored

- The receiving object then collects the unit at the correct time from the buffer

5 4 4 Decoding

Each decoder runs in its own thread and is inherited from the base Decoder class A decoder is bound to two MediaStreams, the input stream and the output stream, see Figure 27 The task of fetching coded units from the input streams (EBs) and storing presentation units into the output stream (PBs) is carried out by the base object This is done as follows

- The decoder gets an AU from the input stream If no data is available, the decoder's thread is suspended till data is available

- The decoder implements its specific decode functionality

- The output is stored in the output MediaStream This operation includes attaching a presentation time stamp to the unit

5 4 5 BIFS Decoder

The BIFS decoder reads in the encoded BIFS scene description file and performs the following

- Retrieves data from the input MediaStream

- Instantiates the root MediaObject, and calls it to parse itself and build the scene tree

- Whenever a node update is detected it calls the appropriate node to parse and update itself

- Whenever an ObjectDescriptor is detected it passes the information to the proper node so the node can create the necessary Decoder and MediaStreams

5 4 6 Flow of Information in the MPEG-4 Player

Figure 27 illustrates the flow of information in the whole application It shows how the BIFS scene is decoded and presented and how the processes described in the previous sections exist in relation to the overall MPEG-4 application

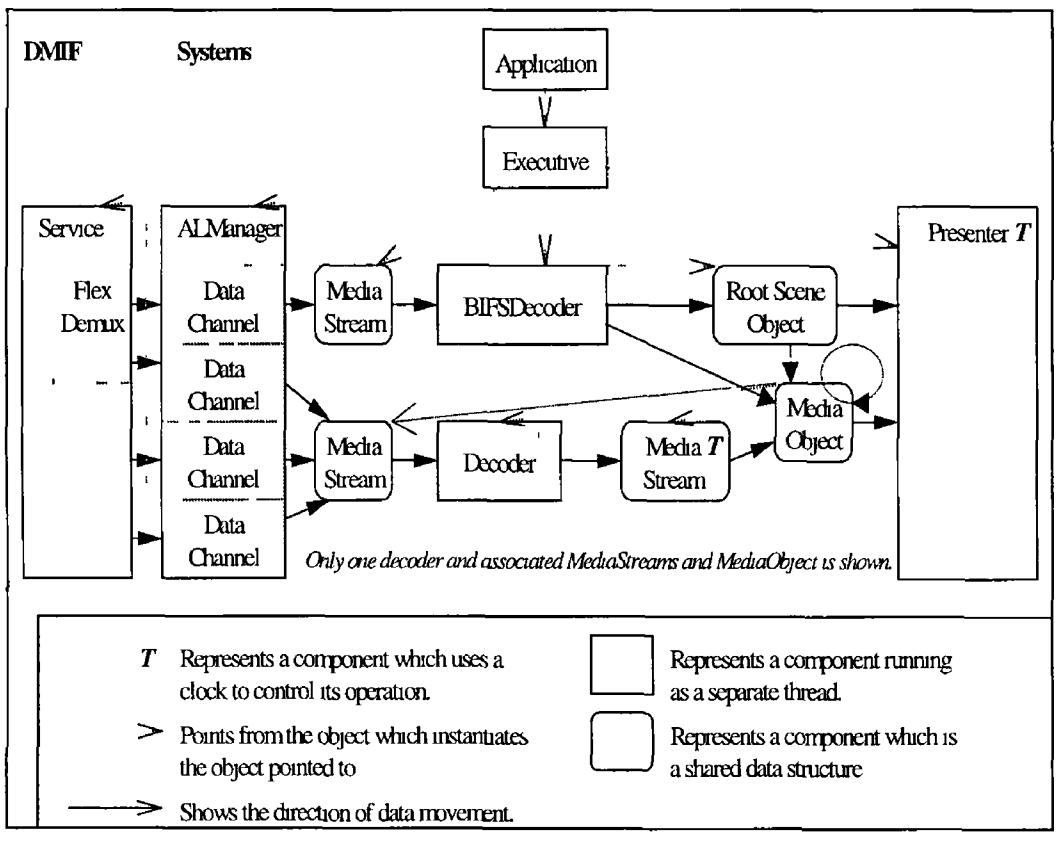


Figure 27 Flow of information in the MPEG-4 Player

5 4 7 2D & 3D Composition in the MPEG-4 Player

Composition is achieved via SGIs' OpenGL graphics library The OpenGL graphics system is a powerful software interface for graphics hardware that allows graphics programmers to produce high-quality colour images of 2D and 3D objects Silicon Graphics Inc developed the technology

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms As such it provides a layer of abstraction between graphics hardware and an application program It is visible to the programmer as a set of routines consisting of about 120 distinct commands Together these routines make up the OpenGL application programming interface (API) The routines allow graphics primitives (points, lines, polygons, bitmaps, and images) to be rendered as well as basic rendering operations such as affine and projective transformations and lighting calculations It also supports advanced rendering features such as texture mapping and antialiasing

No commands for performing windowing tasks or obtaining user input are included in OpenGL, instead, you must work through whatever windowing system controls the particular hardware you're using Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, aeroplanes, or molecules When you build a graphics program using OpenGL, you start with a few simple primitives The sophistication comes from combining the primitives and using them in various modes

5 4 7 1 OpenGL Rendering Pipeline

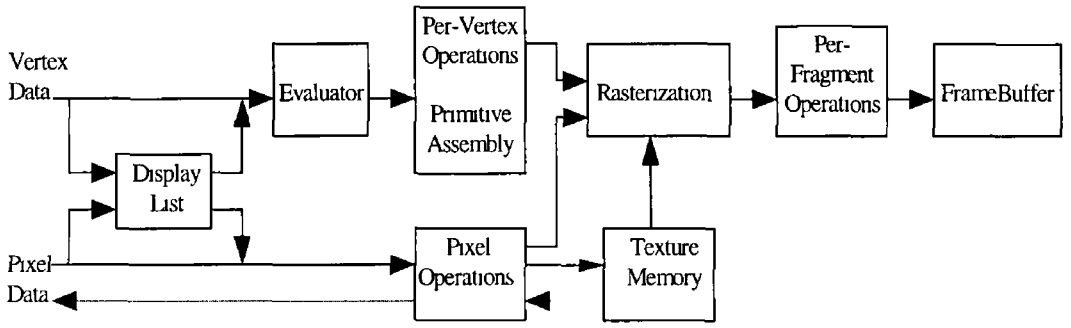


Figure 28 Schematic Diagram of the Order of Operations in OpenGL

Figure 28 shows a schematic diagram of OpenGL Commands enter OpenGL on the left Most commands may be accumulated in a display list for processing at a later time Otherwise, commands are effectively sent through a processing pipeline

The first stage provides an efficient means for approximating curve and surface geometry by evaluating polynomial functions of input values The next stage operates on geometric primitives described by vertices points, line segments, and polygons In this stage vertices are transformed and lit, and primitives are clipped to a viewing volume in preparation for the next stage, rasterization The rasterizer produces a series of framebuffer addresses and values using a two-dimensional description of a point, line segment, or polygon Each fragment so produced represents a portion of a primitive that corresponds to a pixel in the framebuffer Then each fragment may be modified by texture mapping, after which it is fed to the next stage that performs operations on individual fragments before they finally alter the framebuffer These operations include conditional updates into the framebuffer based on incoming and previously stored depth values (to effect depth buffering), blending of incoming fragment colours with stored colours, as well as masking and other logical operations on fragment values

Finally, pixel rectangles and bitmaps (2D images) bypass the vertex processing portion of the pipeline to send a block of fragments directly through rasterization to the individual fragment operations, eventually causing a block of pixels to be written to the framebuffer A unique feature of OpenGL is that pixel rectangles and bitmaps (2D images) are also rasterized to produce fragments, fragments are treated the same no matter if they come from a geometric or image primitive Values may also be read back from the framebuffer or copied from one portion of the framebuffer to another These transfers may include some type of decoding or encoding

More generally, MPEG-4 uses the OpenGL API to compose a 2D and/or 3D scene, allowing for example to

- place AVOs anywhere in a given co-ordinate system,
- group primitive AVOs in order to form compound AVOs,
- modify AVOs attributes using streaming data (e g moving texture belonging to an object, animating a moving head by sending animation parameters),
- update the user's viewing point to enable interactivity anywhere in the scene

5 4 8 An Example MPEG-4 Scene

The following BIFS file is an example of how to describe a scene that contains a number of decoders and BIFS nodes The composited scene is displayed in Figure 29

```
Group {
      children [
            Fog {
                  color 0 0 0 0 0 0
                  visibilityRange 30 0
                  fogType "LINEAR"
            }
  DirectionalLight {
   color 1 1 1
      }
            Viewpoint {
                  fieldOfView 0 785398
            }
            Transform {
                  translation -2 0 0
                  rotation 1 1 0 45
                  children [
                        Shape {
                              appearance Appearance {
                                    texture ImageTexture {
                                          url 2
                                          repeatS FALSE
                                          repeatT FALSE
                                    }
                              }
                              geometry Box {
                                    size 2 2 2
                              }
                        }
                  ]
            }
            Transform {
                  translation 2 0 0
                  scale 0 07 0 07 0 07
                  children [
                        FBA {
                              face Face {
                                    fdp FDP {
                                          faceSceneGraph Group {
                                          }
                                    }
                                    fap FAP {
                                          url 3
                                    }
                              }
                        }
                  ]
            }
            Sound {
                  sound AudioSource {
                        url 4
```

```
                        startTime 0
                        stopTime -1
                  }
            }
      ]
}

SessionStreamAssociation {
      children [
            ObjectDescriptor {
                  objectDescriptorID      2
                  decTypeString           visual/H263
                  configParam       1
            }
            ObjectDescriptor {
                  objectDescriptorID      3
                  decTypeString           visual/FBA
                  configParam       1
            }
            ObjectDescriptor {
                  objectDescriptorID      4
                  decTypeString           audio/G723
                  configParam       2
            }
      ]
}
```

Figure 29 Composition of a BIFS scene in MPEG-4 Player

Chapter

6 CONCLUSIONS AND FUTURE DIRECTIONS

6 1 Introduction

MPEG-4 is the ISO/IEC standard being developed by MPEG (Moving Picture Experts Group), the committee that also developed the Emmy Award winning standards known as MPEG-1 and MPEG-2 The MPEG-4 standard will be the result of an international effort involving hundreds of researchers and engineers from all over the world MPEG-4, whose formal ISO/IEC designation will be ISO/IEC 14496, is to be released in November 1998 and will be an International Standard in January 1999 This release will be known as Version 1 [29]

Work on MPEG-4 will continue after that date, for a Version 2 Version 2, work on which has already started, will add tools to the MPEG-4 Standard Existing tools and profiles from Version 1 will not be replaced in Version 2, technology will be added to MPEG-4 in the form of new profiles

In the previous chapters there has been a description of how the need for the development of an MPEG-4 standard came about Initially an overview of MPEG-4 was given in a layer by layer basis, and then focus was given to the process of developing an efficient method for 2D and 3D scene description and composition The mathematics of 2D and 3D composition and rendering was analysed and developed The evolution of the scene description in MPEG-4 was then analysed It was shown what knowledge and standards currently existed and how a new dynamic approach was developed from this knowledge

In this chapter the future development of the scene description language, the MPEG-4 systems layer, and an overview of the types of applications the final MPEG-4 standard will help develop is presented

6 2 Future Developments Planned in the Scene Description of MPEG-4

Scene description in MPEG-4 will continue with the further development of BIFS, introducing new media objects and MPEG-4 nodes as well as converging with the VRML 2 0 standard, and the introduction of a new adaptive audio visual scene description

6 2 1 The Future of BIFS

Based on the analysis of VRML and BIFS in the previous chapters it is clear that the entire BIFS tool cannot be properly represented in a strictly conformant VRML 2 0 architecture However the interchange and creation of content can be eased, and both the VRML and MPEG-4 community would benefit from, and facilitate the development of, the future potential technology and applications that emerge from the mixing of the computer graphics technology of the VRML consortium, and the compression and streaming expertise of the MPEG group To this purpose the MPEG-4 and VRML consortiums are working towards the following

- MPEG-4 should use all VRML nodes following strictly their semantics and design
- MPEG-4 and VRML should use the same binary encoding
- MPEG-4 shall design its nodes using the same design principles as VRML did In particular, the following rules must apply
    - MPEG-4 should not design nodes that can be efficiently represented by a small set of other existing nodes

- MPEG-4 should use names that are compatible with existing node names to facilitate mutual understanding and technical exchange

6 2 2 Adaptive Audio-Visual Session Format (AAVS)

One of the main disadvantages of BIFS is that the MPEG-4 receiving terminal must have all the media objects defined in the scene implemented in order for it to be rendered correctly This implies that if a scene is developed with a new improved video or audio decoder, or even a completely new decoder, and the receiving terminal doesn't have this implementation we cannot display the scene on this terminal

The Adaptive Audio-Visual Session (AAVS) format specifies interfaces for the interoperation of MPEG-4 media with JAVA code By combining MPEG-4 media and safe executable code, content creators may imbed complex control mechanisms with their media data to intelligently manage the operation of the audio-visual session

It is foreseen that AAVS will provide unique capabilities as a format for session representation

- AAVS will provide interfaces to MPEG-4 multimedia terminals, enabling advanced user interaction and device control

Interactive media applications require both interfaces to user I/O devices as well as media I/O devices The AAVS technology enables such a capability in MPEG-4 by having an adaptive session with downloadable applets An applet is a secure JAVA application that can run over the Internet

- AAVS will provide mechanisms for client-side programmatic control of the audio-visual session

For some types of content, a parametric scene description is sufficient, but for other types of content, a programmatic description may be most appropriate For example, a parametric scene description may require frequent updates across the network, increasing the bandwidth of control information with higher vulnerability to errors In this case, it may be more robust and efficient to generate the scene updates with executable code running on the client side In addition, it may be easier to create a programmatic scene description, such as when a position or graphical parameter changes with time according to a mathematical formula Furthermore, programmatic content may be extended beyond the syntax of a parametric scene description

- AAVS will provide mechanisms for programmatic adaptation of the session to client-side information, thus maximising media quality in the presence of static or dynamic terminal resources

MPEG-4 media is designed to be scalable so that, ideally, a content creator can reuse the same media on multiple MPEG-4 platforms, for example, in a set-top box, a web browser, and or a handheld device AAVS enables the content creator to specify client-side, programmatic control to tailor the media session to the static terminal resource constraints Furthermore, AAVS provides mechanisms for the content creator to specify adaptive session behaviour in the presence of dynamically changing resources

6 3 The Future development of the Systems Layer

As previously described, the systems layer of MPEG-4 helps develop standards for the coding of the combination of, individually coded audio, moving images and related information so that the combination can be used by any application One of its major inputs to the standard has been the development of the scene description format

Systems will provide the following functionalities for the MPEG-4 standard in Version 1

- Scene description for composition (spatio-temporal synchronisation with time response behaviour) of multiple AVOs The scene description provides a rich set of nodes for 2D and 3D composition operators and graphics primitives

- Text with international language support, font and font style selection, timing and synchronisation

- Interactivity, including client and server-based interaction, a general event model for triggering events or routing user actions, general event handling and routing between objects in the scene, upon user or scene triggered events

- The interleaving of multiple streams into a single stream, including timing information (multiplexing)

- Transport layer independence Through the separation of the multiplexing operation into FlexMux and TransMux, support for a large variety of transport facilities is achieved

- The initialisation and continuous management of the receiving terminal's buffers

- Timing identification, synchronisation and recovery mechanisms

- Datasets covering identification of Intellectual Property Rights relating to Audio-visual Objects

Most of these developments have been made and are functioning in the systems software implementation described in chapter 5 Version 2 of the MPEG-4 standard will support, in addition to the tools in Version 1

- Scene description for composition of multiple AVOs This includes 2D/3D objects grouping for ease in editing and composition, spatio-temporal 2D/3D AVO positioning and transformation, and 2D/3D AVO attribute value selection

- Specification of an API for description of AVOs behaviour,

- Specification of APIs for 2D composition,

- Specification of API for 2D/3D composition,

- Support of downloadable executable code,

- Server-side interaction via attribute value modification using standardised parametric description,

- AVOs with descriptors to carry MPEG-7 data (MPEG-7 will define a framework for identifying and describing what is 'inside' the content)

- A number of functionalities in the area of IPR identification and protection are under study for support, and may be provided in MPEG-4 version 2, either by providing hooks or by defining the algorithms within MPEG for automated monitoring and tracking of creations, prevention of unauthorised copying and manipulation, tracking object manipulation and modification history, and supporting transactions between Users, Media Distributors and Rights Holders

For more information on the planned future developments of the various MPEG-4 layers see [1] and [2]

6 4 Future MPEG-4 Applications

MPEG-4 has been developed in order to enable developers to create applications In this section a number of possible applications are listed which are enabled by the tools

and methods currently standardised within MPEG-4 The idea is to describe and highlight possible future usage of MPEG-4 technology Further documentation on possible MPEG-4 applications can be found in [3]

6 4 1 Real Time Communications

Real-time Communications systems are targeted toward applications which encompass two-way human interaction, or one-way applications that impose strict one-way delay constraints A videophone system is a prime example of a two-way real-time system An example of a one-way delay constrained system is a surveillance system

One key feature of real-time systems is that if there is both audio and video present, the audio and video are synchronised so that the viewer is given the impression of lip synchronisation Interaction between the users of two-way systems requires that the overall end-to-end delay will be relatively small and fairly constant

The underlying transport system for real-time communications application is likely to encompass a broad cross section of technologies A key attribute of the real-time communications systems application is the ability to successfully operate over a wide variety of media including low and high mobility wireless, LAN transmission channels, PSTN and ISDN transmission channels Interworking between various media channels should be supported

It is expected that real-time communications systems will operate in a variety of different system configurations including those where the complexity of the encoding/decoding process constitutes a major design constraint Audio and/or visual quality maybe traded off against delay and complexity such that a balance is found between the desire for high quality audio/video and the need to provide low delay operation at a reasonable complexity

6 4 2 Infotainment

As interaction with AVOs is considered as the most important aspect of MPEG-4, infotainment applications, containing a combination of entertainment and information are well within the scope Generally, the users of such systems have the means both to get information about specific subjects of interest and to configure and amuse themselves within a multimedia environment The interactivity aspect includes for example the requesting of additional objects and changing of the content of the existing scene nodes

A key feature of infotainment applications is the manifold of necessarily diversified features Typical infotainment applications will make heavy use of natural and synthetic audio and video in form of e g spoken text and music of all kinds with underlying visual animation For this kind of application it will be necessary to guarantee a high quality of presentation during the whole session if the user shall not become bored of his/her pastime The quality aspects address both high AV quality and time constraints to end-to end latency

MPEG-4 provides an ideal framework for infotainment applications

- It will feature the means to support the utmost multifaceted set of multimedia types to be combined within a presentation scenario in a standardised way

- The composition concepts, which will cover 2D as well as 3D, will be the base for mixing all kinds of data types within a consistent object handling and user interaction paradigm

- MPEG's tradition is to achieve the highest possible quality with existing techniques, which is only adequate for the demanding nature of infotainment applications

6 4 3 Collaborative Scene Visualisation

Collaborative Scene Visualisation supports a class of Computer Supported Co-operative Work (CSCW) applications where groups of people typically working simultaneously in distributed locations leverage visualisation tools to accomplish a task by sharing a common visual information space [31]

A trend of these kind of applications is that they will provide Augmented Reality (AR) A particular feature of these applications is that they not only use dedicated audio-visual streams as usual tele-conferencing applications for interpersonal communication, but also use an additional video streams to achieve AR effects The objective of AR is to create an environment in which a user perceives both real and virtual/synthetic (generated with a computer) objects in a seamless way

From the viewpoint of communication, multiple audio-visual streams of natural and synthetic origins are transferred an audio-visual stream for conferencing, a video stream containing a video shot of the empty office, and a 3D synthetic object stream for the furniture, etc

Like any distributed multimedia system where partly bulk data (video, audio, high resolution image, animation sequence, etc) is transferred, appropriate data coding methods are needed For this end, MPEG-4 is very useful, because of the following reasons

- It supports high performance data compression
- A trade-off between quality and performance can be made by scaling encode and decode complexity, spatial resolution, temporal resolution, and quality
- Content-based coding enables interactivity with objects Real objects can be conveniently manipulated in the same way as virtual objects

- The composition concept of MPEG-4 is very appropriate for organising a scene consisting of real and virtual objects to be transferred among dispersed participants

- Stereoscopic views help a user perceiving a scene

- Face Animation parameters can be used to replace the audio-visual streams used for interpersonal communication to achieve bandwidth reduction The saved bandwidth can be used to improve the quality of the video stream used for AR scenes

As can be deduced from the above examples the MPEG-4 standard will provide a means of creating new and exciting applications

References

[1] MPEG Requirements, Audio, DMIF, SNHC, Systems, Video, "MPEG-4 Overview", Document ISO/IEC JTC1/SC29/WG11 N1909, Fribourg MPEG meeting, October 1997

[2] MPEG Requirements, Audio, DMIF, SNHC, Systems, Video, "Overview of MPEG-4 functionalities supported in MPEG-4 Version 2", Document ISO/IEC JTC1/SC29/WG11 N1914, Fribourg MPEG meeting, October 1997

[3] MPEG Requirements, "MPEG-4 Applications V 2 0", Document ISO/IEC JTC1/SC29/WG11 N1907, Fribourg MPEG meeting, October 1997

[4] MPEG-4 Convenor, "MPEG-4 project description", Document ISO/IEC JTC1/SC29/WG11 N1177, Munich MPEG meeting, January 1996

[5] MPEG-4 Systems, "Text for CD 14496-1 Systems", Document ISO/IEC JTC1/SC29/WG11 N1901, Fribourg MPEG meeting, October 1997

[6] MPEG-4 Systems, "Proposed revision for the MPEG-4 Syntactic Description Language (Rev 2 1)", Document ISO/IEC JTC1/SC29/WG11 N2902, Fribourg MPEG meeting, October 1997

[7] MPEG-4 Audio, "Text for CD 14496-3 Audio", Document ISO/IEC JTC1/SC29/WG11 N1903, Fribourg MPEG meeting, October 1997

[8] MPEG-4 Video, "Text for CD 14496-2 Video", Document ISO/IEC JTC1/SC29/WG11 N1902, Fribourg MPEG meeting, October 1997

[9] Sun Microsystems, The Java Language A White Paper

[10] MPEG-4 Systems, E Cooke, S Lecercle, O Avaro, "MSDL VM A Primer", Document ISO/IEC JTC1/SC29/WG11 N1575, Maceio MPEG meeting, November 1996

[11] MPEG-4 Video Group, "Text of ISO/IEC 14496-2 video verification model V 8 0", Document ISO/IEC JTC1/SC29/WG11 N1796, Stockholm MPEG meeting, July 1997

[12] MPEG-4 Systems, "Systems Working Draft version 2 0", ISO/IEC JTC1/SC29/WG11 N1483, Maceio MPEG meeting, November 1996

[13] Embedded Real Time Development in the Java Language http //www newmonics com

[14] VRML community, "The Virtual Reality Modeling Language Specification ",Version 2 0,August 4, 1996

[15] MPEG-4 Systems, E Cooke, C Bouville, "Proposal for 2D & 3D Composition", ISO/IEC JTC1/SC29/WG11 N1776, Sevilla MPEG meeting, February 1997

[16] MPEG-4 Systems, E Cooke, L Ward, "Proposal for 3D Scene Description and Composition", ISO/IEC JTC1/SC29/WG11 N1777, Sevilla MPEG meeting, February 1997

[17] Cosmo Software, http //cosmo sgi com/, Cosmo Software Homepage

[18] MPEG-4 Systems, E Cooke, L Ward, "3D VM Plug-n-Play Interface for 2D AV objects ', ISO/IEC JTC1/SC29/WG11 N1999, Bristol MPEG meeting, April 1997

[19] MicroSoft,http //www dimensionx com/products/lr/docs/index html, LR Documentation

[20] MicroSoft,http //www dimensionx com/products/lr/docs/tutorial html, LR Tutorial

[21] MPEG-4 Context and Objectives, R Koenen, F Pereira, L Chiariglione, Special Issue of Image Communication on MPEG-4, Volume 9, Issue 4, May 1997

[22] Introduction to MPEG-4, C Reader, Journal of Video Coding, February, 1997

[23] MPEG-4 Audio/Video & Synthetic Graphics/Audio for Mixed Media, P K Doenges, T K Capin, F Lavagetto, J Ostermann, I S Pandzic, Image Communication Journal, Volume 5, No 4, May 1997

[24] MPEG-4 Editorial, Y Q Zhang, F Periera, T Sikora, C Reader, CirSysVideo Journal, No 1, February 1997

[25] The MPEG-4 Video Standard Verification Model, Y Q Zhang, F Periera, T Sikora, C Reader, CirSysVideo Journal, No 1, February 1997

[26] The MPEG-4 Systems and Description Languages A Way Ahead in Audio Visual Information Representation, O Avaro, P Chou, A Eleftheriadis, C Herpel, C Reader, J Signes, Special Issue of Image Communication on MPEG-4, Volume 9, Issue 4, May 1997

[27] Dead-Reckoning Algorithms for Synthetic Objects in MPEG-4 SNHC, T K Capin, I S Pandzic, N Magnenat Thalmann, D Thalmann, Workshop on Synthetic - Natural Hybrid Coding and Three Dimensional Imaging, Rhodes, 1997

[28] MPEG-4 Video Verification Model A Video Encoding/Decoding Algorithm Based on Content Representation, T Ebrahimi, Image Communication Journal, Volume 5, No 4, May 1997

[29] ISO MPEG-4 - An Emerging Standard for Mobile Multimedia Communications, A Puri, A Eleftheriadis, Special Issue of Mobile Networking and Applications Journal on Mobile Multimedia Communications, June 1997

[30] Tests on MPEG-4 Audio Codec Proposals, L Contin, B Edler, D Meares, P Schreiner, Special Issue of Image Communication on MPEG-4, Volume 9, Issue 4, May 1997

[31] MPEG-4 for Networked Collaborative Virtual Environments, T K Capin, I S Pandzic, N Magnenat Thalmann, D Thalmann, IEEE Computer Society Press, 1997

Annex A - Glossary and Acronyms

AAC	Advanced Audio Coding
AAL	ATM Adaptation Layer
AAVS	Adaptive Audio-Visual Session
AL	Adaptation Layer
Access Unit	A logical sub-structure of an Elementary Stream to facilitate random access or bitstream manipulation
ADSL	Asymmetrical Digital Subscriber Line
Alpha plane	Image component providing transparency information (Video)
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
AVO	Audiovisual Object
BAP	Body Animation Parameters
BDP	Body Definition Parameters
BIFS	Binary Format for Scene description
BSAC	Bit-Sliced Arithmetic Coding
CE	Core Experiment
CELP	Code Excited Linear Prediction
DAI	DMIF-Application Interface
DDI	DMIF-DMIF Interface
DMIF	Delivery Multimedia Integration Framework
DSM-CC	Digital Storage Media - Command and Control
DSM-CC U-U	DSM-CC User to User
DSM-CC U-N	DSM-CC User to Network
ES	Elementary Stream A sequence of data that originates from a single producer in the transmitting MPEG-4 Terminal and terminates at a single recipient, e g an AVObject or a Control Entity in the receiving MPEG-4 Terminal It flows through one FlexMux Channel
FAP	Facial Animation Parameters
FBA	Facial and Body Animation
FDP	Facial Definition Parameters
FlexMux layer	Flexible (Content) Multiplex A logical MPEG-4 Systems layer between the Elementary Stream Layer and the TransMux Layer used to interleave one or more Elementary Streams, packetized in Adaptation Layer Protocol Data Units (AL-PDU), into one FlexMux stream
FlexMux stream	A sequence of FlexMux protocol data units originating from one or more FlexMux Channels flowing through one TransMux Channel
FTTC	Fiber To The Curb
GSTN	General Switched Telephone Network
HFC	Hybrid Fiber Coax
HILN	Harmonic Individual Line and Noise
HTTP	HyperText Transfer Protocol

HVXC	Harmonic Vector Excitation Coding
IP	Internet Protocol
IPI	Intellectual Property Identification
IPR	Intellectual Property Rights
ISDN	Integrated Service Digital Network
LAR	Logarithmic Area Ratio
LC	Low Complexity
LPC	Linear Predictive Coding
LSP	Line Spectral Pairs
LTP	Long Term Prediction
mesh	A graphical construct consisting of connected surface elements to describe the geometry/shape of a visual object
MCU	Multipoint Control Unit
MIDI	Musical Instrument Digital Interface
MPEG	Moving Pictures Experts Group
PSNR	Peak Signal to Noise Ratio
QoS	Quality of Service
RTP	Real Time Protocol
RTSP	Real Time Streaming Protocol
Rendering	The process of generating pixels for display
Sprite	A static sprite is a - possibly large - still image, describing panoramic background
SRM	Session and Resource Managers
TCP	Transmission Control Protocol
T/F coder	Time/Frequency Coder
TransMux	Transport Multiplex
TTS	Text-to-speech
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunication System
Viseme	Facial expression associated to a specific phoneme
VLBV	Very Low Bit-rate Video
VRML	Virtual Reality Modeling Language