INTERNATIONAL ORGANISATION FOR STANDARDISATION

ORGANISATION INTERNATIONALE DE NORMALISATION

ISO/IEC JTC 1/SC 29/WG 11

CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC 1/SC 29/WG 11 N 2502

Atlantic City, October 1998

Information technology - Generic coding of audio-visual objects - Part 2: Visual

ISO/IEC FDIS 14496-2

Final Draft International Standard

Modified by the SC 29 Secretariat

ISO/IEC Copyright Office * Case Postale 56 * CH1211 Genève 20 * Switzerland

Contents

7.6.4 Unrestricted motion compensation *

7.6.5 Vector decoding processing and motion-compensation in progressive P-VOP *

7.6.6 Overlapped motion compensation *

7.6.7 Temporal prediction structure *

7.6.8 Vector decoding process of non-scalable progressive B-VOPs *

7.6.9 Motion compensation in non-scalable progressive B-VOPs *

7.7 Interlaced video decoding *

7.7.1 Field DCT and DC and AC Prediction *

7.7.2 Motion compensation *

7.8 Sprite decoding *

7.8.1 Higher syntactic structures *

7.8.2 Sprite Reconstruction *

7.8.3 Low-latency sprite reconstruction *

7.8.4 Sprite reference point decoding *

7.8.5 Warping *

7.8.6 Sample reconstruction *

7.9 Generalized scalable decoding *

7.9.1 Temporal scalability *

7.9.2 Spatial scalability *

7.10 Still texture object decoding *

7.10.1 Decoding of the DC subband *

7.10.2 ZeroTree Decoding of the Higher Bands *

7.10.3 Inverse Quantization *

7.11 Mesh object decoding *

Foreword

(Foreword to be provided by ISO)

Introduction

Purpose

This part of ISO/IEC 14496 was developed in response to the growing need for a coding method that can facilitate access to visual objects in natural and synthetic moving pictures and associated natural or synthetic sound for various applications such as digital storage media, internet, various forms of wired or wireless communication etc. The use of ISO/IEC 14496 means that motion video can be manipulated as a form of computer data and can be stored on various storage media, transmitted and received over existing and future networks and distributed on existing and future broadcast channels.

Application

The applications of ISO/IEC 14496 cover, but are not limited to, such areas as listed below:

IMM Internet Multimedia

IVG Interactive Video Games

IPC Interpersonal Communications (videoconferencing, videophone, etc.)

ISM Interactive Storage Media (optical disks, etc.)

MMM Multimedia Mailing

NDB Networked Database Services (via ATM, etc.)

RES Remote Emergency Systems

RVS Remote Video Surveillance

WMM Wireless Multimedia

Profiles and levels

ISO/IEC 14496 is intended to be generic in the sense that it serves a wide range of applications, bitrates, resolutions, qualities and services. Furthermore, it allows a number of modes of coding of both natural and synthetic video in a manner facilitating access to individual objects in images or video, referred to as content based access. Applications should cover, among other things, digital storage media, content based image and video databases, internet video, interpersonal video communications, wireless video etc. In the course of creating ISO/IEC 14496, various requirements from typical applications have been considered, necessary algorithmic elements have been developed, and they have been integrated into a single syntax. Hence ISO/IEC 14496 will facilitate the bitstream interchange among different applications.

This part of ISO/IEC 14496 includes one or more complete decoding algorithms as well as a set of decoding tools. Moreover, the various tools of this part of ISO/IEC 14496 as well as that derived from ISO/IEC 13818-2 can be combined to form other decoding algorithms. Considering the practicality of implementing the full syntax of ISO/IEC 14496-2, however, a limited number of subsets of the syntax are also stipulated by means of "profile" and "level".

A "profile" is a defined subset of the entire bitstream syntax that is defined by this part of ISO/IEC 14496. Within the bounds imposed by the syntax of a given profile it is still possible to require a very large variation in the performance of encoders and decoders depending upon the values taken by parameters in the bitstream.

In order to deal with this problem "levels" are defined within each profile. A level is a defined set of constraints imposed on parameters in the bitstream. These constraints may be simple limits on numbers. Alternatively they may take the form of constraints on arithmetic combinations of the parameters.

Object based coding syntax

Video object

A *video object* in a scene is an entity that a user is allowed to access (seek, browse) and manipulate (cut and paste). The instances of video objects at a given time are called *video object planes* (VOPs). The encoding process generates a coded representation of a VOP as well as composition information necessary for display. Further, at the decoder, a user may interact with and modify the composition process as needed.

The full syntax allows coding of rectangular as well as arbitrarily shaped video objects in a scene. Furthermore, the syntax supports both nonscalable coding and scalable coding. Thus it becomes possible to handle normal scalabilities as well as object based scalabilities. The scalability syntax enables the reconstruction of useful video from pieces of a total bitstream. This is achieved by structuring the total bitstream in two or more layers, starting from a standalone base layer and adding a number of enhancement layers. The base layer can be coded using a non-scalable syntax, or in the case of picture based coding, even using a syntax of a different video coding standard.

To ensure the ability to access individual objects, it is necessary to achieve a coded representation of its shape. A natural video object consists of a sequence of 2D representations (at different points in time) referred to here as VOPs. For efficient coding of VOPs, both temporal redundancies as well as spatial redundancies are exploited. Thus a coded representation of a VOP includes representation of its shape, its motion and its texture.

Face object

A 3D (or 2D) ***face object*** **is a representation of the human face that is structured for portraying the visual manifestations of speech and facial expressions adequate to achieve visual speech intelligibility and the recognition of the mood of the speaker. A face object is animated by a stream of** ***face animation parameters (FAP)*** **encoded for low-bandwidth transmission in broadcast (one-to-many) or dedicated interactive (point-to-point) communications. The FAPs manipulate key feature control points in a mesh model of the face to produce animated visemes for the mouth (lips, tongue, teeth), as well as animation of the head and facial features like the eyes. FAPs are quantized with careful consideration for the limited movements of facial features, and then prediction errors are calculated and coded arithmetically. The remote manipulation of a face model in a terminal with FAPs can accomplish lifelike visual scenes of the speaker in real-time without sending pictorial or video details of face imagery every frame.**

A simple streaming connection can be made to a decoding terminal that animates a default face model. A more complex session can initialize a custom face in a more capable terminal by downloading ***face definition parameters (FDP)*** **from the encoder. Thus specific background images, facial textures, and head geometry can be portrayed. The composition of specific backgrounds, face 2D/3D meshes, texture attribution of the mesh, etc. is described in ISO/IEC 14496-1. The FAP stream for a given user can be generated at the user?s terminal from video/audio, or from text-to-speech. FAPs can be encoded at bitrates up to 2-3kbit/s at necessary speech rates. Optional temporal DCT coding provides further compression efficiency in exchange for delay. Using the facilities of ISO/IEC 14496-1, a composition of the animated face model and synchronized, coded speech audio (low-bitrate speech coder or text-to-speech) can provide an integrated low-bandwidth audio/visual speaker for broadcast applications or interactive conversation.**

Limited scalability is supported. Face animation achieves its efficiency by employing very concise motion animation controls in the channel, while relying on a suitably equipped terminal for rendering of moving 2D/3D faces with non-normative models held in local memory. Models stored and updated for rendering in the terminal can be simple or complex. To support speech intelligibility, the normative specification of FAPs intends for their selective or complete use as signaled by the encoder. A masking scheme provides for selective transmission of FAPs according to what parts of the face are naturally active from moment to moment. A further control in the FAP stream allows face animation to be suspended while leaving face features in the terminal in a defined quiescent state for higher overall efficiency during multi-point connections.

The Face Animation specification is defined in ISO/IEC 14496-1 and this part of ISO/IEC 14496. This clause is intended to facilitate finding various parts of specification. As a rule of thumb, FAP specification is found in the part 2, and FDP specification in the part 1. However, this is not a strict

rule. For an overview of FAPs and their interpretation, read subclauses "6.1.5.2 Facial animation parameter set", "6.1.5.3 Facial animation parameter units", "6.1.5.4 Description of a neutral face" as well as the Table C-1. The viseme parameter is documented in subclause "7.12.3 Decoding of the viseme parameter fap 1" and the Table C-5 in annex C. The expression parameter is documented in subclause "7.12.4 Decoding of the expression parameter fap 2" and the Table C-3. FAP bitstream syntax is found in subclauses "6.2.10 Face Object", semantics in "6.3.10 Face Object", and subclause "7.12 Face object decoding" explains in more detail the FAP decoding process. FAP masking and interpolation is explained in subclauses "6.3.11.1 Face Object Plane", "7.12.1.1 Decoding of faps", "7.12.5 Fap masking". The FIT interpolation scheme is documented in subclause "7.2.5.3.2.4 FIT" of ISO/IEC 14496-1. The FDPs and their interpretation are documented in subclause "7.2.5.3.2.6 FDP" of ISO/IEC 14496-1. In particular, the FDP feature points are documented in the Figure C-1.

Mesh object

A 2D *mesh object* is a representation of a 2D deformable geometric shape, with which synthetic video objects may be created during a composition process at the decoder, by spatially piece-wise warping of existing video object planes or still texture objects. The instances of mesh objects at a given time are called *mesh object planes* (mops). The geometry of mesh object planes is coded losslessly. Temporally and spatially predictive techniques and variable length coding are used to compress 2D mesh geometry. The coded representation of a 2D mesh object includes representation of its geometry and motion.

Overview of the object based nonscalable syntax

The coded representation defined in the non-scalable syntax achieves a high compression ratio while preserving good image quality. Further, when access to individual objects is desired, the shape of objects also needs to be coded, and depending on the bandwidth available, the shape information can be coded lossy or losslessly.

The compression algorithm employed for texture data is not lossless as the exact sample values are not preserved during coding. Obtaining good image quality at the bitrates of interest demands very high compression, which is not achievable with intra coding alone. The need for random access, however, is best satisfied with pure intra coding. The choice of the techniques is based on the need to balance a high image quality and compression ratio with the requirement to make random access to the coded bitstream.

A number of techniques are used to achieve high compression. The algorithm first uses block-based motion compensation to reduce the temporal redundancy. Motion compensation is used both for causal prediction of the current VOP from a previous VOP, and for non-causal, interpolative prediction from past and future VOPs. Motion vectors are defined for each 16-sample by 16-line region of a VOP or 8-sample by 8-line region of a VOP as required. The prediction error, is further compressed using the discrete cosine transform (DCT) to remove spatial correlation before it is quantised in an irreversible process that discards the less important information. Finally, the shape information, motion vectors and the quantised DCT information, are encoded using variable length codes.

Temporal processing

Because of the conflicting requirements of random access to and highly efficient compression, three main VOP types are defined. Intra coded VOPs (I-VOPs) are coded without reference to other pictures. They provide access points to the coded sequence where decoding can begin, but are coded with only moderate compression. Predictive coded VOPs (P-VOPs) are coded more efficiently using motion compensated prediction from a past intra or predictive coded VOPs and are generally used as a reference for further prediction. Bidirectionally-predictive coded VOPs (B-VOPs) provide the highest degree of compression but require both past and future reference VOPs for motion compensation. Bidirectionally-predictive coded VOPs are never used as references for prediction (except in the case that the resulting VOP is used as a reference for scalable enhancement layer). The organisation of the three VOP types in a sequence is very flexible. The choice is left to the encoder and will depend on the requirements of the application.

Coding of Shapes

In natural video scenes, VOPs are generated by segmentation of the scene according to some semantic meaning. For such scenes, the shape information is thus binary (binary shape). Shape information is also referred to as alpha plane. The binary alpha plane is coded on a macroblock basis by a coder which uses the context information, motion compensation and arithmetic coding.

For coding of shape of a VOP, a bounding rectangle is first created and is extended to multiples of 16´ 16 blocks with extended alpha samples set to zero. Shape coding is then initiated on a 16´ 16 block basis; these blocks are also referred to as binary alpha blocks.

Motion representation - macroblocks

The choice of 16´ 16 blocks (referred to as macroblocks) for the motion-compensation unit is a result of the trade-off between the coding gain provided by using motion information and the overhead needed to represent it. Each macroblock can further be subdivided to 8´ 8 blocks for motion estimation and compensation depending on the overhead that can be afforded.

Depending on the type of the macroblock, motion vector information and other side information is encoded with the compressed prediction error in each macroblock. The motion vectors are differenced with respect to a prediction value and coded using variable length codes. The maximum length of the motion vectors allowed is decided at the encoder. It is the responsibility of the encoder to calculate appropriate motion vectors. The specification does not specify how this should be done.

Spatial redundancy reduction

Both source VOPs and prediction errors VOPs have significant spatial redundancy. This part of ISO/IEC 14496 uses a block-based DCT method with optional visually weighted quantisation, and run-length coding. After motion compensated prediction or interpolation, the resulting prediction error is split into 8´ 8 blocks. These are transformed into the DCT domain where they can be weighted before being quantised. After quantisation many of the DCT coefficients are zero in

value and so two-dimensional run-length and variable length coding is used to encode the remaining DCT coefficients efficiently.

Chrominance formats

This part of ISO/IEC 14496 currently supports the 4:2:0 chrominance format.

Pixel depth

This part of ISO/IEC 14496 supports pixel depths between 4 and 12 bits in luminance and chrominance planes.

Generalized scalability

The scalability tools in this part of ISO/IEC 14496 are designed to support applications beyond that supported by single layer video. The major applications of scalability include internet video, wireless video, multi-quality video services, video database browsing etc. In some of these applications, either normal scalabilities on picture basis such as those in ISO/IEC 13818-2 may be employed or object based scalabilities may be necessary; both categories of scalability are enabled by this part of ISO/IEC 14496.

Although a simple solution to scalable video is the simulcast technique that is based on transmission/storage of multiple independently coded reproductions of video, a more efficient alternative is scalable video coding, in which the bandwidth allocated to a given reproduction of video can be partially re-utilised in coding of the next reproduction of video. In scalable video coding, it is assumed that given a coded bitstream, decoders of various complexities can decode and display appropriate reproductions of coded video. A scalable video encoder is likely to have increased complexity when compared to a single layer encoder. However, this part of ISO/IEC 14496 provides several different forms of scalabilities that address non-overlapping applications with corresponding complexities.

The basic scalability tools offered are temporal scalability and spatial scalability. Moreover, combinations of these basic scalability tools are also supported and are referred to as hybrid scalability. In the case of basic scalability, two layers of video referred to as the lower layer and the enhancement layer are allowed, whereas in hybrid scalability up to four layers are supported.

Object based Temporal scalability

Temporal scalability is a tool intended for use in a range of diverse video applications from video databases, internet video, wireless video and multiview/stereoscopic coding of video. Furthermore, it may also provide a migration path from current lower temporal resolution video systems to higher temporal resolution systems of the future.

Temporal scalability involves partitioning of VOPs into layers, where the lower layer is coded by itself to provide the basic temporal rate and the enhancement layer is coded with temporal prediction with respect to the lower layer. These layers when decoded and temporally multiplexed

yield full temporal resolution. The lower temporal resolution systems may only decode the lower layer to provide basic temporal resolution whereas enhanced systems of the future may support both layers. Furthermore, temporal scalability has use in bandwidth constrained networked applications where adaptation to frequent changes in allowed throughput are necessary. An additional advantage of temporal scalability is its ability to provide resilience to transmission errors as the more important data of the lower layer can be sent over a channel with better error performance, whereas the less critical enhancement layer can be sent over a channel with poor error performance. Object based temporal scalability can also be employed to allow graceful control of picture quality by controlling the temporal rate of each video object under the constraint of a given bit-budget.

Spatial scalability

Spatial scalability is a tool intended for use in video applications involving multi quality video services, video database browsing, internet video and wireless video, i.e., video systems with the primary common feature that a minimum of two layers of spatial resolution are necessary. Spatial scalability involves generating two spatial resolution video layers from a single video source such that the lower layer is coded by itself to provide the basic spatial resolution and the enhancement layer employs the spatially interpolated lower layer and carries the full spatial resolution of the input video source.

An additional advantage of spatial scalability is its ability to provide resilience to transmission errors as the more important data of the lower layer can be sent over a channel with better error performance, whereas the less critical enhancement layer data can be sent over a channel with poor error performance. Further, it can also allow interoperability between various standards.

Hybrid scalability

There are a number of applications where neither the temporal scalability nor the spatial scalability may offer the necessary flexibility and control. This may necessitate use of temporal and spatial scalability simultaneously and is referred to as the hybrid scalability. Among the applications of hybrid scalability are wireless video, internet video, multiviewpoint/stereoscopic coding etc.

Error Resilience

This part of ISO/IEC 14496 provides error robustness and resilience to allow accessing of image or video information over a wide range of storage and transmission media. The error resilience tools developed for this part of ISO/IEC 14496 can be divided into three major categories. These categories include synchronization, data recovery, and error concealment. It should be noted that these categories are not unique to this part of ISO/IEC 14496, and have been used elsewhere in general research in this area. It is, however, the tools contained in these categories that are of interest, and where this part of ISO/IEC 14496 makes its contribution to the problem of error resilience.

Information technology ¾ Very-low bitrate audio-visual coding ¾ Part 2: Visual

1. Scope

This part of ISO/IEC 14496 specifies the coded representation of picture information in the form of natural or synthetic visual objects like video sequences of rectangular or arbitrarily shaped pictures, moving 2D meshes, animated 3D face models and texture for synthetic objects. The coded representation allows for content based access for digital storage media, digital video communication and other applications. ISO/IEC 14496 specifies also the decoding process of the aforementioned coded representation. The representation supports constant bitrate transmission, variable bitrate transmission, robust transmission, content based random access (including normal random access), object based scalable decoding (including normal scalable decoding), object based bitstream editing, as well as special functions such as fast forward playback, fast reverse playback, slow motion, pause and still pictures. Synthetic objects and coding of special 2D/3D meshes, texture, and animation parameters are provided for use with downloadable models to exploit mixed media and the bandwidth improvement associated with remote manipulation of such models. ISO/IEC 14496 is intended to allow some level of interoperability with ISO/IEC 11172-2, ISO/IEC 13818-2 and ITU-T Recommendation H.263.

2. Normative references

The standards contain provisions which through reference in this text, constitute provisions of ISO/IEC 14496. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on ISO/IEC 14496 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards

*** ITU-T Recommendation T.81 (1992)|ISO/IEC 10918-1:1994, *Information technology -Digital compression and coding of continuous-tone still images: Requirements and guidelines.***

*** ISO/IEC 11172-1:1993, *Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 1: Systems.***

*** ISO/IEC 11172-2:1993, *Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 2: Video.***

*** ISO/IEC 11172-3:1993, *Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 3: Audio.***

*** ITU-T Recommendation H.222.0(1995)|ISO/IEC 13818-1:1996, *Information technology - Generic coding of moving pictures and associated audio information: Systems.***

*** ITU-T Recommendation H.262(1995)|ISO/IEC 13818-2:1996,** ***Information technology - Generic coding of moving pictures and associated audio information: Video.***

*** ISO/IEC 13818-3:1996,** ***Information technology - Generic coding of moving pictures and associated audio information - Part 3: Audio.***

*** Recommendations and reports of the CCIR, 1990 XVIIth Plenary Assembly, Dusseldorf, 1990 Volume XI - Part 1 Broadcasting Service (Television) Recommendation ITU-R BT.601-3,** ***Encoding parameters of digital television for studios*****.**

*** CCIR Volume X and XI Part 3 Recommendation ITU-R BR.648,** ***Recording of audio signals*****.**

*** CCIR Volume X and XI Part 3 Report ITU-R 955-2,** ***Satellite sound broadcasting to vehicular, portable and fixed receivers in the range 500 - 3000Mhz.***

*** IEEE Standard Specifications for the Implementations of 8 by 8 Inverse Discrete Cosine Transform, IEEE Std 1180-1990, December 6, 1990.**

*** IEC Publication 908:1987,** ***CD Digital Audio System*****.**

*** IEC Publication 461:1986,** ***Time and control code for video tape recorder.***

*** ITU-T Recommendation H.261 (Formerly CCITT Recommendation H.261** ***), Codec for audiovisual services at px64 kbit/s*****.**

*** ITU-T Recommendation H.263,** ***Video Coding for Low Bitrate Communication*** **.**

3. Definitions

1. **AC coefficient: Any DCT coefficient for which the frequency in one or both dimensions is non-zero.**

2. **B-VOP; bidirectionally predictive-coded video object plane (VOP): A VOP that is coded using motion compensated prediction from past and/or future reference VOPs.**

3. **backward compatibility: A newer coding standard is backward compatible with an older coding standard if decoders designed to operate with the older coding standard are able to continue to operate by decoding all or part of a bitstream produced according to the newer coding standard.**

4. **backward motion vector: A motion vector that is used for motion compensation from a reference VOP at a later time in display order.**

5. **backward prediction: Prediction from the future reference VOP.**

6. **base layer: An independently decodable layer of a scalable hierarchy.**

7. **binary alpha block: A block of size 16x16 pels, colocated with macroblock, representing shape information of the binary alpha map; it is also referred to as a bab.**

8. **binary alpha map: A 2D binary mask used to represent the shape of a video object such that the pixels that are opaque are considered as part of the object where as pixels that are transparent are not considered to be part of the object.**

9. **bitstream; stream: An ordered series of bits that forms the coded representation of the data.**

10. **bitrate: The rate at which the coded bitstream is delivered from the storage medium or network to the input of a decoder.**

11. **block: An 8-row by 8-column matrix of samples, or 64 DCT coefficients (source, quantised or dequantised).**

12. **byte aligned: A bit in a coded bitstream is byte-aligned if its position is a multiple of 8-bits from the first bit in the stream.**

13. **byte: Sequence of 8-bits.**

14. **context based arithmetic encoding: The method used for coding of binary shape; it is also referred to as cae.**

15. **channel: A digital medium or a network that stores or transports a bitstream constructed according to ISO/IEC 14496.**

16. **chrominance format: Defines the number of chrominance blocks in a macroblock.**

17. **chrominance component: A matrix, block or single sample representing one of the two colour difference signals related to the primary colours in the manner defined in the bitstream. The symbols used for the chrominance signals are Cr and Cb.**

18. **coded B-VOP: A B-VOP that is coded.**

19. **coded VOP: A coded VOP is a coded I-VOP, a coded P-VOP or a coded B-VOP.**

20. **coded I-VOP: An I-VOP that is coded.**

21. **coded P-VOP: A P-VOP that is coded.**

22. **coded video bitstream: A coded representation of a series of one or more VOPs as defined in this part of ISO/IEC 14496.**

23. **coded representation: A data element as represented in its encoded form.**

24. **coding parameters: The set of user-definable parameters that characterise a coded video bitstream. Bitstreams are characterised by coding parameters. Decoders are characterised by the bitstreams that they are capable of decoding.**

25. **component: A matrix, block or single sample from one of the three matrices (luminance and two chrominance) that make up a picture.**

26. **composition process: The (non-normative) process by which reconstructed VOPs are composed into a scene and displayed.**

27. **compression: Reduction in the number of bits used to represent an item of data.**

28. **constant bitrate coded video: A coded video bitstream with a constant bitrate.**

29. **constant bitrate: Operation where the bitrate is constant from start to finish of the coded bitstream.**

30. **conversion ratio: The size conversion ratio for the purpose of rate control of shape.**

31. **data element: An item of data as represented before encoding and after decoding.**

32. **DC coefficient: The DCT coefficient for which the frequency is zero in both dimensions.**

33. **DCT coefficient: The amplitude of a specific cosine basis function.**

34. **decoder input buffer: The first-in first-out (FIFO) buffer specified in the video buffering verifier.**

35. **decoder: An embodiment of a decoding process.**

36. **decoding order: The order in which the VOPs are transmitted and decoded. This order is not necessarily the same as the display order.**

37. **decoding (process): The process defined in this part of ISO/IEC 14496 that reads an input coded bitstream and produces decoded VOPs or audio samples.**

38. **dequantisation: The process of rescaling the quantised DCT coefficients after their representation in the bitstream has been decoded and before they are presented to the inverse DCT.**

39. **digital storage media; DSM: A digital storage or transmission device or system.**

40. **discrete cosine transform; DCT: Either the forward discrete cosine transform or the inverse discrete cosine transform. The DCT is an invertible, discrete orthogonal transformation. The inverse DCT is defined in annex A.**

41. display order: The order in which the decoded pictures are displayed. Normally this is the same order in which they were presented at the input of the encoder.

42. editing: The process by which one or more coded bitstreams are manipulated to produce a new coded bitstream. Conforming edited bitstreams must meet the requirements defined in this part of ISO/IEC 14496.

43. encoder: An embodiment of an encoding process.

44. encoding (process): A process, not specified in this part of ISO/IEC 14496, that reads a stream of input pictures or audio samples and produces a valid coded bitstream as defined in this part of ISO/IEC 14496.

45. enhancement layer: A relative reference to a layer (above the base layer) in a scalable hierarchy. For all forms of scalability, its decoding process can be described by reference to the lower layer decoding process and the appropriate additional decoding process for the enhancement layer itself.

46. face animation parameter units, FAPU: Special normalized units (e.g. translational, angular, logical) defined to allow interpretation of FAPs with any facial model in a consistent way to produce reasonable results in expressions and speech pronunciation.

47. face animation parameters, FAP: Coded streaming animation parameters that manipulate the displacements and angles of face features, and that govern the blending of visemes and face expressions during speech.

48. face animation table, FAT: A downloadable function mapping from incoming FAPs to feature control points in the face mesh that provides piecewise linear weightings of the FAPs for controlling face movements.

49. face calibration mesh: Definition of a 3D mesh for calibration of the shape and structure of a baseline face model.

50. face definition parameters, FDP: Downloadable data to customize a baseline face model in the decoder to a particular face, or to download a face model along with the information about how to animate it. The FDPs are normally transmitted once per session, followed by a stream of compressed FAPs. FDPs may include feature points for calibrating a baseline face, face texture and coordinates to map it onto the face, animation tables, etc.

51. face feature control point: A normative vertex point in a set of such points that define the critical locations within face features for control by FAPs and that allow for calibration of the shape of the baseline face.

52. face interpolation transform, FIT: A downloadable node type defined in ISO/IEC 14496-1 for optional mapping of incoming FAPs to FAPs before their application to feature points, through weighted rational polynomial functions, for complex cross-coupling of standard

FAPs to link their effects into custom or proprietary face models.

53. face model mesh: A 2D or 3D contiguous geometric mesh defined by vertices and planar polygons utilizing the vertex coordinates, suitable for rendering with photometric attributes (e.g. texture, color, normals).

54. feathering: A tool that tapers the values around edges of binary alpha mask for composition with the background.

55. flag: A one bit integer variable which may take one of only two values (zero and one).

56. forbidden: The term "forbidden" when used in the clauses defining the coded bitstream indicates that the value shall never be used. This is usually to avoid emulation of start codes.

57. forced updating: The process by which macroblocks are intra-coded from time-to-time to ensure that mismatch errors between the inverse DCT processes in encoders and decoders cannot build up excessively.

58. forward compatibility: A newer coding standard is forward compatible with an older coding standard if decoders designed to operate with the newer coding standard are able to decode bitstreams of the older coding standard.

59. forward motion vector: A motion vector that is used for motion compensation from a reference frame VOP at an earlier time in display order.

60. forward prediction: Prediction from the past reference VOP.

61. frame: A frame contains lines of spatial information of a video signal. For progressive video, these lines contain samples starting from one time instant and continuing through successive lines to the bottom of the frame.

62. frame period: The reciprocal of the frame rate.

63. frame rate: The rate at which frames are be output from the composition process.

64. future reference VOP: A future reference VOP is a reference VOP that occurs at a later time than the current VOP in display order.

65. VOP reordering: The process of reordering the reconstructed VOPs when the decoding order is different from the composition order for display. VOP reordering occurs when B-VOPs are present in a bitstream. There is no VOP reordering when decoding low delay bitstreams.

66. hybrid scalability: Hybrid scalability is the combination of two (or more) types of scalability.

67. interlace: The property of conventional television frames where alternating lines of the

frame represent different instances in time. In an interlaced frame, one of the field is meant to be displayed first. This field is called the first field. The first field can be the top field or the bottom field of the frame.

68. I-VOP; intra-coded VOP: A VOP coded using information only from itself.

69. intra coding: Coding of a macroblock or VOP that uses information only from that macroblock or VOP.

70. intra shape coding: Shape coding that does not use any temporal prediction.

71. inter shape coding: Shape coding that uses temporal prediction.

72. level: A defined set of constraints on the values which may be taken by the parameters of this part of ISO/IEC 14496 within a particular profile. A profile may contain one or more levels. In a different context, level is the absolute value of a non-zero coefficient (see "run").

73. layer: In a scalable hierarchy denotes one out of the ordered set of bitstreams and (the result of) its associated decoding process.

74. layered bitstream: A single bitstream associated to a specific layer (always used in conjunction with layer qualifiers, e. g. "enhancement layer bitstream").

75. lower layer: A relative reference to the layer immediately below a given enhancement layer (implicitly including decoding of *all* layers below this enhancement layer).

76. luminance component: A matrix, block or single sample representing a monochrome representation of the signal and related to the primary colours in the manner defined in the bitstream. The symbol used for luminance is Y.

77. Mbit: 1 000 000 bits.

78. macroblock: The four 8´ 8 blocks of luminance data and the two (for 4:2:0 chrominance format) corresponding 8´ 8 blocks of chrominance data coming from a 16´ 16 section of the luminance component of the picture. Macroblock is sometimes used to refer to the sample data and sometimes to the coded representation of the sample values and other data elements defined in the macroblock header of the syntax defined in this part of ISO/IEC 14496. The usage is clear from the context.

79. mesh: A 2D triangular mesh refers to a planar graph which tessellates a video object plane into triangular patches. The vertices of the triangular mesh elements are referred to as node points. The straight-line segments between node points are referred to as edges. Two triangles are adjacent if they share a common edge.

80. mesh geometry: The spatial locations of the node points and the triangular structure of a mesh.

81. mesh motion: The temporal displacements of the node points of a mesh from one time instance to the next.

82. motion compensation: The use of motion vectors to improve the efficiency of the prediction of sample values. The prediction uses motion vectors to provide offsets into the past and/or future reference VOPs containing previously decoded sample values that are used to form the prediction error.

83. motion estimation: The process of estimating motion vectors during the encoding process.

84. motion vector: A two-dimensional vector used for motion compensation that provides an offset from the coordinate position in the current picture or field to the coordinates in a reference VOP.

85. motion vector for shape: A motion vector used for motion compensation of shape.

86. non-intra coding: Coding of a macroblock or a VOP that uses information both from itself and from macroblocks and VOPs occurring at other times.

87. opaque macroblock: A macroblock with shape mask of all 255?s.

88. P-VOP; predictive-coded VOP: A picture that is coded using motion compensated prediction from the past VOP.

89. parameter: A variable within the syntax of this part of ISO/IEC 14496 which may take one of a range of values. A variable which can take one of only two values is called a flag.

90. past reference picture: A past reference VOP is a reference VOP that occurs at an earlier time than the current VOP in composition order.

91. picture: Source, coded or reconstructed image data. A source or reconstructed picture consists of three rectangular matrices of 8-bit numbers representing the luminance and two chrominance signals. A "coded VOP" was defined earlier. For progressive video, a picture is identical to a frame.

92. prediction: The use of a predictor to provide an estimate of the sample value or data element currently being decoded.

93. prediction error: The difference between the actual value of a sample or data element and its predictor.

94. predictor: A linear combination of previously decoded sample values or data elements.

95. profile: A subset of the syntax of this part of ISO/IEC 14496, defined in terms of Visual Object Types.

96. progressive: The property of film frames where all the samples of the frame represent the same instances in time.

97. quantisation matrix: A set of sixty-four 8-bit values used by the dequantiser.

98. quantised DCT coefficients: DCT coefficients before dequantisation. A variable length coded representation of quantised DCT coefficients is transmitted as part of the coded video bitstream.

99. quantiser scale: A scale factor coded in the bitstream and used by the decoding process to scale the dequantisation.

100. random access: The process of beginning to read and decode the coded bitstream at an arbitrary point.

101. reconstructed VOP: A reconstructed VOP consists of three matrices of 8-bit numbers representing the luminance and two chrominance signals. It is obtained by decoding a coded VOP.

102. reference VOP: A reference VOP is a reconstructed VOP that was coded in the form of a coded I-VOP or a coded P-VOP. Reference VOPs are used for forward and backward prediction when P-VOPs and B-VOPs are decoded.

103. reordering delay: A delay in the decoding process that is caused by VOP reordering.

104. reserved: The term "reserved" when used in the clauses defining the coded bitstream indicates that the value may be used in the future for ISO/IEC defined extensions.

105. scalable hierarchy: coded video data consisting of an ordered set of more than one video bitstream.

106. scalability: Scalability is the ability of a decoder to decode an ordered set of bitstreams to produce a reconstructed sequence. Moreover, useful video is output when subsets are decoded. The minimum subset that can thus be decoded is the first bitstream in the set which is called the base layer. Each of the other bitstreams in the set is called an enhancement layer. When addressing a specific enhancement layer, "lower layer" refers to the bitstream that precedes the enhancement layer.

107. side information: Information in the bitstream necessary for controlling the decoder.

108. run: The number of zero coefficients preceding a non-zero coefficient, in the scan order. The absolute value of the non-zero coefficient is called "level".

109. S-VOP: A picture that is coded using information obtained by warping whole or part of a static sprite.

110. saturation: Limiting a value that exceeds a defined range by setting its value to the maximum or minimum of the range as appropriate.

111. source; input: Term used to describe the video material or some of its attributes before encoding.

112. spatial prediction: prediction derived from a decoded frame of the reference layer decoder used in spatial scalability.

113. spatial scalability: A type of scalability where an enhancement layer also uses predictions from sample data derived from a lower layer without using motion vectors. The layers can have different VOP sizes or VOP rates.

114. static sprite: The luminance, chrominance and binary alpha plane for an object which does not vary in time.

115. start codes: 32-bit codes embedded in that coded bitstream that are unique. They are used for several purposes including identifying some of the structures in the coding syntax.

116. stuffing (bits); stuffing (bytes): Code-words that may be inserted into the coded bitstream that are discarded in the decoding process. Their purpose is to increase the bitrate of the stream which would otherwise be lower than the desired bitrate.

117. temporal prediction: prediction derived from reference VOPs other than those defined as spatial prediction.

118. temporal scalability: A type of scalability where an enhancement layer also uses predictions from sample data derived from a lower layer using motion vectors. The layers have identical frame size, and but can have different VOP rates.

119. top layer: the topmost layer (with the highest layer_id) of a scalable hierarchy.

120. transparent macroblock: A macroblock with shape mask of all zeros.

121. variable bitrate: Operation where the bitrate varies with time during the decoding of a coded bitstream.

122. variable length coding; VLC: A reversible procedure for coding that assigns shorter code-words to frequent events and longer code-words to less frequent events.

123. video buffering verifier; VBV: Part of a hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the variability of the data rate that an encoder or editing process may produce.

124. video complexity verifier; VCV: Part of a hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the

maximum processing requirements of the bitstream that an encoder or editing process may produce.

125. **video memory verifier; VMV: Part of a hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the maximum reference memory requirements of the bitstream that an encoder or editing process may produce.**

126. **video presentation verifier; VPV: Part of a hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the maximum presentation memory requirements of the bitstream that an encoder or editing process may produce.**

127. **video session: The highest syntactic structure of coded video bitstreams. It contains a series of one or more coded video objects.**

128. **viseme: the physical (visual) configuration of the mouth, tongue and jaw that is visually correlated with the speech sound corresponding to a phoneme.**

129. **warping: Processing applied to extract a sprite VOP from a static sprite. It consists of a global spatial transformation driven by a few motion parameters (0,2,4,6,8), to recover luminance, chrominance and shape information.**

130. **zigzag scanning order: A specific sequential ordering of the DCT coefficients from (approximately) the lowest spatial frequency to the highest.**

1. Abbreviations and symbols

The mathematical operators used to describe this part of ISO/IEC 14496 are similar to those used in the C programming language. However, integer divisions with truncation and rounding are specifically defined. Numbering and counting loops generally begin from zero.

1. Arithmetic operators

+ Addition.

- Subtraction (as a binary operator) or negation (as a unary operator).

++ Increment. i.e. *x*++ is equivalent to *x* = *x* + 1

- - Decrement. i.e. *x*-- is equivalent to *x* = *x* - 1

$\left.\begin{matrix}\times\\ *\end{matrix}\right\}$ **Multiplication.**

^ Power.

/ Integer division with truncation of the result toward zero. For example, 7/4 and -7/-4 are truncated to 1 and -7/4 and 7/-4 are truncated to -1.

// Integer division with rounding to the nearest integer. Half-integer values are rounded away from zero unless otherwise specified. For example 3//2 is rounded to 2, and -3//2 is rounded to -2.

/// Integer division with sign dependent rounding to the nearest integer. Half-integer values when positive are rounded away from zero, and when negative are rounded towards zero. For example 3///2 is rounded to 2, and -3///2 is rounded to -1.

//// Integer division with truncation towards the negative infinity.

÷ Used to denote division in mathematical equations where no truncation or rounding is intended.

% Modulus operator. Defined only for positive numbers.

Sign( ) $\mathrm{Sign}(x) = \begin{cases} 1 & x >= 0 \\ -1 & x < 0 \end{cases}$

Abs( ) $\mathrm{Abs}(x) = \begin{cases} x & x >= 0 \\ -x & x < 0 \end{cases}$

$\sum_{i=a}^{i<b} f(i)$ **The summation of the $f(i)$ with *i* taking integral values from *a* up to, but not including *b*.**

2. **Logical operators**

|| Logical OR.

&& Logical AND.

! Logical NOT.

3. Relational operators

> Greater than.

>= Greater than or equal to.

³ Greater than or equal to.

< Less than.

<= Less than or equal to.

£ Less than or equal to.

== Equal to.

!= Not equal to.

max [, ¼ ,] the maximum value in the argument list.

min [, ¼ ,] the minimum value in the argument list.

4. **Bitwise operators**

& AND

| OR

>> Shift right with sign extension.

<< Shift left with zero fill.

5. **Conditional operators**

?:

$$(condition ? a : b) = \begin{cases} a & \text{if } condition \text{ is true,} \\ b & \text{otherwise.} \end{cases}$$

6. **Assignment**

= Assignment operator.

7. **Mnemonics**

The following mnemonics are defined to describe the different data types used in the coded bitstream.

bslbf Bit string, left bit first, where "left" is the order in which bit strings are written in this part of ISO/IEC 14496. Bit strings are generally written as a string of 1s and 0s within single quote marks, e.g. ?1000 0001?. Blanks within a bit string are for ease of reading and have no significance. For convenience large strings are occasionally written in hexadecimal, in this case conversion to a binary in the conventional manner will yield the value of the bit string. Thus the left most hexadecimal digit is first and in each hexadecimal digit the most significant of the four bits is first.

uimsbf Unsigned integer, most significant bit first.

simsbf Signed integer, in twos complement format, most significant (sign) bit first.

vlclbf Variable length code, left bit first, where "left" refers to the order in which the VLC codes are written. The byte order of multibyte words is most significant byte first.

8. **Constants**

P 3,141 592 653 58¼

e 2,718 281 828 45¼

2. **Conventions**
    1. **Method of describing bitstream syntax**

The bitstream retrieved by the decoder is described in subclause 6.2. Each data item in the bitstream is in bold type. It is described by its name, its length in bits, and a mnemonic for its type and order of transmission.

The action caused by a decoded data element in a bitstream depends on the value of that data element and on data elements previously decoded. The decoding of the data elements and definition of the state variables used in their decoding are described in subclause 6.3. The following constructs are used to express the conditions when data elements are present, and are in normal type:

while (condition) {	If the condition is true, then the group of data elements
data_element	occurs next in the data stream. This repeats until the
. . .	condition is not true.
}	
do {	
data_element	The data element always occurs at least once.
. . .	
} while (condition)	The data element is repeated until the condition is not true.
if (condition) {	If the condition is true, then the first group of data
data_element	elements occurs next in the data stream.
. . .	
} else {	If the condition is not true, then the second group of data
data_element	elements occurs next in the data stream.
. . .	
}	
for (i = m; i < n; i++) {	The group of data elements occurs (n-m) times. Conditional
data_element	constructs within the group of data elements may depend
. . .	on the value of the loop control variable i, which is set to
}	m for the first occurrence, incremented by one for
	the second occurrence, and so forth.
/* comment ¼ */	Explanatory comment that may be deleted entirely without
	in any way altering the syntax.

This syntax uses the ?C-code? convention that a variable or expression evaluating to a non-zero value is equivalent to a condition that is true and a variable or expression evaluating to a zero value is equivalent to a condition that is false. In many cases a literal string is used in a condition. For example;

if (video_object_layer_shape == "rectangular") ¼

In such cases the literal string is that used to describe the value of the bitstream element in subclause 6.3. In this example, we see that "rectangular" is defined in a Table 6-14 to be represented by the two bit binary number ?00?.

As noted, the group of data elements may contain nested conditional constructs. For compactness, the brackets { } are omitted when only one data element follows.

data_element [n] data_element [n] is the n+1th element of an array of data.

data_element [m][n] data_element [m][n] is the m+1, n+1th element of a two-dimensional array of data.

data_element [l][m][n] data_element [l][m][n] is the l+1, m+1, n+1th element of a three-dimensional array of data.

While the syntax is expressed in procedural terms, it should not be assumed that subclause 6.2 implements a satisfactory decoding procedure. In particular, it defines a correct and error-free input bitstream. Actual decoders must include means to look for start codes in order to begin decoding correctly, and to identify errors, erasures or insertions while decoding. The methods to identify these situations, and the actions to be taken, are not standardised.

2. Definition of functions

Several utility functions for picture coding algorithm are defined as follows:

1. Definition of next_bits() function

The function next_bits() permits comparison of a bit string with the next bits to be decoded in the bitstream.

2. Definition of bytealigned() function

The function bytealigned () returns 1 if the current position is on a byte boundary, that is the next bit in the bitstream is the first bit in a byte. Otherwise it returns 0.

3. Definition of nextbits_bytealigned() function

The function nextbits_bytealigned() returns a bit string starting from the next byte aligned position. This permits comparison of a bit string with the next byte aligned bits to be decoded in the bitstream. If the current location in the bitstream is already byte aligned and the 8 bits following the current location are ?01111111?, the bits subsequent to these 8 bits are returned. The current location in the bitstream is not changed by this function.

4. Definition of next_start_code() function

The next_start_code() function removes any zero bit and a string of 0 to 7 ?1? bits used for stuffing and locates the next start code.

next_start_code() {	**No. of bits**	**M**
zero_bit	1	?0
while (!bytealigned())		
one_bit	1	?1
}		

This function checks whether the current position is byte aligned. If it is not, a zero stuffing bit followed by a number of one stuffing bits may be present before the start code.

5. Definition of next_resync_marker() function

The next_resync_marker() function removes any zero bit and a string of 0 to 7 ?1? bits used for stuffing and locates the next resync marker; it thus performs similar operation as next_start_code() but for resync_marker.

next_resync_marker() {	**No. of bits**	**Mn**
zero_bit	1	?0
while (!bytealigned())		
one_bit	1	?1
}		

6. Definition of transparent_mb() function

The function transparent_mb() returns 1 if the current macroblock consists only of transparent pixels. Otherwise it returns 0.

7. Definition of transparent_block() function

The function transparent_block(j) returns 1 if the 8x8 with index j consists only of transparent pixels. Otherwise it returns 0. The index value for each block is defined in Figure 6-5.

3. Reserved, forbidden and marker_bit

The terms "reserved" and "forbidden" are used in the description of some values of several fields in the coded bitstream.

The term "reserved" indicates that the value may be used in the future for ISO/IEC defined extensions.

The term "forbidden" indicates a value that shall never be used (usually in order to avoid emulation of start codes).

The term "marker_bit" indicates a one bit integer in which the value zero is forbidden (and it therefore shall have the value ?1?). These marker bits are introduced at several points in the syntax to avoid start code emulation.

The term "zero_bit" indicates a one bit integer with the value zero.

4. Arithmetic precision

In order to reduce discrepancies between implementations of this part of ISO/IEC 14496, the following rules for arithmetic operations are specified.

(a) Where arithmetic precision is not specified, such as in the calculation of the IDCT, the precision shall be sufficient so that significant errors do not occur in the final integer values.

(b) Where ranges of values are given, the end points are included if a square bracket is present, and excluded if a round bracket is used. For example, [a , b) means from a to b, including a but excluding b.

3. Visual bitstream syntax and semantics

1. Structure of coded visual data

Coded visual data can be of several different types, such as video data, still texture data, 2D mesh data or facial animation parameter data.

Synthetic objects and their attribution are structured in a hierarchical manner to support both bitstream scalability and object scalability. ISO/IEC 14496-1 of the specification provides the approach to spatial-temporal scene composition including normative 2D/3D scene graph nodes and their composition supported by Binary Interchange Format Specification. At this level, synthetic and natural object composition relies on ISO/IEC 14496-1 with subsequent (non-normative) rendering performed by the application to generate specific pixel-oriented views of the models.

Coded video data consists of an ordered set of video bitstreams, called layers. If there is only one layer, the coded video data is called non-scalable video bitstream. If there are two layers or more, the coded video data is called a scalable hierarchy.

One of the layers is called base layer, and it can always be decoded independently. Other layers are called enhancement layers, and can only be decoded together with the lower layers (previous layers in the ordered set), starting with the base layer. The multiplexing of these layers is discussed in ISO/IEC 14496-1. The base layer of a scalable set of streams can be coded by other standards. The Enhancement layers shall conform to this part of ISO/IEC 14496. In general the visual bitstream can be thought of as a syntactic hierarchy in which syntactic structures contain one or more subordinate structures.

Visual texture, referred to herein as still texture coding, is designed for maintaining high visual quality in the transmission and rendering of texture under widely varied viewing conditions typical of interaction with 2D/3D synthetic scenes. Still texture coding provides for a multi-layer representation of luminance, color and shape. This supports progressive transmission of the texture for image build-up as it is received by a terminal. Also supported is the downloading of the texture resolution hierarchy for construction of image pyramids used by 3D graphics APIs. Quality and SNR scalability are supported by the structure of still texture coding.

Coded mesh data consists of a single non-scalable bitstream. This bitstream defines the structure and motion of a 2D mesh object. Texture that is to be mapped onto the mesh geometry is coded separately.

Coded face animation parameter data consists of one non-scaleable bitstream. It defines the animation of the facemodel of the decoder. Face animation data is structured as standard formats for downloadable models and their animation controls, and a single layer of compressed face animation parameters used for remote manipulation of the face model. The face is a node in a scene graph that includes face geometry ready for rendering. 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. Upon initial or baseline construction, the face object contains a generic face with a neutral expression. This face can receive FAPs from the bitstream and be subsequently rendered to produce animation of the face. If FDPs are transmitted, the generic face is transformed into a particular face of specific shape and appearance. A downloaded face model via FDPs is a scene graph for insertion in the face node.

1. Visual object sequence

Visual object sequence is the highest syntactic structure of the coded visual bitstream.

A visual object sequence commences with a visual_object_sequence_start_code which is followed by one or more visual objects coded concurrently. The visual object sequence is terminated by a visual_object_sequence_end_code.

2. Visual object

A visual object commences with a visual_object_start_code, is followed by profile and level identification, and a visual object id, and is followed by a video object, a still texture object, a mesh object, or a face object.

3. Video object

A video object commences with a video_object_start_code, and is followed by one or more video object layers.

1. Progressive and interlaced sequences

This part of ISO/IEC 14496 deals with coding of both progressive and interlaced sequences.

The sequence, at the output of the decoding process, consists of a series of reconstructed VOPs separated in time and are readied for display via the compositor.

2. **Frame**

A frame consists of three rectangular matrices of integers; a luminance matrix (Y), and two chrominance matrices (Cb and Cr).

3. **VOP**

A reconstructed VOP is obtained by decoding a coded VOP. A coded VOP may have been derived from either a progressive or interlaced frame.

4. **VOP types**

There are four types of VOPs that use different coding methods:

1. An Intra-coded (I) VOP is coded using information only from itself.

2. A Predictive-coded (P) VOP is a VOP which is coded using motion compensated prediction from a past reference VOP.

3. A Bidirectionally predictive-coded (B) VOP is a VOP which is coded using motion compensated prediction from a past and/or future reference VOP(s).

4. A sprite (S) VOP is a VOP for a sprite object.

1. **I-VOPs and group of VOPs**

I-VOPs are intended to assist random access into the sequence. Applications requiring random access, fast-forward playback, or fast reverse playback may use I-VOPs relatively frequently.

I-VOPs may also be used at scene cuts or other cases where motion compensation is ineffective.

Group of VOP (GOV) header is an optional header that can be used immediately before a coded I-VOP to indicate to the decoder:

1. the modulo part (i.e. the full second units) of the time base for the next VOP after the GOV header in display order

1. if the first consecutive B-VOPs immediately following the coded I-VOP can be reconstructed properly in the case of a random access.

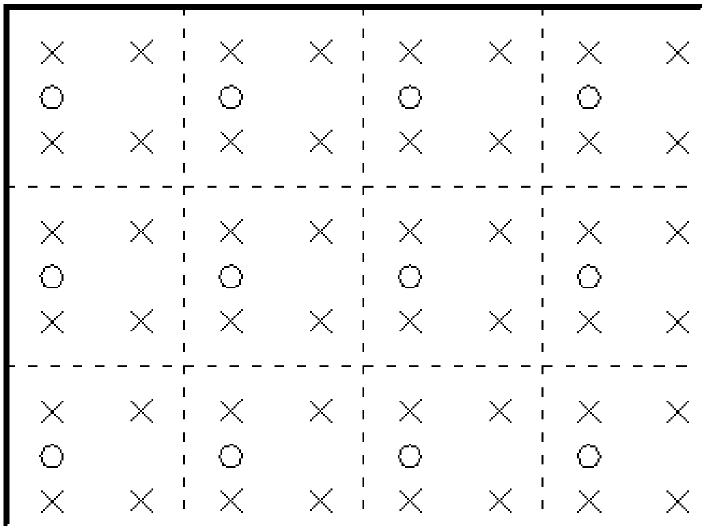
In a non scalable bitstream or the base layer of a scalable bitstream, the first coded VOP following a GOV header shall be a coded I-VOP.

1. **Format**

In this format the Cb and Cr matrices shall be one half the size of the Y-matrix in both horizontal and vertical dimensions. The Y-matrix shall have an even number of lines and samples.

The luminance and chrominance samples are positioned as shown in Figure 6-1.The two variations in the vertical and temporal positioning of the samples for interlaced VOPs are shown in Figure 6-2 and Figure 6-3.

Figure 6-4 shows the vertical and temporal positioning of the samples in a progressive frame.



× Represent luminance samples

○ Represent chrominance samples

Figure -1 -- The position of luminance and chrominance samples in 4:2:0 data

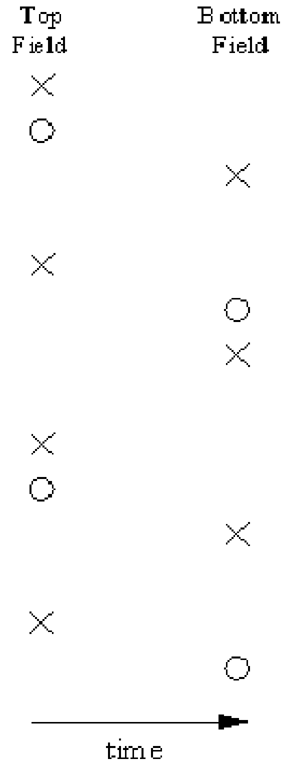


Figure -2 -- Vertical and temporal positions of samples in an interlaced frame with top_field_first=1

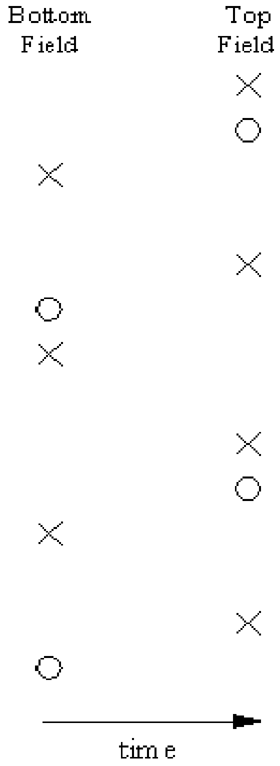


Figure -3 -- Vertical and temporal position of samples in an interlaced frame with top_field_first=0

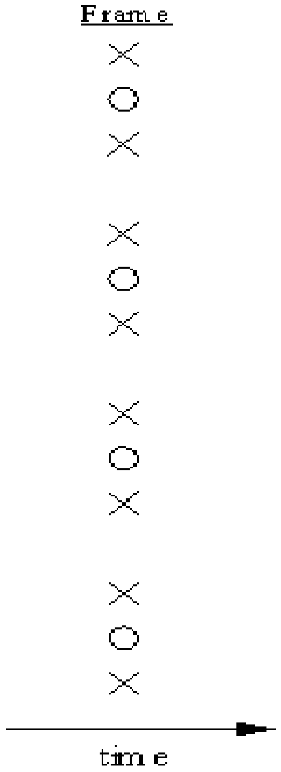


Figure -4 -- Vertical and temporal positions of samples in a progressive frame

The binary alpha plane for each VOP is represented by means of a bounding rectangle as described in clause F.2, and it has always the same number of lines and pixels per line as the luminance plane of the VOP bounding rectangle. The positions between the luminance and chrominance pixels of the bounding rectangle are defined in this clause according to the 4:2:0 format. For the progressive case, each 2x2 block of luminance pixels in the bounding rectangle associates to one chrominance pixel. For the interlaced case, each 2x2 block of luminance pixels of the same field in the bounding rectangle associates to one chrominance pixel of that field.

In order to perform the padding process on the two chrominance planes, it is necessary to generate a binary alpha plane which has the same number of lines and pixels per line as the chrominance planes. Therefore, when non-scalable shape coding is used, this binary alpha plane associated with the chrominance planes is created from the binary alpha plane associated with the luminance plane by the subsampling process defined below:

For each 2x2 block of the binary alpha plane associated with the luminance plane of the bounding rectangle (of the same frame for the progressive and of the same field for the interlaced case), the associated pixel value of the binary alpha plane associated with the chrominance planes is set to 255 if any pixel of said 2x2 block of the binary alpha plane associated with the luminance plane equals 255.

2. VOP reordering

When a video object layer contains coded B-VOPs, the number of consecutive coded B-VOPs is variable and unbounded. The first coded VOP shall not be a B-VOP.

A video object layer may contain no coded P-VOPs. A video object layer may also contain no coded I-VOPs in which case some care is required at the start of the video object layer and within the video object layer to effect both random access and error recovery.

The order of the coded VOPs in the bitstream, also called decoding order, is the order in which a decoder reconstructs them. The order of the reconstructed VOPs at the output of the decoding process, also called the display order, is not always the same as the decoding order and this subclause defines the rules of VOP reordering that shall happen within the decoding process.

When the video object layer contains no coded B-VOPs, the decoding order is the same as the display order.

When B-VOPs are present in the video object layer re-ordering is performed according to the following rules:

If the current VOP in decoding order is a B-VOP the output VOP is the VOP reconstructed from that B-VOP.

If the current VOP in decoding order is a I-VOP or P-VOP the output VOP is the VOP reconstructed from the previous I-VOP or P-VOP if one exists. If none exists, at the start of the video object layer, no VOP is output.

The following is an example of VOPs taken from the beginning of a video object layer. In this example there are two coded B-VOPs between successive coded P-VOPs and also two coded B-VOPs between successive coded I- and P-VOPs. VOP ?1I? is used to form a prediction for VOP ?4P?. VOPs ?4P? and ?1I? are both used to form predictions for VOPs ?2B? and ?3B?. Therefore the order of coded VOPs in the coded sequence shall be ?1I?, ?4P?, ?2B?, ?3B?. However, the decoder shall display them in the order ?1I?, ?2B?, ?3B?, ?4P?.

At the encoder input,

1 2 3 4 5 6 7 8 9 10 11 12 13

I B B P B B P B B I B B P

At the encoder output, in the coded bitstream, and at the decoder input,

1	4	2	3	7	5	6	10	8	9	13	11	12
I	P	B	B	P	B	B	I	B	B	P	B	B

At the decoder output,

1	2	3	4	5	6	7	8	9	10	11	12	13
I	B	B	P	B	B	P	B	B	I	B	B	P

3. **Macroblock**

A macroblock contains a section of the luminance component and the spatially corresponding chrominance components. The term macroblock can either refer to source and decoded data or to the corresponding coded data elements. A skipped macroblock is one for which no information is transmitted. Presently there is only one chrominance format for a macroblock, namely, 4:2:0 format. The orders of blocks in a macroblock is illustrated below:

A 4:2:0 Macroblock consists of 6 blocks. This structure holds 4 Y, 1 Cb and 1 Cr Blocks and the block order is depicted in Figure 6-5.

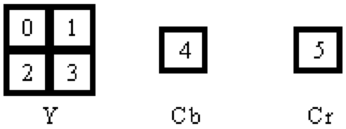


Figure -5 -- 4:2:0 Macroblock structure

The organisation of VOPs into macroblocks is as follows.

For the case of a progressive VOP, the interlaced flag (in the VOP header) is set to "0" and the organisation of lines of luminance VOP into macroblocks is called frame organization and is illustrated in Figure 6-6. In this case, frame DCT coding is employed.

For the case of interlaced VOP, the interlaced flag is set to "1" and the organisation of lines of luminance VOP into macroblocks can be either frame organization or field organization and thus both frame and field DCT coding may be used in the VOP.

- In the case of frame DCT coding, each luminance block shall be composed of lines from two fields alternately. This is illustrated in Figure 6-6.

- In the case of field DCT coding, each luminance block shall be composed of lines from only one of the two fields. This is illustrated in Figure 6-7.

Only frame DCT coding is applied to the chrominance blocks. It should be noted that field based predictions may be applied for these chrominance blocks which will require predictions of 8x4 regions (after half-sample filtering).

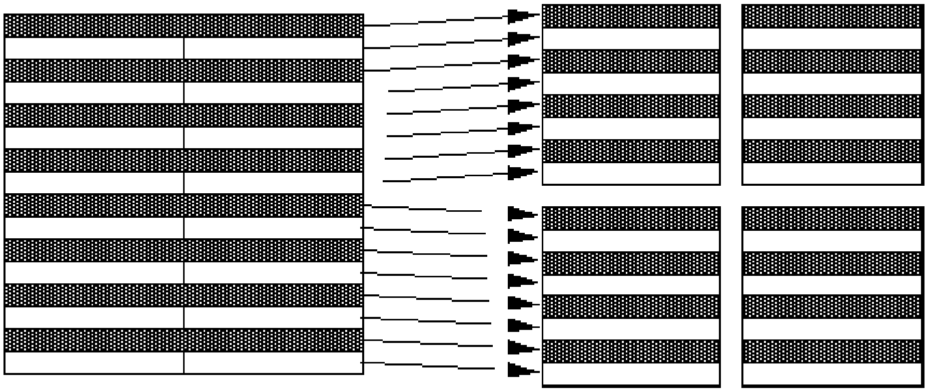


Figure -6 -- Luminance macroblock structure in frame DCT coding

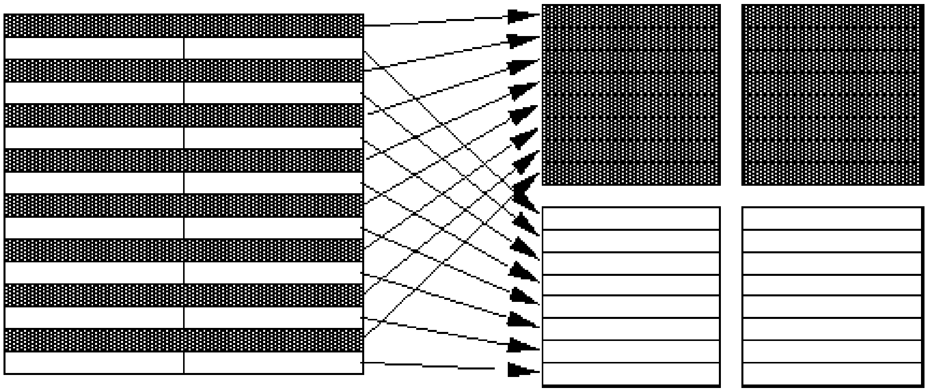


Figure -7 -- Luminance macroblock structure in field DCT coding

1. Block

The term **block** can refer either to source and reconstructed data or to the DCT coefficients or to the corresponding coded data elements.

When the block refers to source and reconstructed data it refers to an orthogonal section of a luminance or chrominance component with the same number of lines and samples. There are 8 lines and 8 samples/line in the block.

1. Mesh object

A 2D triangular *mesh* refers to a tessellation of a 2D visual object plane into triangular patches. The vertices of the triangular patches are called *node points*. The straight-line segments between node points are called *edges*. Two triangles are *adjacent* if they share a common edge.

A *dynamic* 2D mesh consists of a temporal sequence of 2D triangular meshes, where each mesh has the same topology, but node point locations may differ from one mesh to the next. Thus, a dynamic 2D mesh can be specified by the geometry of the initial 2D mesh and motion vectors at the node points for subsequent meshes, where each motion vector points from a node point of the previous mesh in the sequence to the corresponding node point of the current mesh. The dynamic 2D mesh can be used to create 2D animations by mapping texture from e.g. a video object plane onto successive 2D meshes.

A 2D dynamic mesh with *implicit structure* refers to a 2D dynamic mesh of which the initial mesh has either *uniform* or *Delaunay* topology. In both cases, the topology of the initial mesh does not have to be coded (since it is implicitly defined), only the node point locations of the initial mesh have to be coded. Note that in both the uniform and Delaunay case, the mesh is restricted to be *simple*, i.e. it consists of a single connected component without any holes, topologically equivalent to a disk.

A *mesh object* represents the geometry and motion of a 2D triangular mesh. A mesh object consists of one or more *mesh object planes*, each corresponding to a 2D triangular mesh at a certain time instance. An example of a mesh object is shown in the figure below.

A sequence of mesh object planes represents the piece-wise deformations to be applied to a video object plane or still texture object to create a synthetic animated video object. Triangular patches of a video object plane are to be warped according to the motion of corresponding triangular mesh elements. The motion of mesh elements is specified by the temporal displacements of the mesh node points.

The syntax and semantics of the mesh object pertains to the mesh geometry and mesh motion only; the video object to be used in an animation is coded separately. The warping or texture mapping applied to render visual object planes is handled in the context of scene composition. Furthermore, the syntax does not allow explicit encoding of other mesh properties such as colors or texture coordinates.

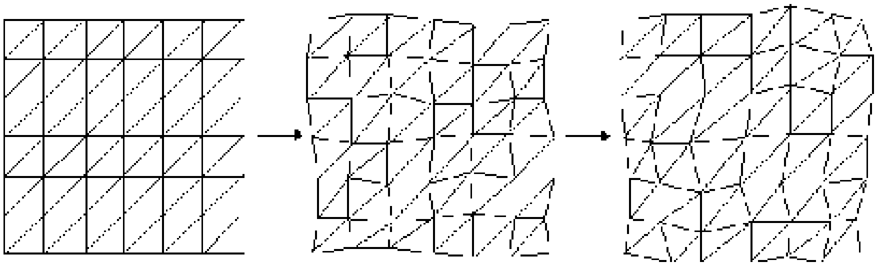


Figure -8 -- Mesh object with uniform triangular geometry

1. Mesh object plane

There are two types of mesh object planes that use different coding methods.

An *intra-coded* mesh object plane codes the geometry of a single 2D mesh. An intra-coded mesh is either of uniform or Delaunay type. In the case of a mesh of uniform type, the mesh geometry is coded by a small set of parameters. In the case of a mesh of Delaunay type, the mesh geometry is coded by the locations of the node points and boundary edge segments. The triangular mesh structure is specified implicitly by the coded information.

A *predictive-coded* mesh object plane codes a 2D mesh using temporal prediction from a past reference mesh object plane. The triangular structure of a predictive-coded mesh is identical to the structure of the reference mesh used for prediction; however, the locations of node points may change. The displacements of node points represent the motion of the mesh and are coded by specifying the motion vectors of node points from the reference mesh towards the predictive-coded mesh.

The locations of mesh node points correspond to locations in a video object or still texture object. Mesh node point locations and motion vectors are represented and coded with half pixel accuracy.

2. Face object

Conceptually the face object consists of a collection of nodes in a scene graph which are animated by the facial object bitstream. 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. Upon construction, the Face object contains a generic face with a neutral expression. This face can already be rendered. It is also immediately capable of receiving the FAPs from the bitstream, which will produce animation of the face: expressions, speech etc. 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 FDP and FAP sets are designed to allow the definition of a facial shape and texture, as well as animation of faces reproducing expressions, emotions and speech pronunciation. The FAPs, if correctly interpreted, will produce reasonably similar high level results in terms of expression and speech pronunciation on different facial models, without the need to initialize or calibrate the model. The FDPs allow the definition of a precise facial shape and texture in the setup phase. If the FDPs are used in the setup phase, it is also possible to produce more precisely the movements of particular facial features. Using a phoneme/bookmark to FAP conversion it is possible to control facial models accepting FAPs via TTS systems. The translation from phonemes to FAPs is not standardized. It is assumed that every decoder has a default face model with default parameters. Therefore, the setup stage is not necessary to create face animation. The setup stage is used to customize the face at the decoder.

1. **Structure of the face object bitstream**

A face object is formed by a temporal sequence of face object planes. This is depicted as follows in Figure 6-9.

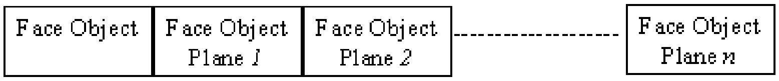
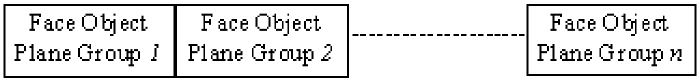


Figure -9 -- Structure of the face object bitstream

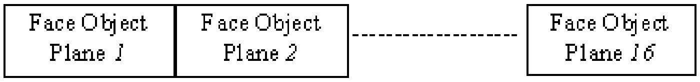
A face object represents a node in an ISO/IEC 14496 scene graph. An ISO/IEC 14496 scene is understood as a composition of Audio-Visual objects according to some spatial and temporal relationships. The scene graph is the hierarchical representation of the ISO/IEC 14496 scene structure (see ISO/IEC 14496-1).

Alternatively, a face object can be formed by a temporal sequence of face object plane groups (called segments for simplicity), where each face object plane group itself is composed of a temporal sequence of 16 face object planes, as depicted in the following:

face object:


Face Object Plane Group *1*
Face Object Plane Group *2*
Face Object Plane Group *n*

face object plane group:



When the alternative face object bitstream structure is employed, the bitstream is decoded by DCT-based face object decoding as described in subclause 7.12.2. Otherwise, the bitstream is decoded by the frame-based face object decoding. Refer to Table C-1 for a specification of default minimum and maximum values for each FAP

2. **Facial animation parameter set**

The FAPs are based on the study of minimal facial actions and are closely related to muscle actions. They represent a complete set of basic facial actions, and therefore allow the representation of most natural facial expressions. Exaggerated values permit the definition of actions that are normally not possible for humans, but could be desirable for cartoon-like characters.

The FAP set contains two high level parameters visemes and expressions. A viseme is a visual correlate to a phoneme. The viseme parameter allows viseme rendering (without having to express them in terms of other parameters) and enhances the result of other parameters, insuring the correct rendering of visemes. Only static visemes which are clearly distinguished are included in the standard set. Additional visemes may be added in future extensions of the standard. Similarly, the expression parameter allows definition of high level facial expressions. The facial expression parameter values are defined by textual descriptions. To facilitate facial animation, FAPs that can be used together to represent natural expression are grouped together in FAP groups, and can be indirectly addressed by using an expression parameter. The expression parameter allows for a very efficient means of animating faces. In annex C, a list of the FAPs is given, together with the FAP grouping, and the definitions of the facial expressions.

3. Facial animation parameter units

All the parameters involving translational movement are expressed in terms of the *Facial Animation Parameter Units (FAPU)*. These units are defined in order to allow interpretation of the FAPs on any facial model in a consistent way, producing reasonable results in terms of expression and speech pronunciation. They correspond to fractions of distances between some key facial features and are defined in terms of distances between feature points. The fractional units used are chosen to allow enough precision. annex C contains the list of the FAPs and the list of the FDP feature points. For each FAP the list contains the name, a short description, definition of the measurement units, whether the parameter is unidirectional (can have only positive values) or bi-directional, definition of the direction of movement for positive values, group number (for coding of selected groups), FDP subgroup number (annex C) and quantisation step size. FAPs act on FDP feature points in the indicated subgroups. The measurement units are shown in Table 6-1, where the notation 3.1.y represents the y coordinate of the feature point 3.1; also refer to Figure 6-10.

Table -1 -- Facial Animation Parameter Units

Description		FA
IRISD0 = 3.1.y - 3.3.y = 3.2.y - 3.4.y	Iris diameter (by definition it is equal to the distance between upper ad lower eyelid) in neutral face	IRISD =
ES0 = 3.5.x - 3.6.x	Eye separation	ES = ES
ENS0 = 3.5.y - 9.15.y	Eye - nose separation	ENS =
MNS0 = 9.15.y - 2.2.y	Mouth - nose separation	MNS =
MW0 = 8.3.x - 8.4.x	Mouth width	MW =
AU	Angle Unit	10^{-5} rad

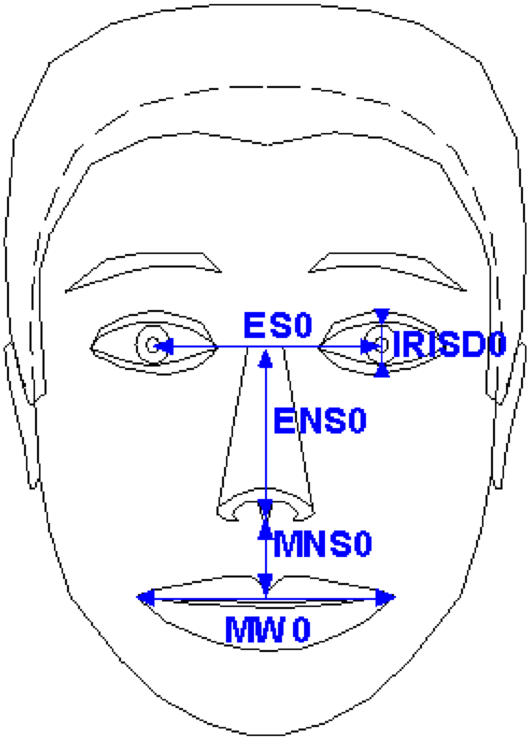


Figure -10 -- The Facial Animation Parameter Units

4. Description of a neutral face

At the beginning of a sequence, the face is supposed to be in a neutral position. Zero values of the FAPs correspond to a neutral face. All FAPs are expressed as displacements from the positions defined in the neutral face. The neutral face is defined as follows:

- the coordinate system is right-handed; head axes are parallel to the world axes
- gaze is in direction of Z axis
- all face muscles are relaxed
- eyelids are tangent to the iris
- the pupil is one third of IRISD0
- lips are in contact; the line of the lips is horizontal and at the same height of lip corners
- the mouth is closed and the upper teeth touch the lower ones
- the tongue is flat, horizontal with the tip of tongue touching the boundary between upper and lower teeth (feature point 6.1 touching 9.11 in annex C)

1. Facial definition parameter set

The FDPs are used to customize the proprietary face model of the decoder to a particular face or to download a face model along with the information about how to animate it. The definition and description of FDP fields is given in annex C. The FDPs are normally transmitted once per session, followed by a stream of compressed FAPs. However, if the decoder does not receive the FDPs, the use of FAPUs ensures that it can still interpret the FAP stream. This insures minimal operation in broadcast or teleconferencing applications. The FDP set is specified in BIFS syntax (see ISO/IEC 14496-1). The FDP node defines the face model to be used at the receiver. Two options are supported:

- calibration information is downloaded so that the proprietary face of the receiver can be configured using facial feature points and optionally a 3D mesh or texture.

- a face model is downloaded with the animation definition of the Facial Animation Parameters. This face model replace the proprietary face model in the receiver.

1. **Visual bitstream syntax**
    1. **Start codes**

Start codes are specific bit patterns that do not otherwise occur in the video stream.

Each start code consists of a start code prefix followed by a start code value. The start code prefix is a string of twenty three bits with the value zero followed by a single bit with the value one. The start code prefix is thus the bit string ?0000 0000 0000 0000 0000 0001?.

The start code value is an eight bit integer which identifies the type of start code. Many types of start code have just one start code value. However video_object_start_code and video_object_layer_start_code are represented by many start code values.

All start codes shall be byte aligned. This shall be achieved by first inserting a bit with the value zero and then, if necessary, inserting bits with the value one before the start code prefix such that the first bit of the start code prefix is the first (most significant) bit of a byte. For stuffing of 1 to 8 bits, the codewords are as follows in Table 6-2.

Table -2-- Stuffing codewords

Bits to be stuffed	**Stuffing Codeword**
1	0
2	01
3	011
4	0111
5	01111
6	011111
7	0111111
8	01111111

Table 6-3 defines the start code values for all start codes used in the visual bitstream.

Table -3 - Start code values

name	**start code value (hexadecimal)**
video_object_start_code	00 through 1F
video_object_layer_start_code	20 through 2F
reserved	30 through AF
visual_object_sequence__start_code	B0
visual_object_sequence_end_code	B1
user_data_start_code	B2
group_of_vop_start_code	B3
video_session_error_code	B4
visual_object_start_code	B5
vop_start_code	B6
reserved	B7-B9
face_object_start_code	BA
face_object_plane_start_code	BB
mesh_object_start_code	BC
mesh_object_plane_start_code	BD
still_texture_object_start_code	BE
texture_spatial_layer_start_code	BF
texture_snr_layer_start_code	C0
reserved	C1-C5
System start codes (see note)	C6 through FF
NOTE System start codes are defined in ISO/IEC 14496-1	

The use of the start codes is defined in the following syntax description with the exception of the video_session_error_code. The video_session_error_code has been allocated for use by a media interface to indicate where uncorrectable errors have been detected.

This syntax for visual bitstreams defines two types of information:

1. Configuration information

a. Global configuration information, referring to the whole group of visual objects that will be simultaneously decoded and composited by a decoder (VisualObjectSequence()).

b. Object configuration information, referring to a single visual object (VO). This is associated with VisualObject().

c. Object layer configuration information, referring to a single layer of a single visual object (VOL) VisualObjectLayer()

2. Elementary stream data, containing the data for a single layer of a visual object.

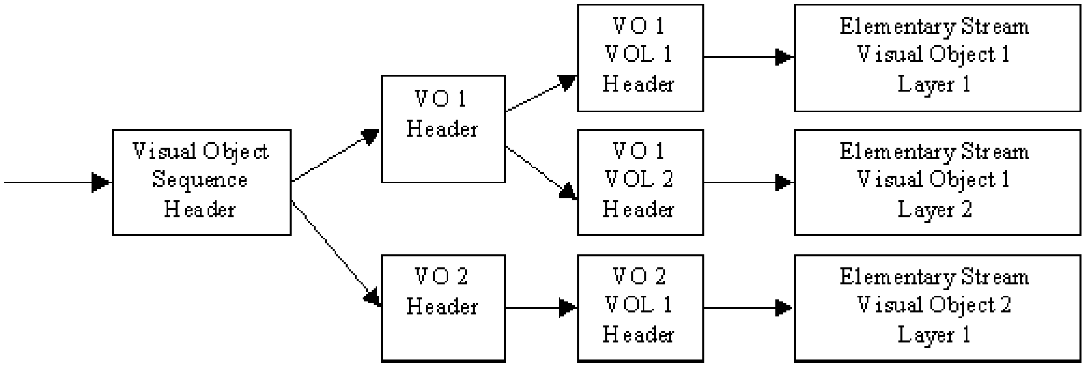

Visual Object Sequence Header
VO 1 Header
VO 2 Header
VO 1 VOL 1 Header
VO 1 VOL 2 Header
VO 2 VOL 1 Header
Elementary Stream Visual Object 1 Layer 1
Elementary Stream Visual Object 1 Layer 2
Elementary Stream Visual Object 2 Layer 1

Figure -11 -- Example Visual Information - Logical Structure

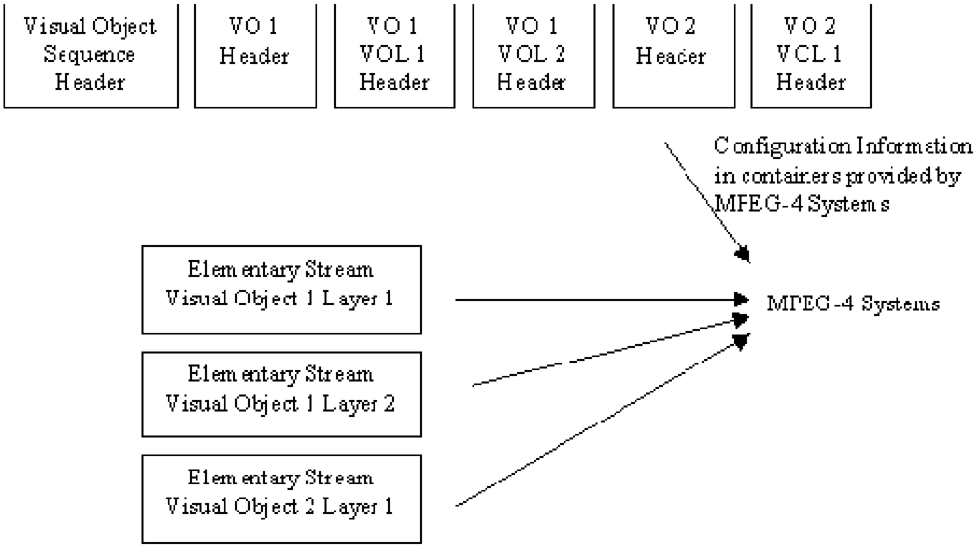

Visual Object Sequence Header
VO 1 Header
VO 1 VOL 1 Header
VO 1 VOL 2 Header
VO 2 Header
VO 2 VCL 1 Header
Configuration Information in containers provided by MFEG-4 Systems
Elementary Stream Visual Object 1 Layer 1
Elementary Stream Visual Object 1 Layer 2
Elementary Stream Visual Object 2 Layer 1
MPEG-4 Systems

Figure -12 -- Example Visual Bitstream - Separate Configuration Information / Elementary Stream.

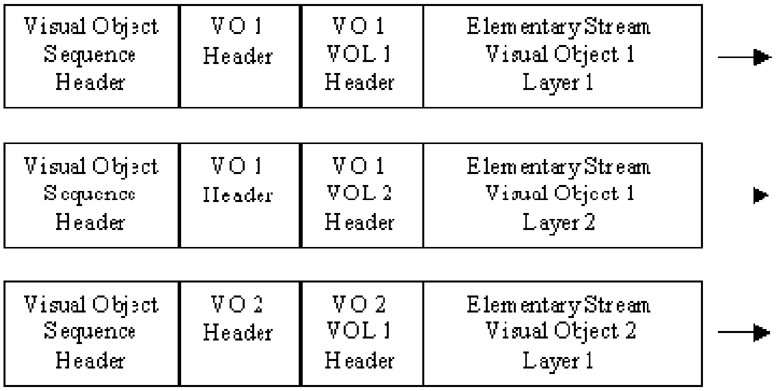

Visual Object Sequence Header
VO 1 Header
VO 1 VOL 1 Header
Elementary Stream Visual Object 1 Layer 1
Visual Object Sequence Header
VO 1 Header
VO 1 VOL 2 Header
Elementary Stream Visual Object 1 Layer 2
Visual Object Sequence Header
VO 2 Header
VO 2 VOL 1 Header
Elementary Stream Visual Object 2 Layer 1

Figure -13 -- Example Visual Bitstream - Combined Configuration Information / Elementary Stream

The following functions are entry points for elementary streams, and entry into these functions defines the breakpoint between configuration information and elementary streams:

1. Group_of_VideoObjectPlane(),

2. VideoObjectPlane(),

3. video_plane_with_short_header(),

4. MeshObject(),

5. FaceObject().

For still texture objects, configuration information ends and elementary stream data begins in StilTextureObject() immediately before the first call to wavelet_dc_decode(), as indicated by the comment in subclause 6.2.8.

There is no overlap of syntax between configuration information and elementary streams.

The configuration information contains all data that is not part of an elementary stream, including that defined by VisualObjectSequence(), VisualObject() and VideoObjectLayer().

ISO/IEC 14496-2 does not provide for the multiplexing of multiple elementary streams into a single bitstream. One visual bitstream contains exactly one elementary stream, which describes one layer of one visual object. A visual decoder must conceptually have a separate entry port for each layer of each object to be decoded.

Visual objects coded in accordance with this Part may be carried within a Systems bitstream as defined by ISO/IEC 14496-1. The coded visual objects may also be free standing or carried within other types of systems. Configuration information may be carried separately from or combined with elementary stream data:

1. *Separate Configuration / Elementary Streams (e.g. Inside ISO/IEC 14496-1 Bitstreams)*

When coded visual objects are carried within a Systems bitstream defined by ISO/IEC 14496-1, configuration information and elementary stream data are always carried separately. Configuration information and elementary streams follow the syntax below, subject to the break points between them defined above. The Systems specification ISO/IEC 14496-1 defines containers that are used to carry Visual Object and Visual Object Layer configuration information. A separate container is used for each object. For video objects, a separate container is also used for each layer. VisualObjectSequence headers are not carried explicitly, but the information is contained in other parts of the Systems bitstream.

2. *Combined Configuration / Elementary Streams*

The elementary stream data associated with a single layer may be wrapped in configuration information defined in accordance with the syntax below. A visual bitstream may contain at most one instance of each of VisualObjectSequence(), VisualObject() and VideoObjectLayer(). The Visual Object Sequence Header must be identical for all streams input simultaneously to a decoder. The Visual Object Headers for each layer of a multilayer object must be identical.

2. Visual Object Sequence and Visual Object

VisualObjectSequence() {	**No. of bits**	**M**
visual_object_sequence_start_code	32	bsl
profile_and_level_indication	8	uir
while (next_bits()== user_data_start_code){		
user_data()		
}		
VisualObject()		
visual_object_sequence_end_code	32	bsl
}		

VisualObject() {	**No. of bits**	**M**
visual_object_start_code	32	bs
is_visual_object_identifier	1	uir
if (is_visual_object_identifier) {		
visual_object_verid	4	uir
visual_object_priority	3	uir
}		
visual_object_type	4	uir
if (visual_object_type == "video ID" || visual_object_type == "still texture ID") {		
video_signal_type()		
}		
next_start_code()		
while (next_bits()== user_data_start_code){		
user_data()		
}		

if (visual_object_type == "video ID") {		
video_object_start_code	32	bs
VideoObjectLayer()		
}		
else if (visual_object_type == "still texture ID") {		
StillTextureObject()		
}		
else if (visual_object_type == "mesh ID") {		
MeshObject()		
}		
else if (visual_object_type == "face ID") {		
FaceObject()		
}		
if (next_bits() != "0000 0000 0000 0000 0000 0001")		
next_start_code()		
}		

video_signal_type() {	**No. of bits**	**M**
video_signal_type	1	bsl
if (video_signal_type) {		
video_format	3	ui
video_range	1	bsl
colour_description	1	bsl
if (colour_description) {		
colour_primaries	8	ui
transfer_characteristics	8	ui
matrix_coefficients	8	ui
}		
}		
}		

1. **User data**

user_data() {	**No. of bits**	**M**
user_data_start_code	32	bsl
while(next_bits() != ‘ 0000 0000 0000 0000 0000 0001’) {		
user_data	8	ui
}		
next_start_code()		
}		

3. **Video Object Layer**

VideoObjectLayer() {	**No. of bits**	**M**
if(next_bits() == video_object_layer_start_code) {		
short_video_header = 0		

video_object_layer_start_code	32	bsl
random_accessible_vol	1	bsl
video_object_type_indication	8	uir
is_object_layer_identifier	1	uir
if (is_object_layer_identifier) {		
video_object_layer_verid	4	uir
video_object_layer_priority	3	uir
}		
a**spect_ratio_info**	4	uir
if (aspect_ratio_info == "extended_PAR") {		
par_width	8	uir
par_height	8	uir
}		
vol_control_parameters	1	bsl
if (vol_control_parameters) {		
chroma_format	2	uir
low_delay	1	uir
vbv_parameters	1	bls
if (vbv_parameters) {		
first_half_bit_rate	15	uir
marker_bit	1	bsl
latter_half_bit_rate	15	uir
marker_bit	1	bsl
first_half_vbv_buffer_size	15	uir
marker_bit	1	bsl
latter_half_vbv_buffer_size	3	uir
first_half_vbv_occupancy	11	uir

marker_bit	1	bls
latter_half_vbv_occupancy	15	uir
marker_bit	1	bls
}		
}		
video_object_layer_shape	2	uir
marker_bit	1	bsl
vop_time_increment_resolution	16	uir
marker_bit	1	bsl
fixed_vop_rate	1	bsl
if (fixed_vop_rate)		
fixed_vop_time_increment	1-16	uir
if (video_object_layer_shape != "binary only") {		
if (video_object_layer_shape == "rectangular") {		
marker_bit	1	bsl
video_object_layer_width	13	uir
marker_bit	1	bsl
video_object_layer_height	13	uir
marker_bit	1	bsl
}		
interlaced	1	bsl
obmc_disable	1	bsl
sprite_enable	1	bsl
if (sprite_enable) {		
sprite_width	13	uir
marker_bit	1	bsl
sprite_height	13	uir

marker_bit	1	bsl
sprite_left_coordinate	13	sir
marker_bit	1	bsl
sprite_top_coordinate	13	sir
marker_bit	1	bsl
no_of_sprite_warping_points	6	uir
sprite_warping_accuracy	2	uir
sprite_brightness_change	1	bsl
low_latency_sprite_enable	1	bsl
}		
not_8_bit	1	bsl
if (not_8_ bit) {		
quant_precision	4	uir
bits_per_pixel	4	uir
}		
if (video_object_layer_shape=="grayscale") {		
no_gray_quant_update	1	bsl
composition_method	1	bsl
linear_composition	1	bsl
}		
quant_type	1	bsl
if (quant_type) {		
load_intra_quant_mat	1	bsl
if (load_intra_quant_mat)		
intra_quant_mat	8*[2-64]	uir
load_nonintra_quant_mat	1	bsl
if (load_nonintra_quant_mat)		

nonintra_quant_mat	8*[2-64]	uir
if(video_object_layer_shape=="grayscale") {		
load_intra_quant_mat_grayscale	1	bsl
if(load_intra_quant_mat_grayscale)		
intra_quant_mat_grayscale	8*[2-64]	uir
load_nonintra_quant_mat_grayscale	1	bsl
if(load_nonintra_quant_mat_grayscale)		
nonintra_quant_mat_grayscale	8*[2-64]	uir
}		
}		
complexity_estimation_disable	1	bsl
if (!complexity_estimation_disable)		
define_vop_complexity_estimation_header()		
resync_marker_disable	1	bsl
data_partitioned	1	bsl
if(data_partitioned)		
reversible_vlc	1	bsl
scalability	1	bsl
if (scalability) {		
hierarchy_type	1	bsl
ref_layer_id	4	uir
ref_layer_sampling_direc	1	bsl
hor_sampling_factor_n	5	uir
hor_sampling_factor_m	5	uir
vert_sampling_factor_n	5	uir
vert_sampling_factor_m	5	uir
enhancement_type	1	bsl

}		
}		
else		
resync_marker_disable	1	bs
next_start_code()		
while (next_bits()== user_data_start_code){		
user_data()		
}		
if (sprite_enable && !low_latency_sprite_enable)		
VideoObjectPlane()		
do {		
if (next_bits() == group_of_vop_start_code)		
Group_of_VideoObjectPlane()		
VideoObjectPlane()		
} while ((next_bits() == group_of_vop_start_code) || (next_bits() == vop_start_code))		
} else {		
short_video_header = 1		
do {		
video_plane_with_short_header()		
} while(next_bits() == short_video_start_marker)		
}		
}		

define_vop_complexity_estimation_header() {	**No. of bits**	**M**
estimation_method	2	ui

if (estimation_method ==?00?){		
shape_complexity_estimation_disable	1	
if (!shape_complexity_estimation_disable) {		bsl
opaque	1	bsl
transparent	1	bsl
intra_cae	1	bsl
inter_cae	1	bsl
no_update	1	bsl
upsampling	1	bsl
}		
texture_complexity_estimation_set_1_disable	1	bsl
if (!texture_complexity_estimation_set_1_disable) {		
intra_blocks	1	bsl
inter_blocks	1	bsl
inter4v_blocks	1	bsl
not_coded_blocks	1	bsl
}		
marker_bit	1	bsl
texture_complexity_estimation_set_2_disable	1	bsl
if (!texture_complexity_ estimation_set_2_disable) {		
dct_coefs	1	bsl
dct_lines	1	bsl
vlc_symbols	1	bsl
vlc_bits	1	bsl
}		
motion_compensation_complexity_disable	1	bsl
If (!motion_compensation_complexity_disable) {		

apm	1	bsl
npm	1	bsl
interpolate_mc_q	1	bsl
forw_back_mc_q	1	bsl
halfpel2	1	bsl
halfpel4	1	bsl
}		
marker_bit	1	bsl
}		
}		

4. **Group of Video Object Plane**

Group_of_VideoObjectPlane() {	**No. of bits**	**M**
group_vop_start_codes	32	bsl
time_code	18	
closed_gov	1	bsl
broken_link	1	bsl
next_start_code()		
while (next_bits()== user_data_start_code){		
user_data()		
}		
}		

5. **Video Object Plane and Video Plane with Short Header**

VideoObjectPlane() {	**No. of bits**	**M**
vop_start_code	32	bsl
vop_coding_type	2	uir

do {		
modulo_time_base	1	bsl
} while (modulo_time_base != ?0?)		
marker_bit	1	bsl
vop_time_increment	1-16	uir
marker_bit	1	bsl
vop_coded	1	bsl
if (vop_coded == ?0?) {		
next_start_code()		
return()		
}		
if ((video_object_layer_shape != "binary only") && (vop_coding_type == "P"))		
vop_rounding_type	1	bsl
if (video_object_layer_shape != "rectangular") {		
if(!(sprite_enable && vop_coding_type == "I")) {		
vop_width	13	uir
marker_bit	1	bsl
vop_height	13	uir
marker_bit	1	bsl
vop_horizontal_mc_spatial_ref	13	sir
marker_bit	1	bsl
vop_vertical_mc_spatial_ref	13	sir
}		
if ((video_object_layer_shape != " binary only") && scalability && enhancement_type)		
background_composition	1	bsl

change_conv_ratio_disable	1	bsl
vop_constant_alpha	1	bsl
if (vop_constant_alpha)		
vop_constant_alpha_value	8	bsl
}		
if (!complexity_estimation_disable)		
read_vop_complexity_estimation_header()		
if (video_object_layer_shape != "binary only") {		
intra_dc_vlc_thr	3	uir
if (interlaced) {		
top_field_first	1	bsl
alternate_vertical_scan_flag	1	bsl
}		
}		
if (sprite_enable && vop_coding_type == "S") {		
if (no_sprite_points > 0)		
sprite_trajectory()		
if (sprite_brightness_change)		
brightness_change_factor()		

if (sprite_transmit_mode != "stop"
&& low_latency_sprite_enable) {		

do {		
sprite_transmit_mode	2	uir
if ((sprite_transmit_mode == "piece") || (sprite_transmit_mode == "update"))		
decode_sprite_piece()		

} while (sprite_transmit_mode != "stop" && sprite_transmit_mode != "pause")		
}		
next_start_code()		
return()		
}		
if (video_object_layer_shape != "binary only") {		
vop_quant	3-9	ui
if(video_object_layer_shape=="grayscale")		
vop_alpha_quant	6	ui
if (vop_coding_type != "I")		
vop_fcode_forward	3	ui
if (vop_coding_type == "B")		
vop_fcode_backward	3	ui
if (!scalability) {		
if (video_object_layer_shape != "rectangular" && vop_coding_type != "I")		
vop_shape_coding_type	1	bsl
motion_shape_texture()		
while (nextbits_bytealigned() == resync_marker) {		
video_packet_header()		
motion_shape_texture()		
}		
}		
else {		
if (enhancement_type) {		

load_backward_shape	1	bsl
if (load_backward_shape) {		
backward_shape_width	13	uir
marker_bit	1	bsl
backward_shape_ height	13	uir
marker_bit	1	bsl
backward_shape_horizontal_mc_spatial_ref	13	sir
marker_bit	1	bsl
backward_shape_vertical_mc_spatial_ref	13	sir
backward_shape()		
load_forward_shape	1	bsl
if (load_forward_shape) {		
forward_shape_width	13	uir
marker_bit	1	bsl
forward_shape_height	13	uir
marker_bit	1	bsl
forward_shape_horizontal_mc_spatial_ref	13	sir
marker_bit	1	bsl
forward_shape_vertical_mc_spatial_ref	13	sir
forward_shape()		
}		
}		
}		
ref_select_code	2	uir
combined_motion_shape_texture()		
}		
}		

else {		
combined_motion_shape_texture()		
while (nextbits_bytealigned() == resync_marker) {		
video_packet_header()		
combined_motion_shape_texture()		
}		
}		
next_start_code()		
}		

1. **Complexity Estimation Header**

read_vop_complexity_estimation_header() {	**No. of bi**
if (estimation_method==?00?){	
if (vop_coding_type=="I"){	
if (opaque) **dcecs_opaque**	8
if (transparent) **dcecs_transparent**	8
if (intra_cae) **dcecs_intra_cae**	8
if (inter_cae) **dcecs_inter_cae**	8
if (no_update) **dcecs_no_update**	8
if (upsampling) **dcecs_upsampling**	8
if (intra_blocks) **dcecs_intra_blocks**	8
if (not_coded_blocks) **dcecs_not_coded_blocks**	8
if (dct_coefs) **dcecs_dct_coefs**	8
if (dct_lines) **dcecs_dct_lines**	8
if (vlc_symbols) **dcecs_vlc_symbols**	8
if (vlc_bits) **dcecs_vlc_bits**	4
}	

if (vop_coding_type=="P"){	
if (opaque) **dcecs_opaque**	8
if (transparent) **dcecs_transparent**	8
if (intra_cae) **dcecs_intra_cae**	8
if (inter_cae) **dcecs_inter_cae**	8
if (no_update) **dcecs_no_update**	8
if (upsampling) **dcecs_upsampling**	8
if (intra) **dcecs_intra_blocks**	8
if (not_coded) **dcecs_not_coded_blocks**	8
if (dct_coefs) **dcecs_dct_coefs**	8
if (dct_lines) **dcecs_dct_lines**	8
if (vlc_symbols) **dcecs_vlc_symbols**	8
if (vlc_bits) **dcecs_vlc_bits**	4
if (inter_blocks) **dcecs_inter_blocks**	8
if (inter4v_blocks) **dcecs_inter4v_blocks**	8
if (apm) **dcecs_apm**	8
if (npm) **dcecs_npm**	8
if (forw_back_mc_q) **dcecs_forw_back_mc_q**	8
if (halfpel2) **dcecs_halfpel2**	8
if (halfpel4) **dcecs_halfpel4**	8
}	
if (vop_coding_type=="B"){	
if (opaque) **dcecs_opaque**	8
if (transparent) **dcecs_transparent**	8
if (intra_cae) **dcecs_intra_cae**	8
if (inter_cae) **dcecs_inter_cae**	8
if (no_update) **dcecs_no_update**	8

if (upsampling) **dcecs_upsampling**	8
if (intra_blocks) **dcecs_intra_blocks**	8
if (not_coded_blocks) **dcecs_not_coded_blocks**	8
if (dct_coefs) **dcecs_dct_coefs**	8
if (dct_lines) **dcecs_dct_lines**	8
if (vlc_symbols) **dcecs_vlc_symbols**	8
if (vlc_bits) **dcecs_vlc_bits**	4
if (inter_blocks) **dcecs_inter_blocks**	8
if (inter4v_blocks) **dcecs_inter4v_blocks**	8
if (apm) **dcecs_apm**	8
if (npm) **dcecs_npm**	8
if (forw_back_mc_q) **dcecs_forw_back_mc_q**	8
if (halfpel2) **dcecs_halfpel2**	8
if (halfpel4) **dcecs_halfpel4**	8
if (interpolate_mc_q) **dcecs_interpolate_mc_q**	8
}	
if (vop_coding_type==?S?){	
if (intra_blocks) **dcecs_intra_blocks**	8
if (not_coded_blocks) **dcecs_not_coded_blocks**	8
if (dct_coefs) **dcecs_dct_coefs**	8
if (dct_lines) **dcecs_dct_lines**	8
if (vlc_symbols) **dcecs_vlc_symbols**	8
if (vlc_bits) **dcecs_vlc_bits**	4
if (inter_blocks) **dcecs_inter_blocks**	8
if (inter4v_blocks) **dcecs_inter4v_blocks**	8
if (apm) **dcecs_apm**	8
if (npm) **dcecs_npm**	8

if (forw_back_mc_q) **dcecs_forw_back_q**	8
if (halfpel2) **dcecs_halfpel2**	8
if (halfpel4) **dcecs_halfpel4**	8
if (interpolate_mc_q) **dcecs_interpolate_mc_q**	8
}	
}	
}	

2. Video Plane with Short Header

video_plane_with_short_header() {	**No. of bi**
short_video_start_marker	22
temporal_reference	8
marker_bit	1
zero_bit	1
split_screen_indicator	1
document_camera_indicator	1
full_picture_freeze_release	1
source_format	3
picture_coding_type	1
four_reserved_zero_bits	4
vop_quant	5
zero_bit	1
do{	
pei	1
if (pei == "1")	
psupp	8
} while (pei == "1")	

gob_number = 0	
for(i=0; i<num_gobs_in_vop; i++)	
gob_layer()	
if(next_bits() == short_video_end_marker)	
short_video _end_marker	22
while(!bytealigned())	
zero_bit	1
}	

gob_layer() {	**No. of bit**
gob_header_empty = 1	
if(gob_number != 0) {	
if (next_bits() == gob_resync_marker) {	
gob_header_empty = 0	
gob_resync_marker	17
gob_number	5
gob_frame_id	2
quant_scale	5
}	
}	
for(i=0; i<num_macroblocks_in_gob; i++)	
macroblock()	
if(next_bits() != gob_resync_marker && nextbits_bytealigned() == gob_resync_marker)	
while(!bytealigned())	
zero_bit	1
gob_number++	
}	

video_packet_header() {	**No. of bit**
next_resync_marker()	
resync_marker	17-23
macroblock_number	1-14
if (video_object_layer_shape != "binary only")	
quant_scale	5
header_extension_code	1

if (header_extension_code) {	
do {	
modulo_time_base	1
} while (modulo_time_base != ?0?)	
marker_bit	1
vop_time_increment	1-16
marker_bit	1
vop_coding_type	2
if (video_object_layer_shape != "binary only") {	
intra_dc_vlc_thr	3
if (vop_coding_type != "I")	
vop_fcode_forward	3
if (vop_coding_type == "B")	
vop_fcode_backward	3
}	
}	
}	

3. Motion Shape Texture

motion_shape_texture() {	**No. of bi**
if (data_partitioned)	
data_partitioned_motion _shape_texture()	
else	
combined_motion_shape_texture()	
}	

combined_motion_shape_texture() {	**No. of bi**
do{	
macroblock()	
} while (nextbits_bytealigned() != resync_marker && nextbits_bytealigned() != ?000 0000 0000 0000 0000 0000?)	
}	

data_partitioned_motion_shape_texture() {	**No. of bi**
if (vop_coding_type == "I") {	
data_partitioned_i_vop()	
} else if (vop_coding_type == "P") {	
data_partitioned_p_vop()	
} else if (vop_coding_type == "B") {	
combined_motion_shape_texture()	
}	
NOTE Data partitioning is not supported in B-VOPs.	

data_partitioned_i_vop() {	**No. of bi**
do{	
if (video_object_layer_shape != "rectangular"){	
bab_type	1-3
if (bab_type >= 4) {	
if (!change_conv_rate_disable) **conv_ratio**	1-2
scan_type	1
binary_arithmetic_code()	
}	
}	

if (!transparent_mb()) {	
mcbpc	1-9
if (mb_type == 4)	
dquant	2
if (use_intra_dc_vlc) {	
for (j = 0; j < 4; j++) {	
if (!transparent_block(j)) {	
dct_dc_size_luminance	2-11
if (dct_dc_size_luminance > 0)	
dct_dc_differential	1-12
if (dct_dc_size_luminance > 8)	
marker_bit	1
}	
}	
for (j = 0; j < 2; j++) {	
dct_dc_size_chrominance	2-12
if (dct_dc_size_chrominance > 0)	
dct_dc_differential	1-12
if (dct_dc_size_chrominance > 8)	
marker_bit	1
}	
}	
}	
} while (next_bits() != dc_marker)	
dc_marker /* 110 1011 0000 0000 0001 */	19
for (i = 0; i < mb_in_video_packet; i++) {	

if (!transparent_mb()) {	
ac_pred_flag	1
cbpy	1-6
}	
}	
for (i = 0; i < mb_in_video_packet; i++) {	
if (!transparent_mb()) {	
for (j = 0; j < block_count; j++)	
block(j)	
}	
}	
}	
NOTE The value of block_count is 6 in the 4:2:0 format. The value of alpha_blc	

data_partitioned_p_vop() {	**No. of bi**
do{	
if (video_object_layer_shape != "rectangular"){	
bab_type	1-7
if ((bab_type == 1) || (bab_type == 6)) {	
mvds_x	1-18
mvds_y	1-18
}	
if (bab_type >= 4) {	
if (!change_conv_rate_disable) **conv_ratio**	1-2
scan_type	1
binary_arithmetic_code()	

}	
}	
if (!transparent_mb()) {	
not_coded	1
if (!not_coded) {	
mcbpc	1-9
if (derived_mb_type < 3)	
motion_coding("forward", derived_mb_type)	
}	
}	
} while (next_bits() != motion_marker)	
motion_marker /* 1 1111 0000 0000 0001 */	17
for (i = 0; i < mb_in_video_packet; i++) {	
if (!transparent_mb()) {	
if (!not_coded){	
if (derived_mb_type >= 3)	
ac_pred_flag	1
cbpy	1-6
if (derived_mb_type == 1 || derived_mb_type == 4)	
dquant	2
if (derived_mb_type >= 3 && use_intra_dc_vlc) {	
for (j = 0; j < 4; j++) {	
if (!transparent_block(j)) {	
dct_dc_size_luminance	2-11
if (dct_dc_size_luminance > 0)	
dct_dc_differential	1-12

if (dct_dc_size_luminance > 8)	
marker_bit	1
}	
}	
for (j = 0; j < 2; j++) {	
dct_dc_size_chrominance	2-12
if (dct_dc_size_chrominance > 0)	
dct_dc_differential	1-12
if (dct_dc_size_chrominance > 8)	
marker_bit	1
}	
}	
}	
}	
}	
for (i = 0; i < mb_in_video_packet; i++) {	
if (!transparent_mb()) {	
if (! not_coded) {	
for (j = 0; j < block_count; j++)	
block(j)	
}	
}	
}	
}	
NOTE The value of block_count is 6 in the 4:2:0 format. The value of alpha_block_count is 4.	

motion_coding(mode, type_of_mb) {	**No. of bit**
motion_vector(mode)	
if (type_of_mb == 2) {	
for (i = 0; i < 3; i++)	
motion_vector(mode)	
}	
}	

4. **Sprite coding**

decode_sprite_piece() {	**No. of bits**	**M**
piece_quant	5	bs
piece_width	9	bs
piece_height	9	bs
marker_bit	1	bs
piece_xoffset	9	bs
piece_yoffset	9	bs
sprite_shape_texture()		
}		

sprite_shape_texture() {	**No. of bits**	**M**
if (sprite_transmit_mode == "piece") {		
for (i=0; i < piece_height; i++) {		
for (j=0; j < piece_width; j++) {		
if (!send_mb()) {		
macroblock()		
}		
}		
}		
}		
if (sprite_transmit_mode == "update") {		
for (i=0; i < piece_height; i++) {		
for (j=0; j < piece_width; j++) {		
macroblock()		
}		
}		
}		
}		

sprite_trajectory() {	**No. of bits**	**M**
for (i=0; i < no_of_sprite_warping_points; i++) {		
warping_mv_code(du[i])		
warping_mv_code(dv[i])		
}		
}		

warping_mv_code(d) {	**No. of bits**	**M**
dmv_length	2-12	uir
if (dmv_length != ?00?)		
dmv_code	1-14	uir
marker_bit	1	bsl
}		

brightness_change_factor() {	**No. of bits**	**M**
brightness_change_factor_size	1-4	uir
brightness_change_factor_code	5-10	uir
}		

6. **Macroblock**

macroblock() {	**No. of bits**	**M**
if (vop_coding_type != "B") {		
if (video_object_layer_shape != "rectangular" && !(sprite_enable && low_latency_sprite_enable && sprite_transmit_mode == "update"))		
mb_binary_shape_coding()		
if (video_object_layer_shape != "binary only") {		
if (!transparent_mb()) {		
if (vop_coding_type != "I" && !(sprite_enable && sprite_transmit_mode == "piece"))		
not_coded	1	bsl
if (!not_coded || vop_coding_type == "I") {		
mcbpc	1-9	vlc

if (!short_video_header && (derived_mb_type == 3 || derived_mb_type == 4))		
ac_pred_flag	1	bsl
if (derived_mb_type != "stuffing")		
cbpy	1-6	vlc
else		
return()		
if (derived_mb_type == 1 || derived_mb_type == 4)		
dquant	2	bsl
if (interlaced)		
interlaced_information()		

if (!(ref_select_code==?11? && scalability)
&& vop_coding_type != "S") {		

if (derived_mb_type == 0 ||
derived_mb_type == 1) {		

motion_vector("forward")		
if (interlaced && field_prediction)		
motion_vector("forward")		
}		
if (derived_mb_type == 2) {		
for (j=0; j < 4; j++)		
if (!transparent_block(j))		
motion_vector("forward")		
}		

}		
for (i = 0; i < block_count; i++)		
if(!transparent_block(i))		
block(i)		
}		
}		
}		
}		
else {		
if (video_object_layer_shape != "rectangular")		
mb_binary_shape_coding()		

if ((co_located_not_coded != 1
|| (scalability && (ref_select_code != ’11’
|| enhancement_type == 1)))
&& video_object_layer_shape != "binary only") {		

if (!transparent_mb()) {		
modb	1-2	vlc
if (modb != ?1?) {		
mb_type	1-4	vlc
if (modb == ?00?)		
cbpb	3-6	vlc
if (ref_select_code != ?00? || !scalability) {		
if (mb_type != "1" && cbpb!=0)		
dbquant	1-2	vlc
if (interlaced)		
interlaced_information()		

if (mb_type == ?01? ||
mb_type == ?0001?) {		

motion_vector("forward")		
if (interlaced && field_prediction)		
motion_vector("forward")		
}		
if (mb_type == ?01? || mb_type == ?001?) {		
motion_vector("backward")		
if (interlaced && field_prediction)		
motion_vector("backward")		
}		
if (mb_type == "1")		
motion_vector("direct")		
}		

if (ref_select_code == ?00? && scalability &&
cbpb !=0) {		

dbquant	1-2	vlc
if (mb_type == ?01? || mb_type == ?1?)		
motion_vector("forward")		
}		
for (i = 0; i < block_count; i++)		
if(!transparent_block(i))		
block(i)		
}		
}		
}		

}		

if(video_object_layer_shape=="grayscale"
&& !transparent_mb()) {		

if(vop_coding_type=="I" || (vop_coding_type=="P"
&& !not_coded
&& (derived_mb_type==3 || derived_mb_type==4))) {		

coda_i	1	bsl
if(coda_i=="coded") {		
ac_pred_flag_alpha	1	bsl
cbpa	1-6	vlc
for(i=0;i<alpha_block_count;i++)		
if(!transparent_block())		
alpha_block(i)		
}		
} else { /* P or B macroblock */		

if(vop_coding_type == "P"
|| co_located_not_coded != 1) {		

coda_pb	1-2	vlc
if(coda_pb=="coded") {		
cbpa	1-6	vlc
for(i=0;i<alpha_block_count;i++)		
if(!transparent_block())		
alpha_block(i)		
}		
}		
}		

}		
}		
NOTE The value of block_count is 6 in the 4:2:0 format. The value of alpha_block_count is 4.		

1. MB Binary Shape Coding

mb_binary_shape_coding() {	**No. of bi**
bab_type	1-7
if ((vop_coding_type == ?P?) || (vop_coding_type == ?B?)) {	
if ((bab_type==1) || (bab_type == 6)) {	
mvds_x	1-18
mvds_y	1-18
}	
}	
if (bab_type >=4) {	
if (!change_conv_ratio_disable)	
conv_ratio	1-2
scan_type	1
binary_arithmetic_code()	
}	
}	

backward_shape () {	**No. of bi**
for(i=0; i<backward_shape_height/16; i++)	
for(j=0; j<backward_shape_width/16; j++) {	
bab_type	1-3
if (bab_type >=4) {	
if (!change_conv_ratio_disable)	
conv_ratio	1-2
scan_type	1
binary_arithmetic_code()	
}	
}	
}	

forward_shape () {	**No. of bi**
for(i=0; i<forward_shape_height/16; i++)	
for(j=0; j<forward_shape_width/16; j++) {	
bab_type	1-3
if (bab_type >=4) {	
if (!change_conv_ratio_disable)	
conv_ratio	1-2
scan_type	1
binary_arithmetic_code()	
}	
}	
}	

2. Motion vector

motion_vector (mode) {	**No. of bi**
if (mode == ?direct") {	
horizontal_mv_data	1-13
vertical_mv_data	1-13
}	
else if (mode == ?forward") {	
horizontal_mv_data	1-13
if ((vop_fcode_forward != 1)&&(horizontal_mv_data != 0))	
horizontal_mv_residual	1-6
vertical_mv_data	1-13
if ((vop_fcode_forward != 1)&&(vertical_mv_data != 0))	
vertical_mv_residual	1-6
}	
else if (mode == ?backward") {	
horizontal_mv_data	1-13
if ((vop_fcode_backward != 1)&&(horizontal_mv_data != 0))	
horizontal_mv_residual	1-6
vertical_mv_data	1-13
if ((vop_fcode_backward != 1)&&(vertical_mv_data != 0))	
vertical_mv_residual	1-6
}	
}	

3. Interlaced Information

interlaced_information( ) {	**No. of bits**	**M**
if ((derived_mb_type == 3) || (derived_mb_type == 4) || (cbp != 0))		
dct_type	1	bsl

if (((vop_coding_type == "P") &&
((derived_mb_type == 0) || (derived_mb_type == 1))) ||
((vop_coding_type == "B") && (mb_type != "1"))) {		

field_prediction	1	bsl
if (field_prediction) {		

if (vop_coding_type == "P" ||
(vop_coding_type == "B" &&
mb_type != "001")) {		

forward_top_field_reference	1	bsl
forward_bottom_field_reference	1	bsl
}		

if ((vop_coding_type == "B") &&
(mb_type != "0001")) {		

backward_top_field_reference	1	bsl
backward_bottom_field_reference	1	bsl
}		
}		
}		
}		

7. **Block**

The detailed syntax for the term "DCT coefficient" is fully described in clause 7.

block(i) {	**No. of bits**	**M**
last = 0		

if(!data_partitioned &&
(derived_mb_type == 3 || derived_mb_type == 4)) {		

if(short_video_header == 1)		
intra_dc_coefficient	8	uir
else if (use_intra_dc_vlc == 1) {		
if (i<4) {		
dct_dc_size_luminance	2-11	vlc
if(dct_dc_size_luminance != 0)		
dct_dc_differential	1-12	vlc
if (dct_dc_size_luminance > 8)		
marker_bit	1	bsl
} else {		
dct_dc_size_chrominance	2-12	vlc
if(dct_dc_size_chrominance !=0)		
dct_dc_differential	1-12	vlc
if (dct_dc_size_chrominance > 8)		
marker_bit	1	bsl
}		
}		
}		
if (pattern_code[i])		
while (! last)		
DCT coefficient	3-24	vlc
}		
NOTE "last" is defined to be the LAST flag resulting from reading the most recent DCT coefficient.		

1. **Alpha Block**

The syntax for DCT coefficient decoding is the same as for block(i) in subclause 6.2.7.

alpha_block(i) {	**No. of bits**	**M**
last = 0		

if(!data_partitioned &&
(vop_coding_type == "I" ||
(vop_coding_type == "P" && !not_coded &&
(derived_mb_type == 3 || derived_mb_type == 4)))) {		

dct_dc_size_alpha	2-11	vlc
if(dct_dc_size_alpha != 0)		
dct_dc_differential	1-12	vlc
if (dct_dc_size_alpha > 8)		
marker_bit	1	bsl
}		
if (pattern_code[i])		
while (! last)		
DCT coefficient	3-24	vlc
}		
NOTE "last" is defined to be the LAST flag resulting from reading the most recent DCT coefficient**.**		

8. Still Texture Object

StillTextureObject() {	**No. of bits**	**Mn**
still_texture_object_start_code	32	bslb
texture_object_id	16	uims
marker_bit	1	bslb
wavelet_filter_type	1	uims
wavelet_download	1	uims
wavelet_decomposition_levels	4	uims
scan_direction	1	bslb

start_code_enable	1	bslb
texture_object_layer_shape	2	uims
quantization_type	2	uims
if (quantization_type == 2) {		
spatial_scalability_levels	4	uims

if (spatial_scalability_levels !=
wavelet_decomposition_levels) {		

use_default_spatial_scalability	1	uims
if (use_default_spatial_layer_size == 0)		
for (i=0; i<spatial_scalability_levels - 1; i++)		
wavelet_layer_index	4	
}		
if (wavelet_download == "1"){		
uniform_wavelet_filter	1	uims
if (uniform_wavelet_filter == "1")		
download_wavelet_filters()		
else		
for (i=0; i<wavelet_decomposition_levels; i++)		
download_wavelet_filters( )		
}		
wavelet_stuffing	3	uims
if(texture_object_layer_shape == "00"){		
texture_object_layer_width	15	uims
marker_bit	1	bslb
texture_object_layer_height	15	uims
marker_bit	1	bslb

}		
else {		
horizontal_ref	15	imsb
marker_bit	1	bslb
vertical_ref	15	imsb
marker_bit	1	bslb
object_width	15	uims
marker_bit	1	bslb
object_height	15	uims
marker_bit	1	bslb
shape_object_decoding ( )		
}		
/* configuration information precedes this point; elementary stream data follows. See annex K */		
for (color = "y", "u", "v"){		
wavelet_dc_decode()		
}		
if(quantization_type == 1){		
TextureLayerSQ ( )		
}		
else if (quantization_type == 2){		
if (start_code_enable == 1) {		
do {		
TextureSpatialLayerMQ ( )		
} while (next_bits() == texture_spatial_layer_start_code)		
} else {		
for (i =0; i<spatial_scalability_levels; i++)		

TextureSpatialLayerMQNSC ( )		
}		
}		
else if (quantization_type == 3){		
for (color = "y", "u", "v")		
do{		
quant_byte	8	uims
} while(quant_byte >>7)		
max_bitplanes	5	uims
if (scan_direction == 0) {		
do {		
TextureSNRLayerBQ ( )		
} while (next_bits() == texture_snr_layer_start_code)		
} else {		
do {		
TextureSpatialLayerBQ ( )		
} while (next_bits() == texture_spatial_layer_start_code)		
}		
}		
}		

1. TextureLayerSQ

TextureLayerSQ() {	**No. of bit**
if (scan_direction == 0) {	
for ("y", "u", "v") {	
do {	

quant_byte	8
} while (quant_byte >> 7)	
for (i=0; i<wavelet_decomposition_levels; i++)	
if (i!=0 || color!= "u","v") {	
max_bitplane[i]	5
if ((i+1)%4==0)	
marker_bit	1
}	
}	
for (i = 0; i<tree_blocks; i++)	
for (color = "y", "u", "v")	
arith_decode_highbands_td()	
} else {	
if (start_code_enable) {	
do {	
TextureSpatialLayerSQ()	
} while (next_bits() == texture_spatial_layer_start_code)	
} else {	
for (i = 0; i< wavelet_decomposition_levels; i++)	
TextureSpatialLayerSQNSC()	
}	
}	
}	

2. TextureSpatialLayerSQ

TextureSpatialLayerSQ() {	**No. of bit**
texture_spatial_layer_start_code	32
texture_spatial_layer_id	5
TextureSpatialLayerSQNSC()	
}	

3. TextureSpatialLayerSQNSC

TextureSpatialLayerSQNSC() {	**No. of bit**
for (color="y","u","v") {	
if ((first_wavelet_layer && color=="y") || (second_wavelet_layer && color=="u","v"))	
do {	
quant_byte	8
} while (quant_byte >> 7)	
if (color =="y")	
max_bitplanes	5
else if (!first_wavelet_layer)	
max_bitplanes	5
}	
arith_decode_highbands_bb()	
}	

4. TextureSpatialLayerMQ

TextureSpatialLayerMQ() {	**No. of bit**
texture_spatial_layer_start_code	32
texture_spatial_layer_id	5
snr_scalability_levels	5
do {	
TextureSNRLayerMQ( )	
} while (next_bits() == texture_snr_layer_start_code)	
}	

5. TextureSpatialLayerMQNSC

TextureSpatialLayerMQNSC() {	**No. of bit**
snr_scalability_levels	5
for (i =0; i<snr_scalability_levels; i++)	
TextureSNRLayerMQNSC ( )	
}	

6. TextureSNRLayerMQ

TextureSNRLayerMQ(){	
texture_snr_layer_start_code	32
texture_snr_layer_id	5
TextureSNRLayerMQNSC()	
}	

7. TextureSNRLayerMQNSC

TextureSNRLayerMQNSC(){	**No. of bit**

if (spatial_scalability_levels == wavelet_decomposition_levels
&& spatial_layer_id == 0) {	

for (color = "y") {		
do {		
quant_byte	8	
} while (quant_byte >> 7)		
for (i=0; i<spatial_layers; i++) {		
max_bitplane[i]	5	
if ((i+1)%4 == 0)		
marker_bit	1	
}		
}		
}		
else {		
for (color="y", "u", "v") {		
do {		
quant_byte	8	
} while (quant_byte >> 7)		
for (i=0; i<spatial_layers; i++) {		
max_bitplane[i]	5	
if ((i+1)%4 == 0)		
marker_bit	1	
}		
}		
}		
if (scan_direction == 0) {		
for (i = 0; i<tree_blocks; i++)		
for (color = "y", "u", "v")		

if (wavelet_decomposition_layer_id != 0 || color != "u", "v")	
arith_decode_highbands_td()	
} else {	
for (i = 0; i< spatial_layers; i++) {	
for (color = "y", "u", "v") {	
if (wavelet_decomposition_layer_id != 0 || color != "u", "v")	
arith_decode_highbands_bb()	
}	
}	
}	
}	

8. TextureSpatialLayerBQ

TextureSpatialLayerBQ() {	**No. of bit**
texture_spatial_layer_start_code	32
texture_spatial_layer_id	5
for (i=0; i<max_bitplanes; i++) {	
texture_snr_layer_start_code	32
texture_snr_layer_id	5
TextureBitPlaneBQ()	
next_start_code()	
}	
}	

9. TextureBitPlaneBQ

TextureBitPlaneBQ () {	**No. of bit**

for (color = "y", "u", "v")	
if (wavelet_decomposition_layer_id == 0){	
all_nonzero[color]	1
if (all_nonzero[color] == 0) {	
all_zero[color]	1
if (all_zero[color]==0) {	
lh_zero[color]	1
hl_zero[color]	1
hh_zero[color]	1
}	
}	
}	
if (wavelet_decomposition_layer_id != 0 ||color != "u", "v"){	
if(all_nonzero[color]==1 || all_zero[color]==0){	
if (scan_direction == 0)	
arith_decode_highbands_bilevel_bb()	
else	
arith_decode_highbands_bilevel_td()	
}	
}	
}	
}	

10. TextureSNRLayerBQ

TextureSNRLayerBQ() {	**No. of bit**
texture_snr_layer_start_code	32
texture_snr_layer_id	5
for (i=0; i<wavelet_decomposition_levels; i++) {	
texture_spatial_layer_start_code	32
texture_spatial_layer_id	5
TextureBitPlaneBQ()	
next_start_code ( )	
}	
}	

11. DownloadWaveletFilters

download_wavelet_filters( ){	**No. of bit**
lowpass_filter_length	4
highpass_filter_length	4
do{	
if (wavelet_filter_type == 0) {	
filter_tap_integer	16
marker_bit	1
} else {	
filter_tap_float_high	16
marker_bit	1
filter_tap_float_low	16
marker_bit	1
}	
} while (lowpass_filter_length--)	
do{	

if (wavelet_filter_type == 0){	
filter_tap_integer	16
marker_bit	1
} else {	
filter_tap_float_high	16
marker_bit	1
filter_tap_float_low	16
marker_bit	1
}	
} while (highpass_filter_length--)	
if (wavelet_filter_type == 0) {	
integer_scale	16
marker_bit	
}	
}	

12. Wavelet dc decode

wavelet_dc_decode() {	**No. of bit**
mean	8
do{	
quant_dc_byte	8
} while(quant_dc_byte >>7)	
do{	
band_offset_byte	8
} while (band_offset_byte >>7)	
do{	
band_max_byte	8
} while (band_max_byte >>7)	
arith_decode_dc()	
}	

13. Wavelet higher bands decode

wavelet_ higher_bands_decode() {	**No. of bit**
do{	
root_max_alphabet_byte	8
} while (root_max_alphabet_byte >>7)	
marker_bit	1
do{	
valz_max_alphabet_byte	8
} while (valz_max_alphabet_byte >>7)	
do{	
valnz_max_alphabet_byte	8
} while (valnz_max_alphabet_byte >>7)	
arith_decode_highbands()	
}	

14. Shape Object Decoding

shape_object_decoding() {	**No. of bits**	**M**
change_conv_ratio_disable	1	bsl
sto_constant_alpha	1	bsl
if (sto_constant_alpha)		
sto_constant_alpha_value	8	bsl
for (i=0; i<((object_width+15)/16)*((object_height+15)/16); i++){		
bab_type	1-2	vlc
if (bab_type ==4) {		
if (!change_conv_ratio_disable)		
conv_ratio	1-2	vlc
scan_type	1	bsl
binary_arithmetic_decode()		
}		
}		
}		

9. **Mesh Object**

MeshObject() {	**No. of bits**	**M**
mesh_object_start_code	32	bsl
do{		
MeshObjectPlane()		
} while (next_bits_bytealigned() == mesh_object_plane_start_code || next_bits_bytealigned() != ?0000 0000 0000 0000 0000 0001?)		
}		

1. Mesh Object Plane

MeshObjectPlane() {	**No. of bit**
MeshObjectPlaneHeader()	
MeshObjectPlaneData()	
}	

MeshObjectPlaneHeader() {	**No. of bit**
if (next_bits_bytealigned()==?0000 0000 0000 0000 0000 0001?){	
next_start_code()	
mesh_object_plane_start_code	32
}	
is_intra	1
mesh_mask	1
temporal_header()	
}	

MeshObjectPlaneData() {	**No. of bit**
if (mesh_mask == 1) {	
if (is_intra == 1)	
mesh_geometry()	
else	
mesh_motion()	
}	
}	

2. Mesh geometry

mesh_geometry() {	**No. of bit**

mesh_type _code	2
if (mesh_type_code == ?01?) {	
nr_of_mesh_nodes_hor	10
nr_of_mesh_nodes_vert	10
marker_bit	1
mesh_rect_size_hor	8
mesh_rect_size_vert	8
triangle_split_code	2
}	
else if (mesh_type_code == ?10?) {	
nr_of_mesh_nodes	16
marker_bit	1
nr_of_boundary_nodes	10
marker_bit	1
node0_x	13
marker_bit	1
node0_y	13
marker_bit	1
for (n=1; n < nr_of_mesh_nodes; n++) {	
delta_x_len_vlc	2-12
if (delta_x_len_vlc)	
delta_x	1-14
delta_y_len_vlc	2-12
if (delta_y_len_vlc)	
delta_y	1-14
}	
}	

}	

3. **Mesh motion**

mesh_motion() {	**No. of bits**	**M**
motion_range_code	3	bsl
for (n=0; n <nr_of_mesh_nodes; n++) {		
node_motion_vector_flag	1	bsl
if (node_motion_vector_flag == ?0?) {		
delta_mv_x_vlc	1-13	vlc
if ((motion_range_code != 1) && (delta_mv_x_vlc != 0))		
delta_mv_x_res	1-6	uir
delta_mv_y_vlc	1-13	vlc
if ((motion_range_code != 1) && (delta_mv_y_vlc != 0))		
delta_mv_y_res	1-6	uir
}		
}		
}		

10. **Face Object**

fba_object() {	**No. of bits**	**M**
face_object_start_code	32	bsl
do {		
fba_object_plane()		
} while(!((nextbits_bytealigned() == ?000 0000 0000 0000 0000 0000?) && (nextbits_bytealigned() != face_object_plane_start_code)))		
}		

1. **Face Object Plane**

fba_object_plane() {	**No. of bit**
fba_object_plane_header()	
fba_object_plane_data()	
}	

fba_object_plane_header() {	**No. of bit**
if (nextbits_bytealigned()==?000 0000 0000 0000 0000 0000?){	
next_start_code()	
fba_object_plane_start_code	32
}	
is_intra	1
fba_object_mask	2
temporal_header()	
}	

fba_object_plane_data() {	**No. of bit**
if(fba_object_mask &?01?) {	
if(is_intra) {	
fap_quant	5
for (group_number = 1; group_number <= 10; group_number++) {	
marker_bit	1
fap_mask_type	2
if(fap_mask_type == ?01?|| fap_mask_type == ?10?)	
fap_group_mask [group_number]	2-16
}	

fba_suggested_gender	1
fba_object_coding_type	1
if(fba_object_coding_type == 0) {	
is_i_new_max	1
is_i_new_min	1
is_p_new_max	1
is_p_new_min	1
decode_new_minmax()	
decode_ifap()	
}	
if(fba_object_coding_type == 1)	
decode_i_segment()	
}	
else {	
if(fba_object_coding_type == 0)	
decode_pfap()	
if(fba_object_coding_type == 1)	
decode_p_segment()	
}	
}	
}	

temporal_header() {	**No. of bit**
if (is_intra) {	
is_frame_rate	1
if(is_frame_rate)	
decode_frame_rate()	
is_time_code	1
if (is_time_code)	
time_code	18
}	
skip_frames	1
if(skip_frames)	
decode_skip_frames()	
}	

2. **Decode frame rate and skip frames**

decode_frame_rate(){	**No. of bit**
frame_rate	8
seconds	4
frequency_offset	1
}	

decode_skip_frames(){	**No. of bit**
do{	
number_of_frames_to_skip	4
} while (number_of_frames_to_skip = "1111")	
}	

3. **Decode new minmax**

decode_new_minmax() {	**No. of bi**
if (is_i_new_max) {	
for (group_number = 2, j=0, group_number <= 10, group_number++)	
for (i=0; i < NFAP[group_number]; i++, j++) {	
if (!(i & 0x3))	
marker_bit	1
if (fap_group_mask[group_number] & (1 <<i))	
i_new_max[j]	5
}	
if (is_i_new_min) {	
for (group_number = 2, j=0, group_number <= 10, group_number++)	
for (i=0; i < NFAP[group_number]; i++, j++) {	
if (!(i & 0x3))	
marker_bit	1
if (fap_group_mask[group_number] & (1 <<i))	
i_new_min[j]	5
}	
if (is_p_new_max) {	
for (group_number = 2, j=0, group_number <= 10, group_number++)	
for (i=0; i < NFAP[group_number]; i++, j++) {	
if (!(i & 0x3))	
marker_bit	1
if (fap_group_mask[group_number] & (1 <<i))	
p_new_max[j]	5

}	
if (is_p_new_min) {	
for (group_number = 2, j=0, group_number <= 10, group_number++)	
for (i=0; i < NFAP[group_number]; i++, j++) {	
if (!(i & 0x3))	
marker_bit	1
if (fap_group_mask[group_number] & (1 <<i))	
p_new_min[j]	5
}	
}	
}	

4. **Decode ifap**

decode_ifap(){	**No. of bit**
for (group_number = 1, j=0; group_number <= 10; group_number++) {	
if (group_number == 1) {	
if(fap_group_mask[1] & 0x1)	
decode_viseme()	
if(fap_group_mask[1] & 0x2)	
decode_expression()	
} else {	
for (i= 0; i<NFAP[group_number]; i++, j++) {	
if(fap_group_mask[group_number] & (1 << i)) {	
aa_decode(ifap_Q[j],ifap_cum_freq[j])	
}	
}	
}	
}	
}	

5. **Decode pfap**

decode_pfap(){	**No. of bit**
for (group_number = 1, j=0; group_number <= 10; group_number++) {	
if (group_number == 1) {	
if(fap_group_mask[1] & 0x1)	
decode_viseme()	
if(fap_group_mask[1] & 0x2)	
decode_expression()	
} else {	
for (i= 0; i<NFAP[group_number]; i++, j++) {	
if(fap_group_mask[group_number] & (1 << i)) {	
aa_decode(pfap_diff[j], pfap_cum_freq[j])	
}	
}	
}	
}	
}	

6. **Decode viseme and expression**

decode_viseme() {	**No. of bit**
aa_decode(viseme_select1Q, viseme_select1_cum_freq)	
aa_decode(viseme_select2Q, viseme_select2_cum_freq)	
aa_decode(viseme_blendQ, viseme_blend_cum_freq)	
viseme_def	1
}	

decode_expression() {	**No. of bit**
aa_decode(expression_select1Q, expression_select1_cum_freq)	
aa_decode(expression_intensity1Q, expression_intensity1_cum_freq)	
aa_decode(expression_select2Q, expression_select2_cum_freq)	
aa_decode(expression_intensity2Q, expression_intensity2_cum_freq)	
aa_decode(expression_blendQ, expression_blend_cum_freq)	
init_face	1
expression_def	1
}	

7. **Face Object Plane Group**

face_object_plane_group() {	**No. of bit**
face_object_plane_start_code	32
is_intra	1
if (is_intra) {	
face_paramset_mask	2
is_frame_rate	1
if(is_frame_rate)	
decode_frame_rate()	
is_time_code	1
if(is_time_code)	
time_code	18
skip_frames	1
if(skip_frames)	
decode_skip_frames()	

if(face_paramset_mask ==?01?) {	
fap_quant_index	5
for (group_number = 1 to 10) {	
marker_bit	1
fap_mask_type	2
if(fap_mask_type == ?01?|| fap_mask_type == ?10?)	
fap_group_mask[group_number]	2-16
}	
decode_i_segment()	
} else {	
face_object_group_prediction()	
}	
next_start_code()	
}	

8. **Face Object Group Prediction**

face_object_group_prediction() {	**No. of bit**
skip_frames	1
if(skip_frames)	
decode_skip_frames()	
if(face_paramset_mask ==?01?) {	
decode_p_segment()	
}	
}	

9. **Decode i_segment**

decode_i_segment(){	**No. of bi**
for (group_number= 1, j=0; group_number<= 10; group_number++) {	
if (group_number == 1) {	
if(fap_group_mask[1] & 0x1)	
decode_i_viseme_segment()	
if(fap_group_mask[1] & 0x2)	
decode_i_expression_segment()	
} else {	
for(i=0; i<NFAP[group_number]; i++, j++) {	
if(fap_group_mask[group_number] & (1 << i)) {	
decode_i_dc(dc_Q[j])	
decode_ac(ac_Q[j])	
}	
}	
}	
}	
}	

10. **Decode p_segment**

decode_p_segment(){	**No. of bit**
for (group_number = 1, j=0; group_number <= 10; group_number++) {	
if (group_number == 1) {	
if(fap_group_mask[1] & 0x1)	
decode_p_viseme_segment()	
if(fap_group_mask[1] & 0x2)	
decode_p_expression_segment()	
} else {	
for (i=0; i<NFAP[group_number]; i++, j++) {	
If(fap_group_mask[group_number] & (1 << i)) {	
decode_p_dc(dc_Q[j])	
decode_ac(ac_Q[j])	
}	
}	
}	
}	
}	

11. **Decode viseme and expression**

decode_i_viseme_segment(){	**No. of bits**	**Mnemon**
viseme_segment_select1q[0]	4	uimsbf
viseme_segment_select2q[0]	4	uimsbf
viseme_segment_blendq[0]	6	uimsbf
viseme_segment_def[0]	1	bslbf
for (k=1; k<16, k++) {		
viseme_segment_select1q_diff[k]		vlclbf
viseme_segment_select2q_diff[k]		vlclbf
viseme_segment_blendq_diff[k]		vlclbf
viseme_segment_def[k]	1	bslbf
}		
}		

decode_p _viseme_segment(){	**No. of bits**	**Mnemon**
for (k=0; k<16, k++) {		
viseme_segment_select1q_diff[k]		vlclbf
viseme_segment_select2q_diff[k]		vlclbf
viseme_segment_blendq_diff[k]		vlclbf
viseme_segment_def[k]	1	bslbf
}		
}		

decode_i_expression_segment(){	**No. of bits**	**Mnemon**
expression_segment_select1q[0]	4	uimsbf
expression_segment_select2q[0]	4	uimsbf
expression_segment_intensity1q[0]	6	uimsbf
expression_segment_intensity2q[0]	6	uimsbf
expression_segment_init_face[0]	1	bslbf
expression_segment_def[0]	1	bslbf
for (k=1; k<16, k++) {		
expression_segment_select1q_diff[k]		vlclbf
expression_segment_select2q_diff[k]		vlclbf
expression_segment_intensity1q_diff[k]		vlclbf
expression_segment_intensity2q_diff[k]		vlclbf
expression_segment_init_face[k]	1	bslbf
expression_segment_def[k]	1	bslbf
}		
}		

decode_p _expression_segment(){	**No. of bits**	**Mnemon**
for (k=0; k<16, k++) {		
expression_segment_select1q_diff[k]		vlclbf
expression_segment_select2q_diff[k]		vlclbf
expression_segment_intensity1q_diff[k]		vlclbf
expression_segment_intensity2q_diff[k]		vlclbf
expression_segment_init_face[k]	1	bslbf
expression_segment_def[k]	1	bslbf
}		
}		

decode_i_dc(dc_q) {	**No. of bits**	**Mnemon**
dc_q	16	simsbf
if(dc_q == -256*128)		
dc_q	31	simsbf
}		

decode_p_dc(dc_q_diff) {	**No. of bits**	**Mnemon**
dc_q_diff		vlclbf
dc_q_diff = dc_q_diff- 256		
if(dc_q_diff == -256)		
dc_q_diff	16	simsbf
if(dc_Q == 0-256*128)		
dc_q_diff	32	simsbf
}		

decode_ac(ac_Q[i]) {	**No. of bits**	**Mnemon**
this = 0		
next = 0		
while(next < 15) {		
count_of_runs		vlclbf
if (count_of_runs == 15)		
next = 16		
else {		
next = this+1+count_of_runs		
for (n=this+1; n<next; n++)		
ac_q[i][n] = 0		
ac_q[i][next]		vlclbf
if(ac_q[i][next] == 256)		
decode_i_dc(ac_q[i][next])		
else		
ac_q[i][next] = ac_q[i][next]-256		
this = next		
}		
}		
}		

2. Visual bitstream semantics

1. Semantic rules for higher syntactic structures

This subclause details the rules that govern the way in which the higher level syntactic elements may be combined together to produce a legal bitstream. Subsequent subclauses detail the semantic meaning of all fields in the video bitstream.

2. Visual Object Sequence and Visual Object

visual_object_sequence_start_code : The visual_object_sequence_start_code is the bit string ?000001B0? in hexadecimal. It initiates a visual session.

profile_and_level_indication: This is an 8-bit integer used to signal the profile and level identification. The meaning of the bits is given in Table G-1.

visual_object_sequence_end_code : The visual_object_sequence_end_code is the bit string ?000001B1? in hexadecimal. It terminates a visual session.

visual_object_start_code: The visual_object_start_code is the bit string ?000001B5? in hexadecimal. It initiates a visual object.

is_visual_object_identifier : This is a 1-bit code which when set to ?1? indicates that version identification and priority is specified for the visual object. When set to ?0?, no version identification or priority needs to be specified.

visual_object_verid: This is a 4-bit code which identifies the version number of the visual object. Its meaning is defined in Table 6-4.

Table -4 -- Meaning of visual_object_verid

visual_object_verid	**Meaning**
0000	reserved
0001	ISO/IEC 14496-2
0010 - 1111	reserved

visual_object_priority: This is a 3-bit code which specifies the priority of the visual object. It takes values between 1 and 7, with 1 representing the highest priority and 7, the lowest priority. The value of zero is reserved.

visual_object_type : The visual_object_type is a 4-bit code given in Table 6-5 which identifies the type of the visual object.

Table -5 -- Meaning of visual object type

code	**visual object type**
0000	reserved
0001	video ID
0010	still texture ID
0011	mesh ID
0100	face ID
0101	reserved
:	:
:	:
1111	reserved

video_object_start_code: The video_object_start_code is a string of 32 bits. The first 27 bits are ?0000 0000 0000 0000 0000 0001 000? in binary and the last 5-bits represent one of the values in the range of ?00000? to ?11111? in binary. The video_object_start_code marks a new video object.

video_object_id: This is given by the last 5-bits of the video_object_start_code. The video_object_id uniquely identifies a video object.

video_signal_type : A flag which if set to ?1? indicates the presence of video_signal_type information.

video_format: This is a three bit integer indicating the representation of the pictures before being coded in accordance with this part of ISO/IEC 14496. Its meaning is defined in Table 6-6. If the video_signal_type() is not present in the bitstream then the video format may be assumed to be "Unspecified video format".

Table -6 -- Meaning of video_format

video_format	Meaning
000	Component
001	PAL
010	NTSC
011	SECAM
100	MAC
101	Unspecified video format
110	Reserved
111	Reserved

video_range: This one-bit flag indicates the black level and range of the luminance and chrominance signals.

colour_description : A flag which if set to ?1? indicates the presence of colour_primaries, transfer_characteristics and matrix_coefficients in the bitstream.

colour_primaries: This 8-bit integer describes the chromaticity coordinates of the source primaries, and is defined in Table 6-7.

Table -7 -- Colour Primaries

Value	Primaries

0	(forbidden)
1	Recommendation ITU-R BT.709 primary x y green 0,300 0,600 blue 0,150 0,060 red 0,640 0,330 white D65 0,3127 0,3290
2	Unspecified Video Image characteristics are unknown.
3	Reserved
4	Recommendation ITU-R BT.470-2 System M primary x y green 0,21 0,71 blue 0,14 0,08 red 0,67 0,33 white C 0,310 0,316
5	Recommendation ITU-R BT.470-2 System B, G primary x y green 0,29 0,60 blue 0,15 0,06 red 0,64 0,33 white D65 0,3127 0,3290
6	SMPTE 170M primary x y green 0,310 0,595 blue 0,155 0,070 red 0,630 0,340 white D65 0,3127 0,3290

7	SMPTE 240M (1987) primary x y green 0,310 0,595 blue 0,155 0,070 red 0,630 0,340 white D65 0,3127 0,3290
8	Generic film (colour filters using Illuminant C) primary x y green 0,243 0,692 (Wratten 58) blue 0,145 0,049 (Wratten 47) red 0,681 0,319 (Wratten 25)
9-255	Reserved

In the case that video_signal_type() is not present in the bitstream or colour_description is zero the chromaticity is assumed to be that corresponding to colour_primaries having the value 1.

transfer_characteristics: This 8-bit integer describes the opto-electronic transfer characteristic of the source picture, and is defined in Table 6-8.

Table -8 -- Transfer Characteristics

Value	**Transfer Characteristic**
0	(forbidden)

1	Recommendation ITU-R BT.709
	V = 1,099 Lc0,45 - 0,099
	for 1³ Lc ³ 0,018
	V = 4,500 Lc
	for 0,018> Lc ³ 0

2	Unspecified Video Image characteristics are unknown.
3	reserved
4	Recommendation ITU-R BT.470-2 System M Assumed display gamma 2,2

5	Recommendation ITU-R BT.470-2 System B, G Assumed display gamma 2,8

6	SMPTE 170M
	V = 1,099 Lc0,45 - 0,099
	for 1³ Lc ³ 0,018
	V = 4,500 Lc
	for 0,018> Lc ³ 0

7	SMPTE 240M (1987)
	V = 1,1115 Lc0,45 - 0,1115
	for Lc³ 0,0228
	V = 4,0 Lc
	for 0,0228> Lc

8	Linear transfer characteristics i.e. V = Lc

9	Logarithmic transfer characteristic (100:1 range)
	V = 1.0-Log10(Lc)/2
	for 1= Lc = 0.01
	V= 0.0
	for 0.01> Lc

10	Logarithmic transfer characteristic (316.22777:1 range)
	V = 1.0-Log10(Lc)/2.5
	for 1= Lc = 0.0031622777
	V= 0.0
	for 0.0031622777> Lc

11-255	reserved

In the case that video_signal_type() is not present in the bitstream or colour_description is zero the transfer characteristics are assumed to be those corresponding to transfer_characteristics having the value 1.

matrix_coefficients: This 8-bit integer describes the matrix coefficients used in deriving luminance and chrominance signals from the green, blue, and red primaries, and is defined in Table 6-9.

In this table:

E?Y is analogue with values between 0 and 1

E?PB and E?PR are analogue between the values -0,5 and 0,5

E?R, E?G and E?B are analogue with values between 0 and 1

White is defined as E?y=1, E?PB=0, E?PR=0; E?R =E?G =E?B=1.

Y, Cb and Cr are related to E?Y, E?PB and E?PR by the following formulae:

if **video_range=0:**

Y = (219 * 2^{n-8} * E?Y) + 2^{n-4}.

Cb = (224 * 2^{n-8} * E?PB) + 2^{n-1}

Cr = (224 * 2^{n-8} * E?PR) + 2^{n-1}

if **video_range=1:**

Y = ((2^{n} -1) * E?Y)

Cb = ((2^{n} -1) * E?PB) + 2^{n-1}

Cr = ((2^{n} -1) * E?PR) + 2^{n-1}

for n bit video.

For example, for 8 bit video,

video_range=0 gives a range of Y from 16 to 235, Cb and Cr from 16 to 240;

video_range=1 gives a range of Y from 0 to 255, Cb and Cr from 0 to 255.

Table -9 -- Matrix Coefficients

Value	**Matrix**
0	(forbidden)
1	Recommendation ITU-R BT.709 E?Y = 0,7152 E?G + 0,0722 E?B + 0,2126 E?R E?PB = -0,386 E?G + 0,500 E?B -0,115 E?R E?PR = -0,454 E?G - 0,046 E?B + 0,500 E?R
2	Unspecified Video Image characteristics are unknown.
3	reserved
4	FCC E?Y = 0,59 E?G + 0,11 E?B + 0,30 E?R E?PB = -0,331 E?G + 0,500 E?B -0,169 E?R E?PR = -0,421 E?G - 0,079 E?B + 0,500 E?R
5	Recommendation ITU-R BT.470-2 System B, G E?Y = 0,587 E?G + 0,114 E?B + 0,299 E?R E?PB = -0,331 E?G + 0,500 E?B -0,169 E?R E?PR = -0,419 E?G - 0,081 E?B + 0,500 E?R
6	SMPTE 170M E?Y = 0,587 E?G + 0,114 E?B + 0,299 E?R E?PB = -0,331 E?G + 0,500 E?B -0,169 E?R E?PR = -0,419 E?G - 0,081 E?B + 0,500 E?R
7	SMPTE 240M (1987) E?Y = 0,701 E?G + 0,087 E?B + 0,212 E?R E?PB = -0,384 E?G + 0,500 E?B -0,116 E?R E?PR = -0,445 E?G - 0,055 E?B + 0,500 E?R
8-255	reserved

In the case that video_signal_type() is not present in the bitstream or colour_description is zero the matrix coefficients are assumed to be those corresponding to matrix_coefficients having the value 1.

In the case that video_signal_type() is not present in the bitstream, video_range is assumed to have the value 0 (a range of Y from 16 to 235 for 8-bit video).

1. **User data**

user_data_start_code: The user_data_start_code is the bit string ‘ 000001B2 ’ in hexadecimal. It identifies the beginning of user data. The user data continues until receipt of another start code.

user_data: This is an 8 bit integer, an arbitrary number of which may follow one another. User data is defined by users for their specific applications. In the series of consecutive user_data bytes there shall not be a string of 23 or more consecutive zero bits.

3. **Video Object Layer**

video_object_layer_start_code: The video_object_layer_start_code is a string of 32 bits. The first 28 bits are ?0000 0000 0000 0000 0000 0001 0010? in binary and the last 4-bits represent one of the values in the range of ?0000? to ?1111? in binary. The video_object_layer_start_code marks a new video object layer.

video_object_layer_id: This is given by the last 4-bits of the video_object_layer_start_code. The video_object_layer_id uniquely identifies a video object layer.

short_video_header: The short_video_header is an internal flag which is set to 1 when an abbreviated header format is used for video content. This indicates video data which begins with a short_video_start_marker rather than a longer start code such as visual_object_ start_code. The short header format is included herein to provide forward compatibility with video codecs designed using the earlier video coding specification ITU-T Recommendation H.263. All decoders which support video objects shall support both header formats (short_video_header equal to 0 or 1) for the subset of video tools that is expressible in either form.

video_plane_with_short_header(): This is a syntax layer encapsulating a video plane which has only the limited set of capabilities available using the short header format.

random_accessible_vol: This flag may be set to "1" to indicate that every VOP in this VOL is individually decodable. If all of the VOPs in this VOL are intra-coded VOPs and some more conditions are satisfied then random_accessible_vol may be set to "1". The flag random_accessible_vol is not used by the decoding process. random_accessible_vol is intended to aid random access or editing capability. This shall be set to "0" if any of the VOPs in the VOL are non-intra coded or certain other conditions are not fulfilled.

video_object_type_indication: Constrains the following bitstream to use tools from the indicated object type only, e.g. Simple Object or Core Object, as shown in Table 6-10.

Table -10 -- FLC table for video_object_type indication

Video Object Type	Code
Reserved	00000000
Simple Object Type	00000001
Simple Scalable Object Type	00000010
Core Object Type	00000011
Main Object Type	00000100
N-bit Object Type	00000101
Basic Anim. 2D Texture	00000110
Anim. 2D Mesh	00000111
Simple Face	00001000
Still Scalable Texture	00001001
Reserved	00001010 - 11111111

is_object_layer_identfier : This is a 1-bit code which when set to ?1? indicates that version identification and priority is specified for the visual object layer. When set to ?0?, no version identification or priority needs to be specified.

video_object_layer_verid: This is a 4-bit code which identifies the version number of the video object layer. Its meaning is defined in Table 6-11. If both visual_object_verid and video_object_layer_verid exist, the semantics of video_object_layer_verid supersedes the other.

Table -11 -- Meaning of video_object_layer_verid

video_object_layer_verid	Meaning
0000	reserved
0001	ISO/IEC 14496-2
0010 - 1111	reserved

video_object_layer_priority : This is a 3-bit code which specifies the priority of the video object layer. It takes values between 1 and 7, with 1 representing the highest priority and 7, the lowest priority. The value of zero is reserved.

aspect_ratio_info: This is a four-bit integer which defines the value of pixel aspect ratio. Table 6-12 shows the meaning of the code. If aspect_ratio_info indicates extended PAR, pixel_aspect_ratio is represented by par_width and par_height. The par_width and par_height shall be relatively prime.

Table -12 -- Meaning of pixel aspect ratio

aspect_ratio_info	pixel aspect ratios
0000	Forbidden
0001	1:1 (Square)
0010	12:11 (625-type for 4:3 picture)
0011	10:11 (525-type for 4:3 picture)
0100	16:11 (625-type stretched for 16:9 picture)
0101	40:33 (525-type stretched for 16:9 picture)
0110-1110	Reserved
1111	extended PAR

par_width: This is an 8-bit unsigned integer which indicates the horizontal size of pixel aspect ratio. A zero value is forbidden.

par_height: This is an 8-bit unsigned integer which indicates the vertical size of pixel aspect ratio. A zero value is forbidden.

vol_control_parameters : This a one-bit flag which when set to ?1? indicates presence of the following parameters: chroma_format, low_delay, and vbv_parameters.

chroma_format: This is a two bit integer indicating the chrominance format as defined in the Table 6-13.

Table -13 -- Meaning of chroma_format

chroma_format	Meaning
00	reserved
01	4:2:0
10	reserved
11	reserved

low_delay : This is a one-bit flag which when set to ?1? indicates the VOL contains no B-VOPs.

vbv_parameters: This is a one-bit flag which when set to ?1? indicates presence of following VBV parameters: first_half_bit_rate, latter_half_bit_rate, first_half_vbv_buffer_size, latter_half_vbv_buffer_size, first_half_vbv_occupancy

and latter_half_vbv_occupancy. The VBV constraint is defined in annex D.

first_half_bit_rate, latter_half_bit_rate: The bit rate is a 30-bit unsigned integer which specifies the bitrate of the bitstream measured in units of 400 bits/second, rounded upwards. The value zero is forbidden. This value is divided to two parts. The most significant bits are in first_half_bit_rate (15 bits) and the least significant bits are in latter_half_bit_rate (15 bits). The marker_bit is inserted between the first_half_bit_rate and the latter_half_bit_rate in order to avoid the resync_marker emulation. The instantaneous video object layer channel bit rate seen by the encoder is denoted by $R_{vol}(t)$ in bits per second. If the bit_rate (i.e. first_half_bit_rate and latter_half_bit_rate) field in the VOL header is present, it defines a peak rate (in units of 400 bits per second; a value of 0 is forbidden) such that $R_{vol}(t)$ <= 400 ´ *bit_rate* Note that $R_{vol}(t)$ counts only visual syntax for the current elementary stream (also see annex D).

first_half_vbv_buffer_size, latter_half_vbv_buffer_size : vbv_buffer_size is an 18-bit unsigned integer. This value is divided into two parts. The most significant bits are in first_half_vbv_buffer_size (15 bits) and the least significant bits are in latter_half_vbv_buffer_size (3 bits), The VBV buffer size is specified in units of 16384 bits. The value 0 for vbv_buffer_size is forbidden. Define B = 16384 ´ vbv_buffer_size to be the VBV buffer size in bits.

first_half_vbv_occupancy, latter_half_vbv_occupancy: The vbv_occupancy is a 26-bit unsigned integer. This value is divided to two parts. The most significant bits are in first_half_vbv_occupancy (11 bits) and the least significant bits are in latter_half_vbv_occupancy (15 bits). The marker_bit is inserted between the first_vbv_buffer_size and the latter_half_vbv_buffer_size in order to avoid the resync_marker emulation. The value of this integer is the VBV occupancy in 64-bit units just before the removal of the first VOP following the VOL header. The purpose for the quantity is to provide the initial condition for VBV buffer fullness.

video_object_layer_shape : This is a 2-bit integer defined in Table 6-14. It identifies the shape type of a video object layer.

Table -14 -- Video Object Layer shape type

Shape format	**Meaning**
00	rectangular
01	binary
10	binary only
11	grayscale

vop_time_increment_resolution : This is a 16-bit unsigned integer that indicates the number of evenly spaced subintervals, called ticks, within one modulo time. One modulo time represents the fixed interval of one second. The value zero is forbidden.

fixed_vop_rate: This is a one-bit flag which indicates that all VOPs are coded with a fixed VOP rate. It shall only be ’1’ if and only if all the distances between the display time of any two successive VOPs in the display order in the video object layer are constant. In this case, the VOP rate can be derived from the fixed_VOP_time_increment. If it is ’0’ the display time between any two successive VOPs in the display order can be variable thus indicated by the time stamps provided in the VOP header.

fixed_vop_time_increment : This value represents the number of ticks between two successive VOPs in the display order. The length of a tick is given by VOP_time_increment_resolution. It can take a value in the range of [0,VOP_time_increment_resolution). The number of bits representing the value is calculated as the minimum number of unsigned integer bits required to represent the above range. fixed_VOP_time_increment shall only be present if fixed_VOP_rate is ’1’ and its value must be identical to the constant given by the distance between the display time of any two successive VOPs in the display order. In this case, the fixed VOP rate is given as
(VOP_time_increment_resolution / fixed_VOP_time_increment). A zero value is forbidden.

EXAMPLE
VOP time = tick ´ vop_time_increment
= vop_time_increment / vop_time_increment_resolution

Table -15 -- Examples of vop_time_increment_resolution, fix_vop_time_increment, and vop_time_increment

Fixed VOP rate = 1/VOP time	**vop_time_increment_ resolution**	**fixed_vop_time_ increment**	**vop_time_increment**
15Hz	15	1	0, 1, 2, 3, 4,?
7.5Hz	15	2	0, 2, 4, 6, 8,?
29.97?Hz	30000	1001	0, 1001, 2002, 3003,?
59.94?Hz	60000	1001	0, 1001, 2002, 3003,?

video_object_layer_width: The video_object_layer_width is a 13-bit unsigned integer representing the width of the displayable part of the luminance component in pixel units. The width of the encoded luminance component of VOPs in macroblocks is (video_object_layer_width+15)/16. The displayable part is left-aligned in the encoded VOPs.

video_object_layer_height : The video_object_layer_height is a 13-bit unsigned integer representing the height of the displayable part of the luminance component in pixel units. The height of the encoded luminance component of VOPs in macroblocks is (video_object_layer_height+15)/16. The displayable part is top-aligned in the encoded VOPs.

interlaced: This is a 1 bit flag which, when set to "1" indicates that the VOP may contain interlaced video. When this flag is set to "0", the VOP is of non-interlaced (or progressive) format.

obmc_disable: This is a one-bit flag which when set to ?1? disables overlapped block motion compensation.

sprite_enable: This is a one-bit flag which when set to ?1? indicates the presence of sprites.

sprite_width : This is a 13-bit unsigned integer which identifies the horizontal dimension of the sprite.

sprite_height : This is a 13-bit unsigned integer which identifies the vertical dimension of the sprite.

sprite_left_coordinate - This is a 13-bit signed integer which defines the left-edge of the sprite. The value of sprite_left_coordinate shall be divisible by two.

sprite_top_coordinate: This is a 13-bit signed integer which defines the top edge of the sprite. The value of sprite_left_coordinate shall be divisible by two.

no_of_sprite_warping_points: This is a 6-bit unsigned integer which represents the number of points used in sprite warping. When its value is 0 and when sprite_enable is set to ?1?, warping is identity (stationary sprite) and no coordinates need to be coded. When its value is 4, a perspective transform is used. When its value is 1,2 or 3, an affine transform is used. Further, the case of value 1 is separated as a special case from that of values 2 or 3. Table 6-16 shows the various choices.

Table -16 -- Number of point and implied warping function

Number of points	**warping function**
0	Stationary
1	Translation
2,3	Affine
4	Perspective

sprite_warping_accuracy - This is a 2-bit code which indicates the quantization accuracy of motion vectors used in the warping process for sprites. Table 6-17 shows the meaning of various codewords

Table -17 -- Meaning of sprite warping accuracy codewords

code	sprite_warping_accuracy
00	½ pixel
01	¼ pixel
10	1/8 pixel
11	1/16 pixel

sprite_brightness_change: This is a one-bit flag which when set to ?1? indicates a change in brightness during sprite warping, alternatively, a value of ?0? means no change in brightness.

low_latency_sprite_enable: This is a one-bit flag which when set to "1" indicates the presence of low_latency sprite, alternatively, a value of "0" means basic sprite.

not_8_bit : This one bit flag is set when the video data precision is not 8 bits per pixel.

quant_precision: This field specifies the number of bits used to represent quantiser parameters. Values between 3 and 9 are allowed. When not_8_bit is zero, and therefore quant_precision is not transmitted, it takes a default value of 5.

bits_per_pixel: This field specifies the video data precision in bits per pixel. It may take different values for different video object layers within a single video object. A value of 12 in this field would indicate 12 bits per pixel. This field may take values between 4 and 12. When not_8_bit is zero and bits_per_pixel is not present, the video data precision is always 8 bits per pixel, which is equivalent to specifying a value of 8 in this field. The same number of bits per pixel is used in the luminance and two chrominance planes. The alpha plane, used to specify shape of video objects, is always represented with 8 bits per pixel.

no_gray_quant_update: This is a one bit flag which is set to ?1? when a fixed quantiser is used for the decoding of grayscale alpha data. When this flag is set to ?0?, the grayscale alpha quantiser is updated on every macroblock by generating it anew from the luminance quantiser value, but with an appropriate scale factor applied. See the description in subclause 7.5.4.3.

composition_method: This is a one bit flag which indicates which blending method is to be applied to the video object in the compositor. When set to ?0?, cross-fading shall be used. When set to ?1?, additive mixing shall be used. See subclause 7.5.4.6.

linear_composition: This is a one bit flag which indicates the type of signal used by the compositing process. When set to ?0?, the video signal in the format from which it was produced by the video decoder is used. When set to ?1?, linear signals are used. See subclause 7.5.4.6.

quant_type: This is a one-bit flag which when set to ?1? that the first inverse quantisation method and when set to ?0? indicates that the second inverse quantisation method is used for inverse quantisation of the DCT coefficients. Both inverse quantisation methods are described in subclause 7.4.4. For the first inverse quantization method, two matrices are used, one for intra blocks the other for non-intra blocks.

The default matrix for intra blocks is:

8	17	18	19	21	23	25	27
17	18	19	21	23	25	27	28
20	21	22	23	24	26	28	30
21	22	23	24	26	28	30	32
22	23	24	26	28	30	32	35
23	24	26	28	30	32	35	38
25	26	28	30	32	35	38	41
27	28	30	32	35	38	41	45

The default matrix for non-intra blocks is:

16	17	18	19	20	21	22	23
17	18	19	20	21	22	23	24
18	19	20	21	22	23	24	25
19	20	21	22	23	24	26	27
20	21	22	23	25	26	27	28
21	22	23	24	26	27	28	30
22	23	24	26	27	28	30	31
23	24	25	27	28	30	31	33

load_intra_quant_mat: This is a one-bit flag which is set to ?1? when intra_quant_mat follows. If it is set to ?0? then there is no change in the values that shall be used.

intra_quant_mat: This is a list of 2 to 64 eight-bit unsigned integers. The new values are in zigzag scan order and replace the previous values. A value of 0 indicates that no more values are transmitted and the remaining, non-transmitted values are set equal to the last non-zero value. The first value shall always be 8 and is not used in the decoding process.

load_nonintra_quant_mat: This is a one-bit flag which is set to ?1? when nonintra_quant_mat follows. If it is set to ?0? then there is no change in the values that shall be used.

nonintra_quant_mat: This is a list of 2 to 64 eight-bit unsigned integers. The new values are in zigzag scan order and replace the previous values. A value of 0 indicates that no more values are transmitted and the remaining, non-transmitted values are set equal to the last non-zero value. The first value shall not be 0.

load_intra_quant_mat_grayscale : This is a one-bit flag which is set to ?1? when intra_quant_mat_grayscale follows. If it is set to ?0? then there is no change in the quantisation matrix values that shall be used.

intra_quant_mat_grayscale: This is a list of 2 to 64 eight-bit unsigned integers defining the grayscale intra alpha quantisation matrix to be used. The semantics and the default quantisation matrix are identical to those of intra_quant_mat.

load_nonintra_quant_mat_grayscale : This is a one-bit flag which is set to ?1? when nonintra_quant_mat_grayscale follows. If it is set to ?0? then there is no change in the quantisation matrix values that shall be used.

nonintra_quant_mat_grayscale : This is a list of 2 to 64 eight-bit unsigned integers defining the grayscale nonintra alpha quantisation matrix to be used. The semantics and the default quantisation matrix are identical to those of nonintra_quant_mat.

complexity_estimation_disable : This is a one-bit flag which, when set to '1', disables complexity estimation header in each VOP.

estimation_method : Setting of the of the estimation method,it is ?00" for Version 1.

shape_complexity_estimation_disable : This is a one-bit flag which when set to '1' disables shape complexity estimation.

opaque : Flag enabling transmission of the number of luminance and chrominance blocks coded using opaque coding mode in % of the total number of blocks (bounding rectangle).

transparent : Flag enabling transmission of the number of luminance and chrominance blocks coded using transparent mode in % of the total number of blocks (bounding rectangle).

intra_cae: Flag enabling transmission of the number of luminance and chrominance blocks coded using IntraCAE coding mode in % of the total number of blocks (bounding rectangle).

inter_cae: Flag enabling transmission of the number of luminance and chrominance blocks coded using InterCAE coding mode in % of the total number of blocks (bounding rectangle).

no_update : Flag enabling transmission of the number of luminance and chrominance blocks coded using no update coding mode in % of the total number of blocks (bounding rectangle).

upsampling : Flag enabling transmission of the number of luminance and chrominance blocks which need upsampling from 4-4- to 8-8 block dimensions in % of the total number of blocks (bounding rectangle).

texture_complexity_estimation_set_1_disable : Flag to disable texture parameter set 1.

intra_blocks: Flag enabling transmission of the number of luminance and chrominance Intra or Intra+Q coded blocks in % of the total number of blocks (bounding rectangle).

inter_blocks : Flag enabling transmission of the number of luminance and chrominance Inter and Inter+Q coded blocks in % of the total number of blocks (bounding rectangle).

inter4v_blocks : Flag enabling transmission of the number of luminance and chrominance Inter4V coded blocks in % of the total number of blocks (bounding rectangle).

not_coded_blocks : Flag enabling transmission of the number of luminance and chrominance Non Coded blocks in % of the total number of blocks (bounding rectangle).

texture_complexity_estimation_set_2_disable : Flag to disable texture parameter set 2.

dct_coefs: Flag enabling transmission of the number of DCT coefficients % of the maximum number of coefficients (coded blocks).

dct_lines : Flag enabling transmission of the number of DCT8x1 in % of the maximum number of DCT8x1 (coded blocks).

vlc_symbols : Flag enabling transmission of the average number of VLC symbols for macroblock.

vlc_bits: Flag enabling transmission of the average number of bits for each symbol.

motion_compensation_complexity_disable : Flag to disable motion compensation parameter set.

apm (Advanced Prediction Mode): Flag enabling transmission of the number of luminance block predicted using APM in % of the total number of blocks for VOP (bounding rectangle).

npm (Normal Prediction Mode): Flag enabling transmission of the number of luminance and chrominance blocks predicted using NPM in % of the total number of luminance and chrominance for VOP (bounding rectangle).

interpolate_mc_q : Flag enabling transmission of the number of luminance and chrominance interpolated blocks in % of the total number of blocks for VOP (bounding rectangle).

forw_back_mc_q: Flag enabling transmission of the number of luminance and chrominance predicted blocks in % of the total number of blocks for VOP (bounding rectangle).

halfpel2: Flag enabling transmission of the number of luminance and chrominance block predicted by a half-pel vector on one dimension (horizontal or vertical) in % of the total number of blocks (bounding rectangle).

halfpel4: Flag enabling transmission of the number of luminance and chrominance block predicted by a half-pel vector on two dimensions (horizontal and vertical) in % of the total number of blocks (bounding rectangle).

resync_marker_disable : This is a one-bit flag which when set to ?1? indicates that there is no resync_marker in coded VOPs. This flag can be used only for the optimization of the decoder operation. Successful decoding can be carried out without taking into account the value of this flag.

data_partitioned: This is a one-bit flag which when set to ?1? indicates that the macroblock data is rearranged differently, specifically, motion vector data is separated from the texture data (i.e., DCT coefficients).

reversible_vlc: This is a one-bit flag which when set to ?1? indicates that the reversible variable length tables (Table B-23, Table B-24 and Table B-25) should be used when decoding DCT coefficients. These tables can only be used when data_partition flag is enabled. Note that this flag shall be treated as ?0? in B-VOPs. Use of escape sequence (Table B-24 and Table B-25) for encoding the combinations listed in Table B-23 is prohibited.

scalability: This is a one-bit flag which when set to ?1? indicates that the current layer uses scalable coding. If the current layer is used as base-layer then this flag is set to ?0?.

hierarchy_type : The hierarcical relation between the associated hierarchy layer and its hierarchy embedded layer is defined as shown in Table 6-18.

Table -18 -- Code table for hierarchy_type

Description	**Code**
ISO/IEC 14496-2 Spatial Scalability	0
ISO/IEC 14496-2 Temporal Scalability	1

ref_layer_id: This is a 4-bit unsigned integer with value between 0 and 15. It indicates the layer to be used as reference for prediction(s) in the case of scalability.

ref_layer_sampling_direc: This is a one-bit flag which when set to ?1? indicates that the resolution of the reference layer (specified by reference_layer_id) is higher than the resolution of the layer being coded. If it is set to ?0? then the reference layer has the same or lower resolution then the resolution of the layer being coded.

hor_sampling_factor_n: This is a 5-bit unsigned integer which forms the numerator of the ratio used in horizontal spatial resampling in scalability. The value of zero is forbidden.

hor_sampling_factor_m: This is a 5-bit unsigned integer which forms the denominator of the ratio used in horizontal spatial resampling in scalability. The value of zero is forbidden.

vert_sampling_factor_n: This is a 5-bit unsigned integer which forms the numerator of the ratio used in vertical spatial resampling in scalability. The value of zero is forbidden.

vert_sampling_factor_m: This is a 5-bit unsigned integer which forms the denominator of the ratio used in vertical spatial resampling in scalability. The value of zero is forbidden.

enhancement_type: This is a 1-bit flag which is set to ?1? when the current layer enhances the partial region of the reference layer. If it is set to ?0? then the current layer enhances the entire region of the reference layer. The default value of this flag is ?0?.

4. **Group of Video Object Plane**

group_vop_start_code: The group_vop_start_code is the bit string ?000001B3? in hexadecimal. It identifies the beginning of a GOV header.

time_code: This is a 18-bit integer containing the following: time_code_hours, time_code_minutes, marker_bit and time_code_seconds as shown in Table 6-19. The parameters correspond to those defined in the IEC standard publication 461 for "time and control codes for video tape recorders". The time code specifies the modulo part (i.e. the full second units) of the time base for the first object plane (in display order) after the GOV header.

Table -19 -- Meaning of time_code

time_code	**range of value**	**No. of bits**	**Mnemonic**
time_code_hours	0 - 23	5	uimsbf
time_code_minutes	0 - 59	6	uimsbf
marker_bit	1	1	bslbf
time_code_seconds	0 - 59	6	uimsbf

closed_gov: This is a one-bit flag which indicates the nature of the predictions used in the first consecutive B-VOPs (if any) immediately following the first coded I-VOP after the GOV header .The closed_gov is set to ?1? to indicate that these B-VOPs have been encoded using only backward prediction or intra coding. This bit is provided for use during any editing which occurs after encoding. If the previous pictures have been removed by editing, broken_link may be set to ?1? so that a decoder may avoid displaying these B-VOPs following the first I-VOP following the group of plane header. However if the closed_gov bit is set to ?1?, then the editor may choose not to set the broken_link bit as these B-VOPs can be correctly decoded.

broken_link: This is a one-bit flag which shall be set to ?0? during encoding. It is set to ?1? to indicate that the first consecutive B-VOPs (if any) immediately following the first coded I-VOP following the group of plane header may not be correctly decoded because the reference frame which is used for prediction is not available (because of the action of editing). A decoder may use this flag to avoid displaying frames that cannot be correctly decoded.

5. **Video Object Plane and Video Plane with Short Header**

vop_start_code: This is the bit string ?000001B6? in hexadecimal. It marks the start of a video object plane.

vop_coding_type : The vop_coding_type identifies whether a VOP is an intra-coded VOP (I), predictive-coded VOP (P), bidirectionally predictive-coded VOP (B) or sprite coded VOP (S). The meaning of vop_coding_type is defined in Table 6-20.

Table -20 -- Meaning of vop_coding_type

vop_coding_type	**coding method**
00	intra-coded (I)
01	predictive-coded (P)
10	bidirectionally-predictive-coded (B)
11	sprite (S)

modulo_time_base: This value represents the local time base in one second resolution units (1000 milliseconds). It consists of a number of consecutive ?1? followed by a ?0?. Each ?1? represents a duration of one second that have elapsed. For I- and P-VOPs of a non scalable bitstream and the base layer of a scalable bitstream, the number of ?1?s indicate the number of seconds elapsed since the synchronization point marked by time_code of the previous GOV header or by modulo_time_base of the previously decoded I- or P-VOP, in decoding order. For B-VOP of non scalable bitstream and base layer of scalable bitstream, the number of ?1?s indicate the number of seconds elapsed since the synchronization point marked in the previous GOV header, I-VOP, or P-VOP, in display order. For I-, P-, or B-VOPs of enhancement layer of scalable bitstream, the number of ?1?s indicate the number of seconds elapsed since the synchronization point marked in the previous GOV header, I-VOP, P-VOP, or B-VOP, in display order.

vop_time_increment : This value represents the absolute vop_time_increment from the synchronization point marked by the modulo_time_base measured in the number of clock ticks. It can take a value in the range of [0,vop_time_increment_resolution). The number of bits representing the value is calculated as the minimum number of unsigned integer bits required to represent the above range. The local time base in the units of seconds is recovered by dividing this value by the vop_time_increment_resolution.

vop_coded: This is a 1-bit flag which when set to ?0? indicates that no subsequent data exists for the VOP. In this case, the following decoding rule applies: For an arbitrarily shaped VO (i.e. when the shape type of the VO is either ?binary? or ?binary only?), the alpha plane of the reconstructed VOP shall be completely transparent. For a rectangular VO (i.e. when the shape type of the VO is

?rectangular?), the corresponding rectangular alpha plane of the VOP, having the same size as its luminance component, shall be completely transparent. If there is no alpha plane being used in the decoding and composition process of a rectangular VO, the reconstructed VOP is filled with the respective content of the immediately preceding VOP for which vop_coded!=0.

vop_rounding_type : This is a one-bit flag which signals the value of the parameter rounding_control used for pixel value interpolation in motion compensation for P-VOPs. When this flag is set to ?0?, the value of rounding_control is 0, and when this flag is set to ?1?, the value of rounding_control is 1. When vop_rounding_type is not present in the VOP header, the value of rounding_control is 0.

vop_width: This is a 13-bit unsigned integer which specifies the horizontal size, in pixel units, of the rectangle that includes the VOP. The width of the encoded luminance component of VOP in macroblocks is (vop_width+15)/16. The rectangle part is left-aligned in the encoded VOP. A zero value is forbidden.

vop_height: This is a 13-bit unsigned integer which specifies the vertical size, in pixel units, of the rectangle that includes the VOP. The height of the encoded luminance component of VOP in macroblocks is (vop_height+15)/16. The rectangle part is top-aligned in the encoded VOP. A zero value is forbidden.

vop_horizontal_mc_spatial_ref: This is a 13-bit signed integer which specifies, in pixel units, the horizontal position of the top left of the rectangle defined by horizontal size of vop_width. The value of vop_horizontal_mc_spatial_ref shall be divisible by two. This is used for decoding and for picture composition.

vop_vertical_mc_spatial_ref : This is a 13-bit signed integer which specifies, in pixel units, the vertical position of the top left of the rectangle defined by vertical size of vop_width. The value of vop_vertical_mc_spatial_ref shall be divisible by two for progressive and divisible by four for interlaced motion compensation. This is used for decoding and for picture composition.

background_composition: This flag only occurs when scalability flag has a value of "1. This flag is used in conjunction with enhancement_type flag. If enhancement_type is "1" and this flag is "1", background composition specified in subclause 8.1 is performed. If enhancement type is "1" and this flag is "0", any method can be used to make a background for the enhancement layer.

change_conv_ratio_disable: This is a 1-bit flag which when set to ?1? indicates that conv_ratio is not sent at the macroblock layer and is assumed to be 1 for all the macroblocks of the VOP. When set to ?0?, the conv_ratio is coded at macroblock layer.

vop_constant_alpha: This bit is used to indicate the presence of vop_constant_alpha_value. When this is set to one, vop_constant_alpha_value is included in the bitstream.

vop_constant_alpha_value: This is an unsigned integer which indicates the scale factor to be applied as a post processing phase of binary or grayscale shape decoding. See subclause 7.5.4.2.

intra_dc_vlc_thr: This is a 3-bit code allows a mechanism to switch between two VLC?s for coding of Intra DC coefficients as per Table 6-21.

Table -21 -- Meaning of intra_dc_vlc_thr

index	**meaning of intra_dc_vlc_thr**	**code**
0	Use Intra DC VLC for entire VOP	000
1	Switch to Intra AC VLC at running Qp >=13	001
2	Switch to Intra AC VLC at running Qp >=15	010
3	Switch to Intra AC VLC at running Qp >=17	011
4	Switch to Intra AC VLC at running Qp >=19	100
5	Switch to Intra AC VLC at running Qp >=21	101
6	Switch to Intra AC VLC at running Qp >=23	110
7	Use Intra AC VLC for entire VOP	111

Where running Qp is defined as the DCT quantization parameter for luminance and chrominance used for immediately previous coded macroblock, except for the first coded macroblock in a VOP or a video packet. At the first coded macroblock in a VOP or a video packet, the running Qp is defined as the quantization parameter value for the current macroblock.

top_field_first: This is a 1-bit flag which when set to "1" indicates that the top field (i.e., the field containing the top line) of reconstructed VOP is the first field to be displayed (output by the decoding process). When top_field_first is set to "0" it indicates that the bottom field of the reconstructed VOP is the first field to be displayed.

alternate_vertical_scan_flag: This is a 1-bit flag which when set to "1" indicates the use of alternate vertical scan for interlaced VOPs.

sprite_transmit_mode: This is a 2-bit code which signals the transmission mode of the sprite object. At video object layer initialization, the code is set to "piece" mode. When all object and quality update pieces are sent for the entire video object layer, the code is set to the "stop"mode. When an object piece is sent, the code is set to "piece" mode. When an update piece is being sent, the code is set to the "update" mode. When all sprite object pieces andquality update pieces for the current VOP are sent, the code is set to "pause" mode. Table 6-22 shows the different sprite transmit modes.

Table -22 -- Meaning of sprite transmit modes

code	**sprite_transmit_mode**
00	stop
01	piece
10	update
11	pause

vop_quant: This is an unsigned integer which specifies the absolute value of quant to be used for dequantizing the macroblock until updated by any subsequent dquant, dbquant, or quant_scale. The length of this field is specified by the value of the parameter quant_precision. The default length is 5-bits which carries the binary representation of quantizer values from 1 to 31 in steps of 1.

vop_alpha_quant: This is a an unsigned integer which specifies the absolute value of the initial alpha plane quantiser to be used for dequantising macroblock grayscale alpha data. The alpha plane quantiser cannot be less than 1.

vop_fcode_forward: This is a 3-bit unsigned integer taking values from 1 to 7; the value of zero is forbidden. It is used in decoding of motion vectors.

vop_fcode_backward: This is a 3-bit unsigned integer taking values from 1 to 7; the value of zero is forbidden. It is used in decoding of motion vectors.

vop_shape_coding_type: This is a 1 bit flag which specifies whether inter shape decoding is to be carried out for the current P VOP. If vop_shape_coding_type is equal to ?0?, intra shape decoding is carried out, otherwise inter shape decoding is carried out.

Coded data for the top-left macroblock of the bounding rectangle of a VOP shall immediately follow the VOP header, followed by the remaining macroblocks in the bounding rectangle in the conventional left-to-right, top-to-bottom scan order. Video packets shall also be transmitted following the conventional left-to-right, top-to-bottom macroblock scan order. The last MB of one video packet is guaranteed to immediately precede the first MB of the following video packet in the MB scan order.

load_backward_shape: This is a one-bit flag which when set to ?1? implies that the backward shape of the previous VOP in the same layer is copied to the forward shape for the current VOP and the backward shape of the current VOP is decoded from the bitstream. When this flag is set to ?0?, the forward shape of the previous VOP is copied to the forward_shape of the current VOP and the backward shape of the previous VOP in the same layer is copied to the backward shape of the current VOP. This flag shall be ?1? when (1) background_composition is ?1? and vop_coded of the previous VOP in the same layer is ?0? or (2) background_composition is ?1? and the current VOP is the first VOP in the current layer.

backward_shape_width : This is a 13-bit unsigned integer which specifies the horizontal size, in pixel units, of the rectangle that includes the backward shape. A zero value is forbidden.

backward_shape_height : This is a 13-bit unsigned integer which specifies the vertical size, in pixel units, of the rectangle that includes the backward shape. A

zero value is forbidden.

backward_shape_horizontal_mc_spatial_ref: This is a 13-bit signed integer which specifies, in pixel units, the horizontal position of the top left of the rectangle that includes the backward shape. This is used for decoding and for picture composition.

backward_shape_vertical_mc_spatial_ref : This is a 13-bit signed integer which specifies, in pixel units, the vertical position of the top left of the rectangle that includes the backward shape. This is used for decoding and for picture composition.

backward_shape(): The decoding process of the backward shape is identical to the decoding process for the shape of I-VOP with binary only mode (video_object_layer_shape = "10").

load_forward_shape: This is a one-bit flag which when set to ?1? implies that the forward shape is decoded from the bitstream. This flag shall be ?1? when (1) background_composition is ?1? and vop_coded of the previous VOP in the same layer is ?0? or (2) background_composition is ?1? and the current VOP is the first VOP in the current layer.

forward_shape_width : This is a 13-bit unsigned integer which specifies the horizontal size, in pixel units, of the rectangle that includes the forward shape. A zero value is forbidden.

forward_shape_height : This is a 13-bit unsigned integer which specifies the vertical size, in pixel units, of the rectangle that includes the forward shape. A zero value is forbidden.

forward_shape_horizontal_mc_spatial_ref: This is a 13-bit signed integer which specifies, in pixel units, the horizontal position of the top left of the rectangle that includes the forward shape. This is used for decoding and for picture composition.

forward_shape_vertical_mc_spatial_ref : This is a 13-bit signed integer which specifies, in pixel units, the vertical position of the top left of the rectangle that includes the forward shape. This is used for decoding and for picture composition.

forward_shape(): The decoding process of the backward shape is identical to the decoding process for the shape of I-VOP with binary only mode (video_object_layer_shape = "10").

ref_select_code : This is a 2-bit unsigned integer which specifies prediction reference choices for P- and B-VOPs in enhancement layer with respect to decoded reference layer identified by ref_layer_id. The meaning of allowed values is specified in Table 7-13 and Table 7-14.

resync_marker: This is a binary string of at least 16 zero?s followed by a one?0 0000 0000 0000 0001?. For an I-VOP or a VOP where video_object_layer_shape has the value "binary_only", the resync marker is 16 zeros followed by a one. The length of this resync marker is dependent on the value of vop_fcode_forward, for a P-VOP, and the larger value of either vop_fcode_forward and vop_fcode_backward for a B-VOP. The relationship between the length of the resync_marker and appropriate fcode is given by 16 + fcode. The resync_marker is (15+fcode) zeros followed by a one. It is only present when resync_marker_disable flag is set to ?0?. A resync marker shall only be located immediately before a macroblock and aligned with a byte

macroblock_number: This is a variable length code with length between 1 and 14 bits. It identifies the macroblock number within a VOP. The number of the top-left macroblock in a VOP shall be zero. The macroblock number increases from left to right and from top to bottom. The actual length of the code depends on the total number of macroblocks in the VOP calculated according to Table 6-23, the code itself is simply a binary representation of the macroblock number.

Table -23 -- Length of macroblock_number code

length of macroblock_number code	**((vop_width+15)/16) * ((vop_height+15)/16)**
1	1-2
2	3-4
3	5-8
4	9-16
5	17-32
6	33-64
7	65-128
8	129-256
9	257-512
10	513-1024
11	1025-2048
12	2049-4096
13	4097-8192
14	8193-16384

quant_scale: This is an unsigned integer which specifies the absolute value of quant to be used for dequantizing the macroblock of the video packet until updated by any subsequent dquant. The length of this field is specified by the value of the

parameter quant_precision. The default length is 5-bits.

header_extension_code: This is a 1-bit flag which when set to ?1? indicates the prescence of additional fields in the header. When header_extension_code is is se to ?1?, modulo_time_base, vop_time_increment and vop_coding_type are also included in the video packet header. Furthermore, if the vop_coding_type is equal to either a P or B VOP, the appropriate fcodes are also present.

use_intra_dc_vlc: The value of this internal flag is set to 1 when the values of intra_dc_thr and the DCT quantiser for luminance and chrominace indicate the usage of the intra DC VLCs shown in Table B-13 - Table B-15 for the decoding of intra DC coefficients. Otherwise, the value of this flag is set to 0.

motion_marker: This is a 17-bit binary string ?1 1111 0000 0000 0001?. It is only present when the data_partitioned flag is set to ?1?.In the data partitioning mode, a motion_marker is inserted after the motion data (prior to the texture data). The motion_marker is unique from the motion data and enables the decoder to determine when all the motion information has been received correctly.

dc_marker: This is a 19 bit binary string ?110 1011 0000 0000 0001?. It is present when the data_partitioned flag is set to ?1?. It is used for I-VOPs. In the data partitioning mode, a dc_marker is inserted into the bitstream after the mcbpc, dquant and dc data but before the ac_pred flag and remaining texture information.

1. **Definition of DCECS variable values**

   The semantic of all complexity estimation parameters is defined at the VO syntax level. DCECS variables represent % values. The actual % values have been converted to 8 bit words by normalization to 256. To each 8 bit word a binary 1 is added to prevent start code emulation (i.e 0% = ?00000001?, 99.5% = ?11111111? and is conventionally considered equal to one). The binary ?00000000? string is a forbidden value. The only parameter expressed in their absolute value is the dcecs_vlc_bits parameter expressed as a 4 bit word.

   dcecs_opaque : 8 bit number representing the % of luminance and chrominance blocks using opaque coding mode on the total number of blocks (bounding rectangle).

   dcecs_transparent : 8 bit number representing the % of luminance and chrominance blocks using transparent coding mode on the total number of blocks (bounding rectangle).

   dcecs_intra_cae : 8 bit number representing the % of luminance and chrominance blocks using IntraCAE coding mode on the total number of

blocks (bounding rectangle).

dcecs_inter_cae : 8 bit number representing the % of luminance and chrominance blocks using InterCAE coding mode on the total number of blocks (bounding rectangle).

dcecs_no_update : 8 bit number representing the % of luminance and chrominance blocks using no update coding mode on the total number of blocks (bounding rectangle).

dcecs_upsampling : 8 bit number representing the % of luminance and chrominance blocks which need upsampling from 4-4- to 8-8 block dimensions on the total number of blocks (bounding rectangle).

dcecs_intra_blocks : 8 bit number representing the % of luminance and chrominance Intra or Intra+Q coded blocks on the total number of blocks (bounding rectangle).

dcecs_not_coded_blocks: 8 bit number representing the % of luminance and chrominance Non Coded blocks on the total number of blocks (bounding rectangle).

dcecs_dct_coef s: 8 bit number representing the % of the number of DCT coefficients on the maximum number of coefficients (coded blocks).

dcecs_dct_lines: 8 bit number representing the % of the number of DCT8x1 on the maximum number of DCT8x1 (coded blocks).

dcecs_vlc_symbols: 8 bit number representing the average number of VLC symbols for macroblock.

dcecs_vlc_bits : 4 bit number representing the average number of bits for each symbol.

dcecs_inter_blocks : 8 bit number representing the % of luminance and chrominance Inter and Inter+Q coded blocks on the total number of blocks (bounding rectangle).

dcecs_inter4v_blocks : 8 bit number representing the % of luminance and chrominance Inter4V coded blocks on the total number of blocks (bounding rectangle).

dcecs_apm (Advanced Prediction Mode): 8 bit number representing the % of the number of luminance block predicted using APM on the total number of blocks for VOP (bounding rectangle).

dcecs_npm (Normal Prediction Mode): 8 bit number representing the % of luminance and chrominance blocks predicted using NPM on the total number of luminance and chrominance blocks for VOP (bounding rectangle).

dcecs_forw_back_mc_q: 8 bit number representing the % of luminance and chrominance predicted blocks on the total number of blocks for VOP (bounding rectangle).

dcecs_halfpel2 : 8 bit number representing the % of luminance and chrominance blocks predicted by a half-pel vector on one dimension (horizontal or vertical) on the total number of blocks (bounding rectangle).

dcecs_halfpel4 : 8 bit number representing the % of luminance and chrominance blocks predicted by a half-pel vector on two dimensions (horizontal and vertical) on the total number of blocks (bounding rectangle).

dcecs_interpolate_mc_q: 8 bit number representing the % of luminance and chrominance interpolated blocks in % of the total number of blocks for VOP (bounding rectangle).

2. **Video Plane with Short Header**

video_plane_with_short_header() - This data structure contains a video plane using an abbreviated header format. Certain values of parameters shall have pre-defined and fixed values for any video_plane_with_short_header, due to the limited capability of signaling information in the short header format. These parameters having fixed values are shown in Table 6-24.

Table -24 -- Fixed Settings for video_plane_with_short_header()

Parameter	**Value**
video_object_layer_shape	"rectangular"
obmc_disable	1
quant_type	0
resync_marker_disable	1
data_partitioned	0
block_count	6
reversible_vlc	0
vop_rounding_type	0
vop_fcode_forward	1
vop_coded	1
interlaced	0
complexity_estimation_disable	1
use_intra_dc_vlc	0
scalability	0
not_8_bit	0
bits_per_pixel	8
colour_primaries	1
transfer_characteristics	1
matrix_coefficients	6

short_video_start_marker : This is a 22-bit start marker containing the value ?0000 0000 0000 0000 1000 00?. It is used to mark the location of a video plane having the short header format. short_video_start_marker shall be byte aligned by the insertion of zero to seven zero-valued bits as necessary to achieve byte alignment prior to short_video_start_marker.

temporal_reference : This is an 8-bit number which can have 256 possible values. It is formed by incrementing its value in the previously transmitted video_plane_with_short_header() by one plus the number of non-transmitted pictures (at 30000/1001 Hz) since the previously transmitted picture. The arithmetic is performed with only the eight LSBs.

split_screen_indicator: This is a boolean signal that indicates that the upper and lower half of the decoded picture could be displayed side by side. This bit has no direct effect on the encoding or decoding of the video plane.

document_camera_indicator: This is a boolean signal that indicates that the video content of the vop is sourced as a representation from a document camera or graphic representation, as opposed to a view of natural video content. This bit has no direct effect on the encoding or decoding of the video plane.

full_picture_freeze_release : This is a boolean signal that indicates that resumption of display updates should be activated if the display of the video content has been frozen due to errors, packet losses, or for some other reason such as the receipt of a external signal. This bit has no direct effect on the encoding or decoding of the video plane.

source_format : This is an indication of the width and height of the rectangular video plane represented by the video_plane_with_short_header. The meaning of this field is shown in Table 6-25. Each of these source formats has the same vop time increment resolution which is equal to 30000/1001 (approximately 29.97) Hz and the same width:height pixel aspect ratio (288/3):(352/4), which equals 12:11 in relatively prime numbers and which defines a CIF picture as having a width:height picture aspect ratio of 4:3.

Table -25 -- Parameters Defined by source_format Field

source_format value	**Source Format Meaning**	**vop_width**	**vop_height**	**num_macroblocks_in_ gob**	**r v**
000	reserved	reserved	reserved	reserved	r
001	sub-QCIF	128	96	8	6
010	QCIF	176	144	11	9
011	CIF	352	288	22	1
100	4CIF	704	576	88	1
101	16CIF	1408	1152	352	1
110	reserved	reserved	reserved	reserved	r
111	reserved	reserved	reserved	reserved	r

picture_coding_type : This bit indicates the vop_coding_type. When equal to zero, the vop_coding_type is "I", and when equal to one, the vop_coding_type is "P".

four_reserved_zero_bits : This is a four-bit field containing bits which are

reserved for future use and equal to zero.

pei: This is a single bit which, when equal to one, indicates the presence of a byte of psupp data following the pei bit.

psupp: This is an eight bit field which is present when pei is equal to one. The pei + psupp mechanism provides for a reserved method of later allowing the definition of backward-compatible data to be added to the bitstream. Decoders shall accept and discard psupp when pei is equal to one, with no effect on the decoding of the video data. The pei and psupp combination pair may be repeated if present. The ability for an encoder to add pei and psupp to the bitstream is reserved for future use.

gob_number: This is a five-bit number which indicates the location of video data within the video plane. A group of blocks (or GOB) contains a number of macroblocks in raster scanning order within the picture. For a given gob_number, the GOB contains the num_macroblocks_per_gob macroblocks starting with macroblock_number = gob_number * num_macroblocks_per_gob. The gob_number can either be read from the bitstream or inferred from the progress of macroblock decoding as shown in the syntax description pseudo-code.

num_gobs_in_vop: This is the number of GOBs in the vop. This parameter is derived from the source_format as shown in Table 6-25.

gob_layer(): This is a layer containing a fixed number of macroblocks in the vop. Which macroblocks which belong to each gob can be determined by gob_number and num_macroblocks_in_gob.

gob_resync_marker: This is a fixed length code of 17 bits having the value ?0000 0000 0000 0000 1? which may optionally be inserted at the beginning of each gob_layer(). Its purpose is to serve as a type of resynchronization marker for error recovery in the bitstream. The gob_resync_marker codes may (and should) be byte aligned by inserting zero to seven zero-valued bits in the bitstream just prior to the gob_resync_marker in order to obtain byte alignment. The gob_resync_marker shall not be present for the first GOB (for which gob_number = 0).

gob_number: This is a five-bit number which indicates which GOB is being processed in the vop. Its value may either be read following a gob_resync_marker or may be inferred from the progress of macroblock decoding. All GOBs shall appear in the bitstream of each video_plane_with_short_header(), and the GOBs shall appear in a strictly increasing order in the bitstream. In other words, if a gob_number is read from the bitstream after a gob_resync_marker, its value must be the same as the value that would have been inferred in the absence of the

gob_resync_marker.

gob_frame_id: This is a two bit field which is intended to help determine whether the data following a gob_resync_marker can be used in cases for which the vop header of the video_plane_with_short_header() may have been lost. gob_frame_id shall have the same value in every GOB header of a given video_plane_with_short_header(). Moreover, if any field among the split_screen_indicator or document_camera_indicator or full_picture_freeze_release or source_format or picture_coding_type as indicated in the header of a video_plane_with_short_header() is the same as for the previous transmitted picture in the same video object, gob_frame_id shall have the same value as in that previous video_plane_with_short_header(). However, if any of these fields in the header of a certain video_plane_with_short_header() differs from that in the previous transmitted video_plane_with_short_header() of the same video object, the value for gob_frame_id in that picture shall differ from the value in the previous picture.

num_macroblocks_in_gob : This is the number of macroblocks in each group of blocks (GOB) unit. This parameter is derived from the source_format as shown in Table 6-25.

short_video_end_marker : This is a 22-bit end of sequence marker containing the value ?0000 0000 0000 0000 1111 11?. It is used to mark the end of a sequence of video_plane_with_short_header(). short_video_end_marker may (and should) be byte aligned by the insertion of zero to seven zero-valued bits to achieve byte alignment prior to short_video_end_marker.

3. Shape coding

bab_type: This is a variable length code between 1 and 7 bits. It indicates the coding mode used for the bab. There are seven bab_types as depicted in Table 6-26 . The VLC tables used depend on the decoding context i.e. the bab_types of blocks already received. For I-VOPs, the context-switched VLC table of Table B-27 is used. For P-VOPs and B-VOPs, the context switched table of Table B-28 is used.

Table -26 -- List of bab_types and usage

bab_type	Semantic	Used in
0	MVDs==0 && No Update	P,B VOPs
1	MVDs!=0 && No Update	P,B VOPs
2	transparent	All VOP types
3	opaque	All VOP types
4	intraCAE	All VOP types
5	MVDs==0 && interCAE	P,B VOPs
6	MVDs!=0 && interCAE	P,B VOPs

The bab_type determines what other information fields will be present for the bab shape. No further shape information is present if the bab_type = 0, 2 or 3. Opaque means that all pixels of the bab are part of the object. Transparent means that none of the bab pixels belong to the object. IntraCAE means the intra-mode CAE decoding will be required to reconstruct the pixels of the bab. No_update means that motion compensation is used to copy the bab from the previous VOP?s binary alpha map. InterCAE means the motion compensation and inter_mode CAE decoding are used to reconstruct the bab. MVDs refers to the motion vector difference for shape.

mvds_x: This is a VLC code between 1 and 18 bits. It represents the horizontal element of the motion vector difference for the bab. The motion vector difference is in full integer precision. The VLC table is shown is Table B-29.

mvds_y: This is a VLC code between 1 and 18 bits. It represents the vertical element of the motion vector difference for the bab. The motion vector difference is in full integer precision. If mvds_x is ?0?, then the VLC table of Table B-30 , otherwise the VLC table of Table B-29 is used.

conv_ratio: This is VLC code of length 1-2 bits. It specifies the factor used for sub-sampling the 16x16 pixel bab. The decoder must up-sample the decoded bab by this factor. The possible values for this factor are 1, 2 and 4 and the VLC table used is given in Table B-31.

scan_type: This is a 1-bit flag where a value of ?0? implies that the bab is in transposed form i.e. the BAB has been transposed prior to coding. The decoder must then transpose the bab back to its original form following decoding. If this flag is ?1?, then no transposition is performed.

binary_arithmetic_code(): This is a binary arithmetic decoder representing the pixel values of the bab. This code may be generated by intra cae or inter cae depending on the bab_type. Cae decoding relies on the knowledge of intra_prob[] and inter_prob[], probability tables given in annex B.

4. **Sprite coding**

warping_mv_code(dmv) : The codeword for each differential motion vector consists of a VLC indicating the length of the dmv code (dmv_length) and a FLC, dmv_code-, with dmv_length bits. The codewords are listed in Table B-33.

brightness_change_factor (): The codeword for brightness_change_factor consists of a variable length code denoting brightness_change_factor_size and a fix length code, brightness_change_factor, of brightness_change_factor_size bits (sign bit included). The codewords are listed in Table B-34.

send_mb(): This function returns 1 if the current macroblock has already been sent previously and "not coded". Otherwise it returns 0.

piece_quant: This is a 5-bit unsigned interger which indicates the quant to be used for a sprite-piece until updated by a subsequent dquant. The piece_quant carries the binary representation of quantizer values from 1 to 31 in steps of 1.

piece_width: This value specifies the width of the sprite piece measured in macroblock units.

piece_height : This value specifies the height of the sprite piece measured in macroblock units.

piece_xoffset: This value specifies the horizontal offset location, measured in macroblock units from the left edge of the sprite object, for the placement of the sprite piece into the sprite object buffer at the decoder.

piece_yoffset: This value specifies the vertical offset location, measured in macroblock units from the top edge of the sprite object.

decode_sprite_piece (): It decodes a selected region of the sprite object or its update. It also decodes the parameters required by the decoder to properly incorporate the pieces. All the static-sprite-object pieces will be encoded using a subset of the I-VOP syntax. And the static-sprite-update pieces use a subset of the P-VOP syntax. The sprite update is defined as the difference between the original sprite texture and the reconstructed sprite assembled from all the sprite object pieces.

sprite_shape_texture(): For the static-sprite-object pieces, shape and texture are coded using the macroblock layer structure in I-VOPs. And the static-sprite-update pieces use the P-VOP inter-macroblock syntax -- except that there are no motion vectors and shape information included in this syntax structure. Macroblocks raster scanning is employed to encode a sprite piece; however, whenever the scan encounters a macroblock which has been part of some previously sent sprite piece, then the block is not coded and the corresponding macroblock layer is empty.

6. **Macroblock related**

not_coded: This is a 1-bit flag which signals if a macroblock is coded or not. When set to?1? it indicates that a macroblock is not coded and no further data is included in the bitstream for this macroblock; decoder shall treat this macroblock as ?inter? with motion vector equal to zero and no DCT coefficient data. When set to ?0? it indicates that the macroblock is coded and its data is included in the bitstream.

mcbpc: This is a variable length code that is used to derive the macroblock type and the coded block pattern for chrominance . It is always included for coded macroblocks. Table B-6 and Table B-7 list all allowed codes for mcbpc in I- and P-VOPs respectively. The values of the column "MB type" in these tables are used as the variable "derived_mb_type" which is used in the respective syntax part for motion and texture decoding. In P-vops using the short video header format (i.e., when short_video_header is 1), mcbpc codes indicating macroblock type 2 shall not be used.

ac_pred_flag: This is a 1-bit flag which when set to ?1? indicates that either the first row or the first column of ac coefficients are differentially coded for intra coded macroblocks.

cbpy: This variable length code represents a pattern of non-transparent luminance blocks with at least one non intra DC transform coefficient, in a macroblock. Table B-8 - Table B-11 indicate the codes and the corresponding patterns they indicate for the respective cases of intra- and inter-MBs.

dquant: This is a 2-bit code which specifies the change in the quantizer, quant, for I- and P-VOPs. Table 6-27 lists the codes and the differential values they represent. The value of quant lies in range of 1 to $2^{\text{quant\_precision}}$ -1; if the value of quant after adding dquant value is less than 1 or exceeds $2^{\text{quant\_precision}}$-1, it shall be correspondingly clipped to 1 and $2^{\text{quant\_precision}}$-1. If quant_precision takes its default value of 5, the range of allowed values for quant is [1:31].

Table -27 -- dquant codes and corresponding values

dquant code	value
00	-1
01	-2
10	1
11	2

co_located_not_coded : The value of this internal flag is set to 1 when the current VOP is a B-VOP, the future reference VOP is a P-VOP, and the co-located macroblock in the future reference VOP is skipped (i.e. coded as not_coded = ’1’). Otherwise the value of this flag is set to 0. The co-located macroblock is the macroblock which has the same horizontal and vertical index with the current macroblock in the B-VOP. If the co-located macroblock lies outside of the bounding rectangle, this macroblock is considered to be not skipped.

modb : This is a variable length code present only in coded macroblocks of B-VOPs. It indicates whether mb_type and/or cbpb information is present for a macroblock. The codes for modb are listed in Table B-3.

mb_type : This variable length code is present only in coded macroblocks of B-VOPs. Further, it is present only in those macroblocks for which one motion vector is included. The codes for mb_type are shown in Table B-4 for B-VOPs for no scalability and in Table B-5 for B-VOPs with scalability. When mb_type is not present (i.e. modb==?1?) for a macroblock in a B-VOP, the macroblock type is set to the default type. The default macroblock type for the enhancement layer of spatially scalable bitstreams (i.e. ref_select_code == ’00’ && scalability = ’1’) is "forward mc + Q". Otherwise, the default macroblock type is "direct".

cbpb : This is a 3 to 6 bit code representing coded block pattern in B-VOPs, if indicated by modb. Each bit in the code represents a coded/no coded status of a block; the leftmost bit corresponds to the top left block in the macroblock. For each non-transparent blocks with coefficients, the corresponding bit in the code is set to ?1?. When cbpb is not present (i.e. modb==?1? or ?01?) for a macroblock in a B-VOP, no coefficients are coded for all the non-transparent blocks in this macroblock.

dbquant: This is a variable length code which specifies the change in quantizer for B-VOPs. Table 6-28 lists the codes and the differential values they represent. If

the value of quant after adding dbquant value is less than 1 or exceeds $2^{quant\_precision}$-1, it shall be correspondingly clipped to 1 and $2^{quant\_precision}$-1. If quant_precision takes its default value of 5, the range of allowed values for the quantzer for B-VOPs is [1:31].

Table -28 -- dbquant codes and corresponding values

dbquant code	**value**
10	-2
0	0
11	2

coda_i: This is a one-bit flag which is set to "1" to indicate that all the values in the grayscale alpha macroblock are equal to 255 (AlphaOpaqueValue). When set to "0", this flag indicates that one or more 8x8 blocks are coded according to cbpa.

ac_pred_flag_alpha: This is a one-bit flag which when set to ?1? indicates that either the first row or the first column of ac coefficients are to be differentially decoded for intra alpha macroblocks. It has the same effect for alpha as the corresponding luminance flag.

cbpa: This is the coded block pattern for grayscale alpha texture data. For I, P and B VOPs, this VLC is exactly the same as the INTER (P) cbpy VLC described in Table B-8 - Table B-11. cbpa is followed by the alpha block data which is coded in the same way as texture block data. Note that grayscale alpha blocks with alpha all equal to zero (transparent) are not included in the bitstream.

coda_pb : This is a VLC indicating the coding status for P or B alpha macroblocks. The semantics are given in the table below (Table 6-29). When this VLC indicates that the alpha macroblock is all opaque, this means that all values are set to 255 (AlphaOpaqueValue).

Table -29 -- coda_pb codes and corresponding values

coda_pb	**Meaning**
1	alpha residue all zero
01	alpha macroblock all opaque
00	alpha residue coded

1. **MB Binary Shape Coding**

bab_type: This defines the coding type of the current bab according to Table B-27 and Table

B-28 for intra and inter mode, respectively.

mvds_x: This defines the size of the x-component of the differential motion vector for the current bab according to Table B-29.

mvds_y: This defines the size of the y-component of the differential motion vector for the current bab according to Table B-29 if mvds_x!=0 and according to Table B-30 if mvds_x==0.

conv_ratio : This defines the upsampling factor according to Table B-31 to be applied after decoding the current shape information

scan_type: This defines according to Table 6-30 whether the current bordered to be decoded bab and the eventual bordered motion compensated bab need to be transposed

Table -30 -- scan_type

scan_type	**meaning**
0	transpose bab as in matrix transpose
1	do not transpose

binary_arithmetic_code() -This is a binary arithmetic decoder that defines the context dependent arithmetically to be decoded binary shape information. The meaning of the bits is defined by the arithmetic decoder according to subclause 7.5.3

2. **Motion vector**

horizontal_mv_data: This is a variable length code, as defined in Table B-12, which is used in motion vector decoding as described in subclause 7.6.3.

vertical_mv_data : This is a variable length code, as defined in Table B-12, which is used in motion vector decoding as described in subclause 7.6.3.

horizontal_mv_residual: This is an unsigned integer which is used in motion vector decoding as described in subclause 7.6.3. The number of bits in the bitstream for horizontal_mv_residual, r_size, is derived from either vop_fcode_forward or vop_fcode_backward as follows;

r_size = vop_fcode_forward - 1 or r_size = vop_fcode_backward - 1

vertical_mv_residual: This is an unsigned integer which is used in motion vector decoding as described in subclause 7.6.3. The number of bits in the bitstream for vertical_mv_residual, r_size, is derived from either vop_fcode_forward or vop_fcode_backward as follows;

r_size = vop_fcode_forward - 1 or r_size = vop_fcode_backward - 1

3. **Interlaced Information**

dct_type : This is a 1-bit flag indicating whether the macroblock is frame DCT coded or field DCT coded. If this flag is set to "1", the macroblock is field DCT coded; otherwise, the macroblock is frame DCT coded. This flag is only present in the bitstream if the interlaced flag is set to "1" and the macroblock is coded (coded blcok pattern is non-zero) or intra-coded. Boundary blocks are always coded in frame-based mode.

field_prediction : This is a 1-bit flag indicating whether the macroblock is field predicted or frame predicted. This flag is set to ?1? when the macroblock is predicted using field motion vectors. If it is set to ?0? then frame prediction (16x16 or 8x8) will be used. This flag is only present in the bitstream if the interlaced flag is set to "1" and the derived_mb_type is "0" or "1" in the P-VOP or an non-direct mode macroblock in the B-VOP.

forward_top_field_reference: This is a 1-bit flag which indicates the reference field for the forward motion compensation of the top field. When this flag is set to ?0?, the top field is used as the reference field. If it is set to ?1? then the bottom field will be used as the reference field. This flag is only present in the bitstream if the field_prediction flag is set to "1" and the macroblock is not backward predicted.

forward_bottom_field_reference : This is a 1-bit flag which indicates the reference field for the forward motion compensation of the bottom field. When this flag is set to ?0?, the top field is used as the reference field. If it is set to ?1? then the bottom field will be used as the reference field. This flag is only present in the bitstream if the field_prediction flag is set to "1" and the macroblock is not backward predicted.

backward_top_field_reference: This is a 1-bit flag which indicates the reference field for the backward motion compensation of the top field. When this flag is set to ?0?, the top field is used as the reference field. If it is set to ?1? then the bottom field will be used as the reference field. This flag is only present in the bitstream if the field_prediction flag is set to "1" and the macroblock is not forward predicted.

backward_bottom_field_reference: This is a 1-bit flag which indicates the reference field for the backward motion compensation of the bottom field. When this flag is set to ?0?, the top field is used as the reference field. If it is set to ?1? then the bottom field will be used as the reference field.. This flag is only present in the bitstream if the field_prediction flag is set to "1" and the macroblock is not forward predicted.

7. **Block related**

intra_dc_coefficient: This is a fixed length code that defines the value of an intra DC coefficient when the short video header format is in use (i.e., when short_video_header is "1"). It is transmitted as a fixed length unsigned integer code of size 8 bits, unless this integer has the value 255. The values 0 and 128 shall not be used - they are reserved. If the integer value is 255, this is interpreted as a signalled value of 128. The integer value is then multiplied by a dc_scaler value of 8 to produce the reconstructed intra DC coefficient value.

dct_dc_size_luminance: This is a variable length code as defined in Table B-13 that is used to derive the value of the differential dc coefficients of luminance values in blocks in intra macroblocks. This value categorizes the coefficients according to their size.

dct_dc_differential: This is a variable length code as defined in Table B-15 that is used to derive the value of the differential dc coefficients in blocks in intra macroblocks. After identifying the category of the dc coefficient in size from dct_dc_size_luminance or dct_dc_size_chrominance, this value denotes which actual difference in that category occurred.

dct_dc_size_chrominance: This is a variable length code as defined in Table B-14 that is used to derive the value of the differential dc coefficients of chrominance values in blocks in intra macroblocks. This value categorizes the coefficients according to their size.

pattern_code[i]: The value of this internal flag is set to 1 if the block or alpha block with the index value i includes one or more DCT coefficients that are decoded using at least one of Table B-16 to Table B-25. Otherwise the value of this flag is set to 0.

1. **Alpha block related**

dct_dc_size_alpha : This is a variable length code for coding the alpha block dc coefficient. Its semantics are the same as dct_dc_size_luminance in subclause 6.3.7.

8. **Still texture object**

still_texture_object_start_code: The still_texture_object_start_code is a string of 32 bits. The first 24 bits are ?0000 0000 0000 0000 0000 0001? and the last 8 bits are defined in Table 6-3.

texture_object_id: This is given by 16-bits representing one of the values in the range of ?0000 0000 0000 0000? to ?1111 1111 1111 1111? in binary. The texture_object_layer_id uniquely identifies a texture object layer.

wavelet_filter_type : This field indicates the arithmetic precision which is used for the wavelet decomposition as the following:

Table -31 -- Wavelet type

wavelet_filter_type	**Meaning**
0	integer
1	Double float

wavelet_download: This field indicates if the 2-band filter bank is specificed in the bitstream:

Table -32 -- Wavelet downloading flag

wavelet_download	**meaning**
0	default filters
1	specified in bitstream

The default filter banks are described in subclause B.2.2.

wavelet_decomposition_levels: This field indicates the number of levels in the wavelet decomposition of the texture.

scan_direction: This field indicates the scan order of AC coefficients. In single-quant and multi-quant mode, if this flag is ‘0?, then the coefficients are scanned in the tree-depth fashion. If it is ‘1?, then they are scanned in the subband by subband fashion. In bilevel_quant mode, if the flag is ‘0?, then they are scanned in bitplane by bitplane fashion. Within each bitplane, they are scanned in a subband by subband fashion. If it is "1", they are scanned from the low wavelet decomposition layer to high wavelet decomposition layer. Within each wavelet decomposition layer, they are scanned from most significant bitplane down to the least significant bitplane.

start_code_enable: If this flag is enabled (disable =0; enabled = 1), the start code followed by an ID to be inserted in to each spatial scalability layer and/or each SNR scalability layer.

texture_object_layer_shape: This is a 2-bit integer defined in Table 6-33. It identifies the shape type of a texture object layer.

Table -33 -- Texture Object Layer Shape type

texture_object_layer_shape	**Meaning**
00	rectangular
01	binary
10	reserved
11	reserved

quantization_type: This field indicates the type of quantization as shown in Table 6-34.

Table -34 -- The quantization type

quantization_type	**Code**
single quantizer	01
multi quantizer	10
bi-level quantizer	11

spatial_scalability_levels: This field indicates the number of spatial scalability layers supported in the bitstream. This number can be from 1 to wavelet_decomposition_levels.

use_default_spatial_scalability : This field indicates how the spatial scalability levels are formed. If its value is one, then default spatial scalability is used, starting from (¼)^(spatial_scalability_levels-1)-th of the full resolution up to the full resolution, where ^ is a power operation. If its value is zero, the spatial scalability is specified by wavelet_layer_index described below.

wavelet_layer_index : This field indicates the identification number of wavelet_decomposition layer used for spatial scalability. The index starts with 0 (i.e., root_band) and ends at (wavelet_decomposition_levels-1) (i.e., full resolution).

uniform_wavelet_filter: If this field is "1", then the same wavelet filter is applied for all wavelet layers. If this field is "0", then different wavelet filters may be applied for the wavelet decomposition. Note that the same filters are used for both luminance and chromanence. Since the chromanence?s width and height is half that of the luminance, the last wavelet filter applied to the luminance is skipped when the chromanence is synthesized.

wavelet_stuffing : These 3 stuffing bits are reserved for future expansion. It is currently defined to be ?111?.

texture_object_layer_width: The texture_object_layer_width is a 15-bit unsigned integer representing the width of the displayable part of the luminance component in pixel units. A zero value is forbidden.

texture_object_layer_height: The texture_object_layer_height is a 15-bit unsigned integer representing the height of the displayable part of the luminance component in pixel units. A zero value is forbidden.

horizontal_ref : This is a 15-bit integer which specifies, in pixel units, the horizontal position of the top left of the rectangle defined by horizontal size of object_width. The value of horizontal_ref shall be divisible by two. This is used for decoding and for picture composition.

vertical_ref : This is a 15-bit integer which specifies, in pixel units, the vertical position of the top left of the rectangle defined by vertical size of object_height. The value of vertical_ref shall be divisible by two. This is used for decoding and for picture composition.

object_width: This is a 15-bit unsigned integer which specifies the horizontal size, in pixel units, of the rectangle that includes the object. A zero value is forbidden.

object_height: This is a 15-bit unsigned integer which specifies the vertical size, in pixel units, of the rectangle that includes the object. A zero value is forbidden.

quant_byte : This field defines one byte of the quantization step size for each scalability layer. A zero value is forbidden. The quantization step size parameter, quant, is decoded using the function get_param( ) : quant = get_param(7);

max_bitplanes : This field indicates the number of maximum bitplanes in bilevel_quant mode.

1. **Texture Layer Decoding**

   tree_blocks : The tree block is that wavelet coefficients are organized in a tree structure which is rooted in the low-low band (DC band) of the wavelet decomposition, then extends into the higher frequency bands at the same spatial location. Note the DC band is encoded separately.

   spatial_layers : This field is equivalent to the maximum number of the wavelet decomposition layers in that scalability layer.

   arith_decode_highbands_td() : This is an arithmetic decoder for decoding the quantized coefficient values of the higher bands (all bands except DC band) within a single tree block. The bitstream is generated by an adaptive arithmetic encoder. The arithmetic decoding relies on the initialization of the uniform probability distribution models described in subclause B.2.2. This decoder uses only integer arithmetic. It also uses an adaptive

probability model based on the frequency counts of the previously decoded symbols. The maximum range (or precision) specified is (2^16) - 1 (16 bits). The maximum frequency count for the magnitude and residual models is 127, and for all other models it is 127. The arithmetic coder used is identical to the one used in arith_decode_highbands_bilevel_td().

texture_spatial_layer_start_code: The texture_spatial_layer_start_code is a string of 32 bits. The 32 bits are ?0000 0000 0000 0000 0000 0001 1011 1111? in binary. The texture_spatial_layer_start_code marks the start of a new spatial layer.

texture_spatial_layer_id : This is given by 5-bits representing one of the values in the range of ?00000? to ?11111? in binary. The texture_spatial_layer_id uniquely identifies a spatial layer.

arith_decode_highbands_bb() : This is an arithmetic decoder for decoding the quantized coefficient values of the higher bands (all bands except DC band) within a single band. The bitstream is generated by an adaptive arithmetic encoder. The arithmetic decoding relies on the initialization of the uniform probability distribution models described in subclause B.2.2. This decoder uses arithmetic. It also uses an adaptive probability model based on the frequency counts of the previously decoded symbols. The maximum range (or precision) specified is (2^16) - 1 (16 bits). The maximum frequency count for the magnitude and residual models is 127, and for all other models it is 127.

snr_scalability_levels : This field indicates the number of levels of SNR scalability supported in this spatial scalability level.

texture_snr_layer_start_code: The texture_snr_layer_start_code is a string of 32 bits. The 32 bits are ?0000 0000 0000 0000 0000 0001 1100 0000? in binary. The texture_snr_layer_start_code marks the start of a new snr layer.

texture_snr_layer_id: This is given by 5-bits representing one of the values in the range of ?00000? to ?11111? in binary. The texture_snr_layer_id uniquely identifies an SNR layer.

NOTE All the start codes start at the byte boundary. Appropriate number of bits is stuffed before any start code to byte-align the bitstream.

all_nonzero : This flag indicates whether some of the subbands of the current layer contain only zero coefficients. The value ?0? for this flag indicates that one or more of the subbands contain only zero coefficients. The value ?1? for this flag indicates the all the subbands contain some nonzero coefficients

all_zero: This flag indicates whether all the coefficients in the current layer are zero or not. The value ?0? for this flag indicates that the layer contains some nonzero coefficients. The value ?1? for this flag indicates that the layer only contains zero coefficients, and therefore the layer is skipped.

lh_zero, hl_zero, hh_zero : This flag indicates whether the LH/HL/HH subband of the current layer contains only all zero coefficients. The value ?1? for this flag indicates that the LH/HL/HH subband contains only zero coefficients, and therefore the subband is skipped. The value ?0? for this flag indicates that the LH/HL/HH subband contains some nonzero coefficients

arith_decode_highbands_bilevel_bb() : This is an arithmetic decoder for decoding the quantized coefficient values of the higher bands in the bilevel_quant mode (all bands except DC band). The bitstream is generated by an adaptive arithmetic encoder. The arithmetic decoding relies on the initialization of the uniform probability distribution models described. The arith_decode_highbands_bilevel() function uses bitplane scanning, and a different probability model as described in subclause B.2.2. In this mode, The maximum range (or precision) specified is (2^16) - 1 (16 bits). The maximum frequency count is 127. It uses the lh/hl/hh_zero flags to see if any of the LH/HL/HH are all zero thus not decoded . For example if lh_zero=1 and hh_zero=1 only hl_zero is decoded.

arith_decode_highbands_bilevel_td() : This is an arithmetic decoder for decoding the quantized coefficient values of the higher bands in the bilevel_quant mode (all bands except DC band). The bitstream is generated by an adaptive arithmetic encoder. The arithmetic decoding relies on the initialization of the uniform probability distribution models described. The arith_decode_highbands_bilevel() function uses bitplane scanning, and a different probability model as described in subclause B.2.2. In this mode, The maximum range (or precision) specified is (2^16) - 1 (16 bits). The maximum frequency count is 127. It uses the lh/hl/ll_zero flags to see if any of the LH/HL/HH are all zero thus not decoded. For example if lh_zero=1 and hh_zero=1 only hl_zero is decoded.

lowpass_filter_length: This field defines the length of the low pass filter in binary ranging from "0001" (length of 1) to "1111" (length of 15.)

highpass_filter_length: This field defines the length of the high pass filter in binary ranging from "0001" (length of 1) to "1111" (length of 15.)

filter_tap_integer: This field defines an integer filter coefficient in a 16 bit signed integer. The filter coefficients are decoded from the left most tap to the right most tap order.

filter_tap_float_high: This field defines the left 16 bits of a floating filter coefficient which is defined in 32-bit IEEE floating format. The filter coefficients are decoded from the left most tap to the right most tap order.

filter_tap_float_low: This field defines the right 16 bits of a floating filter coefficient which is defined in 32-bit IEEE floating format. The filter coefficients are decoded from the left most tap to the right most tap order.

integer_scale: This field defines the scaling factor of the integer wavelet, by which the output of each composition level is divided by an integer division operation. A zero value is forbidden.

mean: This field indicates the mean value of one color component of the texture.

quant_dc_byte: This field indicates the quantization step size for one color component of the DC subband. A zero value is forbidden. The quantization step size parameter, quant_dc, is decoded using the function get_param( ): quant = get_param(7);

band_offset_byte : This field defines one byte of the absolute value of the parameter band_offset. This parameter is added to each DC band coefficient obtained by arithmetic decoding. The parameter band_offset is decoded using the function get_param( ):

band_offset = -get_param(7);

where function get_param() is defined as

```
int get_param(int nbit)
{
int count = 0;
int word =0;
int value = 0;
int module = 1<<(nbit);

do{
word= get_next_word_from_bitstream( nbit+1);
value += (word & (module-1) ) << (count * nbit);
count ++;
} while( word>> nbit);
```

```
 return value;

 }
```

The function get_next_word_from_bitstream(x) reads the next x bits from the input bitstream.

band_max_byte : This field defines one byte of the maximum value of the DC band. The parameter band_max_value is decoded using function get_param( ):

band_max_value = get_param(7);

arith_decode_dc(): This is an arithmetic decoder for decoding the quantized coefficient values of DC band only. No zerotree symbol is decoded since the VAL is assumed for all DC coefficient values. This bitstream is generated by an adaptive arithmetic encoder. The arithmetic decoding relies on the initialization of a uniform probability distribution model described in subclause B.2.2. The arith_decode_dc() function uses the same arithmetic decoder as described in arith_decode_highbands_td() but it uses different scanning, and a different probability model (*DC*).

root_max_alphabet_byte : This field defines one byte of the maximum absolute value of the quantized coefficients of the three lowest AC bands. This parameter is decoded using the function get_param( ):

root_max_alphabet = get_param (7);

valz_max_alphabet_byte-- This field defines one byte of the maximum absolute value of the quantized coefficients of the 3 highest AC bands. The parameter valz_max is decoded using the function get_param( ):

valz_max_alphabet = get_param (7);

valnz_max_alphabet_byte : This field defines one byte of the maximum absolute value of the quantized coefficients which belong to the middle AC bands (the bands between the 3 lowest and the 3 highest AC bands). The parameter valnz_max_alphabet is decoded using the function get_param( ):

valnz_max_alphabet = get_param (7);

2. **Shape Object decoding**

change_conv_ratio_disable: This specifies whether conv_ratio is encoded at the shape object decoding function. If it is set to "1" when disable.

sto_constant_alpha: This is a 1-bit flag when set to ?1?, the opaque alpha values of the binary mask are replaced with the alpha value specified by sto_constant_alpha_value.

sto_constant_alpha_value: This is an 8-bit code that gives the alpha value to replace the opaque pixels in the binary alpha mask. Value ?0? is forbidden.

bab_type: This is a variable length code of 1-2 bits. It indicates the coding mode used for the bab. There are three bab_types as depicted in Table 6-35. The VLC tables used depend on the decoding context i.e. the bab_types of blocks already received.

Table -35 -- List of bab_types and usage

bab_type	Semantic	code
2	transparent	10
3	opaque	0
4	intraCAE	11

The bab_type determines what other information fields will be present for the bab shape. No further shape information is present if the bab_type = 2 or 3. opaque means that all pixels of the bab are part of the object. transparent means that none of the bab pixels belong to the object. IntraCAE means the intra-mode CAE decoding will be required to reconstruct the pixels of the bab.

conv_ratio: This is VLC code of length 1-2 bits. It specifies the factor used for sub-sampling the 16x16 pixel bab. The decoder must up-sample the decoded bab by this factor. The possible values for this factor are 1, 2 and 4 and the VLC table used is given in Table B-31.

scan_type: This is a 1-bit flag where a value of ?0? implies that the bab is in transposed form i.e. the bab has been transposed prior to coding. The decoder must then transpose the bab back to its original form following decoding. If this flag is ?1?, then no transposition is performed.

binary_arithmetic_decode(): This is a binary arithmetic decoder representing the pixel values of the bab. Cae decoding relies on the knowledge of intra_prob[], probability tables given in annex B.

9. **Mesh object**

mesh_object_start_code: The mesh_object_start_code is the bit string ?000001BC? in hexadecimal. It initiates a mesh object.

1. **Mesh object plane**

mesh_object_plane_start_code : The mesh_object_plane_start_code is the bit string ?000001BD? in hexadecimal. It initiates a mesh object plane.

is_intra: This is a 1-bit flag which when set to ?1? indicates that the mesh object is coded in intra mode. When set to ?0? it indicates that the mesh object is coded in predictive mode.

2. **Mesh geometry**

mesh_type_code: This is a 2-bit integer defined in Table 6-36. It indicates the type of initial mesh geometry to be decoded.

Table -36 -- Mesh type code

mesh type code	mesh geometry
00	forbidden
01	uniform
10	Delaunay
11	reserved

nr_of_mesh_nodes_hor: This is a 10-bit unsigned integer specifying the number

of nodes in one row of a uniform mesh.

nr_of_mesh_nodes_vert : This is a 10-bit unsigned integer specifying the number of nodes in one column of a uniform mesh.

mesh_rect_size_hor: This is a 8-bit unsigned integer specifying the width of a rectangle of a uniform mesh (containing two triangles) in half pixel units.

mesh_rect_size_vert: This is a 8-bit unsigned integer specifying the height of a rectangle of a uniform mesh (containing two triangles) in half pixel units.

triangle_split_code: This is a 2-bit integer defined in Table 6-37. It specifies how rectangles of a uniform mesh are split to form triangles.

Table -37 -- Specification of the triangulation type

triangle split code	**Split**
00	top-left to right bottom
01	bottom-left to top right
10	alternately top-left to bottom-right and bottom-left to top-right
11	alternately bottom-left to top-right and top-left to bottom-right

nr_of_mesh_nodes: This is a 16-bit unsigned integer defining the total number of nodes (vertices) of a (non-uniform) Delaunay mesh. These nodes include both interior nodes as well as boundary nodes.

nr_of_boundary_nodes : This is a 10-bit unsigned integer defining the number of nodes (vertices) on the boundary of a (non-uniform) Delaunay mesh.

node0_x: This is a 13-bit signed integer specifying the x-coordinate of the first boundary node (vertex) of a mesh in half-pixel units with respect to a local coordinate system.

node0_y: This is a 13-bit signed integer specifying the y-coordinate of the first boundary node (vertex) of a mesh in half-pixel units with respect to a local coordinate system.

delta_x_len_vlc: This is a variable-length code specifying the length of the delta_x code that follows. The delta_x_len_vlc and delta_x codes together specify the difference between the x-coordinates of a node (vertex) and the previously encoded node (vertex). The definition of the delta_x_len_vlc and

delta_x codes are given in Table B-33, the table for sprite motion trajectory coding.

delta_x: This is an integer that defines the value of the difference between the x-coordinates of a node (vertex) and the previously encoded node (vertex) in half pixel units. The number of bits in the bitstream for delta_x is delta_x_len_vlc.

delta_y_len_vlc: This is a variable-length code specifying the length of the delta_y code that follows. The delta_y_len_vlc and delta_y codes together specify the difference between the y-coordinates of a node (vertex) and the previously encoded node (vertex). The definition of the delta_y_len_vlc and delta_y codes are given in Table B-33, the table for sprite motion trajectory coding.

delta_y: This is an integer that defines the value of the difference between the y-coordinates of a node (vertex) and the previously encoded node (vertex) in half pixel units. The number of bits in the bitstream for delta_y is delta_y_len_vlc.

3. Mesh motion

motion_range_code: This is a 3-bit integer defined in Table 6-38. It specifies the dynamic range of motion vectors in half pel units.

Table -38 -- motion range code

motion range code	motion vector range
1	[-32, 31]
2	[-64, 63]
3	[-128, 127]
4	[-256, 255]
5	[-512, 511]
6	[-1024, 1023]
7	[-2048, 2047]

node_motion_vector_flag: This is a 1 bit code specifying whether a node has a zero motion vector. When set to ?1? it indicates that a node has a zero motion vector, in which case the motion vector is not encoded. When set to ?0?, it indicates the node has a nonzero motion vector and that motion vector data shall follow.

delta_mv_x_vlc : This is a variable-length code defining (together with delta_mv_x_res) the value of the difference in the x-component of the motion

vector of a node compared to the x-component of a predicting motion vector. The definition of the delta_mv_x_vlc codes are given in Table B-12, the table for motion vector coding (MVD). The value delta_mv_x_vlc is given in half pixel units.

delta_mv_x_res : This is an integer which is used in mesh node motion vector decoding using an algorithm equivalent to that described in the section on video motion vector decoding, subclause 7.6.3. The number of bits in the bitstream for delta_mv_x_res is motion_range_code-1.

delta_mv_y_vlc : This is a variable-length code defining (together with delta_mv_y_res) the value of the difference in the y-component of the motion vector of a node compared to the y-component of a predicting motion vector. The definition of the delta_mv_y_vlc codes are given in Table B-12, the table for motion vector coding (MVD). The value delta_mv_y_vlc is given in half pixel units.

delta_mv_y_res : This is an integer which is used in mesh node motion vector decoding using an algorithm equivalent to that described in the section on video motion vector decoding, subclause 7.6.3. The number of bits in the bitstream for delta_mv_y_res is motion_range_code-1.

10. Face object

fba_object_start_code: The fba_object_start_code is the bit string ?000001BA? in hexadecimal. It initiates a face object.

fba_object_coding_type: This is a 1-bit integer indicating which coding method is used. Its meaning is described in Table 6-39.

Table -39 -- fba_object_coding_type

type value	Meaning
0	predictive coding
1	DCT (face_object_plane_group)

fba_suggested_gender: This is a 1-bit integer indicating the suggested gender for the face model. It does not bind the decoder to display a facial model of suggested gender, but indicates that the content would be more suitable for display with the facial model of indicated gender, if the decoder can provide one. If fba_suggested_gender is 1, the suggested gender is male, otherwise it is female.

1. Face object plane

face_paramset_mask : This is a 2-bit integer defined in Table 6-40. It

indicates whether FAP data are present in the face_frame.

Table -40 -- Face parameter set mask

mask value	Meaning
00	unused
01	FAP present
10	reserved
11	reserved

face_object_plane_start_code : The face_frame_start_code is the bit string ?000001BB? in hexadecimal. It initiates a face object plane.

is_frame_rate: This is a 1-bit flag which when set to ?1? indicates that frame rate information follows this bit field. When set to ?0? no frame rate information follows this bit field.

is_time_code: This is a 1-bit flag which when set to ?1? indicates that time code information follows this bit field. When set to ?0? no time code information follows this bit field.

time_code : This is a 18-bit integer containing the following: time_code_hours, time_code_minutes, marker_bit and time_code_seconds as shown in Table 6-41. The parameters correspond to those defined in the IEC standard publication 461 for "time and control codes for video tape recorders". The time code specifies the modulo part (i.e. the full second units) of the time base for the current object plane.

Table -41 -- Meaning of time_code

time_code	range of value	No. of bits	Mnemonic
time_code_hours	0 - 23	5	uimsbf
time_code_minutes	0 - 59	6	uimsbf
marker_bit	1	1	bslbf
time_code_seconds	0 - 59	6	uimsbf

skip_frames : This is a 1-bit flag which when set to ?1? indicates that information follows this bit field that indicates the number of skipped frames. When set to ?0? no such information follows this bit field.

fap_mask_type: This is a 2-bit integer. It indicates if the group mask will be present for the specified fap group, or if the complete faps will be present;

its meaning is described in Table 6-42. In the case the type is ?10? the ?0? bit in the group mask indicates interpolate fap.

Table -42 -- fap mask type

mask type	Meaning
00	no mask nor fap
01	group mask
10	group mask?
11	fap

fap_group_mask [group_number]: This is a variable length bit entity that indicates, for a particular group_number which fap is represented in the bitstream. The value is interpreted as a mask of 1-bit fields. A 1-bit field in the mask that is set to ?1? indicates that the corresponding fap is present in the bitstream. When that 1-bit field is set to ?0? it indicates that the fap is not present in the bitstream. The number of bits used for the fap_group_mask depends on the group_number, and is given in Table 6-43.

Table -43 -- fap group mask bits

group_number	No. of bits
1	2
2	16
3	12
4	8
5	4
6	5
7	3
8	10
9	4
10	4

NFAP[group_number] **:** This indicates the number of FAPs in each FAP group. Its values are specified in the following table:

Table -44 -- NFAP definition

group_number	NFAP[group_number]
1	2
2	16
3	12
4	8
5	4
6	5
7	3
8	10
9	4
10	4

fap_quant : This is a 5-bit unsigned integer which is the quantization scale factor used to compute the FAPi table step size.

is_i_new_max: This is a 1-bit flag which when set to ?1? indicates that a new set of maximum range values for I frame follows these 4, 1-bit fields.

is_i_new_min: This is a 1-bit flag which when set to ?1? indicates that a new set of minimum range values for I frame follows these 4, 1-bit fields.

is_p_new_max: This is a 1-bit flag which when set to ?1? indicates that a new set of maximum range values for P frame follows these 4, 1-bit fields.

is_p_new_min: This is a 1-bit flag which when set to ?1? indicates that a new set of minimum range values for P frame follows these 4, 1-bit fields.

2. **Face Object Prediction**

   skip_frames : This is a 1-bit flag which when set to ?1? indicates that information follows this bit field that indicates the number of skipped frames. When set to ?0? no such information follows this bit field.

3. **Decode frame rate and frame skip**

   frame_rate: This is an 8 bit unsigned integer indicating the reference frame rate of the sequence.

   seconds: This is a 4 bit unsigned integer indicating the fractional reference frame rate. The frame rate is computed as follows frame rate = (frame_rate + seconds/16).

frequency_offset: This is a 1-bit flag which when set to ?1? indicates that the frame rate uses the NTSC frequency offset of 1000/1001. This bit would typically be set when frame_rate = 24, 30 or 60, in which case the resulting frame rate would be 23.97, 29.94 or 59.97 respectively. When set to ?0? no frequency offset is present. I.e. if (frequency_offset ==1) frame rate = (1000/1001) * (frame_rate + seconds/16).

number_of_frames_to_skip: This is a 4-bit unsigned integer indicating the number of frames skipped. If the number_of_frames_to skip is equal to 15 (pattern "1111") then another 4-bit word follows allowing to skip up to 29 frames(pattern "11111110"). If the 8-bits pattern equals "11111111", then another 4-bits word will follow and so on, and the number of frames skipped is incremented by 30. Each 4-bit pattern of ?1111? increments the total number of frames to skip with 15.

4. **Decode new minmax**

i_new_max[j]: This is a 5-bit unsigned integer used to scale the maximum value of the arithmetic decoder used in the I frame.

i_new_min[j]: This is a 5-bit unsigned integer used to scale the minimum value of the arithmetic decoder used in the I frame.

p_new_max[j]: This is a 5-bit unsigned integer used to scale the maximum value of the arithmetic decoder used in the P frame.

p_new_min[j]: This is a 5-bit unsigned integer used to scale the minimum value of the arithmetic decoder used in the P frame.

5. **Decode viseme and expression**

viseme_def: This is a 1-bit flag which when set to ?1? indicates that the mouth FAPs sent with the viseme FAP may be stored in the decoder to help with FAP interpolation in the future.

expression_def: This is a 1-bit flag which when set to ?1? indicates that the FAPs sent with the expression FAP may be stored in the decoder to help with FAP interpolation in the future.

6. **Face object plane group**

face_object_plane_start_code: Defined in subclause 6.3.10.1.

is_intra: This is a 1-bit flag which when set to ?1? indicates that the face object is coded in intra mode. When set to ?0? it indicates that the face object is coded in predictive mode.

face_paramset_mask: Defined in subclause 6.3.10.1.

is_frame_rate: Defined in subclause 6.3.10.1.

is_time_code: Defined in subclause 6.3.10.1.

time_code: Defined in subclause 6.3.10.1.

skip_frames: Defined in subclause 6.3.10.1.

Fap_quant_index: This is a 5-bit unsigned integer used as the index to a fap_scale table for computing the quantization step size of DCT coefficients. The value of fap_scale is specified in the following list:

fap_scale[0 - 31] = { 1, 1, 2, 3, 5, 7, 8, 10, 12, 15, 18, 21, 25, 30, 35, 42,

50, 60, 72, 87, 105, 128, 156, 191, 234, 288, 355, 439, 543, 674, 836, 1039}

fap_mask_type: Defined in subclause 6.3.10.1.

fap_group_mask [group_number] : Defined in subclause 6.3.10.1.

7. **Face Object Group Prediction**

skip_frames: See the definition in subclause 6.3.10.1.

8. **Decode frame rate and frame skip**

frame_rate: See the definition in subclause 6.3.10.3.

frequency_offset: See the definition in subclause 6.3.10.3.

number_of_frames_to_skip : See the definition in subclause 6.3.10.3.

9. **Decode viseme_segment and expression_segment**

viseme_segment_select1q[k]: This is the quantized value of viseme_select1 at frame k of a viseme FAP segment.

viseme_segment_select2q[k]: This is the quantized value of viseme_select2 at frame k of a viseme FAP segment.

viseme_segment_blendq[k]: This is the quantized value of viseme_blend at frame k of a viseme FAP segment.

viseme_segment_def[k]: This is a 1-bit flag which when set to ?1? indicates

that the mouth FAPs sent with the viseme FAP at frame k of a viseme FAP segment may be stored in the decoder to help with FAP interpolation in the future.

viseme_segment_select1q_diff[k] : This is the prediction error of viseme_select1 at frame k of a viseme FAP segment.

viseme_segment_select2q_diff[k] : This is the prediction error of viseme_select2 at frame k of a viseme FAP segment.

viseme_segment_blendq_diff[k] : This is the prediction error of viseme_blend at frame k of a viseme FAP segment.

expression_segment_select1q[k] : This is the quantized value of expression_select1 at frame k of an expression FAP segment.

expression_segment_select2q[k] : This is the quantized value of expression_select2 at frame k of an expression FAP segment.

expression_segment_intensity1q[k] : This is the quantized value of expression_intensity1 at frame k of an expression FAP segment

expression_segment_intensity2q[k] : This is the quantized value of expression_intensity2 at frame k of an expression FAP segment

expression_segment_select1q_diff[k] : This is the prediction error of expression_select1 at frame k of an expression FAP segment.

expression_segment_select2q_diff[k] : This is the prediction error of expression_select2 at frame k of an expression FAP segment.

expression_segment_intensity1q_diff[k] : This is the prediction error of expression_intensity1 at frame k of an expression FAP segment.

expression_segment_intensity2q_diff[k] : This is the prediction error of expression_intensity2 at frame k of an expression FAP segment.

expression_segment_init_face[k] : This is a 1-bit flag which indicates the value of init_face at frame k of an expression FAP segment.

expression_segment_def[k] : This is a 1-bit flag which when set to ?1? indicates that the FAPs sent with the expression FAP at frame k of a viseme FAP segment may be stored in the decoder to help with FAP interpolation in the future.

10. **Decode i_dc, p_dc, and ac**

dc_q: This is the quantized DC component of the DCT coefficients. For an intra FAP segment, this component is coded as a signed integer of either 16 bits or 31 bits. The DCT quantization parameters of the 68 FAPs are specified in the following list:

DCTQP[1 - 68] = {1, 1, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5,

7.5, 7.5, 7.5, 15, 15, 15, 15, 5, 10, 10,

10, 10, 425, 425, 425, 425, 5, 5, 5, 5,

7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 20, 20,

20, 20, 10, 10, 10, 10, 255, 170, 255, 255,

7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5,

15, 15, 15, 15, 10, 10, 10, 10}

For DC coefficients, the quantization stepsize is obtained as follows:

qstep[i] = fap_scale[fap_quant_inex] * DCTQP[i] ÷ 3.0

dc_q_diff: This is the quantized prediction error of a DC coefficient of an inter FAP segment. Its value is computed by subtracting the decoded DC coefficient of the previous FAP segment from the DC coefficient of the current FAP segment. It is coded by a variable length code if its value is within [-255, +255]. Outside this range, its value is coded by a signed integer of 16 or 32 bits.

count_of_runs: This is the run length of zeros preceding a non-zero AC coefficient.

ac_q[i][next]: This is a quantized AC coefficients of a segment of FAPi. For AC coefficients, the quantization stepsize is three times larger than the DC quantization stepsize and is obtained as follows:

qstep[i] = fap_scale[fap_quant_inex] * DCTQP[i]

1. **The visual decoding process**

   This clause specifies the decoding process that the decoder shall perform to recover visual data from the coded bitstream. As shown in Figure 7-1, the visual decoding process includes several decoding processes such as shape-motion-texture decoding, still texture decoding, mesh decoding, and face decoding processes. After decoding the coded bitstream, it is then sent to the compositor to integrate various visual objects.

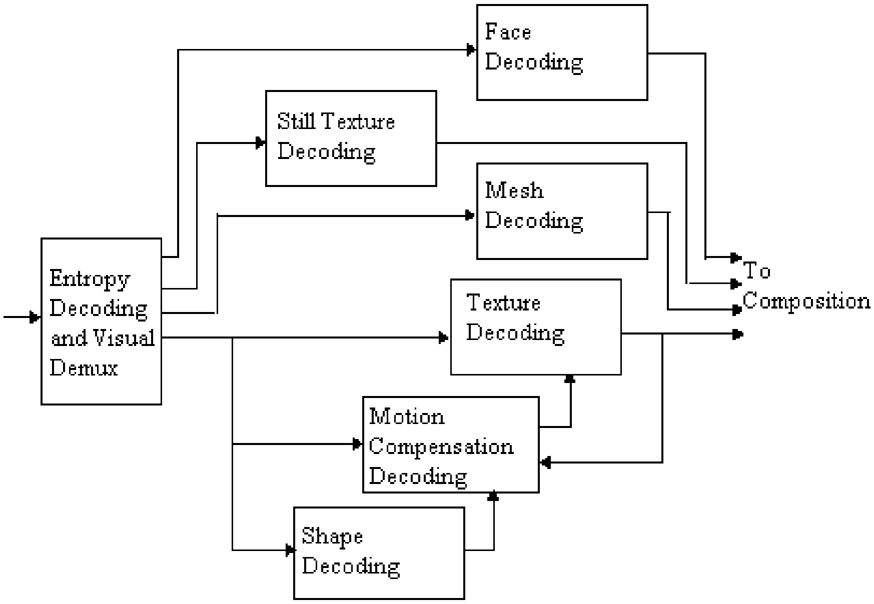


Figure -1 -- A high level view of basic visual decoding; specialized decoding such as scalable, sprite and error resilient decoding are not shown

In subclauses 7.1 through 7.9 the VOP decoding process is specified in which shape, motion, texture decoding processes are the major contents. The still texture object decoding is described in subclauses 7.10. Subclause 7.11 includes the mesh decoding process, and subclause 7.12 features the face object decoding process. The output of the decoding process is explained in subclause 7.13.

1. Video decoding process

This subclause specifies the decoding process that a decoder shall perform to recover VOP data from the coded video bitstream.

With the exception of the Inverse Discrete Cosine Transform (IDCT) the decoding process is defined such that all decoders shall produce numerically identical results. Any decoding process that produces identical results to the process described here, by definition, complies with this part of ISO/IEC 14496.

The IDCT is defined statistically such that different implementations for this function are allowed. The IDCT specification is given in annex A.

Figure 7-2 is a diagram of the Video Decoding Process without any scalability feature. The diagram is simplified for clarity. The same decoding scheme is applied when decoding all the VOPs of a given session

NOTE Throughout this part of ISO/IEC 14496 two dimensional arrays are represented as *name*[q][p] where ?q? is the index in the vertical dimension and ?p? the index in the horizontal dimension.

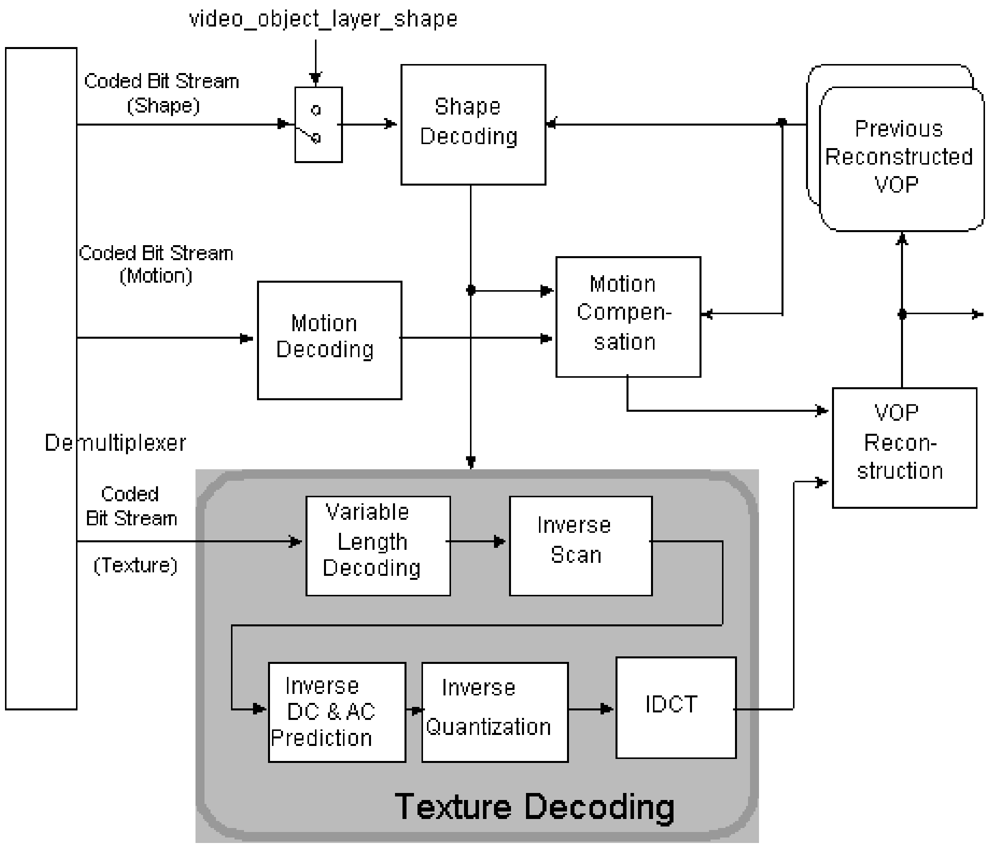


Figure -2 -- Simplified Video Decoding Process

The decoder is mainly composed of three parts: shape decoder, motion decoder and texture decoder. The reconstructed VOP is obtained by combining the decoded shape, texture and motion information.

2. **Higher syntactic structures**

The various parameters and flags in the bitstream for VideoObjectLayer(), Group_of_VideoObjectPlane(), VideoObjectPlane(), video_plane_with_short_header(), macroblock() and block(), as well as other syntactic structures related to them shall be interpreted as discussed earlier. Many of these parameters and flags affect the decoding process. Once all the macroblocks in a given VOP have been processed, the entire VOP will have been reconstructed. In case the bitstream being decoded contains B-VOPs, reordering of VOPs may be needed as discussed in subclause 6.1.3.7.

3. **VOP reconstruction**

The luminance and chrominance values of a VOP from the decoded texture and motion information are reconstructed as follows:

1. In case of INTRA macroblocks, the luminance and chrominance values f[y][x] from the decoded texture data form the luminance and chrominance values of the VOP: d[y][x] = f[y][x].

1. In case of INTER macroblocks, first the prediction values p[y][x] are calculated using the decoded motion vector information and the texture information of the respective reference VOPs. Then, the decoded texture data f[y][x] is added to the prediction values, resulting in the final luminance and chrominance values of the VOP: d[y][x] = p[y][x] + f[y][x]

1. Finally, the calculated luminance and chrominance values of the reconstructed VOP are saturated so that

0 £ d[y][x] £ $2^{bits\_per\_pixel}$ - 1 , for all x, y.

1. Texture decoding

This subclause describes the process used to decode the texture information of a VOP. The process of video texture decoding is given in Figure 7-3.

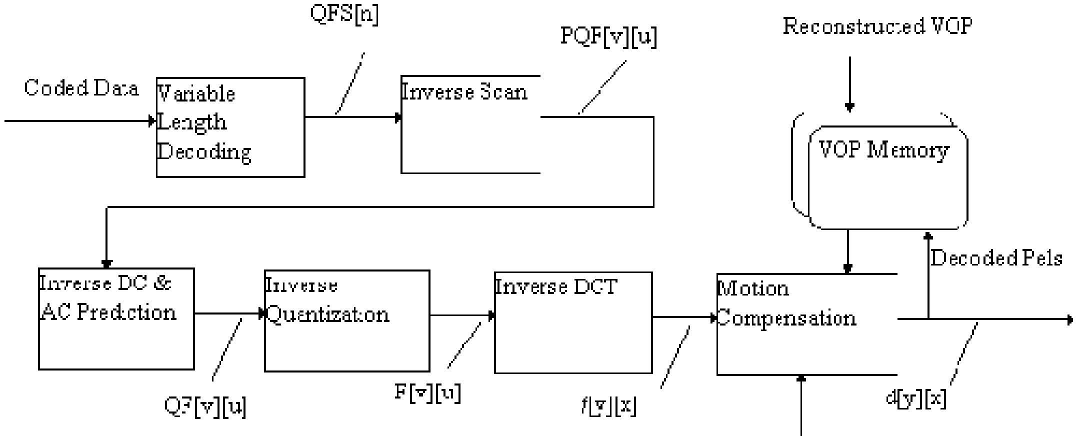


Figure -3 -- Video Texture Decoding Process

1. Variable length decoding

This subclause explains the decoding process. Subclause 7.4.1.1 specifies the process used for the DC coefficients (n=0) in an intra coded block. (n is the index of the coefficient in the appropriate zigzag scan order). Subclause 7.4.1.2 specifies the decoding process for all other coefficients; AC coefficients ($n \neq 0$) and DC coefficients in non-intra coded blocks.

1. DC coefficients decoding in intra blocks

Differential DC coefficients in blocks in intra macroblocks are encoded as variable length code denoting dct_dc_size as defined in Table B-13 and Table B-14 in annex B, and a fixed length code dct_dc_differential (Table B-15). The dct_dc_size categorizes the dc coefficients according to their "size". For each category additional bits are appended to the dct_dc_size code to uniquely identify which difference in that category actually occurred (Table B-15). This is done by appending a fixed length code, dct_dc_differential, of dct_dc_size bits. The final value of the decoded dc coefficient is the sum of this differential dc value and the predicted value.

When short_video_header is 1, the dc coefficient of an intra block is not coded differentially. It is instead transmitted as a fixed length unsigned integer code of size 8 bits, unless this integer has the value 255. The values 0 and 128 shall not be used - they are reserved. If the integer value is 255, this is interpreted as a signaled value of 128.

2. Other coefficients

The ac coefficients are obtained by decoding the variable length codes to produce EVENTs. An EVENT is a combination of a last non-zero coefficient indication (LAST; "0": there are more nonzero coefficients in this block, "1": this is the last nonzero coefficient in this block), the number of successive zeros preceding the coded coefficient (RUN), and the non-zero value of the coded coefficient (LEVEL).

When short_video_header is 1, the most commonly occurring EVENTS are coded with the variable length codes given in Table B-17 (for all coefficients other than intra DC whether in intra or inter blocks). The last bit "s" denotes the sign of level, "0" for positive and "1" for negative.

When short_video_header is 0, the variable length code table is different for intra blocks and inter blocks. The most commonly occurring EVENTs for the luminance and chrominance components of intra blocks in this case are decoded by referring to Table B-16. The most commonly occurring EVENTs for the luminance and chrominance components of inter blocks in this case are decoded by referring to Table B-17. The last bit "s" denotes the sign of level, "0" for positive and "1" for negative. The combinations of (LAST, RUN, LEVEL) not represented in these tables are decoded as described in subclause 7.4.1.3.

3. **Escape code**

Many possible EVENTS have no variable length code to represent them. In order to encode these statistically rare combinations an Escape Coding method is used. The escape codes of DCT coefficients are encoded in five modes. The first three of these modes are used when short_video_header is 0 and in the case that the reversible VLC tables are not used, and the fourth is used when short_video_header is 1. In the case that the reversible VLC tables are used, the fifth escape coding method as in Table B-23 is used. Their decoding process is specified below.

Type 1 : ESC is followed by "0", and the code following ESC + "0" is decoded as a variable length code using the standard Tcoef VLC codes given in Table B-16 and Table B-17, but the values of LEVEL are modified following decoding to give the restored value LEVEL^{S}, as follows:

LEVEL^{S}= sign(LEVEL^{+}) x [abs(LEVEL^{+}) + LMAX]

where LEVEL^{+} is the value after variable length decoding and LMAX is obtained from Table B-19 and Table B-20 as a function of the decoded values of RUN and LAST.

Type 2 : ESC is followed by "10", and the code following ESC + "10" is decoded as a variable length code using the standard Tcoef VLC codes given in Table B-16 and Table B-17, but the values of RUN are modified following decoding to give the restored value RUN^{S}, as follows:

RUN^{S}= RUN^{+} + (RMAX + 1)

where RUN^{+} is the value after variable length decoding. RMAX is obtained from Table B-21 and Table B-22 as a function of the decoded values of LEVEL and LAST.

Type 3 : ESC is followed by "11", and the code following ESC + "11" is decoded as fixed length codes. This type of escape codes are represented by 1-bit LAST, 6-bit RUN and 12-bit LEVEL. A marker bit is inserted before and after the 12-bit-LEVEL in order to avoid the resync_marker emulation. Use of this escape sequence for encoding the combinations listed in Table B-16 and Table B-17 is prohibited. The codes for RUN and LEVEL are given in Table B-18.

Type 4: The fourth type of escape code is used if and only if short_video_header is 1. In this case, the 15 bits following ESC are decoded as fixed length codes represented by 1-bit LAST, 6-bit RUN and 8-bit LEVEL. The values 0000 0000 and 1000 000 for LEVEL are not used (they are reserved).

4. **Intra dc coefficient decoding for the case of switched vlc encoding**

At the VOP layer, using quantizer value as the threshold, a 3 bit code (intra_dc_vlc_thr) allows switching between 2 VLCs (DC Intra VLC and AC Intra VLC) when decoding DC coefficients of Intra macroblocks, see Table 6-21.

NOTE When the intra AC VLC is turned on, Intra DC coefficients are not handled separately any more, but treated the same as all other coefficients. That means that a zero Intra DC coefficient will not be coded but will simply increase the run for the following AC coefficients. The definitions of mcbpc and cbpy in subclause 6.3.6 are changed accordingly.

2. **Inverse scan**

This subclause specifies the way in which the one dimensional data, QFS[n] is converted into a two-dimensional array of coefficients denoted by PQF[v][u] where u and v both lie in the range of 0 to 7. Let the data at the output of the variable length decoder be denoted by QFS[n] where n is in the range of 0 to 63. Three scan patterns are defined as shown in Figure 7-4. The scan that shall be used is determined by the following method. For intra blocks, if acpred_flag=0, zigzag scan is selected for all blocks in a macroblock. Otherwise, DC prediction direction is used to select a scan on block basis. For instance, if the DC prediction refers to the horizontally adjacent block, alternate-vertical scan is selected for the current block. Otherwise (for DC prediction referring to vertically adjacent block), alternate-horizontal scan is used for the current block. For all other blocks, the 8x8 blocks of transform coefficients are scanned in the "zigzag" scanning direction.

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

Figure -4 -- (a) Alternate-Horizontal scan (b) Alternate-Vertical scan (c) Zigzag scan

3. **Intra dc and ac prediction for intra macroblocks**

This subclause specifies the prediction process for decoding of coefficients. This prediction process is only carried out for intra-macroblocks (I-MBs) and when short_video_header is "0". When short_video_header is "1" or the macroblock is not an I-MB, this prediction process is not performed.

1. **DC and AC Prediction Direction**

This adaptive selection of the DC and AC prediction direction is based on comparison of the horizontal and vertical DC gradients around the block to be decoded. Figure 7-5 shows the three blocks surrounding the block to be decoded. Block ?X?, ?A?, ?B? and ?C? respectively refer to the current block, the left block, the above-left block, and the block immediately above, as shown.

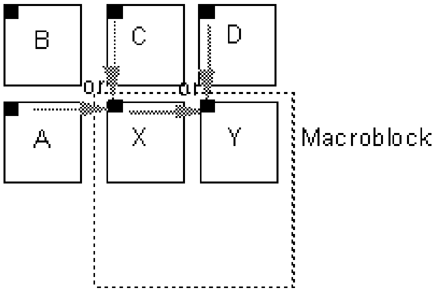


Figure -5 -- Previous neighboring blocks used in DC prediction

The inverse quantized DC values of the previously decoded blocks, F[0][0], are used to determine the direction of the DC and AC prediction as follows.

```
if ( $|F_A[0][0] - F_B[0][0]| < |F_B[0][0] - F_C[0][0]|$)
predict from block C
else
predict from block A
```

If any of the blocks A, B or C are outside of the VOP boundary, or the video packet boundary, or they do not belong to an intra coded macroblock, their F[0][0] values are assumed to take a value of $2^{(bits\_per\_pixel+2)}$ and are used to compute the prediction values.

2. **Adaptive DC Coefficient Prediction**

The adaptive DC prediction method involves selection of either the F[0][0] value of immediately previous block or that of the block immediately above it (in the previous row of blocks) depending on the prediction direction determined above.

```
if (predict from block C)
$QF_X[0][0] = PQF_X[0][0] + F_C[0][0]$ // dc_scaler
else
$QF_X[0][0] = PQF_X[0][0] + F_A[0][0]$ // dc_scaler
```

dc_scalar is defined in Table 7-1. This process is independently repeated for every block of a macroblock using the appropriate immediately horizontally adjacent block ?A? and immediately vertically adjacent block ?C?.

DC predictions are performed similarly for the luminance and each of the two chrominance components.

3. **Adaptive ac coefficient prediction**

This process is used when ac_pred_flag = ?1?, which indicates that AC prediction is performed when decoding the coefficients.

Either coefficients from the first row or the first column of a previous coded block are used to predict the co-sited coefficients of the current block. On a block basis, the best direction (from among horizontal and vertical directions) for DC coefficient prediction is also used to select the direction for AC coefficients prediction; thus, within a macroblock, for example, it becomes possible to predict each block independently from either the horizontally adjacent previous block or the vertically adjacent previous block. The AC coefficients prediction is illustrated in Figure 7-6.

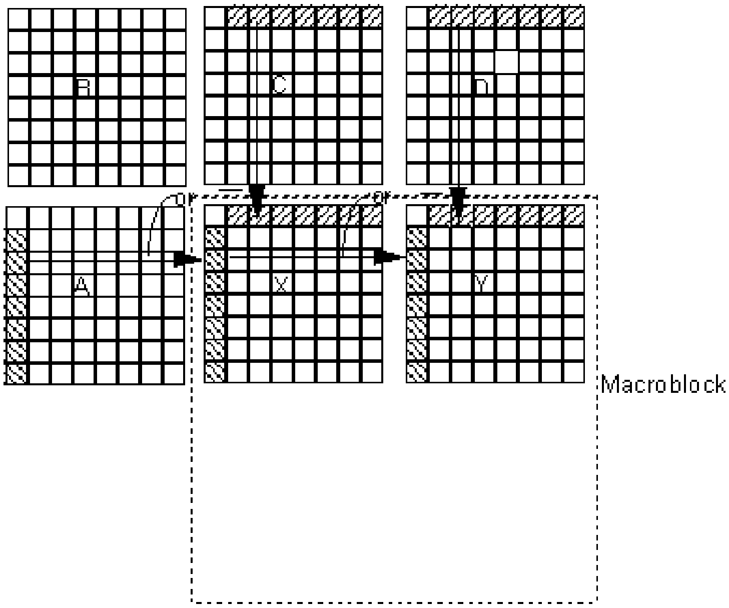


Figure -6 -- Previous neighboring blocks and coefficients used in AC prediction

To compensate for differences in the quantization of previous horizontally adjacent or vertically adjacent blocks used in AC prediction of the current block, scaling of prediction coefficients becomes necessary. Thus the prediction is modified so that the predictor is scaled by the ratio of the current quantisation stepsize and the quantisation stepsize of the predictor block. The definition is given in the equations below.

If block ?A? was selected as the predictor for the block for which coefficient prediction is to be performed, calculate the first column of the quantized AC coefficients as follows.

$$QF_X[0][i] = PQF_X[0][i] + (QF_A[0][i] * QP_A) // QP_X \quad i = 1 \text{ to } 7$$

If block ?C? was selected as the predictor for the block for which coefficient prediction is to be performed, calculate the first row of the quantized AC coefficients as follows.

$$QF_X[j][0] = PQF_X[j][0] + (QF_C[j][0] * QP_C) // QP_X \quad i = 1 \text{ to } 7$$

If the prediction block (block ’A’ or block ’C’) is outside of the boundary of the VOP or video packet, then all the prediction coefficients of that block are assumed to be zero.

4. **Saturation of QF[v][u]**

The quantized coefficients resulting from the DC and AC Prediction are saturated to lie in the range [-2048, 2047]. Thus:

$$QF[v][u] = \begin{cases} 2047 & QF[v][u] > 2047 \\ QF[v][u] & -2048 \le QF[v][u] \le 2047 \\ -2048 & QF[v][u] < -2048 \end{cases}$$

4. **Inverse quantisation**

The two-dimensional array of coefficients, $QF[v][u]$, is inverse quantised to produce the reconstructed DCT coefficients. This process is essentially a multiplication by the quantiser step size. The quantiser step size is modified by two mechanisms; a weighting matrix is used to modify the step size within a block and a scale factor is used in order that the step size can be modified at the cost of only a few bits (as compared to encoding an entire new weighting matrix).

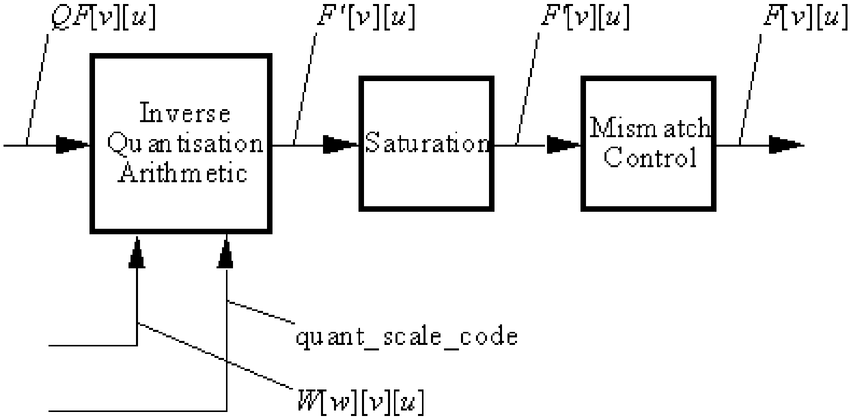


Figure -7 -- Inverse quantisation process

Figure 7-7 illustrates the overall inverse quantisation process. After the appropriate inverse quantisation arithmetic the resulting coefficients, $F''[v][u]$, are saturated to yield $F'[v][u]$ and then a mismatch control operation is performed to give the final reconstructed DCT coefficients, $F[v][u]$.

NOTE Attention is drawn to the fact that the method of achieving mismatch control in this part of ISO/IEC 14496 is identical to that employed by ISO/IEC 13818-2.

1. **First inverse quantisation method**

This subclause specifies the first of the two inverse quantisation methods. The method described here is used when quant_type equals 1.

1. **Intra dc coefficient**

The DC coefficients of intra coded blocks shall be inverse quantised in a different manner to all other coefficients.

In intra blocks *F??* [0][0] shall be obtained by multiplying *QF* [0][0] by a constant multiplier,

The reconstructed DC values are computed as follows.

F??[0][0] = *dc_scaler* * *QF*[0][0]

When short_video_header is 1, dc_scaler is 8, otherwise dc_scaler is defined in Table 7-1.

2. **Other coefficients**

All coefficients other than the DC coefficient of an intra block shall be inverse quantised as specified in this subclause. Two weighting matrices are used. One shall be used for intra macroblocks and the other for non-intra macroblocks. Each matrix has a default set of values which may be overwritten by down-loading a user defined matrix.

Let the weighting matrices be denoted by $W[w][v][u]$ where w takes the values 0 to 1 indicating which of the matrices is being used. $W[0][v][u]$ is for intra macroblocks, and $W[1][v][u]$ is for non-intra macroblocks. The value of *quantiser_scale* is determined by vop_quant, dquant, dbquant, and quant_scale for luminance and chrominance, and additionally by vop_quant_alpha for grayscale alpha. For example, the value of *quantiser scale* for luminance and chrominance shall be an integer from 1 to 31 when not_8_bit == ?0?. The following equation specifies the arithmetic to reconstruct $F''[v][u]$ from $QF[v][u]$ (for all coefficients except intra DC coefficients).

$$F'[v][u] = \begin{cases} 0, \text{ if } QF[v][u] = 0 \\ ((2 \times QF[v][u] + k) \times W[w][v][u] \times quantiser\_scale) \ / 16, \text{ if } QF[v][u] \neq 0 \end{cases}$$

$$\text{where :}$$

$$k = \begin{cases} 0 & \text{intra blocks} \\ Sign(QF[v][u]) & \text{non-intra blocks} \end{cases}$$

NOTE The above equation uses the "/" operator as defined in subclause 4.1.

2. Second inverse quantisation method

This subclause specifies the second of the two inverse quantisation methods. . The method described here is used for all the coefficients other than the DC coefficient of an intra block when quant_type==0. In the second inverse quantization method, the DC coefficient of an intra block is quantized using the same method as in the first inverse quantization method (see subclause 7.4.4.1.1). The quantization parameter *quantiser_scale* may take integer values from 1 to $2^{\text{quant\_precision}}$-1. The quantization stepsize is equal to twice the *quantiser_scale*.

1. Dequantisation

$$|F''[v][u]| = \begin{cases} 0, \quad \text{if } QF[v][u] = 0, \\ (2 \times |QF[v][u]| + 1) \times quantiser\_scale, & \text{if } QF[v][u] \neq 0, \ quantiser\_scale \text{ is odd,} \\ (2 \times |QF[v][u]| + 1) \times quantiser\_scale - 1, & \text{if } QF[v][u] \neq 0, \ quantiser\_scale \text{ is even.} \end{cases}$$

The sign of $QF[v][u]$ is then incorporated to obtain $F''[v][u]$: $F''[v][u]$= Sign($QF[v][u]$)´ $|F''[v][u]|$

3. Nonlinear inverse DC quantisation

NOTE This subclause is valid for both quantization methods.

Within an Intra macroblock for which short_video_header is 0, luminance blocks are called type 1 blocks, chroma blocks are classified as type 2. When short_video_header is 1, the inverse quantization of DC intra coefficients is equivalent to using a fixed value of dc_scaler = 8, as described above in subclause 7.4.1.1.

- DC coefficients of Type 1 blocks are quantized by Nonlinear Scaler for Type 1
- DC coefficients of Type 2 blocks are quantized by Nonlinear Scaler for Type 2

Table 7-1 specifies the nonlinear dc_scaler expressed in terms of piece-wise linear characteristics.

Table -1 -- Non linear scaler for DC coefficients of DCT blocks, expressed in terms of relation with quantizer_scale

Component:Type	**dc_scaler for quantiser_scale range**			
	1 through 4	**5 through 8**	**9 through 24**	**>= 25**
Luminance: Type1	8	2x quantiser_scale	quantiser_scale +8	2 x quantiser_scale -16
Chrominance: Type2	8	(quantiser_scale +13)/2	quantiser_scale -6	

1. **Saturation**

The coefficients resulting from the Inverse Quantisation Arithmetic are saturated to lie in the range [- 2bits_per_pixel + 3 , 2bits_per_pixel + 3 - 1]. Thus:

$$F'[v][u] = \begin{cases} 2^{\text{bits\_per\_pixel}+3} - 1 & F''[v][u] > 2^{\text{bits\_per\_pixel}+3} - 1 \\ F''[v][u] & -2^{\text{bits\_per\_pixel}+3} \le F''[v][u] \le 2^{\text{bits\_per\_pixel}+3} - 1 \\ -2^{\text{bits\_per\_pixel}+3} & F''[v][u] < -2^{\text{bits\_per\_pixel}+3} \end{cases}$$

2. **Mismatch control**

This mismatch control is only applicable to the first inverse quantization method. Mismatch control shall be performed by any process equivalent to the following. Firstly all of the reconstructed, saturated coefficients, *F'[v*][*u*] in the block shall be summed. This value is then tested to determine whether it is odd or even. If the sum is even then a correction shall be made to just one coefficient; *F*[7][7]. Thus:

$$sum = \sum_{v=0}^{v<8} \sum_{u=0}^{u<8} F'[v][u]$$

$$F[v][u] = F'[v][u] \text{ for all } u, v \text{ except } u = v = 7$$

$$F[7][7] = \begin{cases} F'[7][7] & \text{if } sum \text{ is odd} \\ \left.\begin{cases} F'[7][7] - 1 & \text{if } F'[7][7] \text{ is odd} \\ F'[7][7] + 1 & \text{if } F'[7][7] \text{ is even} \end{cases}\right\} & \text{if } sum \text{ is even} \end{cases}$$

NOTE 1 It may be useful to note that the above correction for *F* [7][7] may simply be implemented by toggling the least significant bit of the twos complement representation of the coefficient. Also since only the "oddness" or "evenness" of the *sum* is of interest an exclusive OR (of just the least significant bit) may be used to calculate "*sum*".

NOTE 2 Warning. Small non-zero inputs to the IDCT may result in zero output for compliant IDCTs. If this occurs in an encoder, mismatch may occur in some pictures in a decoder that uses a different compliant IDCT. An encoder should avoid this problem and may do so by checking the output of its own IDCT. It should ensure that it never inserts any non-zero coefficients into the bitstream when the block in question reconstructs to zero through its own IDCT function. If this action is not taken by the encoder, situations can arise where large and very visible mismatches between the state of the encoder and decoder occur.

3. **Summary of quantiser process for method 1**

In summary, the method 1 inverse quantisation process is any process numerically equivalent to:

```
for (v=0; v<8;v++) {

for (u=0; u<8;u++) {
```

```
if (QF[v][u] == 0)
F??[v][u] = 0;
else if ( (u==0) && (v==0) && (macroblock_intra) ) {
F’’[v][u] = dc_scaler * QF[v][u];
} else {
if ( macroblock_intra ) {
F’’[v][u] = ( QF[v][u] * W[0][v][u] * quantiser_scale * 2 ) / 32;
} else {
F’’[v][u] = ( ( ( QF[v][u] * 2 ) + Sign(QF[v][u]) ) * W[1][v][u]
* quantiser_scale ) / 32;
}
}
}
}
sum = 0;
for (v=0; v<8;v++) {
for (u=0; u<8;u++) {
if ( F?’[v][u] > $2^{bits\_per\_pixel + 3}$ - 1 ) {
F?[v][u] = $2^{bits\_per\_pixel + 3}$ - 1;
} else {
if ( F?’[v][u] < $-2^{bits\_per\_pixel + 3}$ ) {
F?[v][u] = $-2^{bits\_per\_pixel + 3}$ ;
} else {
F?[v][u] = F’?[v][u];
}
}
sum = sum + F?[v][u];
F[v][u] = F?[v][u];
}
```

```
}

if ((sum & 1) == 0) {

if ((F[7][7] & 1) != 0) {

F[7][7] = F'[7][7] - 1;

} else {

F[7][7] = F'[7][7] + 1;

}

}
```

1. Inverse DCT

Once the DCT coefficients, F[u][v] are reconstructed, the inverse DCT transform defined in annex A shall be applied to obtain the inverse transformed values, $f[y][x]$.These values shall be saturated so that: -2^{N_bit} £ f[y][x] £ 2^{N_bit} - 1 , for all x, y.

1. Shape decoding

Binary shape decoding is based on a block-based representation. The primary coding methods are block-based context-based binary arithmetic decoding and block-based motion compensation. The primary data structure used is denoted as the binary alpha block (bab). The bab is a square block of binary valued pixels representing the opacity/transparency for the pixels in a specified block-shaped spatial region of size 16x16 pels. In fact, each bab is co-located with each texture macroblock.

1. Higher syntactic structures

1. VOL decoding

If video_object_layer_shape is equal to ?00? then no binary shape decoding is required. Otherwise, binary shape decoding is carried out.

2. VOP decoding

If video_object_layer_shape is not equal to ?00? then, for each subsequent VOP, the dimensions of the bounding rectangle of the reconstructed VOP are obtained from:

- vop_width
- vop_height

If these decoded dimensions are not multiples of 16, then the values of vop_width and vop_height are rounded up to the nearest integer, which is a multiple of 16.

Additionally, in order to facilitate motion compensation, the horizontal and spatial position of the VOP are obtained from:

- vop_horizontal_mc_spatial_ref
- vop_vertical_mc_spatial_ref

These spatial references may be different for each VOP but the same coordinate system must be used for all VOPs within a vol. Additionally, the decoded spatial references must have an even value.

- vop_shape_coding_type

This flag is used in error resilient mode and enables the use of intra shape codes in P-VOPs. Finally, in the VOP class, it is necessary to decode

- change_conv_ratio_disable

This specifies whether conv_ratio is encoded at the macroblock layer.

Once the above elements have been decoded, the binary shape decoder may be applied to decode the shape of each macroblock within the bounding rectangle.

1. **Macroblock decoding**

The shape information for each macroblock residing within the bounding rectangle of the VOP is decoded into the form of a 16x16 bab.

1. **Mode decoding**

Each bab belongs to one of seven types listed in Table 7-2. The type information is given by the bab_type field which influences decoding of further shape information. For I-VOPs only three out of the seven modes are allowed as shown in Table 7-2.

Table -2 -- List of bab types

bab_type	**Semantic**	**Used in**
0	MVDs==0 && No Update	P- ,B-VOPs
1	MVDs!=0 && No Update	P- ,B-VOPs
2	Transparent	All VOP types
3	Opaque	All VOP types
4	IntraCAE	All VOP types
5	MVDs==0 && interCAE	P- ,B-VOPs
6	MVDs!=0 && interCAE	P- ,B-VOPs

1. **I-VOPs**

Suppose that f(x,y) is the bab_type of the bab located at (x,y), where x is the BAB column number and y is the BAB row number. The code word for the bab_type at the position (i,j) is determined as follows. A context C is computed from previously decoded bab_type?s.

C = 27*(f(i-1,j-1)-2) + 9*(f(i,j-1)-2) + 3*(f(i+1,j-1)-2) + (f(i-1,j)-2)

If f(x,y) references a bab outside the current VOP, bab_type is assumed to be transparent for that bab (i.e. f(x,y)=2). The bab_type of babs outside the current video packet is also assumed to be transparent. The VLC used to decode bab_type for the current bab is switched according to the value of the context C. This context-switched VLC table is given in Table B-27.

2. **P- and B-VOPs**

The decoding of the current bab_type is dependent on the bab_type of the co-located bab in the reference VOP. The reference VOP is either a forward reference VOP or a backward reference VOP. The forward reference VOP is defined as the most recent non-empty (i.e. vop_coded != 0) I- or P-VOP in the past, while the backward VOP is defined as the most recently decoded I- or P-VOP in the future. If the current VOP is a P-VOP, the forward reference VOP is selected as the reference VOP. If the current VOP is a B-VOP the following decision rules are applied:

1. If one of the reference VOPs is empty, the non-empty one (forward/backward) is selected as the reference VOP for the current B-VOP.

2. If both reference VOPs are non-empty, the forward reference VOP is selected if its temporal distance to the current B-VOP is not larger than that of the backward reference VOP, otherwise, the backward one is chosen.

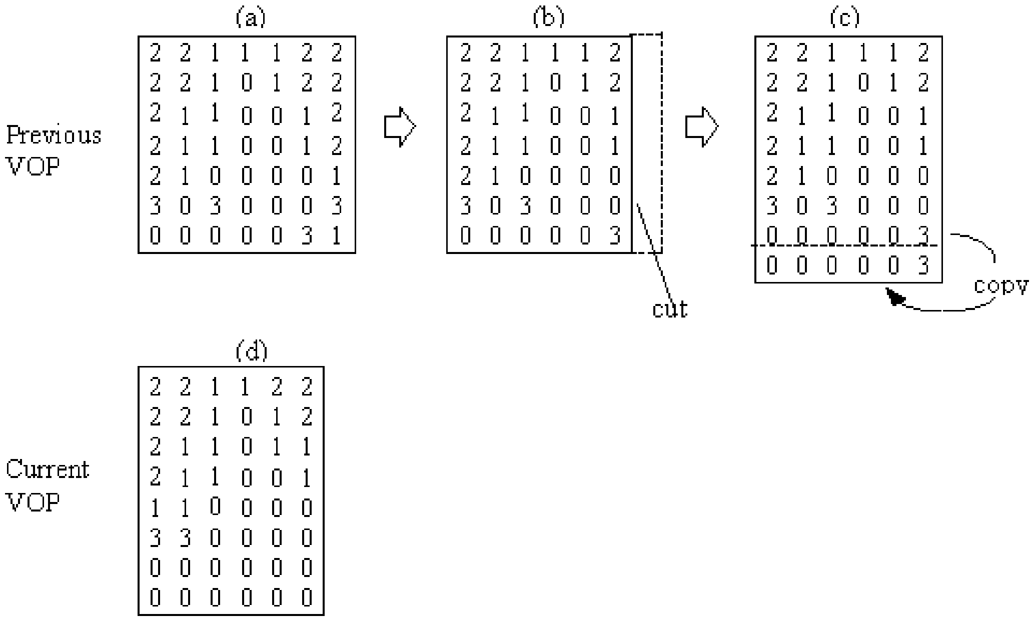
In the special cases when closed_gov == 1 and the forward reference VOP belongs to the previous GOV, the current B-VOP takes the backward VOP as reference.

If the sizes of the current and reference VOPs are different, some babs in the current VOP may not have a co-located equivalent in the reference VOP. Therefore the bab_type matrix of the reference VOP is manipulated to match the size of the current VOP. Two rules are defined for that purpose, namely a cut rule and a copy rule:

- ***cut rule.*** If the number of lines (respectively columns) is smaller in the current VOP than in the reference VOP, the bottom lines (respectively rightmost columns) are eliminated from the reference VOP such that both VOP sizes match.

- ***copy rule.*** If the number of lines (respectively columns) is larger in the current VOP than in the reference VOP, the bottom line (respectively rightmost column) is replicated as many times as needed in the reference VOP such that both VOP sizes match.

An example is shown in Figure 7-8 where both rules are applied.



(a)

2	2	1	1	1	2	2
2	2	1	0	1	2	2
2	1	1	0	0	1	2
2	1	1	0	0	1	2
2	1	0	0	0	0	1
3	0	3	0	0	0	3
0	0	0	0	0	3	1

(b)

2	2	1	1	1	2
2	2	1	0	1	2
2	1	1	0	0	1
2	1	1	0	0	1
2	1	0	0	0	0
3	0	3	0	0	0
0	0	0	0	0	3

(c)

2	2	1	1	1	2
2	2	1	0	1	2
2	1	1	0	0	1
2	1	1	0	0	1
2	1	0	0	0	0
3	0	3	0	0	0
0	0	0	0	0	3
0	0	0	0	0	3

(d)

2	2	1	1	2	2
2	2	1	0	1	2
2	1	1	0	1	1
2	1	1	0	0	1
1	1	0	0	0	0
3	3	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Figure -8 -- Example of size fitting between current VOP and reference VOP. The numbers represent the type of each bab

The VLC to decode the current bab_type is switched according to the value of bab_type of the co-located bab in the reference VOP. This context-switched VLC tables for P and B VOPs are given in Table B-28. If the type of the bab is transparent, then the current bab is filled with zero (transparent) values. A similar procedure is carried out if the type is opaque, where the reconstructed bab is filled with values of 255 (opaque). For both transparent and opaque types, no further decoding of shape-related data is required for the current bab. Otherwise further decoding steps are necessary, as listed in Table 7-3. Decoding for motion compensation is described in subclause 7.5.2.2, and cae decoding in subclause 7.5.2.5.

Table -3 -- Decoder components applied for each type of bab

bab_type	**Motion compensation**	**CAE decoding**
0	yes	no
1	yes	no
2	no	no
3	no	no
4	no	yes
5	yes	yes
6	yes	yes

1. **Binary alpha block motion compensation**

   Motion Vector of shape (MVs) is used for motion compensation (MC) of shape. The value of MVs is reconstructed as described in subclause 7.5.2.3. Integer pixel motion compensation is carried out on a 16x16 block basis according to subclause 7.5.2.4. Overlapped MC, half sample MC and 8x8 MC are not carried out.

   If bab_type is MVDs==0 && No Update or MVDs!=0 && No Update then the motion compensated bab is taken to be the decoded bab, and no further decoding of the bab is necessary. Otherwise, cae decoding is required.

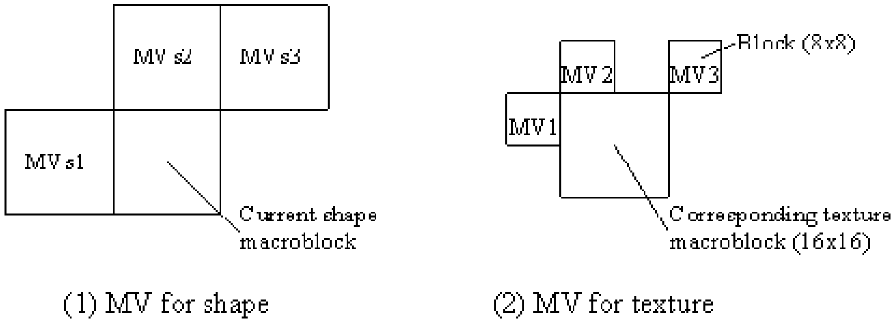
2. **Motion vector decoding**

   If bab_type indicates that MVDs!=0, then mvds_x and mvds_y are VLC decoded. For decoding mvds_x, the VLC given in Table B-29 is used. The same table is used for decoding mvds_y, unless the decoded value of mvds_x is zero. If mvds_x == 0, the VLC given in Table B-30 is used for decoding mvds_y. If bab_type indicates that MVDs==0, then both mvds_x and mvds_y are set to zero.

   The integer valued shape motion vector MVs=(mvs_x,mvs_y) is determined as the sum of a predicted motion vector MVPs and MVDs = (mvds_x,mvds_y), where MVPs is determined as follows.

   MVPs is determined by analysing certain candidate motion vectors of shape (MVs) and motion vectors of selected texture blocks (MV) around the MB corresponding to the current bab. They are located and denoted as shown in Figure 7-9 where MV1, MV2 and MV3 are rounded up to integer values towards 0. If the selected texture block is a field predicted macroblock, then MV1, MV2 or MV3 are generated by averaging the two field motion vectors and rounding toward zero. Regarding the texture MV's, the convention is that a MB possessing only 1 MV is considered the same as a MB possessing 4 MV's, where the 4 MV's are equal. By traversing MVs1, MVs2, MVs3, MV1, MV2 and MV3 in this order, MVPs is determined by taking the first encountered

MV that is defined. If no candidate motion vectors is defined, MVPs = (0,0).



(1) MV for shape

(2) MV for texture

Figure -9

-- Candidates for MVPs

In the case that video_object_layer_shape is "binary_only" or vop_coding_type indicates B-VOP, MVPs is determined by considering the motion vectors of shape (MVs1, MVs2 and MVs3) only. The following subclauses explain the definition of MVs1, MVs2, MVs3, MV1, MV2 and MV3 in more detail.

Defining candidate predictors from texture motion vectors:

One shape motion vector predictor MV_i (i =1,2,3) is defined for each block located around the current bab according to Figure 7-9 (2). The definition only depends on the transparency of the reference MB. MVi is set to the corresponding block vector as long as it is in a non-transparent reference MB, otherwise, it is not defined. Note that if a reference MB is outside the current VOP or video packet, it is treated as a transparent MB.

Defining candidate predictors from shape motion vectors:

The candidate motion vector predictors MVs_i are defined by the shape motion vectors of neighbouring bab located according to Figure 7-9 (1). The MVs_i are defined according to Table 7-4.

Table -4 -- Definition of candidate shape motion vector predictors MVs1, MVs2, and MVs3 from shape motion vectors for P and B-VOPs. Note that interlaced modes are not included

Shape mode of reference MB	MVs_i for each reference shape block-i (a shape block is
MVDs == 0 or MVDs !=0 bab_type 0, 1, 5,6	The retrieved shape motion vector of the said reference defined as MVs_i . Note that MVs_i is defined, and hence vali if the reconstructed shape block is transparent.
all_0, bab_type 2	MVs_i is undefined
all=255, bab_type 3	MVs_i is undefined
Intra, bab_type 4	MVs_i is undefined

If the reference MB is outside of the current video packet, MV_i and MVs_i are undefined.

3. **Motion compensation**

For inter mode babs (bab_type = 0,1,5 or 6), motion compensation is carried out by simple MV displacement according to the MVs.

Specifically, when bab_type is equal to 0 or 1 i.e. for the no-update modes, a displaced block of 16x16 pixels is copied from the binary alpha map of the previously decoded I or P VOP for which vop_coded is not equal to ?0?. When the bab_type is equal to 5 or 6 i.e. when interCAE decoding is required, then the pixels immediately bordering the displaced block (to the left, right, top and bottom) are also copied from the most recent valid reference VOP?s (as defined in subclause 6.3.5**)** binary alpha map into a temporary shape block of 18x18 pixels size (see Figure 7-12). If the displaced position is outside the bounding rectangle, then these pixels are assumed to be "transparent".

If the current VOP is a B-VOP the following decision rules are applied:

- If one of the reference VOPs is empty (i.e. VOP_coded is 0), the non-empty one (forward/backward) is selected as the reference VOP for the current B-VOP.

- If both reference VOPs are non-empty, the forward reference VOP is selected if its temporal distance to the current B-VOP is not larger than that of the backward reference VOP, otherwise, the backward one is chosen.

In the special cases when closed_gov == 1 and the forward reference VOP belongs to the previous GOV, the current B-VOP takes the backward VOP as reference.

1. Context based arithmetic decoding

Before decoding the binary_arithmetic_code field, border formation (see subclause 7.5.2.5.2) needs to be carried out. Then, if the scan_type field is equal to 0, the bordered to-be decoded bab and the eventual bordered motion compensated bab need to be transposed (as for matrix transposition). If change_conv_rate_disable is equal to 0, then conv_ratio is decoded to determine the size of the sub-sampled BAB, which is 16/conv_ratio by 16/conv_ratio pixels large. If change_conv_rate_disable is equal to 1, then the decoder assumes that the bab is not subsampled and thus the size is simply 16x16 pixels. Binary_arithmetic_code is then decoded by a context-based arithmetic decoder as follows. The arithmetic decoder is firstly initialised (see subclause 7.5.3.3). The pixels of the sub-sampled bab are decoded in raster order. At each pixel,

1. A context number is computed based on a template, as described in subclause 7.5.2.5.1.

2. The context number is used to access the probability table (Table B-32).

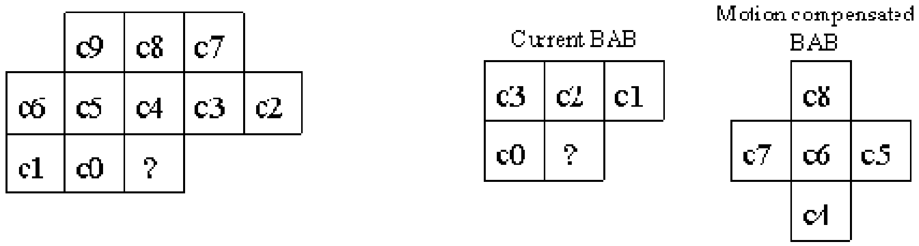
3. Using the accessed probability value, the next bits of binary_arithmetic_code are decoded by the arithmetic decoder to give the decoded pixel value.

When all pixels in sub-sampled BAB have been decoded, the arithmetic decoder is terminated (see subclause 7.5.3.6).

If the scan_type field is equal to 0, the decoded bab is transposed. Then up-sampling is carried out if conv_ratio is different from 1, as described in subclause 7.5.2.5.3. Then the decoded bab is copied into the decoded shape map.

1. Context computation

For INTRA coded BABs, a 10 bit context $C = \sum_k c_k \cdot 2^k$ is built for each pixel as illustrated in Figure 7-10 (a), where c_k=0 for transparent pixels and c_k=1 for opaque pixels.



(a) (b)

Figure -10 -- (a) The INTRA template (b) The INTER template where c6 is aligned with the pixel to be decoded. The pixel to be decoded is marked with ???

For INTER coded BABs, temporal redundancy is exploited by using pixels from the bordered motion compensated BAB (depicted in Figure 7-12) to make up part of the context. Specifically, a 9 bit context $C = \sum_k c_k \cdot 2^k$ **is built as illustrated in Figure 7-10 (b).**

There are some special cases to note.

- **When building contexts, any pixels outside the bounding rectangle of the current VOP to the left and above are assumed to be zero (transparent).**
- **When building contexts, any pixels outside the space of the current video packet to the left and above are assumed to be zero (transparent).**
- **The template may cover pixels from BABs which are unknown at decoding time. Unknown pixels are defined as area U in Figure 7-11.**
- **The values of these unknown pixels are defined by the following procedure:**
  - **When constructing the INTRA context, the following steps are taken in the sequence**

    1. if (c7 is unknown) c7=c8,
    2. if (c3 is unknown) c3=c4,
    3. if (c2 is unknown) c2=c3.

  - When constructing the INTER context, the following conditional assignment is performed.

    if (c1 is unknown) c1=c2

1. Border formation

When decoding a BAB, pixels from neighbouring BABs shall be used to make up the context. For both the INTRA and INTER cases, a 2 pixel wide border about the current BAB is used where pixels values are known, as depicted in Figure 7-11.

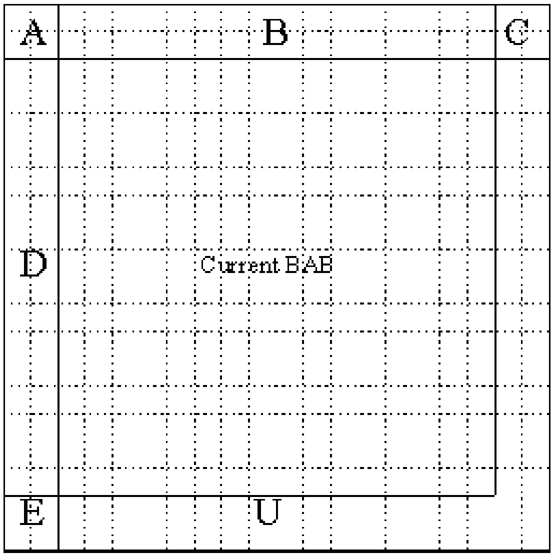


Figure -11 -- Bordered BAB. A: TOP_LEFT_BORDER. B: TOP_BORDER. C: TOP_RIGHT_BORDER. D: LEFT_BORDER. E: BOTTOM_LEFT_BORDER. U: pixels which are unknown when decoding the current BAB

If the value of conv_ratio is not equal to 1, a sub-sampling procedure is further applied to the BAB borders for both the current BAB and the motion compensated BAB.

The border of the current BAB is partitioned into 5 regions:

- TOP_LEFT_BORDER, which contains pixels from the BAB located to the upper-left of the current BAB and which consists of 2 lines of 2 pixels
- TOP_BORDER, which contains pixels from the BAB located above the current BAB and which consists of 2 lines of 16 pixels
- TOP_RIGHT_BORDER, which contains pixels from the BAB located to the upper-right of the current BAB and which consists of 2 lines of 2 pixels
- LEFT_BORDER, which contains pixels from the BAB located to the left of the current BAB and which consists of 2 columns of 16 pixels

1. BOTTOM_LEFT_BORDER, which contains pixels from the BAB located to the bottom-left of the current BAB and which consists of 2 lines of 2 pixels

The TOP_LEFT_BORDER and TOP_RIGHT_BORDER are not sub-sampled, and kept as they are. The TOP_BORDER and LEFT_BORDER are sub-sampled such as to obtain 2 lines of 16/conv_ratio pixels and 2 columns of 16/conv_ratio pixels, respectively.

The sub-sampling procedure is performed on a line-basis for TOP_BORDER, and a column-basis for LEFT_BORDER. For each line (respectively column), the following algorithm is applied: the line (respectively column) is split into groups of conv_ratio pixels. For each group of pixels, one pixel is associated in the sub-sampled border. The value of the pixel in the sub-sampled border is OPAQUE if half or more pixels are OPAQUE in the corresponding group. Otherwise the pixel is TRANSPARENT.

The 2x2 BOTTOM_LEFT_BORDER is filled by replicating downwards the 2 bottom border samples of the LEFT_BORDER after the down-sampling (if any).

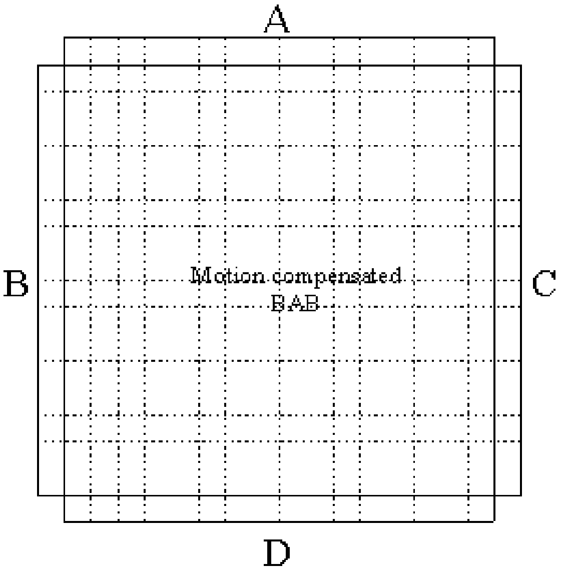


Figure -12 -- Bordered motion compensated BAB. A: TOP_BORDER. B: LEFT_BORDER. C: RIGHT_BORDER. D: BOTTOM_BORDER

In the case of a motion compensated BAB, the border is also partitioned into 4, as shown Figure 7-12:

- TOP_BORDER, which consists of a line of 16 pixels
- LEFT_BORDER, which consists of a column of 16 pixels
- RIGHT_BORDER, which consists of a column of 16 pixels
- BOTTOM_BORDER, which consists of a line of 16 pixels

The very same sub-sampling process as described above is applied to each of these borders.

1. **Upsampling**

When conv_ratio is different from 1, up-sampling is carried out for the BAB. This is illustrated in Figure 7-13 where "O" in this figure is the coded pixel and "X" is the interpolated pixel. To compute the value of the interpolated pixel, a filter context from the neighboring pixels is first calculated. For the pixel value calculation, the value of "0" is used for a transparent pixel, and "1" for an opaque pixel. The values of the interpolated pixels (Pi, i=1,2,3,4, as shown in Figure 7-14) can then be determined by the following equation:

P1 : if(4*A + 2*(B+C+D) + (E+F+G+H+I+J+K+L) > Th[Cf]) then "1" else "(

P2 : if(4*B + 2*(A+C+D) + (E+F+G+H+I+J+K+L) > Th[Cf]) then "1" else "(

P3 : if(4*C + 2*(B+A+D) + (E+F+G+H+I+J+K+L) > Th[Cf]) then "1" else "(

P4 : if(4*D + 2*(B+C+A) + (E+F+G+H+I+J+K+L) > Th[Cf]) then "1" else "(

The 8-bit filter context, Cf, is calculated as follows:

$$C_f = \sum_k c_k \cdot 2^k$$

Based on the calculated Cf, the threshold value (Th[Cf]) can be obtained from the look-up table as follows:

Th[256] = {

3, 6, 6, 7, 4, 7, 7, 8, 6, 7, 5, 8, 7, 8, 8, 9,

6, 5, 5, 8, 5, 6, 8, 9, 7, 6, 8, 9, 8, 7, 9, 10,

6, 7, 7, 8, 7, 8, 8, 9, 7, 10, 8, 9, 8, 9, 9, 10,

7, 8, 6, 9, 6, 9, 9, 10, 8, 9, 9, 10, 11, 10, 10, 11,

6, 9, 5, 8, 5, 6, 8, 9, 7, 10, 10, 9, 8, 7, 9, 10,

7, 6, 8, 9, 8, 7, 7, 10, 8, 9, 9, 10, 9, 8, 10, 9,

7, 8, 8, 9, 6, 9, 9, 10, 8, 9, 9, 10, 9, 10, 10, 9,

8, 9, 11, 10, 7, 10, 10, 11, 9, 12, 10, 11, 10, 11, 11, 12,

6, 7, 5, 8, 5, 6, 8, 9, 5, 6, 6, 9, 8, 9, 9, 10,

5, 8, 8, 9, 6, 7, 9, 10, 6, 7, 9, 10, 9, 10, 10, 11,

7, 8, 6, 9, 8, 9, 9, 10, 8, 7, 9, 10, 9, 10, 10, 11,

8, 9, 7, 10, 9, 10, 8, 11, 9, 10, 10, 11, 10, 11, 9, 12,

7, 8, 6, 9, 8, 9, 9, 10, 10, 9, 7, 10, 9, 10, 10, 11,

8, 7, 7, 10, 7, 8, 8, 9, 9, 10, 10, 11, 10, 11, 11, 12,

8, 9, 9, 10, 9, 10, 10, 9, 9, 10, 10, 11, 10, 11, 11, 12,

9, 10, 10, 11, 10, 11, 11, 12, 10, 11, 11, 12, 11, 12, 12, 13 };

TOP_LEFT_BORDER, TOP_RIGHT_BORDER, sub-sampled TOP_BORDER and sub-sampled LEFT_BORDER described in the previous subclause are used. The other pixels outside the BAB are extended from the outermost pixels inside the BAB as shown in Figure 7-13.

In the case that conv_ratio is 4, the interpolation is processed twice. The above mentioned borders of 4x4 BAB are used for the interpolation from 4x4 to 8x8, and top-border (respectively left-border) for the interpolation from 8x8 to 16x16 are up-sampled from the 4x4 BAB top-border (respectively left-border) by simple repetition.

When the BAB is on the left (and/or top) border of VOP, the borders outside VOP are set to zero value. The upsampling filter shall not use pixel values outside of the current video packet.

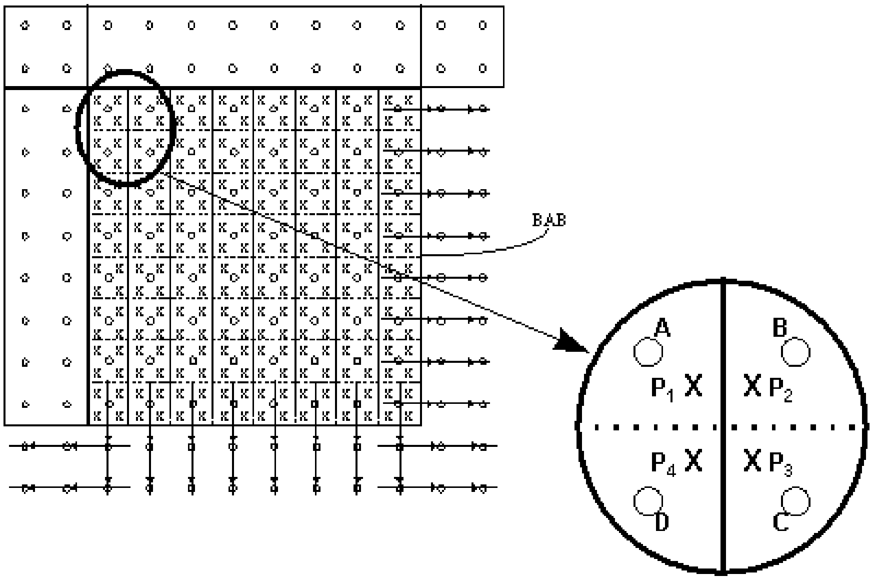
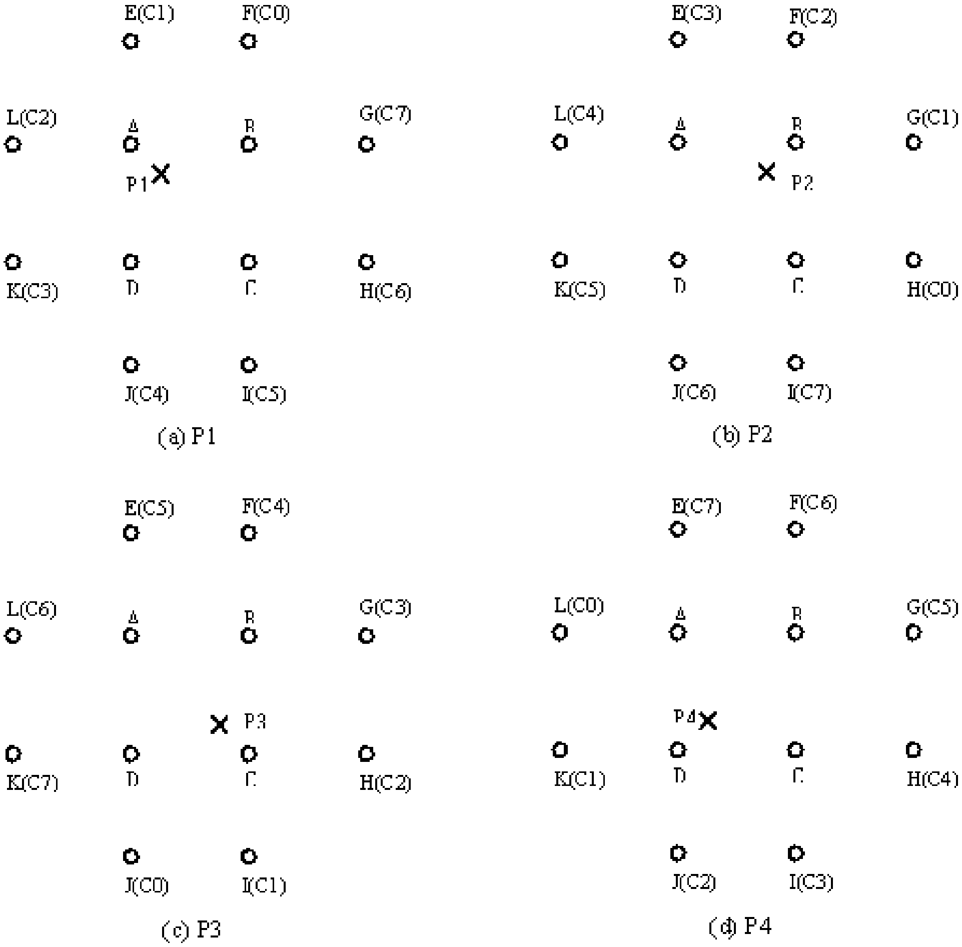


Figure -13 -- Upsampling



(a) P1

(b) P2

(c) P3

(d) P4

Figure -14 -- Interpolation filter and interpolation construction

2. Down-sampling process in inter case

If bab_type is ?5? or ?6? (see Table 7-3), downsampling of the motion compensated bab is needed for calculating the 9 bit context in the case that conv_ratio is not 1. The motion compensated bab of size 16x16 pixels is down sampled to bab of size 16/conv_ratio by 16/conv_ratio pixels by the following rules:

- conv_ratio==2

  If the average of pixel values in 2 by 2 pixel block is equal to or greater than 127.5 the pixel value of the downsampled bab is set to 255 otherwise it is set to 0.

- conv_ratio==4

  If the average of pixel values in 4 by 4 pixel block is equal to or greater than 127.5 the pixel value of the downsampled bab is set to 255 otherwise it is set to 0.

1. Arithmetic decoding

Arithmetic decoding consists of four main steps:

- Removal of stuffed bits
- Initialization which is performed prior to the decoding of the first symbol
- Decoding of the symbol themselves. The decoding of each symbol may be followed by a re-normalization step.
- Termination which is performed after the decoding of the last symbol

1. Registers, symbols and constants

Several registers, symbols and constants are defined to describe the arithmetic decoder.

- HALF: 32-bit fixed point constant equal to ½ (0x80000000)
- QUARTER: 32-bit fixed point constant equal to ¼ (0x40000000)
- L: 32-bit fixed point register. Contains the lower bound of the interval
- R: 32-bit fixed point register. Contains the range of the interval.
- V: 32-bit fixed point register. Contains the value of the arithmetic code. V is always larger than or equal to L and smaller than L+R.
- p0: 16-bit fixed point register. Probability of the ?0? symbol.
- p1: 16-bit fixed point register. Probability of the ?1? symbol.
- LPS: boolean. Value of the least probable symbol (?0? or ?1?).
- bit: boolean. Value of the decoded symbol.
- pLPS: 16-bit fixed point register. Probability of the LPS.
- rLPS: 32-bit fixed point register. Range corresponding to the LPS.

1. Bit stuffing

In order to avoid start code emulation, 1?s are stuffed into the bitstream whenever there are too many successive 0?s. If the first MAX_HEADING bits are 0?s, then a 1 is transmitted after the MAX_HEADING-th 0. If MAX_MIDDLE or more 0?s are sent successively a 1 is inserted after the MAX_MIDDLE-th 0. If the number of trailing 0?s is larger than MAX_TRAILING, then a 1 is appended to the stream. The decoder shall properly skip these inserted 1?s when reading data into the V register (see subclauses 7.5.3.3 and 7.5.3.5).

MAX_HEADING equals 3, MAX_MIDDLE equals 10, and MAX_TRAILIING equals 2.

2. Initialization

The lower bound L is set to 0, the range R to HALF-0x1 (0x7fffffff) and the first 31 bits are read in register V.

3. **Decoding a symbol**

When decoding a symbol, the probability p0 of the ?0? symbol is provided according to the context computed in subclause 7.5.2.5.1 and using Table B-32. p0 uses a 16-bit fixed-point number representation. Since the decoder is binary, the probability of the ?1? symbol is defined to be 1 minus the probability of the ?0? symbol, i.e. p1 = 1-p0.

The least probable symbol LPS is defined as the symbol with the lowest probability. If both probabilities are equal to ½ (0x8000), the ?0? symbol is considered to be the least probable.

The range rLPS associated with the LPS may simply be computed as R*pLPS: The 16 most significant bits of register R are multiplied by the 16 bits of pLPS to obtain the 32 bit rLPS number.

The interval [L,L+R) is split into two intervals [L,L+R-rLPS) and [L+R-rLPS,L+R). If V is in the latter interval then the decoded symbol is equal to LPS. Otherwise the decoded symbol is the opposite of LPS. The interval [L,L+R) is then reduced to the sub-interval in which V lies.

After the new interval has been computed, the new range R might be smaller than QUARTER. If so, re-normalization is carried out, as described below.

4. **Re-normalization**

As long as R is smaller than QUARTER, re-normalization is performed.

- If the interval [L,L+R) is within [0,HALF), the interval is scaled to [2L,2L+2R). V is scaled to 2V.
- If the interval [L,L+R) is within [HALF,1) the interval is scaled to [2(L-HALF),2(L-HALF)+2R). V is scaled to 2(V-HALF).
- Otherwise the interval is scaled to [2(L-QUARTER),2(L-QUARTER)+2R). V is scaled to 2(V-QUARTER).

After each scaling, a bit is read and copied into the least significant bit of register V.

1. **Termination**

After the last symbol has been decoded, additional bits need to be "consumed". They were introduced by the encoder to guarantee decodability.

In general 3 further bits need to be read. However, in some cases, only two bits need to be read. These cases are defined by:

- if the current interval covers entirely [QUARTER-0x1,HALF)
- if the current interval covers entirely [HALF-0x1, 3QUARTER)

After these additional bits have been read, 32 bits shall be "unread", i.e. put the content of register V back into the bit buffer.

1. **Software**

The example software for arithmetic decoding for binary shape decoding is included in annex B.

1. **Grayscale Shape Decoding**

Grayscale alpha plane decoding is achieved by the separate decoding of a support region and the values of the alpha channel. The support region is transmitted by using the binary shape as described above. The alpha values are transmitted as texture data with arbitrary shape, using almost the same coding method as is used for the luminance texture channel.

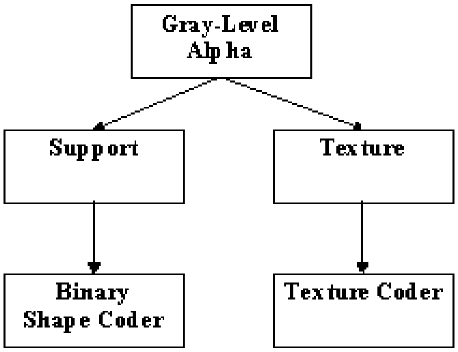


Figure -15 -- Grayscale shape coding

All samples which are indicated to be transparent by the binary shape data, must be set to zero in the decoded grayscale alpha plane. Within the VOP, alpha samples have the values produced by the grayscale alpha decoding process. Decoding of binary shape information is not dependent on the decoding of grayscale alpha. The alpha values are decoded into 16x16 macroblocks in the same way as the luminance channel (see subclauses 7.4 and 7.6). The 16x16 blocks of alpha values are referred to as alpha macroblocks hereafter. The data for each alpha macroblock is present in the bitstream immediately following the texture data for the corresponding texture macroblock. Any aspect of alpha decoding that is not covered in this document should be assumed to be the same as for the decoding of luminance.

1. **Grayscale Alpha COD Modes**

   When decoding grayscale alpha macroblocks, CODA is first encountered and indicates the coding status for alpha. It is important to understand that the macroblock syntax elements for alpha are still present in the bitstream for P or B macroblocks even if the texture syntax elements indicate "not-coded" (not_coded=?1?). In this respect, the decoding of the alpha and texture data are independent. The only exception is for BVOPs when the colocated PVOP texture macroblock is skipped. In this case, no syntax is transmitted for texture or grayscale alpha, as both types of macroblock are skipped.

   For macroblocks which are completely transparent (indicated by the binary shape coding), no alpha syntax elements are present and the grayscale alpha samples must all be set to zero (transparent). If CODA="all opaque" (I, P or B macroblocks) or CODA="not coded" (P or B macroblocks) then no more alpha data is present. Otherwise, other alpha syntax elements follow, including the coded block pattern (cbpa), followed by alpha texture data for those 8x8 blocks which are coded and non-transparent, as is the case for regular luminance macroblock texture data.

   When CODA="all opaque", the corresponding decoded alpha macroblock is filled with a constant value of 255. This value will be called AlphaOpaqueValue.

2. **Alpha Plane Scale Factor**

   For both binary and grayscale shape, the VOP header syntax element "vop_constant_alpha" can be used to scale the alpha plane. If this bit is equal to ?1?, then each pixel in the decoded VOP is scaled before output, using vop_constant_alpha_value. The scaling formula is:

   scaled_pixel = (original_pixel * (vop_constant_alpha_value + 1)) / 256

   Scaling is applied at the output of the decoder, such that the decoded original values, not the scaled values are used as the source for motion compensation.

3. **Gray Scale Quantiser**

When no_gray_quant_update is equal to "1", the grayscale alpha quantiser is fixed for all macroblocks to the value indicated by vop_alpha_quant. Otherwise, the grayscale quantiser is reset at each new macroblock to a value that depends on the current texture quantiser (after any update by dquant). The relation is:

current_alpha_quant = (current_texture_quant * vop_alpha_quant) / vop_quant

The resulting value of current_alpha_quant must then be clipped so that it never becomes less than 1.

4. **Intra Macroblocks**

When the texture mb_type indicates an intra macroblock in IVOPs or PVOPs, the grayscale alpha data is also decoded using intra mode.

The intra dc value is decoded in the same way as for luminance, using the same non-linear transform to convert from alpha_quant to DCScalarA. However, intra_dc_vlc_thr is not used for alpha, and therefore AC coeffiecient VLCs are never used to code the differential intra dc coefficient.

DC prediction is used in the same way as for luminance. However, when coda_i indicates that a macroblock is all opaque, a synthetic intra dc value is created for each block in the current macroblock so that adjacent macroblocks can correctly obtain intra dc prediction values. The synthetic intra dc value is given as:

BlockIntraDC = (((AlphaOpaqueValue * 8) + (DcScalerA>>1)) / DcScalerA) * DcScalerA

AlphaOpaqueValue is described in subclause 7.5.4.1.

The intra cbpa VLC makes use of the *inter* cbpy VLC table, but the intra alpha block DCT coefficients are decoded in the same manner as with luminance intra macroblocks.

5. **Inter Macroblocks and Motion Compensation**

Motion compensation is carried out for PVOPs and BVOPs, using the 8x8 or 16x16 luminance motion vectors, in the same way as for luminance data, except that regular motion compensation is used instead of OBMC. Forward, backward, bidirectional and direct mode motion compensation are used for BVOPs. Where the luminance motion vectors are not present because the texture macroblock is skipped, the exact same style of non-coded motion compensation used for luminance is applied to the alpha data (but without OBMC). Note that this does not imply that the alpha macroblock is skipped, because an error signal to update the resulting motion compensated alpha macroblock may still be present if indicated by coda_pb. When the colocated PVOP texture macroblock is skipped for BVOPs, then the alpha macroblock is assumed to be skipped with no syntax transmitted.

cbpa and the alpha inter DCT coefficients are decoded in the same way as with luminance cbpy and inter DCT cofficients

6. **Method to be used when blending with greyscale alpha signal**

The following explains the blending method to be applied to the video object in the compositor, which is controlled by the composition_method flag and the linear_composition flag. The linear_composition flag is informative only, and the decoder may ignore it and proceed as if it had the value 0. However, it is normative that the composition_method flag be acted upon.

The descriptions below show the processing taking place in YUV space; note that the processing can of course be implemented in RGB space to obtain equivalent results.

composition_method=0 (cross-fading)

If layer N, with an n-bit alpha signal, is overlaid over layer M to generate a new layer P, the composited Y, U, V and alpha values are:

$Pyuv = ((2^n-1 - Nalpha) * Myuv + (Nalpha * Nyuv)) / (2^n-1)$

$Palpha = (2^n-1)$

composition_method=1 (Additive mixing)

If layer N, with an n-bit alpha signal, is overlaid over layer M to generate a new layer P, the composited Y, U, V and alpha values are:

{ Myuv ..... Nalpha = 0

Pyuv = {

{ (Myuv - BLACK) - ((Myuv - BLACK) * Nalpha) / (2^n-1)+ Nyuv ..... Nalpha > 0

(this is equivalent to Pyuv = Myuv*(1-alpha) + Nyuv, taking account of black level and the fact that the video decoder does not produce an output in areas where alpha=0)

$Palpha = Nalpha + Malpha - (Nalpha*Malpha) / (2^n-1)$

where

BLACK is the common black value of foreground and background objects.

NOTE The compositor must convert foreground and background objects to the same black value and signal range before composition. The black level of each video object is specified by the video_range bit in the video_signal_type field, or by the default value if the field is not present. (The RGB values of synthetic objects are specified in a range from 0 to 1, as described in ISO/IEC 14496-1).

- linear_composition = 0: The compositing process is carried out using the video signal in the format from which it is produced by the video decoder, that is, without converting to linear signals. Note that because video signals are usually non-linear ("gamma-corrected"), the composition will be approximate.
- linear_composition = 1: The compositing process is carried out using linear signals, so the output of the video decoder is converted to linear if it was originally in a non-linear form, as specified by the video_signal_type field. Note that the alpha signal is always linear, and therefore requires no conversion.

1. **Motion compensation decoding**

In order to perform motion compensated prediction on a per VOP basis, a special padding technique, i.e. the macroblock-based repetitive padding, is applied for the reference VOP. The details of these techniques are described in the following subclauses.

Since a VOP may have arbitrary shape, and this shape can change from one instance to another, conventions are necessary to ensure the consistency of the motion compensation process.

The absolute (frame) coordinate system is used for referencing every VOP. At every given instance, a bounding rectangle that includes the shape of that VOP, as described in subclause 7.5, is defined. The left and top corner, in the absolute coordinates, of the bounding rectangle is decoded from VOP spatial reference. Thus, the motion vector for a particular feature inside a VOP, e.g. a macroblock, refers to the displacement of the feature in absolute coordinates. No alignment of VOP bounding rectangles at different time instances is performed.

In addition to the above motion compensation processing, three additional processes are supported, namely, unrestricted motion compensation, four MV motion compensation,and overlapped motion compensation. Note that in all three modes, macroblock-based padding of the arbitrarily shaped reference VOP is performed for motion compensation.

1. **Padding process**

The padding process defines the values of luminance and chrominance samples outside the VOP for prediction of arbitrarily shaped objects. Figure 7-16 shows a simplified diagram of this process.

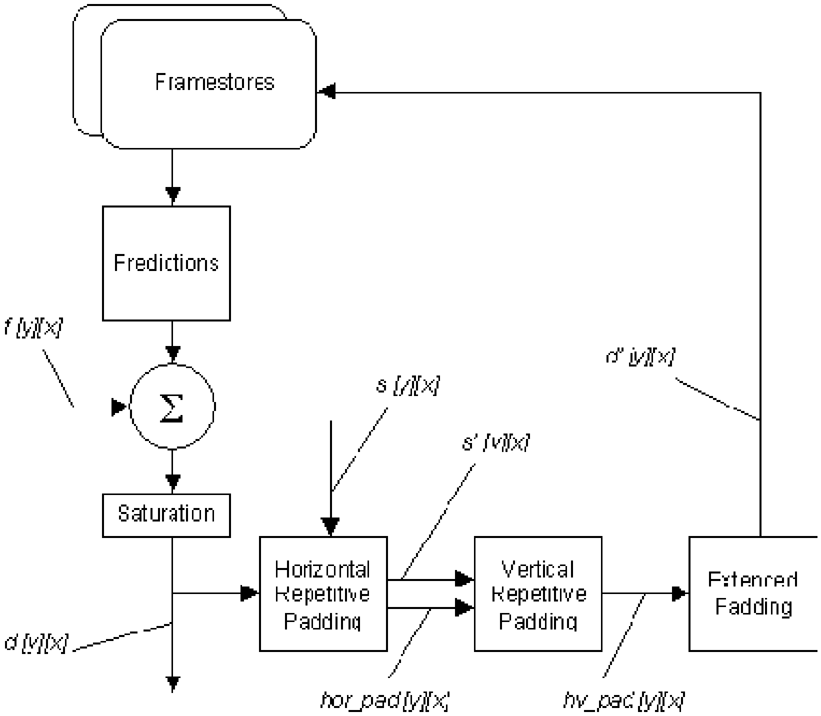


Figure -16 -- Simplified padding process

A decoded macroblock *d[y][x]* is padded by referring to the corresponding decoded shape block *s[y][x]*. The luminance component is padded per 16 x 16 samples, while the chrominance components are padded per 8 x 8 samples. A macroblock that lies on the VOP boundary (hereafter referred to as a boundary macroblock) is padded by replicating the boundary samples of the VOP towards the exterior. This process is divided into horizontal repetitive padding and vertical repetitive padding. The remaining macroblocks that are completely outside the VOP (hereafter referred to as exterior macroblocks) are filled by extended padding.

NOTE The padding process is applied to all macroblocks inside the bounding rectangle of a VOP. The bounding rectangle of the luminance component is defined by vop_width and vop_height extended to multiple of 16, while that of the chrominance components is defined by (vop_width>>1) and (vop_height>>1) extended to multiple of 8.

1. **Horizontal repetitive padding**

Each sample at the boundary of a VOP is replicated horizontally to the left and/or right direction in order to fill the transparent region outside the VOP of a boundary macroblock. If there are two boundary sample values for filling a sample outside of a VOP, the two boundary samples are averaged (//2).

hor_pad[y][x] is generated by any process equivalent to the following example. For every line with at least one shape sample *s[y][x]* == 1(inside the VOP) :

```
for (x=0; x<N; x++) {
    if (s[y][x] == 1) { hor_pad[y][x] = d[y][x]; s?[y][x] = 1; }
    else {
        if ( s[y][x?] == 1 && s[y][x"] == 1) {
            hor_pad[y][x] = (d[y][x?]+ d[y][x"])//2;
            s?[y][x] = 1;
        } else if ( s[y][x?] == 1 ) {
            hor_pad[y][x] = d[y][x?]; s?[y][x] = 1;
        } else if ( s[y][x"] == 1 ) {
            hor_pad[y][x] = d[y][x"]; s?[y][x] = 1;
        }
    }
}
```

where *x?* is the location of the nearest valid sample (*s[y][x?]* == 1) at the VOP boundary to the left of the current location *x*, *x"* is the location of the nearest boundary sample to the right, and *N* is the number of samples of a line in a macroblock. *s?[y][x]* is initialized to 0.

2. **Vertical repetitive padding**

The remaining unfilled transparent horizontal samples (where *s?[y][x]* == 0) from subclause 7.6.1.1 are padded by a similar process as the horizontal repetitive padding but in the *vertical* direction. The samples already filled in subclause 7.6.1.1 are treated as if they were inside the VOP for the purpose of this vertical pass.

hv_pad[y][x] is generated by any process equivalent to the following example. For every column of *hor_pad[y][x]* :

```
for (y=0; y<M; y++) {
    if (s?[y][x] == 1)
        hv_pad[y][x] =hor_pad[y][x];
    else {
        if ( s?[y?][x] == 1 && s?[y"][x] == 1 )
            hv_pad[y][x] = (hor_pad[y?][x] + hor_pad[y"][x])//2;
        else if ( s?[y?][x] == 1 )
hv_pad[y][x] = hor_pad[y?][x];
else if (s?[y"][x] == 1 )
hv_pad[y][x] = hor_pad[y"][x];
}
}
```

where *y?* is the location of the nearest valid sample (*s?[y?][x]* == 1) above the current location *y* at the boundary of *hv_pad*, *y"* is the location of the nearest boundary sample below *y*, and *M* is the number of samples of a column in a macroblock.

3. **Extended padding**

Exterior macroblocks immediately next to boundary macroblocks are filled by replicating the samples at the border of the boundary macroblocks. Note that the boundary macroblocks have been completely padded in subclause 7.6.1.1 and subclause 7.6.1.2. If an exterior macroblock is next to more than one boundary macroblocks, one of the macroblocks is chosen, according to the following convention, for reference.

The boundary macroblocks surrounding an exterior macroblock are numbered in priority according to Figure 7-17. The exterior macroblock is then padded by replicating upwards, downwards, leftwards, or rightwards the row of samples from the horizontal or vertical border of the boundary macroblock having the largest priority number.

The remaining exterior macroblocks (not located next to any boundary macroblocks) are filled with $2^{\text{bits\_per\_pixel-1}}$. For 8-bit luminance component and associated chrominance this implies filling with 128.

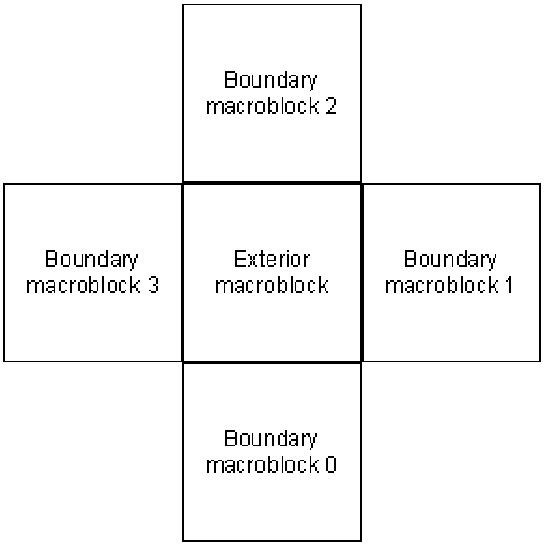


Figure -17 -- Priority of boundary macroblocks surrounding an exterior macroblock

4. **Padding for chrominance components**

Chrominance components are padded according to subclauses 7.6.1.1 through 7.6.1.3 for each 8 x 8 block. The padding is performed by referring to a shape block generated by decimating the shape block of the corresponding luminance component. This decimating of the shape block is performed by the subsampling process described in subclause 6.1.3.6.

5. **Padding of interlaced macroblocks**

   Macroblocks of interlaced VOP (interlaced = 1) are padded according to subclauses 7.6.1.1 through 7.6.1.3. The vertical padding of the luminance component, however, is performed for each field independently. A sample outside of a VOP is therefore filled with the value of the nearest boundary sample of the same field. Completely transparent blocks are padded with $2^{\text{bits\_per\_pixel-1}}$. Chrominance components of interlaced VOP are padded according to subclause 7.6.1.4, however, based on fields to enhance subjective quality of display in 4:2:0 format. The padding method described in this subclause is not used outside the bounding rectangle of the VOP.

6. **Vector padding technique**

The vector padding technique is applied to generate the vectors for the transparent blocks within a non-transparent macroblock, for an INTRA-coded macroblock and for a skipped macroblock. It works in a similar way as the horizontal followed by the vertical repetitive padding, and can be simply regarded as the repetitive padding performed on a 2x2 block except that the padded values are two dimensional vectors. A macroblock has four 8x8 luminance blocks, let {MVx[i], MVy[i], i=0,1,2,3} and {Transp[i], i=0,1,2,3} be the vectors and the transparencies of the four 8x8 blocks, respectively, the vector padding is any process numerically equivalent to:

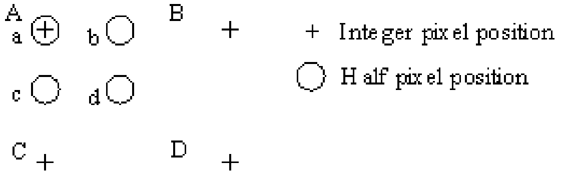
```
if (the macroblock is INTRA-coded, skipped ) {

MVx[0] = MVx[1] = MVx[2] = MVx[3] = 0

MVy[0] = MVy[1] = MVy[2] = MVy[3] = 0

} else {

if(Transp[0] == TRANSPARENT) {

MVx[0]=(Transp[1] != TRANSPARENT) ? MVx[1] :((Transp[2]!=TRANSPARENT) ? MVx[2]:MVx[3]));

MVy[0]=(Transp[1] != TRANSPARENT) ? MVy[1]:((Transp[2]!=TRANSPARENT) ? MVy[2]:MVy[3]));

}

if(Transp[1] == TRANSPARENT) {

MVx[1]=(Transp[0] != TRANSPARENT) ? MVx[0] :((Transp[3]!=TRANSPARENT) ? MVx[3]:MVx[2]));

MVy[1]=(Transp[0] != TRANSPARENT) ? MVy[0]:((Transp[3]!=TRANSPARENT) ? MVy[3]:MVy[2]));

}

if(Transp[2] == TRANSPARENT) {

MVx[2]=(Transp[3] != TRANSPARENT) ? MVx[3] :((Transp[0]!=TRANSPARENT) ? MVx[0]:MVx[1]));

MVy[2]=(Transp[3] != TRANSPARENT) ? MVy[3]:((Transp[0]!=TRANSPARENT) ? MVy[0]:MVy[1]));
```

```
}

if(Transp[3] == TRANSPARENT) {

MVx[3]=(Transp[2] != TRANSPARENT) ? MVx[2] :((Transp[1]!=TRANSPARENT) ? MVx[1]:MVx[0]));

MVy[3]=(Transp[2] !=TRANSPARENT) ? MVy[2]:((Transp[1]!=TRANSPARENT) ? MVy[1]:MVy[0]));

}

}
```

Vector padding is only used in I- and P-VOPs, it is applied on a macroblock directly after it is decoded. The block vectors after padding are used in the P-VOP vector decoding and binary shape decoding, and in the B-VOP direct mode decoding.

2. **Half sample interpolation**

Pixel value interpolation for block matching when rounding is used corresponds to bilinear interpolation as depicted in Figure 7-18. The value of rounding_control is defined using the vop_rounding_type bit in the VOP header (see subclause 6.3.5). Note that the samples outside the padded region cannot be used for interpolation.



a = A,
b = (A + B + 1 - rounding_control) / 2
c = (A + C + 1 - rounding_control) / 2,
d = (A + B + C + D + 2 - rounding_control) / 4

Figure -18 -- Interpolation scheme for half sample search

3. **General motion vector decoding process**

To decode a motion vector (MVx, MVy), the differential motion vector (MVDx, MVDy) is extracted from the bitstream by using the variable length decoding. Then it is added to a motion vector predictor (Px, Py) component wise to form the final motion vector. The general motion vector decoding process is any process that is equivalent to the following one. All calculations are carried out in halfpel units in the following. This process is generic in the sense that it is valid for the motion vector decoding in interlaced/progressive P- and B-VOPs except that the generation of the predictor (Px, Py) may be different.

r_size = vop_*fcode* - 1

f = 1 << *r_size*

high = (32 * *f*) - 1;

low = ((-32) * *f*);

```
range = ( 64 * f );
if ( (f == 1) || (horizontal_mv_data == 0) )
    MVDx = horizontal_mv_data;
else {
    MVDx = ( ( Abs (horizontal_mv_data) - 1 ) * f ) + horizontal_mv_residual + 1;
    if (horizontal_mv_data < 0)
        MVDx = - MVDx;
}
if ( (f == 1) || (vertical_mv_data == 0) )
    MVDy = vertical_mv_data;
else {
    MVDy = ( ( Abs(vertical_mv_data) - 1 ) * f ) + vertical_mv_residual + 1;
    if (vertical_mv_data < 0)
        MVDy = - MVDy;
}


MVx = Px + MVDx;
if ( MVx < low )
    MVx = MVx + range;
if (MVx > high)
    MVx = MVx - range;
MVy = Py + MVDy;
if ( MVy < low )
    MVy = MVy + range;
if (MVy > high)
    MVy = MVy - range;
```

The parameters in the bitstream shall be such that the components of the reconstructed differential motion vector, *MVDx* and *MVDy*, shall lie in the range [*low*:*high]*. In addition the components of the reconstructed motion vector, *MVx* and *MVy*, shall also lie in the range [*low* : *high*]. The allowed range [low : high] for the motion vectors depends on the parameter vop_fcode; it is shown in Table 7-5.

The variables *r_size*, *f*, *MVDx*, *MVDy*, *high* , *low* and *range* are temporary variables that are not used in the remainder of this part of ISO/IEC 14496. The parameters horizontal_mv_data, vertical_mv_data, horizontal_mv_residual and vertical_mv_residual are parameters recovered from the bitstream.

The variable *vop_fcode* refers either to the parameter vop_fcode_forward or to the parameter vop_fcode_backward which have been recovered from the bitstream, depending on the respective prediction mode. In the case of P-VOP prediction only forward prediciton applies. In the case of B-VOP prediction, forward as well as backward prediction may apply.

Table -5 -- Range for motion vectors

vop_fcode_forward or vop_fcode_backward	**motion vector range in halfsample units [low:high]**
1	[-32,31]
2	[-64,63]
3	[-128,127]
4	[-256,255]
5	[-512,511]
6	[-1024,1023]
7	[-2048,2047]

If the current macroblock is a field motion compensated macroblock, then the same prediction motion vector (Px, Py) is used for both field motion vectors. Because the vertical component of a field motion vector is integral, the vertical differential motion vector encoded in the bitstream is

$$MVy = MVDy_{field} + PY / 2$$

4. **Unrestricted motion compensation**

Motion vectors are allowed to point outside the decoded area of a reference VOP when (and only when) the short video header format is not in use (i.e., when short_video_header is 0). For an arbitrary shape VOP, the decoded area refers to the area within the bounding rectangle, padded as described in subclause 7.6.1. A bounding rectangle is defined by vop_width and vop_height extended to multiple of 16. When a sample referenced by a motion vector stays outside the decoded VOP area, an edge sample is used. This edge sample is retrieved by limiting the motion vector to the last full pel position inside the decoded VOP area. Limitation of a motion vector is performed on a sample basis and separately for each component of the motion vector, as depicted in Figure 7-19.

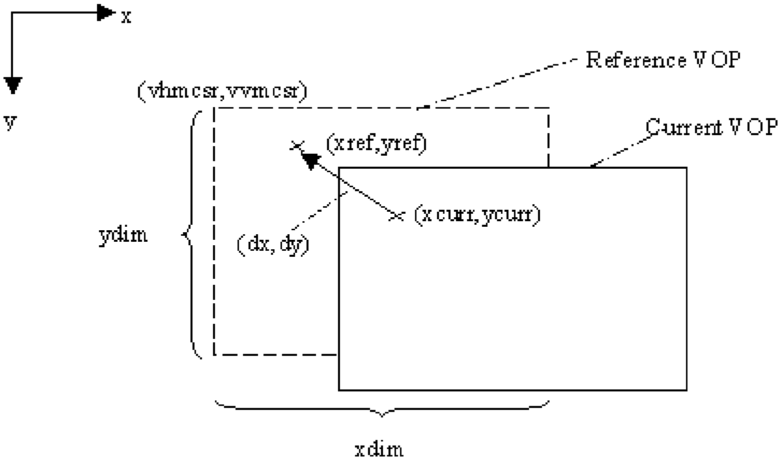


Figure -19 -- Unrestricted motion compensation

The coordinates of a reference sample in the reference VOP, (yref, xref) is determined as follows :

xref = MIN (MAX (xcurr+dx, vhmcsr), xdim+vhmcsr-1)

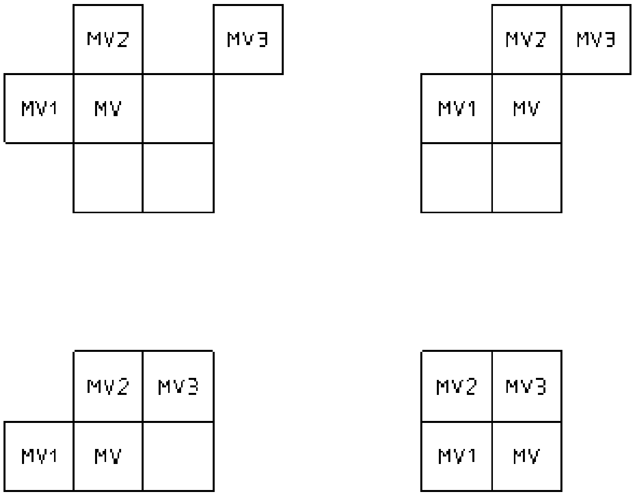
yref = MIN (MAX (ycurr+dy, vvmcsr), ydim+vvmcsr-1)

where vhmcsr = vop_horizontal_mc_spatial_reference, vvmcsr = vop_vertical_mc_spatial_reference, (ycurr, xcurr) are the coordinates of a sample in the current VOP, (yref, xref) are the coordinates of a sample in the reference VOP, (dy, dx) is the motion vector, and (ydim, xdim) are the dimensions of the bounding rectangle of the reference VOP. All coordinates are related to the absolute coordinate system shown in Figure 7-19. Note that for rectangular VOP, a reference VOP is defined by video_object_layer_width and video_object_layer_height. For an arbitrary shape VOP, a reference VOP of luminance is defined by vop_width and vop_height extended to multiple of 16, while that of chrominance is defined by (vop_width>>1 **)** and (vop_height>>1) extended to multiple of 8.

5. **Vector decoding processing and motion-compensation in progressive P-VOP**

An inter-coded macroblock comprises either one motion vector for the complete macroblock or K (1< K<=4) motion vectors, one for each non-transparent 8x8 pel blocks forming the 16x16 pel macroblock, as is indicated by the mcbpc code.

For decoding a motion vector, the horizontal and vertical motion vector components are decoded differentially by using a prediction, which is formed by a median filtering of three vector candidate predictors (MV1, MV2, MV3) from the spatial neighbourhood macroblocks or blocks already decoded. The spatial position of candidate predictors for each block vector is depicted in Figure 7-20. In the case of only one motion vector present for the complete macroblock, the top-left case in Figure 7-20 is applied. When the short video header format is in use (i.e., when short_video_header is "1"), only one motion vector shall be present for a macroblock.



.

Figure -20 -- Definition of the candidate predictors MV1, MV2 and MV3 for each of the luminance blocks in a macroblock

The following four decision rules are applied to obtain the value of the three candidate predictors:

1. If a candidate predictor MVi is in a transparent spatial neighbourhood macroblock or in a transparent block of the current macroblock it is not valid, otherwise, it is set to the corresponding block vector.
2. If one and only one candidate predictor is not valid, it is set to zero.
3. If two and only two candidate predictors are not valid, they are set to the third candidate predictor.
4. If all three candidate predictors are not valid, they are set to zero.

Note that any neighbourhood macroblock outside the current VOP or video packet or outside the current GOB (when short_video_header is "1") for which gob_header_empty is "0" is treated as transparent in the above sense. The median value of the three candidates for the same component is computed as predictor, denoted by Px and Py:

$$Px = Median(MV1x, MV2x, MV3x)$$
$$Py = Median(MV1y, MV2y, MV3y)$$

For instance, if MV1=(-2,3), MV2=(1,5) and MV3=(-1,7), then Px = -1 and Py = 5. The final motion vector is then obtained by using the general decoding process defined in the subclause 7.6.3.

If four vectors are used, each of the motion vectors is used for all pixels in one of the four luminance blocks in the macroblock. The numbering of the motion vectors is equivalent to the numbering of the four luminance blocks as given in Figure 6-5. Motion vector MVDCHR for both chrominance blocks is derived by calculating the sum of the *K* luminance vectors, that corresponds to *K* 8x8 blocks that do not lie outside the VOP shape and dividing this sum by 2**K;* the component values of the resulting sixteenth/twelfth/eighth/fourth sample resolution vectors are modified towards the nearest half sample position as indicated below.

Table -6 -- Modification of sixteenth sample resolution chrominance vector components

sixteenth pixel position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	//16
resulting position	0	0	0	1	1	1	1	1	1	1	1	1	1	1	2	2	//2

Table -7 -- Modification of twelfth sample resolution chrominance vector components

twelfth pixel position	0	1	2	3	4	5	6	7	8	9	10	11	//12
resulting position	0	0	0	1	1	1	1	1	1	1	2	2	//2

Table -8 -- Modification of eighth sample resolution chrominance vector components

eighth pixel position	0	1	2	3	4	5	6	7	//8
resulting position	0	0	1	1	1	1	1	2	//2

Table -9 -- Modification of fourth sample resolution chrominance vector components

fourth pixel position	0	1	2	3	//4
resulting position	0	1	1	1	//2

Half sample values are found using bilinear interpolation as described in subclause 7.6.2. The prediction for luminance is obtained by overlapped motion compensation as described in subclause 7.6.6 if indicated by obmc_disable==0. The prediction for chrominance is obtained by applying the motion vector MVDCHR to all pixels in the two chrominance blocks.

1. Overlapped motion compensation

This subclause specifies the overlapped motion compensation process. This process is performed when the flag obmc_disable=0.

Each pixel in an 8*8 luminance prediction block is a weighted sum of three prediction values, divided by 8 (with rounding). In order to obtain the three prediction values, three motion vectors are used: the motion vector of the current luminance block, and two out of four "remote" vectors:

- the motion vector of the block at the left or right side of the current luminance block;
- the motion vector of the block above or below the current luminance block.

For each pixel, the remote motion vectors of the blocks at the two nearest block borders are used. This means that for the upper half of the block the motion vector corresponding to the block above the current block is used, while for the lower half of the block the motion vector corresponding to the block below the current block is used. Similarly, for the left half of the block the motion vector corresponding to the block at the left side of the current block is used, while for the right half of the block the motion vector corresponding to the block at the right side of the current block is used.

The creation of each pixel, $\bar{p}(i,j),$ in an 8*8 luminance prediction block is governed by the following equation

$$\bar{p}(i,j) = (q(i,j) \times H_0(i,j) + r(i,j) \times H_1(i,j) + s(i,j) \times H_2(i,j) + 4) // 8,$$

where $q(i,j),\ r(i,j),$ and $s(i,j)$ are the pixels from the referenced picture as defined by

$$q(i,j) = p(i + MV_x^0, j + MV_y^0),$$
$$r(i,j) = p(i + MV_x^1, j + MV_y^1),$$
$$s(i,j) = p(i + MV_x^2, j + MV_y^2).$$

Here, (MV_x^0, MV_y^0) denotes the motion vector for the current block, (MV_x^1, MV_y^1) denotes the motion vector of the block either above or below, and (MV_x^2, MV_y^2) denotes the motion vector either to the left or right of the current block as defined above.

The matrices $H_0(i,j), H_1(i,j)$ and $H_2(i,j)$ are defined in Figure 7-21, Figure 7-22, and Figure 7-23, where (i,j) denotes the column and row, respectively, of the matrix.

If one of the surrounding blocks was not coded, the corresponding remote motion vector is set to zero. If one of the surrounding blocks was coded in intra mode, the corresponding remote motion vector is replaced by the motion vector for the current block. If the current block is at the border of the VOP and therefore a surrounding block is not present, the corresponding remote motion vector is replaced by the current motion vector. In addition, if the current block is at the bottom of the macroblock, the remote motion vector corresponding with an 8*8 luminance block in the macroblock below the current macroblock is replaced by the motion vector for the current block.

4	5	5	5	5	5	5	4
5	5	5	5	5	5	5	5
5	5	6	6	6	6	5	5
5	5	6	6	6	6	5	5
5	5	6	6	6	6	5	5
5	5	6	6	6	6	5	5
5	5	5	5	5	5	5	5
4	5	5	5	5	5	5	4

Figure -21 -- Weighting values, H_0, for prediction with motion vector of current luminance block

2	2	2	2	2	2	2	2
1	1	2	2	2	2	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	2	2	2	2	1	1
2	2	2	2	2	2	2	2

Figure -22 -- Weighting values, H1 , for prediction with motion vectors of the luminance blocks on top or bottom of current luminance block

2	1	1	1	1	1	1	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	1	1	1	1	1	1	2

Figure -23 -- Weighting values, H_2 , for prediction with motion vectors of the luminance blocks to the left or right of current luminance block

1. **Temporal prediction structure**

1. **A forward reference VOP is defined as a most recently decoded I- orP-VOP in the past for which "vop_coded==1". A backward reference VOP is defined as the most recently decoded I- or P-VOP in the future, regardless of its value for "vop_coded".**
1. **A target P-VOP shall make reference to the forward reference VOP**
1. **A target B-VOP can make reference**

   - **to the forward and/or the backward reference VOP, if for the backward reference VOP "vop_coded==1"**
   - **only to the forward reference VOP, if for the backward reference VOP "vop_coded==0"**

1. **Note that for the reference VOP selection of binary shape coding the rules stated in subclause 7.5.2.4 shall be applied**

The temporal prediction structure is depicted in Figure 7-24.

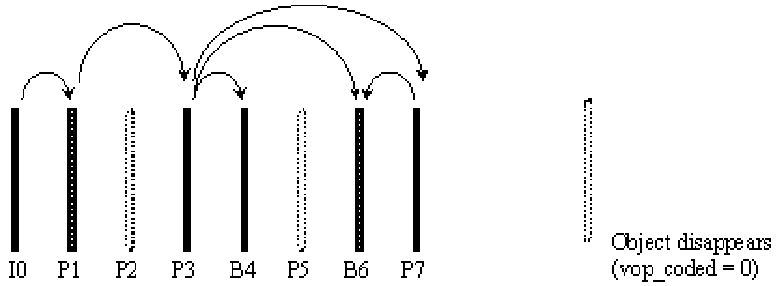


Figure -24 -- Temporal Prediction Structure

1. **Vector decoding process of non-scalable progressive B-VOPs**

In B-VOPs there are three kinds of vectors, namely, 16x16 forward vector, 16x16 backward vector and the delta vector for the direct mode. The vectors are decoded with respect to the corresponding vector predictors. The basic decoding process of a differential vector is the exactly same as defined in P-VOPs except that for the delta vector of the direct mode the f_code is always one. The vector is then reconstructed by adding the decoded differential vector to the corresponding vector predictor. The vector predictor for the delta vector is always set to zero, while the forward and backward vectors have their own vector predictors, which are reset to zero only at the beginning of each macroblock row. The vector predictors are updated in the following three cases:

- **after decoding a macroblock of forward mode only the forward predictor is set to the decoded forward vector**
- **after decoding a macroblock of backward mode only the backward predictor is set to the decoded backward vector.**
- **after decoding a macroblock of bi-directional mode both the forward and backward predictors are updated separately with the decoded vectors of the same type (forward/backward).**

1. **Motion compensation in non-scalable progressive B-VOPs**

In B-VOPs the overlapped motion compensation (OBMC) is not employed. The motion-compensated prediction of B-macroblock is generated by using the decoded vectors and taking reference to the padded forward/backward reference VOPs as defined below. Arbitrarily shaped reference VOPs shall be padded accordingly.

1. **Basic motion compensation procedure**

All of the ISO/IEC 14496-2 motion compensation techniques are based on

the formation of a prediction block, pred[i][j] of dimension (width, height), from a reference image, ref[x][y]. The coordinates of the current block (or macroblock) in the reference VOP is (x,y), the motion half-pel resolution motion vector is (dx_halfpel, dy_halfpel). The pseudo-code for this procedure is given below.

The component_width() and component_height() function give the coded VOP dimensions for the current component. For luminance, component_width() is video_object_layer_width for a rectangular VOP or vop_width otherwise rounded up to the next multiple of 16. The luminance component_height() is defined similarly. The chrominance dimensions are one half of the corresponding luminance dimension.

```
clip_ref(ref, x, y)
{
return(ref[MIN(MAX(x, 0), component_width(ref) - 1)]
[MIN(MAX(y, 0), component_height(ref) - 1)]);
}
mc(pred, /* prediction block */
ref, /* reference component */
x, y, /* ref block coords for MV=(0, 0) */
width, height, /* reference block dimensions */
dx_halfpel, dy_halfpel, /* half-pel resolution motion vector */
rounding, /* rounding control (0 or 1 ) */
pred_y0, /* field offset in pred blk (0 or 1) */
ref_y0, /* field offset in ref blk (0 or 1) */
y_incr) /* vertical increment (1 or 2) */
{
dx = dx_halfpel >> 1;
dy = y_incr * (dy_halfpel >> y_incr);
if (dy_halfpel & y_incr) {
if (dx_halfpel & 1) {
for (iy = 0; iy < height; iy += y_incr) {
for (ix = 0; ix < width; ix++) {
```

```
x_ref = x + dx + ix;
y_ref = y + dy + iy + ref_y0;
pred[ix][iy + pred_y0] =
(clip_ref(ref, x_ref + 0, y_ref + 0) +
clip_ref(ref, x_ref + 1, y_ref + 0) +
clip_ref(ref, x_ref + 0, y_ref + y_incr) +
clip_ref(ref, x_ref + 1, y_ref + y_incr) +
2 - rounding) >> 2;
}
}
} else {
for (iy = 0; iy < height; iy += y_incr) {
for (ix = 0; ix < width; ix++) {
x_ref = x + dx + ix;
y_ref = y + dy + iy + ref_y0;
pred[ix][iy + pred_y0] =
(clip_ref(ref, x_ref, y_ref + 0) +
clip_ref(ref, x_ref, y_ref + y_incr) +
1 - rounding) >> 1;
}
}
}
} else {
if (dx_halfpel & 1) {
for (iy = 0; iy < height; iy += y_incr) {
for (ix = 0; ix < width; ix++) {
x_ref = x + dx + ix;
y_ref = y + dy + iy + ref_y0;
pred[ix][iy + pred_y0] =
(clip_ref(ref, x_ref + 0, y_ref) +
clip_ref(ref, x_ref + 1, y_ref) +
1 - rounding) >> 1;
}
}
} else {
for (iy = 0; iy < height; iy += y_incr) {
for (ix = 0; ix < width; ix++) {
x_ref = x + dx + ix;
y_ref = y + dy + iy + ref_y0;
pred[ix][iy + pred_y0] =
clip_ref(ref, x_ref, y_ref);
}
}
}
}
}
```

2. **Forward mode**

Only the forward vector (MVFx,MVFy) is applied in this mode. The prediction blocks Pf_Y, Pf_U, and Pf_V are generated from the forward reference VOP, ref_Y_for for luminance component and ref_U_for and ref_V_for for chrominance components, as follows:

```
mc(Pf_Y, ref_Y_for, x, y, 16, 16, MVFx, MVFy, 0, 0, 0, 1);
mc(Pf_U, ref_U_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1);
mc(Pf_V, ref_V_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1);
```

where (MVFx_chro, MVFy_chro) is motion vector derived from the luminance motion vector by dividing each component by 2 then rounding on a basis of Table 7-9. Here (and hereafter) the function MC is defined in subclause 7.6.9.

3. **Backward mode**

Only the backward vector (MVBx,MVBy) is applied in this mode. The prediction blocks Pb_Y, Pb_U, and Pb_V are generated from the backward reference VOP, ref_Y_back for luminance component and ref_U_back and ref_V_back for chrominance components, as follows:

mc(Pb_Y, ref_Y_back, x, y, 16, 16, MVBx, MVBy, 0, 0, 0, 1);

mc(Pb_U, ref_U_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1);

mc(Pb_V, ref_V_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1);

where (MVBx_chro, MVBy_chro) is motion vector derived from the luminance motion vector by dividing each component by 2 then rounding on a basis of Table 7-9.

4. **Bi-directional mode**

Both the forward vector (MVFx,MVFy) and the backward vector (MVBx,MVBy) are applied in this mode. The prediction blocks Pi_Y, Pi_U, and Pi_V are generated from the forward and backward reference VOPs by doing the forward prediction, the backward prediction and then averaging both predictions pixel by pixel as follows.

mc(Pf_Y, ref_Y_for, x, y, 16, 16, MVFx, MVFy, 0, 0, 0, 1);

mc(Pf_U, ref_U_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1);

mc(Pf_V, ref_V_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1);

mc(Pb_Y, ref_Y_back, x, y, 16, 16, MVBx, MVBy, 0, 0, 0, 1);

mc(Pb_U, ref_U_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1);

mc(Pb_V, ref_V_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1);

Pi_Y[i][j] = (Pf_Y[i][j] + Pb_Y[i][j] + 1)>>1; i,j=0,1,2?15;

Pi_U[i][j] = (Pf_U[i][j] + Pb_U[i][j] + 1)>>1; i,j=0,1,2?8;

Pi_V[i][j] = (Pf_V[i][j] + Pb_V[i][j] + 1)>>1; i,j=0,1,2?8;

where (MVFx_chro, MVFy_chro) and (MVBx_chro, MVBy_chro) are motion vectors derived from the forward and backward luminance motion vectors by dividing each component by 2 then rounding on a basis of Table 7-9, respectively.

5. **Direct mode**

This mode uses direct bi-directional motion compensation derived by employing I- or P-VOP macroblock motion vectors and scaling them to derive forward and backward motion vectors for macroblocks in B-VOP. This is the only mode which makes it possible to use motion vectors on 8x8 blocks. Only one delta motion vector is allowed per macroblock.

1. **Formation of motion vectors for the direct mode**

The direct mode utilises the motion vectors (MVs) of the co-located macroblock in the most recently decoded I- or P-VOP. The co-located macroblock is defined as the macroblock which has the same horizontal and vertical index with the current macroblock in the B-VOP. The MV vectors are the block vectors of the co-located macroblock after applying the vector padding defined in subclause 7.6.1.6. If the co-located macroblock is transparent and thus the MVs are not available, the direct mode is still enabled by setting

MV vectors to zero vectors.

2. **Calculation of vectors**

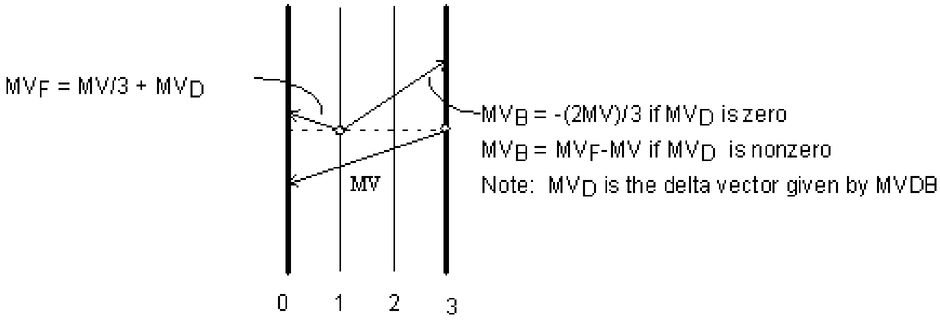

MVF = MV/3 + MVD
MVB = -(2MV)/3 if MVD is zero
MVB = MVF-MV if MVD is nonzero
Note: MVD is the delta vector given by MVDB
MV
0
1
2
3

Figure -25 -- Direct Bi-directional Prediction

Figure 7-25 shows scaling of motion vectors. The calculation of forward and backward motion vectors involves linear scaling of the collocated block in temporally next I- or P-VOP, followed by correction by a delta vector (MVDx,MVDy). The forward and the backward motion vectors are {(MVFx[i],MVFy[i]), (MVBx[i],MVBy[i]), i = 0,1,2,3} and are given in half sample units as follows.

MVFx[i] = (TRB x MVx[i]) / TRD + MVDx

MVBx[i] = (MVDx==0)? ((TRB - TRD) x MVx[i]) / TRD : MVFx[i] - MVx[i]

MVFy[i] = (TRB x MVy[i]) / TRD + MVDy

MVBy[i] = (MVDy==0)? ((TRB - TRD) x MVy[i]) / TRD : MVFy[i] - MVy[i]

i = 0,1,2,3.

where {(MVx[i],MVy[i]), i = 0,1,2,3} are the MV vectors of the co-located macroblock, TRD is the difference in temporal reference of the B-VOP and the previous reference VOP. TRD is the difference in temporal reference of the temporally next reference VOP with temporally previous reference VOP, assuming B-VOPs or skipped VOPs in between.

3. **Generation of prediction blocks**

Motion compensation for luminance is performed individually on 8x8 blocks to generate a macroblock. The process of generating a prediction block simply consists of using computed forward and backward motion vectors {(MVFx[i],MVFy[i]), (MVBx[i],MVBy[i]), i = 0,1,2,3} to obtain appropriate blocks from reference VOPs and averaging these blocks, same as the case of bi-directional mode except that motion compensation is performed on 8x8 blocks.

For the motion compensation of both chrominance blocks, the forward motion vector (MVFx_chro, MVFy_chro) is calculated by the sum of K forward luminance motion vectors dividing by 2K and then rounding toward the nearest half sample position as defined in Table 7-6 to Table 7-9. The backward motion vector (MVBx_chro, MVBy_chro) is derived in the same way. The rest process is the same as the chrominance motion compensation of the bi-directional mode described in subclause 7.6.9.4.

6. **Motion compensation in skipped macroblocks**

If the co-located macroblock in the most recently decoded I- or P-VOP is skipped, the current B-macroblock is treated as the forward mode with the zero motion vector (MVFx,MVFy). If the modb equals to ?1? the current B-macroblock is reconstructed by using the direct mode with zero delta vector.

1. Interlaced video decoding

This subclause specifies the additional decoding process that a decoder shall perform to recover VOP data from the coded bitstream when the interlaced flag in the VOP header is set to "1". Interlaced information (subclause 6.3.6.3) specifies the method to decode bitstream of interlaced VOP.

1. Field DCT and DC and AC Prediction

When dct_type flag is set to ?1? (field DCT coding), DCT coefficients of luminance data are formed such that each 8x8 block consists of data from one field as being shown in Figure 6-7. DC and optional AC (see "ac_pred_flag") prediction will be performed for a intra-coded macroblock. For the intra macroblocks which have dct_type flag being set to "1", DC/AC prediction are performed to field blocks shown in Figure 7-26. After taking inverse DCT, all luminance blocks will be inverse permuted back to (frame) macroblock. Chrominance (block) data are not effected by dct_type flag.

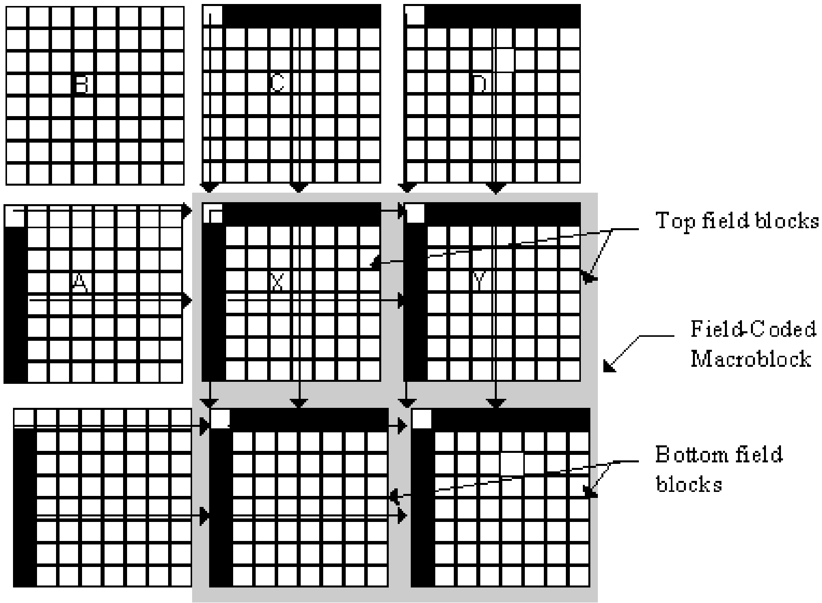


Figure -26 -- Previous neighboring blocks used in DC/AC prediction for interlaced intra blocks.

2. Motion compensation

For non-intra macroblocks in P- and B-VOPs, motion vectors are extracted syntactically following subclause 6.2.6 "Macroblock". The motion vector decoding is performed separately on the horizontal and vertical components.

1. Motion vector decoding in P-VOP

For each component of motion vector in P-VOPs, the median value of the candidate predictor vectors for the same component is computed and add to corresponding component of the motion vector difference obtained from the bitstream. To decode the motion vectors in a P-VOP, the decoder shall first extract the differential motion vectors ($(MVDx_{f1}, MVDy_{f1})$ and $(MVDx_{f2}, MVDy_{f2})$ for top and bottom fields of a field predicted macroblock, respectively) by a use of variable length decoding and then determine the predictor vector from three candidate vectors. These candidate predictor vectors are generated from the three motion vectors of three spatial neighborhood decoded macroblocks or blocks as follows.

CASE 1 :

If the current macroblock is a field predicted macroblock and none of the coded spatial neighborhood macroblocks is a field predicted macroblock, then candidate predictor vectors MV1, MV2, and MV3 are defined by Figure 7-27. If the candidate block *i* is not in four MV motion (8x8) mode, MV*i* represents the motion vector for the macroblock. If the candidate block *i* is in four MV motion (8x8) mode, the 8x8 block motion vector closest to the upper left block of the current MB is used. The predictors for the horizontal and vertical components are then computed by

$$P_x = Median(MV1x, MV2x, MV3x)$$
$$P_y = Median(MV1y, MV2y, MV3y).$$

For differential motion vectors both fields use the same predictor and motion vectors are recovered by

$$MVx_{f1} = MVDx_{f1} + P_x$$
$$MVy_{f1} = 2 * (MVDy_{f1} + (P_y / 2))$$
$$MVx_{f2} = MVDx_{f2} + P_x$$
$$MVy_{f2} = 2 * (MVDy_{f2} + (P_y / 2))$$

where "/" is integer division with truncation toward 0. Note that all motion vectors described above are specified as integers with one LSB representing a half-pel displacement. The vertical component of field motion vectors always even (in half-pel frame coordinates). Vertical half-pel interpolation between adjacent lines of the same field is denoted by MVy_{fi} be an odd multiple of 2 (e.g. -2,2,6,..) No vertical interpolation is needed when MVy_{fi} is an multiple of 4 (it is a full pel value).

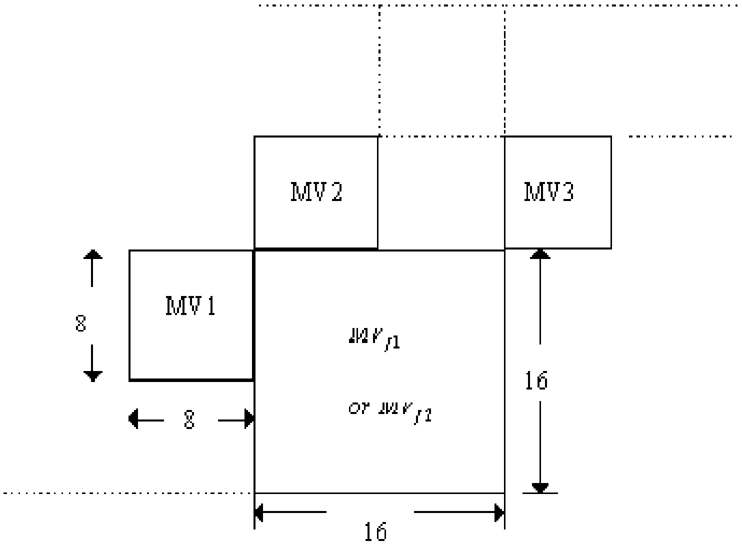


Figure -27 -- Example of motion vector prediction for field predicted macroblocks (Case1)

CASE 2 :

If the current macroblock or block is frame predicted macroblock or block and if at least one of the coded spatial neighborhood macroblocks is a field predicted macroblock, then the candidate predictor vector for each field predicted macroblock will be generated by averaging two field motion vectors such that all fractional pel offsets are mapped into the half-pel displacement. Each component (P_x or P_y) of the final predictor vector is the median value of the candidate predictor vectors for the same component. The motion vector is recovered by

$$MVx = MVDx + P_x$$
$$MVy = MVDy + P_y .$$

where

$$P_x = Median\left\{MV1x, Div2Round(MVx_{f1} + MVx_{f2}), MV3x\right\},$$
$$P_y = Median\left\{MV1y, Div2Round(MVy_{f1} + MVy_{f2}), MV3y\right\},$$

Div2Round(*x*) is defined as follows: *Div2Round*(*x*) = (*x* >> 1) | (*x* & 1).

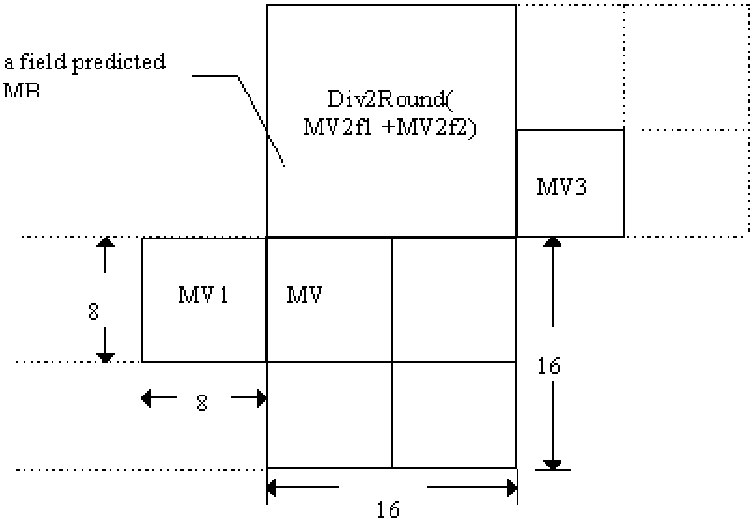


Figure -28 -- Example of motion vector prediction for field predicted macroblocks (Case 2)

CASE 3 :

Assume that the current macroblock is a field predicted macroblock and at least one of the coded spatial neighborhood macroblocks is a field predicted macroblock. If the candidate block *i* is field predicted, the candidate predictor vector MV*i* will be generated by averaging two field motion vectors such that all fractional pel offsets are mapped into the half-pel displacement as discribed in CASE 2. If the candidate block *i* is neither in four MV motion (8x8) mode nor in field prediction mode, MV*i* represents the frame motion vector for the macroblock. If the candidate block *i* is in four MV motion (8x8) mode, the 8x8 block motion vector closest to the upper left block of the current MB is used. The predictors for the horizontal and vertical components are then computed by

$$P_x = Median(MV1x, MV2x, MV3x)$$
$$P_y = Median(MV1y, MV2y, MV3y)$$

where

$$MVi\,x = Div2Round(MVx_{f1} + MVx_{f2}),$$
$$MVi\,y = Div2Round(MVy_{f1} + MVy_{f2}),$$

for some *i* in {1,2,3}.

For differential motion vectors both fields use the same predictor and motion vectors are recovered by (see both Figure 7-27 and Figure 7-28)

$$MVx_{f1} = MVDx_{f1} + P_x$$
$$MVy_{f1} = 2 * (MVDy_{f1} + (P_y / 2))$$
$$MVx_{f2} = MVDx_{f2} + P_x$$
$$MVy_{f2} = 2 * (MVDy_{f2} + (P_y / 2))$$

The motion compensated prediction macroblock is calculated calling the "field_compensate_one_reference" using the motion vectors calculated above. The top_field_ref, bottom_field_ref, and rounding type come directly from the syntax as forward_top_field_reference, forward_bottom_field_reference and vop_rounding_type respectively. The reference VOP is defined such the the even lines (0, 2, 4, ...) are the top field and the odd lines (1, 3, 5, ...) are the bottom field.

```
field_motion_compensate_one_reference(

luma_pred, cb_pred, cr_pred, /* Prediction component pel array */

luma_ref, cb_ref, cr_ref, /* Reference VOP pel arrays */

mv_top_x, mv_top_y, /* top field motion vector */

mv_bot_x, mv_bot_y, /* bottom field motion vector */

top_field_ref, /* top field reference */

bottom_field_ref, /* bottom field reference */

x, y, /* current luma macroblock coords */

rounding_type) /* rounding type */

{

mc(luma_pred, luma_ref, x, y, 16, 16, mv_top_x, mv_top_y,

rounding_type, 0, top_field_ref, 2);

mc(luma_pred, luma_ref, x, y, 16, 16, mv_bot_x, mv_bot_y,

rounding_type, 1, bottom_field_ref, 2);
mc(cb_pred, cb_ref, x/2, y/2, 8, 8,

Div2Round(mv_top_x), Div2Round(mv_top_y),

rounding_type, 0, top_field_ref, 2);

mc(cr_pred, cr_ref, x/2, y/2, 8, 8,

Div2Round(mv_top_x), Div2Round(mv_top_y),

rounding_type, 0, top_field_ref, 2);

mc(cb_pred, cb_ref, x/2, y/2, 8, 8,

Div2Round(mv_bot_x), Div2Round(mv_bot_y),

rounding_type, 0, top_field_ref, 2);

mc(cr_pred, cr_ref, x/2, y/2, 8, 8,

Div2Round(mv_bot_x), Div2Round(mv_bot_y),

rounding_type, 0, top_field_ref, 2);

}
```

In the case that obmc_disable is "0", the OBMC is not applied if the current MB is field-predicted. If the current MB is frame-predicted (including 8x8 mode) and some adjacent MBs are field-predicted, the motion vectors of those field-predicted MBs for OBMC are computed in the same manner as the candidate predictor vectors for field-predicted MBs are.

2. **Motion vector decoding in B-VOP**

For interlaced B-VOPs, a macroblock can be coded using (1) direct coding, (2) 16x16 motion compensation (includes forward, backward & bidirectional modes), or (3) field motion compensation (includes forward, backward & bidirectional modes). Motion vector in half sample accuracy will be employed for a 16x16 macroblock being coded. Chrominance vectors are derived by scaling of luminance vectors using the rounding tables described in Table 7-9 (i.e. by applying *Div2Round* to the luminance motion vectors). These coding modes except direct coding mode allow switching of quantizer from the one previously in use. Specification of dquant, a differential quantizer involves a 2-bit overhead as discussed earlier. In direct coding mode, the quantizer value for previous coded macroblock is used.

For interlaced B-VOP motion vector predictors, four prediction motion vectors (PMVs) are used:

Table -10 -- Prediction motion vector allocation for interlaced P-VOPs

Function	**PMV**
Top field forward	0
Bottom field forward	1
Top field backward	2
Bottom field backward	3

These PMVs are used as follows for the different macroblock prediction modes:

Table -11 -- Prediction motion vectors for interlaced B-VOP decoding

Macroblock mode	PMVs used	PMVs updated
Direct	none	none
Frame forward	0	0,1
Frame backward	2	2,3
Frame bidirectional	0,2	0,1,2,3
Field forward	0,1	0,1
Field backward	2,3	2,3
Field bidirectional	0,1,2,3	0,1,2,3

The PMVs used by a macroblock are set to the value of current macroblock motion vectors after being used.

When a frame macroblock is decoded, the two field PMVs (top and bottom field) for each prediction direction are set to the same frame value. The PMVs are reset to zero at the beginning of each row of macroblocks. The predictors are not zeroed by skipped macroblocks or direct mode macroblocks.

The frame based motion compensation modes are described in subclause 7.6. The field motion compensation modes are calculated using the "field_motion_compensate_one_reference()" pseudo code function described above. The field forward mode is denoted by mb_type == "0001" and field_prediction == "1". The PMV update and calculation of the motion compensated prediction is shown below. The luma_fwd_ref_VOP[][], cb_fwd_ref_VOP[][], cr_fwd_ref_VOP[][] denote the entire forward (past) anchor VOP pixel arrays. The coordinates of the upper left corner of the luminance macroblock is given by (x, y) and MVD[].x and MVD[].y denote an array of the motion vector differences in the order they occur in the bitstream for the current macroblock.

```
PMV[0].x = PMV[0].x + MVD[0].x;

PMV[0].y = 2 * (PMV[0].y / 2 + MVD[0].y);

PMV[1].x = PMV[1].x + MVD[1].x;

PMV[1].y = 2 * (PMV[1].y / 2 + MVD[1].y);

field_motion_compensate_one_reference(

luma_pred, cb_pred, cr_pred,

luma_fwd_ref_VOP, cb_fwd_ref_VOP, cr_fwd_ref_VOP,

PMV[0].x, PMV[0].y, PMV[1].x, PMV[1].y,

forward_top_field_reference,

forward_bottom_field_reference,

x, y, 0);
```

The field backward mode is denoted by mb_type == "001" and field_prediction == "1". The PMV update and prediction calculation is outlined the following pseudo code. The luma_bak_ref_VOP[][], cb_bak_ref_VOP[][], cr_bak_ref_VOP[][] denote the entire backward (future) anchor VOP pixel arrays.

```
PMV[2].x = PMV[2].x + MVD[0].x;

PMV[2].y = 2 * (PMV[2].y / 2 + MVD[0].y);

PMV[3].x = PMV[1].x + MVD[1].x;

PMV[3].y = 2 * (PMV[3].y / 2 + MVD[1].y);

field_motion_compensate_one_reference(

luma_pred, cb_pred, cr_pred,

luma_bak_ref_VOP, cb_bak_ref_VOP, cr_bak_ref_VOP,

PMV[2].x, PMV[2].y, PMV[3].x, PMV[3].y,

backward_top_field_reference,

backward_bottom_field_reference,

x, y, 0);
```

The bidirectional field prediction is used when mb_type == "01" and field_prediction == "1". The prediction macroblock (in luma_pred[][], cb_pred[][], and cr_pred[][]) is calculated by:

```
for (mv = 0; mv < 4; mv++) {

PMV[mv].x = PMV[mv].x + MVD[mv].x;
```

```
PMV[mv].y = 2 * (PMV[mv].y / 2 + MVD[mv].y);
}
field_motion_compensate_one_reference(
luma_pred_fwd, cb_pred_fwd, cr_pred_fwd,
luma_fwd_ref_VOP, cb_fwd_ref_VOP, cr_fwd_ref_VOP,
PMV[0].x, PMV[0].y, PMV[1].x, PMV[1].y,
forward_top_field_reference,
forward_bottom_field_reference,
x, y, 0);
field_motion_compensate_one_reference(
luma_pred_bak, cb_pred_bak, cr_pred_bak,
luma_bak_ref_VOP, cb_bak_ref_VOP, cr_bak_ref_VOP,
PMV[2].x, PMV[2].y, PMV[3].x, PMV[3].y,
backward_top_field_reference,
backward_bottom_field_reference,
x, y, 0);
for (iy = 0; iy < 16; iy++) {
for (ix = 0; ix < 16; ix++) {
luma_pred[ix][iy] = (luma_pred_fwd[ix][iy] +
luma_pred_bak[ix][iy] + 1) >> 1;
}
}
for (iy = 0; iy < 8; iy++) {
for (ix = 0; ix < 8; ix++) {
cb_pred[ix][iy] = (cb_pred_fwd[ix][iy] +
cb_pred_bak[ix][iy] + 1) >> 1;
cr_pred[ix][iy] = (cr_pred_fwd[ix][iy] +
cr_pred_bak[ix][iy] + 1) >> 1;
}
```

```
}
```

The direct mode prediction can be either progressive (see subclause 7.6.9.5) or interlaced as described below. Interlaced direct mode is used when ever the co-located macroblock (macroblock with the same coordinates) of the future anchor VOP has field_predition flag is "1". Note that if the future macroblock is skipped, or intra, the direct mode prediction is progressive. Otherwise, interlaced direct mode prediction is used.

Interlaced direct coding mode is an extension of progressive direct coding mode. Four derived field motion vectors are calculated from the forward field motion vectors of the co-located future anchor VOP, a single differential motion vector and the temporal position of the B-VOP fields with respect to the fields of the past and future anchor VOPs. The four derived field motion vectors are denoted mvf[0] (top field forward) mvf[1], (bottom field forward), mvb[0] (top field backward), and mvb[1] (bottom field backward). MV[i] is the future anchor picture motion vector for the top (i == 0) and bottom (i == 1) fields. Only one delta motion vector (used for both field), MVD[0], occurs in the bitstream for the field direct mode predicted macroblock. MVD[0] is decoded assuming f_code == 1 regardless of the number in VOP header. The interlaced direct mode prediction (in luma_pred[][], cb_pred[][] and cr_pred[][]) is calculated as shown below.

```
for (i = 0; i < 2; i++) {
mvf[i].x = (TRB[i] * MV[i].x) / TRD[i] + MVD[0].x;
mvf[i].y = (TRB[i] * MV[i].y) / TRD[i] + MVD[0].y;
mvb[i].x = (MVD[i].x == 0) ?
(((TRB[i] - TRD[i]) * MV[i].x) / TRD[i]) :
mvf[i].x - MV[i].x);
mvb[i].y = (MVD[i].y == 0) ?
(((TRB[i] - TRD[i]) * MV[i].y) / TRD[i]) :
mvf[i].y - MV[i].y);
field_motion_compensate_one_reference(
luma_pred_fwd, cb_pred_fwd, cr_pred_fwd,
luma_fwd_ref_VOP, cb_fwd_ref_VOP, cr_fwd_ref_VOP,
mvf[0].x, mvf[0].y, mvf[1].x, mvf[1].y,
colocated_future_mb_top_field_reference,
colocated_future_mb_bottom_field_reference,
x, y, 0);
field_motion_compensate_one_reference(
luma_pred_bak, cb_pred_bak, cr_pred_bak,
luma_bak_ref_VOP, cb_bak_ref_VOP, cr_bak_ref_VOP,
mvb[1].x, mvb[1].y, mvb[1].x, mvb[1].y,
0, 1, x, y, 0);
for (iy = 0; iy < 16; iy++) {
for (ix = 0; ix < 16; ix++) {
luma_pred[ix][iy] = (luma_pred_fwd[ix][iy] +
luma_pred_bak[ix][iy] + 1) >> 1;
}
}
for (iy = 0; iy < 8; iy++) {
for (ix = 0; ix < 8; ix++) {
cb_pred[ix][iy] = (cb_pred_fwd[ix][iy] +
cb_pred_bak[ix][iy] + 1) >> 1;
cr_pred[ix][iy] = (cr_pred_fwd[ix][iy] +
cr_pred_bak[ix][iy] + 1) >> 1;
}
}
```

The temporal references (TRB[i] and TRD[i]) are distances in time expressed in field periods. Figure 7-29 shows how they are defined for the case where i is 0 (top field of the B-VOP). The bottom field is analogously.

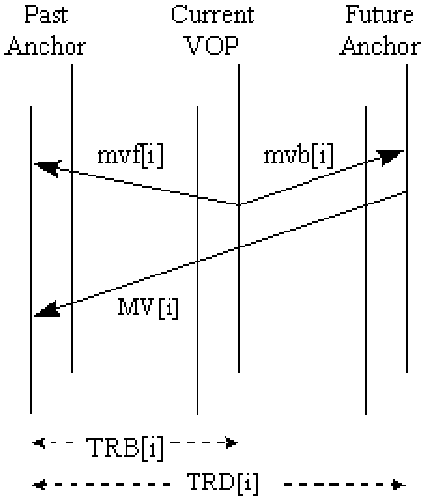


Figure -29 -- Interlaced direct mode

The calculation of TRD[i] and TRB[i] depends not only on the current field, reference field, and frame temporal references, but also on whether the current video is top field first or bottom field first.

$$\text{TRD}[i] = 2*(\text{T(future)}//\text{Tframe} - \text{T(past)}//\text{Tframe}) + d\ [i]$$

$$\text{TRB}[i] = 2*(\text{T(current)}//\text{Tframe} - \text{T(past)}//\text{Tframe}) + d\ [i]$$

where T(future), T(current) and T(past) are the cumulative VOP times calculated from modulo_time_base and vop_time_increment of the future, current and past VOPs in display order. Tframe is the frame period determined by

$$\text{Tframe} = \text{T(first\_B\_VOP)} - \text{T(past\_anchor\_of\_first B\_VOP)}$$

where first_B_VOP denotes the first B-VOP following the Video Object Layer syntax. The important thing about Tframe is that the period of time between consecutive fields which constitute an interlaced frame is assuemed to be 0.5 * Tframe for purposes of scaling the motion vectors.

The value of d is determined from Table 7-12; it is a function of the current field parity (top or bottom), the reference field of the co-located macroblock (macroblock at the same coordinates in the furture anchor VOP), and the value of top_field_first in the B-VOP?s video object plane syntax.

Table -12 -- Selection of the parameter δ

future anchor VOP reference fields of the co-located macroblock		**top_field_first == 0**		**top_field_first == 1**	
Top field reference	**Bottom field reference**	**Top field, d [0]**	**Bottom field, d [1]**	**Top field, d [0]**	**Bottom field, d [1]**
0	0	0	-1	0	1
0	1	0	0	0	0
1	0	1	-1	-1	1
1	1	1	0	-1	0

The top field prediction is based on the top field motion vector of the P-VOP macroblock of the future anchor picture. The past reference field is the reference field selected by the co-located macroblock of the future anchor picture for the top field. Analogously, the bottom field predictor is the average of pixels obtained from the future anchor?s bottom field and the past anchor field referenced by the bottom field motion vector of the corresponding macroblock of the future anchor picture. When interlaced direct mode is used, vop_time_increment_resolution must be the smallest integer greater than or equal to the number of frames per second. In each VOP, vop_time_increment counts individual frames within a second.

2. Sprite decoding

The subclause specifies the additional decoding process for a sprite video object. The sprite decoding can operate in two modes: basic sprite decoding and low-latency sprite decoding. Figure 7-30 is a diagram of the sprite decoding process. It is simplified for clarity.

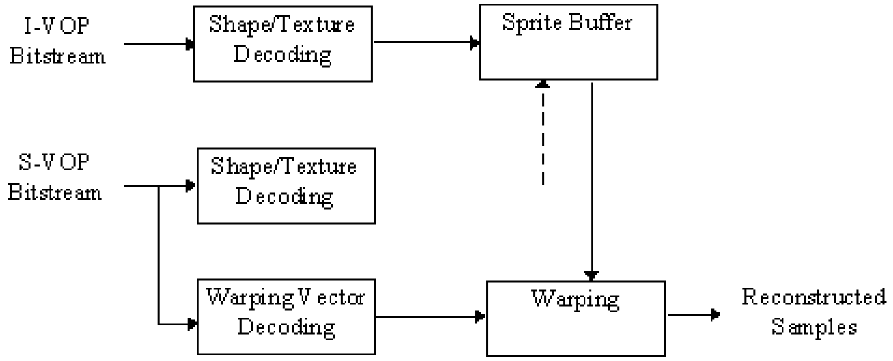


Figure -30 -- The sprite decoding process

1. Higher syntactic structures

The various parameters in the VOL and VOP bitstreams shall be interpreted as described in clause 6. When sprite_enable == ?1?, vop_coding_type shall be "I" only for the initial VOP in a VOL for basic sprites (i.e. low_latency_sprite_enable == ?0?), and all the other VOPs shall be S-VOPs (i.e.

vop_coding_type == "S"). The reconstructed I-VOP in a VOL for basic sprites is not displayed but stored in a sprite memory, and will be used by all the remaining S-VOPs in the same VOL. An S-VOP is reconstructed by applying warping to the VOP stored in the sprite memory, using the warping parameters (i.e. a set of motion vectors) embedded in the VOP bitstream. Alternatively, in a VOL for low-latency sprites (i.e. low_latency_sprite_enable == ?1?), these S-VOPs can update the information stored in the sprite memory before applying warping.

2. Sprite Reconstruction

The luminance, chrominance and grayscale alpha data of a sprite are stored in two-dimensional arrays. The width and height of the luminance array are specified by sprite_width and sprite_height respectively. The samples in the sprite luminance, chrominance and grayscale alpha arrays are addressed by two-dimensional integer pairs $(i?, j?)$ and $(i_c?, j_c?)$ as defined in the following:

- Top left luminance and grayscale alpha sample
  $(i?, j?)$ = (sprite_left_coordinate, sprite_top_coordinate)
- Bottom right luminance and grayscale alpha sample
  $(i?, j?)$ = (sprite_left_coordinate + sprite_width - 1,
  sprite_top_coordinate + sprite_height - 1)
- Top left chrominance sample
  $(i_c?, j_c?)$ = (sprite_left_coordinate / 2, sprite_top_coordinate / 2)
- Bottom right chrominance sample
  $(i_c?, j_c?)$ = (sprite_left_coordinate / 2 + sprite_width// 2 - 1,
  sprite_top_coordinate / 2 + sprite_height// 2 - 1).

Likewise, the addresses of the luminance, chrominance and grayscale alpha samples of the VOP currently being decoded are defined in the following:

- Top left sample of luminance and grayscale alpha
  (i, j) = (0, 0) for rectangular VOPs, and
  (i, j) = (vop_horizontal_mc_spatial_ref, vop_vertical_mc_spatial_ref) for non-rectangular VOPs
- Bottom right sample of luminance and grayscale alpha
  (i, j) = (video_object_layer_width - 1, video_object_layer_height - 1) for rectangular VOPs, and
  (i, j) = (vop_horizontal_mc_spatial_ref + vop_width - 1,
  vop_vertical_mc_spatial_ref + vop_height - 1) for non-rectangular VOPs
- Top left sample of chrominance
  (i_c, j_c) = (0, 0) for rectangular VOPs, and
  (i_c, j_c) = (vop_horizontal_mc_spatial_ref / 2, vop_vertical_mc_spatial_ref / 2) for non-rectangular VOPs
- Bottom right sample of chrominance
  (i_c, j_c) = (video_object_layer_width / 2 - 1, video_object_layer_height / 2 - 1) for rectangular VOPs, and
  (i_c, j_c) = (vop_horizontal_mc_spatial_ref / 2 + vop_width// 2 - 1,
  vop_vertical_mc_spatial_ref / 2 + vop_height// 2 - 1) for non-rectangular VOPs

1. Low-latency sprite reconstruction

This subclause allows a large static sprite to be reconstructed at the decoder by properly incorporating its corresponding pieces. There are two types of pieces recognized by the decoder-object and update. The decoded sprite object-piece (i.e., embedded in a S-VOP with low_latence_sprite_enable==1 and sprite_transmit_mode=="piece") is a highly quantized version of the original sprite piece while the sprite update-piece (i.e., sprite_transmit_mode=="update") is a residual designed to improve upon the quality of decoded object-piece. Sprite pieces are rectangular pieces of texture (and shape for the object-piece) and can contain "holes," corresponding to macroblocks, that do not need to be decoded. Five parameters are required by the decoder to properly incorporate the pieces: piece_quant, piece_width, piece_height, piece_xoffset, and piece_yoffset.

Macroblocks raster scanning is employed to decode each piece. However, whenever the scan encounters a macroblock which has been part of some previously sent sprite piece, then the macroblock is not decoded and its corresponding macroblock layer is empty. In that case, the decoder treats the macroblock as a hole in the current sprite piece. Since a macroblock can be refined as long as there is some available bandwidth, more than one update may be decoded per macroblock and the holes for a given refinement step have no relationship to the holes of later refinement steps. Therefore, the decoding process of a hole for an update piece is different than that for the object-piece. For the object-piece, no information is decoded at all and the decoder must "manage" where "holes" lie. (see subclause 7.8.3.1). For the update-piece, the not_coded bit is decoded to indicate whether or not one more refinement should be decoded for this given macroblock. (see subclause 7.8.3.2). Note that a hole could be non-transparent and have had shape information decoded previously. Multiple intermingled object-pieces and update-pieces may be decoded at the same current VOP. Part of a sequence could consist for example of rapidly showing a zooming out effect, a panning to the right, a zooming in, and finally a panning to the left. In this case, the first decoded object-piece covers regions on all four sides of the previous VOP transmitted piece, which is now treated as a hole and not decoded again. The second decoded object-piece relates to the right panning, and the third object-piece is a smaller left-panning piece due to the zooming-in effect. Finally, the last piece is different; instead of an object, it contains the update for some previous object-piece of zooming-in (thus, the need to update to refine for higher quality). All four pieces will be decoded within the same VOP. When sprite_transmit_mode = ="pause," the decoder recognizes that all sprite object-pieces and update-pieces for the current VOP session have been sent. However, when sprite_transmit_mode = "stop," the decoder understands that all object and update-pieces have been sent for the entire video object layer, not just for the current VOP. session. In addition, once all object-pieces or update-pieces have been decoded during a VOP session (i.e., signaled by sprite_transmit_mode == "pause" or sprite_transmit_mode == "stop"), the static sprite is padded (as defined in subclause 7.6.1), then the portion to be displayed is warped, to complete the current VOP session.

For the S-VOPs (i.e., vop_coding_type == "S"), the macroblock layer syntax of object-pieces is the same as those of I-VOP. Therefore, shape and texture are decoded using the macroblock layer structure in I-VOPs with the quantization of intra macroblocks. The syntax of the update-pieces is similar to the P-VOP inter-macroblock syntax with the quantization of non-intra macroblocks); however, the differences are indicated in Table B-1, specifically that there are no motion vectors and shape information included in this decoder syntax structure. In summary, this decoding process supports the construction of any large sprite image progressively, both spatially and in terms of quality.

1. **Decoding of holes in sprite object-piece**

   Implementation of macroblock scanning must account for the possibility that a macroblock uses prediction based on some macroblock sent in a previous piece. When an object-piece with holes is decoded, the decoder in the process of reconstruction acts as if the whole original piece were decoded, but actually only the bitstream corresponding to the "new macroblock" is received. Whenever macroblocks raster scanning encounters a hole, the decoder needs to manage the retrieval of relevant information (e.g. DCT quantization parameters, AC and DC prediction parameters, and BAB bordering values) from the corresponding macroblock decoded earlier.

2. **Decoding of holes in sprite update-pieces**

In contrast to the send_mb() used by the object-pieces, the update-pieces use the not_coded bit. When not_coded = 1 in the P-VOP syntax, the decoder recognizes that the corresponding macroblock is not refined by the current sprite update-piece. When not_coded = 0 in the P-VOP syntax, the decoder recognizes that this macroblock is refined. The prediction for the update piece is obtained by extracting the "area" of the static sprite defined by (piece_width, piece_height, piece_xoffset, piece_yoffset). This area is then padded and serves as prediction for the update pieces. Since there is no shape information included in an update-piece, the result of its transparent_mb() is retrieved from the corresponding macroblock in the object-piece decoded earlier. In addition, an update macroblock cannot be transmitted before its corresponding object macroblock. As a result, the very first sprite piece transmitted in the low-latency mode shall be an object-piece.

2. **Sprite reference point decoding**

The syntatic elements in sprite_trajectory () and below shall be interpreted as specified in clause 6. du[i] and dv[i] (0 =< i < no_sprite_point) specifies the mapping between indexes of some reference points in the VOP and the corresponding reference points in the sprite. These points are referred to as VOP reference points and sprite reference points respectively in the rest of the specification.

The index values for the VOP reference points are defined as:
(i_0, j_0) = (0, 0) when video_object_layer_shape == ?rectangle?, and
(vop_horizontal_mc_spatial_ref, vop_vetical_mc_spatial_ref) otherwise,
$(i_1, j_1) = (i_0+W, j_0)$,
$(i_2, j_2) = (i_0, j_0 + H)$,
$(i_3, j_3) = (i_0+W, j_0+H)$

where W = video_object_layer_width and H = video_object_layer_height when video_object_layer_shape == ?rectangle? or W = vop_width and H = vop_height otherwise. Only the index values with subscripts less than no_sprite_point shall be used for the rest of the decoding process.

The index values for the sprite reference points shall be calculated as follows:

$(i_0?, j_0?) = (s / 2)\ (2\ i_0 + du[0], 2\ j_0 + dv[0])$

$(i_1?, j_1?) = (s / 2)\ (2\ i_1 + du[1] + du[0], 2\ j_1 + dv[1] + dv[0])$

$(i_2?, j_2?) = (s / 2)\ (2\ i_2 + du[2] + du[0], 2\ j_2 + dv[2] + dv[0])$

$(i_3?, j_3?) = (s / 2)\ (2\ i_3 + du[3] + du[2] + du[1] + du[0], 2\ j_3 + dv[3] + dv[2] + dv[1] + dv[0])$

where $i_0?$, $j_0?$, etc are integers in $\frac{1}{s}$ pel accuracy, where s is specified by sprite_warping_accuracy. Only the index values with substcripts less than no_sprite_point need to be calculated.

When no_of_sprite_warping_points == 2 or 3, the index values for the *virtual sprite points* are additionally calculated as follows:

$(i_1??, j_1??) = (16\ (i_0 + W?) + ((W - W?)\ (r\ i_0? - 16\ i_0) + W?\ (r\ i_1? - 16\ i_1))\ //\ W,$
$16\ j_0 + ((W - W?)\ (r\ j_0? - 16\ j_0) + W?\ (r\ j_1? - 16\ j_1))\ //\ W)$
$(i_2??, j_2??) = (16\ i_0 + ((H - H?)\ (r\ i_0? - 16\ i_0) + H?\ (r\ i_2? - 16\ i_2))\ //\ H,$
$16\ (j_0 + H?) + ((H - H?)\ (r\ j_0? - 16\ j_0) + H?\ (r\ j_2? - 16\ j_2))\ //\ H)$

where $i_1??$, $j_1??$, $i_2??$, and $j_2??$ are integers in $\frac{1}{16}$ pel accuracy, and $r = 16/s$. $W?$ and $H?$ are defined as the smallest integers that satisfy the following condition:

W? = 2a , H? = 2b , W? ³ W, H? ³ H, a > 0, b > 0, both a and b are integers.

The calculation of $i_2??$, and $j_2??$ is not necessary when no_of_sprite_warping_points == 2.

3. Warping

For any pixel (i, j) inside the VOP boundary, $(F(i, j), G(i, j))$ and $(F_c(i_c, j_c), G_c(i_c, j_c))$ are computed as described in the following. These quantities are then used for sample reconstruction as specified in subclause 7.8.6. The following notations are used to simplify the description:

$$I = i - i_0,$$
$$J = j - j_0,$$
$$I_c = 4\, i_c - 2\, i_0 + 1,$$
$$J_c = 4\, j_c - 2\, j_0 + 1,$$

When no_of_sprite_warping_point == 0,

$(F(i, j), G(i, j)) = (s\, i, s\, j),$
$(F_c(i_c, j_c), G_c(i_c, j_c)) = (s\, i_c, s\, j_c).$

When no_of_sprite_warping_point == 1,

$(F(i, j), G(i, j)) = (i_0? + sI, j_0? + s\, J),$
$(F_c(i_c, j_c), G_c(i_c, j_c)) = i_0? /// 2 + s\, (i_c - i_0 / 2), j_0? /// 2 + s\, (\, j_c - j_0 / 2)).$

When no_of_sprite_warping_points == 2,

$(F(i, j), G(i, j)) = (\, i_0? + ((- r\, i_0? + i_1??)\, I + (r\, j_0? - j_1??)\, J) /// (W?\, r)\, ,$
$j_0? + ((- r\, j_0? + j_1??)\, I + (- r\, i_0? + i_1??)\, J) /// (W?\, r)),$
$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((- r\, i_0? + i_1\, ??)\, I_c + (r\, j_0? - j_1??)\, J_c + 2\, W?\, r\, i_0? - 16W?) /// (4\, W?\, r),$
$((- r\, j_0? + j_1??)\, I_c + (- r\, i_0? + i_1??)\, J_c + 2\, W?\, r\, j_0? - 16W?) /// (4\, W?\, r)).$

According to the definition of $W?$ and $H?$ (i.e. $W? = 2\mathrm{a}$ and $H? = 2\mathrm{b}$), the divisions by "///" in these functions can be replaced by binary shift operations. By this replacement, the above equations can be rewritten as:

$(F(i, j), G(i, j)) = (\, i_0? + (((- r\, i_0? + i_1??)\, I + (r\, j_0? - j_1??)\, J + 2\mathrm{a}^{+r\ -1}) >> (\mathrm{a} + r\,))\, ,$
$j_0? + (((- r\, j_0? + j_1??)\, I + (- r\, i_0? + i_1??)\, J + 2\mathrm{a}^{+r\ -1}) >> (\mathrm{a} + r\,)),$
$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((- r\, i_0? + i_1\, ??)\, I_c + (r\, j_0? - j_1??)\, J_c + 2\, W?\, r\, i_0? - 16W? + 2\mathrm{a}^{+r\ +1}) >> (\mathrm{a} + r + 2),$
$((- r\, j_0? + j_1??)\, I_c + (- r\, i_0? + i_1??)\, J_c + 2\, W?\, r\, j_0? - 16W? + 2\mathrm{a}^{+r\ +1}) >> (\mathrm{a} + r + 2)),$
where $2\mathrm{r} = r$.

When no_of_sprite_warping_points == 3,

$(F(i, j), G(i, j)) = (i_0? + ((- r\, i_0? + i_1??)\, H?\, I + (- r\, i_0?+ i_2??)W?\, J) /// (W?H?r),$
$j_0? + ((- r\, j_0? + j_1??)\, H?\, I + (- r\, j_0?+ j_2??)W?\, J) /// (W?H?r)),$
$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((- r\, i_0? + i_1??)\, H?\, I_c + (- r\, i_0?+ i_2??)W?\, J_c + 2\, W?H?r\, i_0? - 16W?H?) ///$
$(4W?H?r),$
$((- r\, j_0? + j_1??)\, H?\, I_c + (- r\, j_0?+ j_2??)W?\, J_c + 2\, W?H?r\, j_0? - 16W?H?) /// (4W?H?r)).$

According to the definition of $W?$ and $H?$, the computation of these functions can be simplified by dividing the denominator and numerator of division beforehand by $W?$ (when $W? < H?$) or $H?$ (when $W?$ ³ $H?$). As in the case of no_of_sprite_warping_points == 2, the divisions by "///" in these functions can be replaced by binary shift operations. For example, when $W?$ ³ $H?$ (i.e. a ³ b) the above equations can be rewritten as:

$(F(i, j), G(i, j)) = (i_0? + (((- r\, i_0? + i_1??)\, I + (- r\, i_0?+ i_2??)\, 2\mathrm{a}^{-b}\, J + 2\mathrm{a}^{+r\ -1}) >> (\mathrm{a} + r\,)),$

$j_0? + (((- r\ j_0? + j_1??)\ I + (- r\ j_0?+ j_2??)\ 2a^{-b}\ J + 2a^{+r -1}) >> (a +r))),$
$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((- r\ i_0? + i_1??)\ I_c + (- r\ i_0?+ i_2??)\ 2a^{-b}\ J_c + 2W?r\ i_0? - 16W? + 2a^{+r +1}) >> (a +r +2),$
$((- r\ j_0? + j_1??)\ I_c + (- r\ j_0?+ j_2??)\ 2a^{-b}\ J_c + 2W?r\ j_0? - 16W? + 2a^{+r +1}) >> (a +r +2)).$

When no_of_sprite_warping_point == 4,

$(F(i, j), G(i, j)) = ((a\ i + b\ j + c) /// (g\ i + h\ j + D\ W\ H),$
$(d\ i + e\ j + f) /// (g\ i + h\ j + D\ W\ H)),$
$(F_c(i_c, j_c), G_c(i_c, j_c)) = ((2\ a\ I_c + 2\ b\ J_c + 4\ c - (g\ I_c + h\ J_c + 2\ D\ W\ H)\ s) /// (4gI_c +4\ hJ_c +8D\ W\ H),$
$(2\ d\ I_c + 2\ e\ J_c + 4\ f - (g\ I_c + h\ J_c + 2\ D\ W\ H)\ s) /// (4\ g\ I_c +4\ hJ_c +8D\ W\ H))$

where

$g = ((i_0? - i_1? - i_2? + i_3?)\ (j_2? - j_3?) - (i_2? - i_3?)\ (j_0? - j_1? - j_2? + j_3?))\ H\ ,$
$h = ((i_1? - i_3?)\ (j_0? - j_1? - j_2? + j_3?) - (i_0? - i_1? - i_2? + i_3?)\ (j_1? - j_3?))\ W\ ,$
$D = (i_1? - i_3?)\ (j_2? - j_3?) - (i_2? - i_3?)\ (j_1? - j_3?),$
$a = D\ (i_1? - i_0?)\ H + g\ i_1?\ ,$
$b = D\ (i_2? - i_0?)\ W + h\ I_2?,$
$c = D\ i_0?\ W\ H,$
$d = D\ (j_1? - j_0?)\ H + g\ j_1?,$
$e = D\ (j_2? - j_0?)\ W + h\ j_2?,$
$f = D\ j_0?\ W\ H.$

A set of parameters that causes the denominator of any of the the above equations to be zero for any pixel in a opaque or boundary macroblock is disallowed. The implementor should be aware that a 32bit register may not be sufficient for representing the denominator or the numerator in the above transform functions for affine and perspective transform. The usage of a 64 bit floating point representation should be sufficient in such case.

4. **Sample reconstruction**

The reconstructed value Y of the luminance sample (i, j) in the currently decoded VOP shall be defined as

$$Y = ((s - r_j)((s -r_i)\ Y_{00} + r_i\ Y_{01}) + r_j\ ((s - r_i)\ Y_{10} + r_i\ Y_{11})) // s^2,$$

where Y_{00}, Y_{01}, Y_{10}, Y_{11} represent the sprite luminance sample at ($F(i, j)$////s, $G(i, j)$////s), ($F(i, j)$////s + 1,$G(i, j)$////s), ($F(i, j)$////s, $G(i, j)$////s + 1), and ($F(i, j)$////s + 1,$G(i, j)$////s + 1) respectively, and $r_i = F(i, j) -(F(i, j)////s)s$ and $r_j = G(i, j) - (G(i, j)////s)s$. Figure 7-31 illustrates this process.

In case any of Y_{00}, Y_{01}, Y_{10} and Y_{11} lies outside the sprite luminance binary mask, it shall be obtained by the padding process as defined in subclause 7.6.1.

When brightness_change_in_sprite == 1, the final reconstructed luminance sample (i, j) is further computed as $Y = Y *$ (brightness_change_factor * 0.01 + 1), clipped to the range of [0, 255].

Similarly, the reconstructed value C of the chrominance sample (i_c, j_c) in the currently decoded VOP shall be define as

$$C = ((s - r_j)((s -r_i)\ C_{00} + r_i\ C_{01}) + r_j\ ((s - r_i)\ C_{10} + r_i\ C_{11})) // s^2,$$

where C_{00}, C_{01}, C_{10}, C_{11} represent the sprite chrominance sample at ($F_c(i_c, j_c)////s$, $G_c(i_c, j_c)////s$), ($F_c(i_c, j_c)////s + 1$, $G_c(i_c, j_c)////s$), ($F_c(i_c, j_c)////s$, $G_c(i_c, j_c)////s + 1$), and ($F_c(i_c, j_c)////s + 1$, $G_c(i_c, j_c)////s + 1$) respectively, and $r_i = F_c(i_c, j_c) - (F_c(i_c, j_c))////s)s$ and $r_j = G_c(i_c, j_c) - (G_c(i_c, j_c)////s)s$. In case any of C_{00}, C_{01}, C_{10} and C_{11} lies outside the sprite chrominance binary mask, it shall be obtained by the padding process as defined in subclause 7.6.1.

The same method is used for the reconstruction of grayscale alpha and luminance samples. The reconstructed value A of the grayscale alpha sample (i, j) in the currently decoded VOP shall be defined as

$$A = ((s - r_j)((s - r_i)\, A_{00} + r_i\, A_{01}) + r_j\, ((s - r_i)\, A_{10} + r_i\, A_{11}))\ //\ s^2,$$

where A_{00}, A_{01}, A_{10}, A_{11} represent the sprite grayscale alpha sample at ($F(i, j)////s$, $G(i, j)////s$), ($F(i, j)////s + 1$,$G(i, j)////s$), ($F(i, j)////s$, $G(i, j)////s + 1$), and ($F(i, j)////s + 1$,$G(i, j)////s + 1$) respectively, and $r_i = F(i, j) -(F(i, j)////s)s$ and $r_j = G(i, j) - (G(i, j)////s)s$. In case any of A_{00}, A_{01}, A_{10} and A_{11} lies outside the sprite luminance binary mask, it shall be obtained by the padding process as defined in subclause 7.6.1.

The reconstructed value of luminance binary mask sample $BY(i,j)$ shall be computed following the identical process for the luminance sample. However, corresponding binary mask sample values shall be used in place of luminance samples Y_{00}, Y_{01}, Y_{10}, Y_{11}. Assume the binary mask sample opaque is equal to 255 and the binary mask sample transparent is equal to 0. If the computed value is bigger or equal to 128, $BY(i, j)$ is defined as opaque. Otherwise, $BY(i, j)$ is defined as transparent. The chrominance binary mask samples shall be reconstructed by decimating of the corresponding 2 x 2 adjacent luminance binary mask samples as specified in subclause 7.6.1.4.

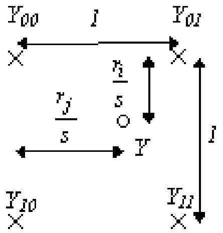


Figure -31 -- Pixel value interpolation (it is assumed that sprite samples are located on an integer grid)

1. Generalized scalable decoding

This subclause specifies the additional decoding process required for decoding scalable coded video.

The scalability framework is referred to as generalized scalability which includes the spatial and the temporal scalabilities. The temporal scalability offers scalability of the temporal resolution, and the spatial scalability offers scalability of the spatial resolution. Each type of scalability involves more than one layer. In the case of two layers, consisting of a lower layer and a higher layer; the lower layer is referred to as the base layer and the higher layer is called the enhancement layer.

In the case of temporal scalability, both rectangular VOPs as well as arbitrary shaped VOPs are supported. In the case of spatial scalability, only rectangular VOPs are supported. Figure 7-32 shows a high level decoder structure for generalized scalability.

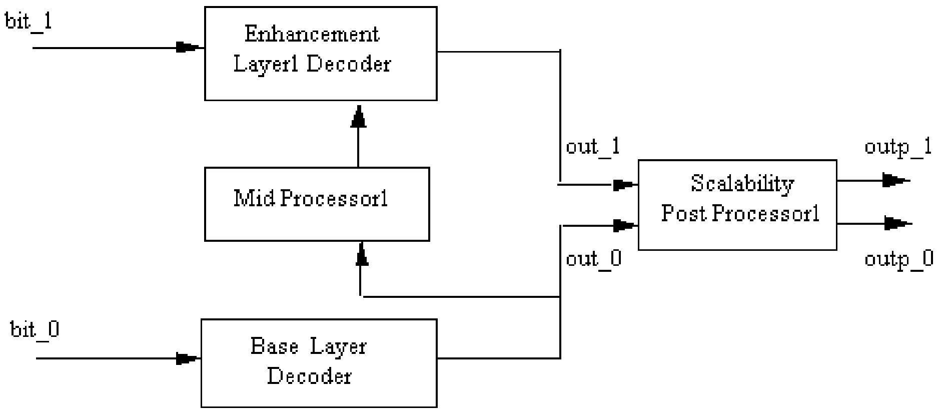


Figure -32 -- High level decoder structure for generalized scalability

The base layer and enhancement layer bitstreams are input for decoding by the corresponding base layer decoder and enhancement layer decoder.

When spatial scalability is to be performed, mid processor 1 performs spatial up or down sampling of input. The scalability post processor performs any necessary operations such as spatial up or down sampling of the decoded base layer for display resulting at outp_0 while the enhancement layer without resolution conversion may be output as outp_1.

When temporal scalability is to be performed, the decoding of base and enhancement layer bitstreams occurs in the corresponding base and enhancement layer decoders as shown. In this case, mid processor 1 does not perform any spatial resolution conversion. The post processor simply outputs the base layer VOPs without any conversion, but temporally multiplexes the base and enhancement layer VOPs to produce higher temporal resolution enhancement layer.

The reference VOPs for prediction are selected by ref_select_code as specified in Table 7-13 and Table 7-14. In coding of P-VOPs belonging to an enhancement layer, the forward reference is one of the following four: the most recently decoded VOP of enhancement layer, the most recent VOP of the reference layer in display order, the next VOP of the reference layer in display order, or the temporally coincident VOP in the reference layer.

In B-VOPs, the forward reference is one of the following two: the most recently decoded enhancement VOP or the most recent reference layer VOP in display order. The backward reference is one of the following three: the temporally coincident VOP in the reference layer, the most recent reference layer VOP in display order, or the next reference layer VOP in display order.

Table -13 -- Prediction reference choices in enhancement layer P-VOPs for scalability

ref_select_code	forward prediction reference
00	Most recently decoded enhancement VOP belonging to the same layer.
01	Most recently VOP in display order belonging to the reference layer.
10	Next VOP in display order belonging to the reference layer.
11	Temporally coincident VOP in the reference layer (no motion vectors)

Table -14 -- Prediction reference choices in enhancement layer B-VOPs for scalability

ref_select_code	forward temporal reference	backward temporal reference
00	Most recently decoded enhancement VOP of the same layer	Temporally coincident VOP in the reference layer (no motion vectors)
01	Most recently decoded enhancement VOP of the same layer.	Most recent VOP in display order belonging to the reference layer.
10	Most recently decoded enhancement VOP of the same layer.	Next VOP in display order belonging to the reference layer.
11	Most recently VOP in display order belonging to the reference layer.	Next VOP in display order belonging to the reference layer.

1. **Temporal scalability**

   Temporal scalability involves two layers, a lower layer and an enhancement layer. Both the lower and the enhancement layers process the same spatial resolution. The enhancement layer enhances the temporal resolution of the lower layer and if temporally remultiplexed with the lower layer provides full temporal rate.

   1. **Base layer and enhancement layer**

      In the case of temporal scalability, the decoded VOPs of the enhancement layer are used to increase the frame rate of the base layer. Figure 7-33 shows a simplified diagram of the motion compensation process for the enhancement layer using temporal scalability.

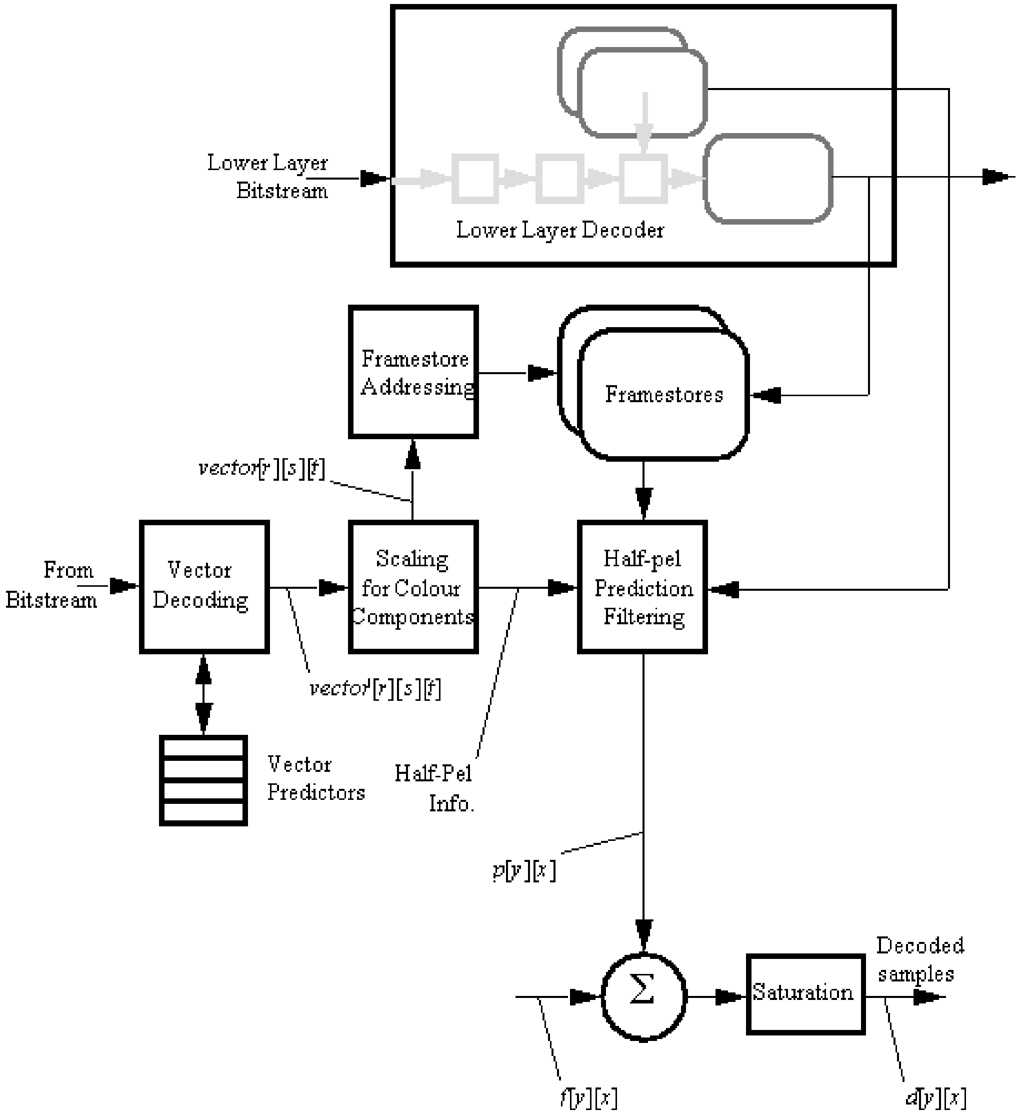


Figure -33 -- Simplified motion compensation process for temporal scalability

Predicted samples p[y][x] are formed either from frame stores of base layer or from frame stores of enhancement layer. The difference data samples f[y][x] are added to p[y][x] to form the decoded samples d[y][x].

There are two types of enhancement structures indicated by the "enhancement_type" flag. When the value of enhancement_type is "1", the enhancement layer increases the temporal resolution of a partial region of the base layer. When the value of enhancement_type is "0", the enhancement layer increases the temporal resolution of an entire region of the base layer.

2. **Base layer**

The decoding process of the base layer is the same as non-scalable decoding process.

3. **Enhancement layer**

The VOP of the enhancement layer is decoded as either I-VOP, P-VOP or B-VOP. The shape of the VOP is either rectangular (video_object_layer_id is "00") or arbitrary (video_object_layer_id is "01").

1. **Decoding of I-VOPs**

   The decoding process of I-VOPs in enhancement layer is the same as non-scalable decoding process.

2. **Decoding of P-VOPs**

   The reference layer is indicated by ref_layer_id in Video Object Layer class. Other decoding process is the same as non-scalable P-VOPs except the process specified in subclauses 7.9.1.3.4 and 7.9.1.3.5.

   For P-VOPs, the ref_select_code is either "00", "01" or "10".

   When the value of ref_select_code is "00", the prediction reference is set by the most recently decoded VOP belonging to the same layer.

   When the value of ref_select_code is "01", the prediction reference is set by the previous VOP in display order belonging to the reference layer.

   When the value of ref_select_code is "10", the prediction reference is set by the next VOP in display order belonging to the reference layer.

3. **Decoding of B-VOPs**

   The reference layer is indicated by ref_layer_id in Video Object Layer class. Other decoding process is the same as non-scalable B-VOPs except the process specified in subclauses 7.9.1.3.4 and 7.9.1.3.5.

   For B-VOPs, the ref_select_code is either "01", "10" or "11".

   When the value of ref_select_code is "01", the forward prediction reference is set by the most recently decoded VOP belonging to the same layer and the backward prediction reference is set by the previous VOP in display order belonging to the reference layer.

   When the value of ref_select_code is "10", the forward prediction reference is set by the most recently decoded VOP belonging to the same layer, and the backward prediction reference is set by the next VOP in display order belonging to the reference layer.

   When the value of ref_select_code is "11", the forward prediction reference is set by the previous VOP in display order belonging to the reference layer and the backward prediction reference is set by the next VOP in display order belonging to the reference layer. The picture type of the reference VOP shall be either I or P (vop_coding_type = "00" or "01").

   When the value of ref_select_code is "01" or "10", direct mode is not allowed. modb shall always exist in each macroblock, i.e. the macroblock is not skipped even if the co-located macroblock is skipped.

4. **Decoding of arbitrary shaped VOPs**

   Prediction for arbitrary shape in P-VOPs or in B-VOPs is same as the one in the base layer (see subclause 7.5.2.1.2).

   For arbitrary shaped VOPs with the value of enhancement_type being "1", the shape of the reference VOP is defined as an all opaque rectangle whose size is the same as the reference layer when the shape of reference layer is rectangular (video_object_layer_shape

= "00").

When the value of ref_select_code is "11" and the value of enhancement_type is "1", modb shall always exist in each macroblock, i.e. the macroblock is not skipped even if the co-located macroblock is skipped.

5. **Decoding of backward and forward shape**

Backward shape and forward shape are used in the background composition process specified in subclause 8.1. The backward shape is the shape of the enhanced object at the next VOP in display order belonging to the reference layer. The forward shape is the shape of the enhanced object at the previous VOP in display order belonging to the reference layer.

For the VOPs with the value of enhancement_type being "1", backward shape is decoded when the load_backward_shape is "1" and forward shape is decoded when load_forward_shape is "1".

When the value of load_backward_shape is "1" and the value of load_forward_shape is "0", the backward shape of the previous VOP is copied to the forward shape for the current VOP. When the value of load_backward_shape is "0", the backward shape of the previous VOP is copied to the backward shape for the current VOP and the forward shape of the previous VOP is copied to the forward shape for the current VOP.

The decoding process of backward and forward shape is the same as the decoding process for the shape of I-VOP with binary only mode (video_object_layer_shape = "10").

2. **Spatial scalability**
   1. **Base Layer and Enhancement Layer**

In the case of spatial scalability, the enhancement bitstream is used to increase the resolution of the image. When the output with lower resolution is required, only the base layer is decoded. When the output with higher resolution is required, both the base layer and the enhancement layer are decoded.

Figure 7-34 is a diagram of the video decoding process with spatial scalability.

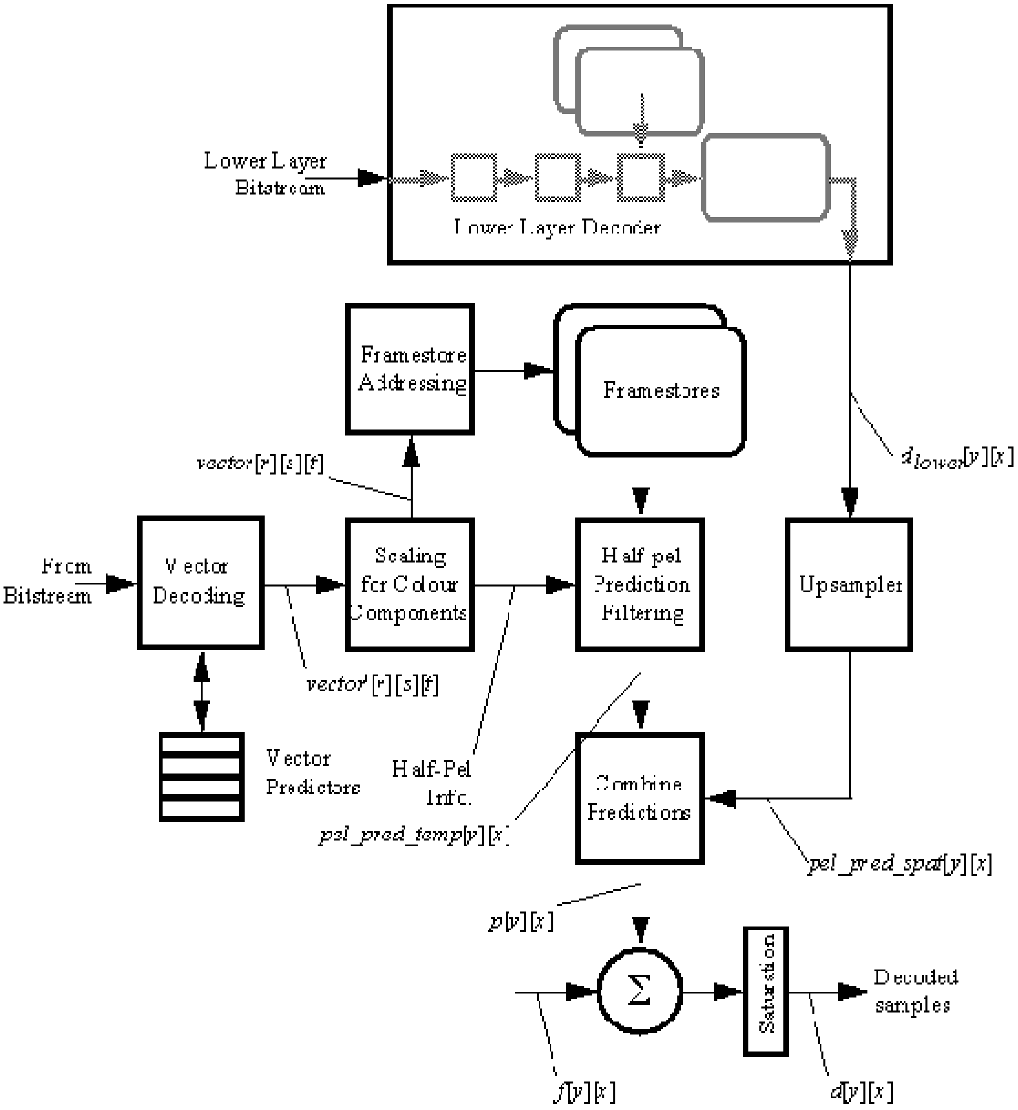


Figure -34 -- Simplified motion compensation process for spatial scalability

2. **Decoding of Base Layer**

The decoding process of the base layer is the same as nonscalable decoding process.

3. **Prediction in the enhancement layer**

A motion compensated temporal prediction is made from reference VOPs in the enhancement layer. In addition, a spatial prediction is formed from the lower layer decoded frame (dlower[y][x]). These predictions are selected individually or combined to form the actual prediction.

In the enhancement layer, the forward prediction in P-VOP and the backward prediction in B-VOP are used as the spatial prediction. The reference VOP is set to the temporally coincident VOP in the base layer. The forward prediction in B-VOP is used as the temporal prediction from the enhancement layer VOP. The reference VOP is set to the most recently decoded VOP of the enhancement layer. The interpolate prediction in B-VOP is the combination of these predictions.

In the case that a macroblock is not coded, either because the entire macroblock is skipped or the specific macroblock is not coded there is no coefficient data. In this case f[y][x] is zero, and the decoded samples are simply the prediction, p[y][x].

4. **Formation of spatial prediction**

Forming the spatial prediction requires definition of the spatial resampling process. The formation is performed at the mid-processor. The resampling process is defined for a whole VOP, however, for decoding of a macroblock, only the 16x16 region in the upsampled VOP, which corresponds to the position of this macroblock, is needed.

The spatial prediction is made by resampling the lower layer reconstructed VOP to the same sampling grid as the enhancement layer. In the first step, the lower layer VOP is subject to vertical resampling. Then, the vertically resampled image is subject to horizontal resampling.

5. **Vertical resampling**

The image subject to vertical resampling, $d_{lower}[y][x]$, is resampled to the enhancement layer vertical sampling grid using linear interpolation between the sample sites according to the following formula, where vert_pic is the resulting image:

vert_pic[yh][x] = (16 - phase) * d_{lower} [y1][x] + phase * d_{lower} [y2][x]

where

yh = output sample coordinate in vert_pic

y1 = (yh * vertical_sampling_factor_m) / vertical_sampling_factor_n

y2 = y1 + 1 if y1 < video_object_layer_height - 1

y1 otherwise

phase = (16 * ((yh * vertical_sampling_factor_m) %

vertical_sampling_factor_n)) // vertical_sampling_factor_n

where video_object_layer_height is the height of the reference VOL.

Samples which lie outside the vertically upsampled reconstructed frame which are required for upsampling are obtained by border extension of the vertically upsampled reconstructed frame.

NOTE The calculation of phase assumes that the sample position in the enhancement layer at yh = 0 is spatially coincident with the first sample position of the lower layer. It is recognised that this is an approximation for the chrominance component if the chroma_format == 4:2:0.

6. **Horizontal resampling**

The image subject to horizontal resampling, $vert\_pict[y][x]$, is resampled to the enhancement layer horizontal sampling grid using linear interpolation between the sample sites according to the following formula, where hor_pic is the resulting image:

hor_pic[y][xh] = ((16 - phase) * vert_pic[y][x1] + phase * vert_pic[y][x2]) // 256

where

xh = output sample coordinate in hor_pic

x1 = (xh * horizontal_sampling_factor_m) / horizontal_sampling_factor_n

x2 = x1 + 1 if x1 < video_object_layer_width - 1

x1 otherwise

phase = (16 * ((xh * horizontal_sampling_factor_m) % horizontal_sampling_factor_n)) // horizontal_sampling_factor_n

where video_object_layer_width is the width of the reference VOL.

Samples which lie outside the lower layer reconstructed frame which are required for upsampling are obtained by border extension of the lower layer reconstructed frame.

7. **Selection and combination of spatial and temporal predictions**

The spatial and temporal predictions can be selected or combined to form the actual prediction in B-VOP. The spatial prediction is referred to as "backward prediction", while the temporal prediction is referred to as "forward prediction". The combination of these predictions can be used as "interpolate prediction". In the case of P-VOP, only the spatial prediction (prediction from the reference layer) can be used as the forward prediction. The prediction in the enhancement layer is defined in the following formulae.

pel_pred[y][x] = pel_pred_temp[y][x] (forward in B-VOP)

pel_pred[y][x] = pel_pred_spat[y][x] = hor_pict[y][x] (forward in P-VOP and backward in B-VOP)

pel_pred[y][x] = (pel_pred_temp[y][x] + pel_pred_spat[y][x])//2 (Interpolate in B-VOP)

pel_pred_temp[y][x] is used to denote the temporal prediction (formed within the enhancement layer). pel_pred_spat[y][x] is used to denote the prediction formed from the lower layer. pel_pred[y][x] is denoted the resulting prediction.

8. **Decoding process of enhancement layer**

The VOP in the enhancement layer is decoded as either I-VOP, P-VOP or

B-VOP.

9. Decoding of I-VOPs

   The decoding process of the I-VOP in the enhancement layer is the same as the non_scalable decoding process.

10. Decoding of P-VOPs

   In P-VOP, the ref_select_code shall be "11", i.e., the prediction reference is set to the temporally coincident VOP in the base layer. The reference layer is indicated by ref_layer_id in VideoObjectLayer class. In the case of spatial prediction, the motion vector shall be set to 0 at the decoding process and is not encoded in the bitstream.

   A variable length codeword giving information about the macroblock type and the coded block pattern for chrominance is mcbpc. The codewords for mcbpc in the enhancement layer are the same as the base layer and shown in Table B-7. mcbpc shall be included in coded macroblocks.

   The macroblock type gives information about the macroblock and which data elements are present. Macroblock types and included elements in the enhancement layer bitstream are listed in subclause B.1.1.

   In the case of the enhancement layer of spatial scalability, INTER4V shall not be used. The macroblock of INTER or INTER+Q is encoded using the spatial prediction.

11. Decoding of B-VOPs

In B-VOP, the ref_select_code shall be "00", i.e., the backward prediction reference is set to the temporally coincident VOP in the base layer, and the forward prediction reference is set to the most recently decoded VOP in the enhancement layer. In the case of spatial prediction, the motion vector shall be set to 0 at the decoding process and is not encoded in the bitstream.

modb shall be present in coded macroblocks belonging to B-VOPs. The codeword is the same as the base layer and is shown in Table B-3. In case mb_type does not exist the default shall be set to "Forward MC" (prediction from the last decoded VOP in the same reference layer). modb shall be encoded in all macroblocks. If its value is equal to ?1?, further information is not transmitted for this macroblock. The decoder treats the prediction of this macroblock as forward MC with motion vector equal to zero.

mb_type is present only in coded macroblocks belonging to B-VOPs. The mb_type gives information about the macroblock and which data elements are present. mb_type and included elements in the enhancement layer bitstream are listed in Table B-5.

In the case of the enhancement layer of spatial scalability, direct mode shall not be used. The decoding process of the forward motion vectors are the same as the base layer.

2. Still texture object decoding

The block diagram of the decoder is shown in Figure 7-35.

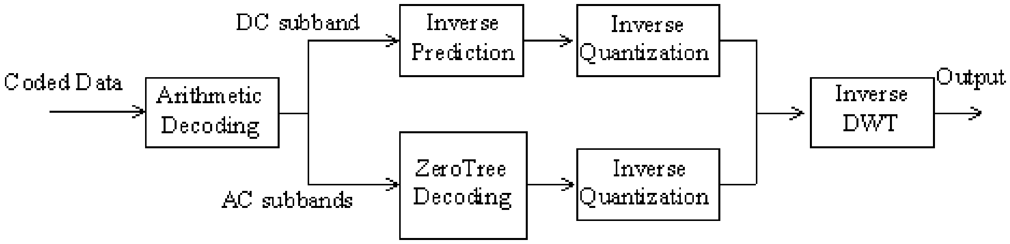


Figure -35 -- Block diagram of the decoder

The basic modules of a zero-tree wavelet based decoding scheme are as follows:

1. Arithmetic decoding of the DC subband using a predictive scheme.
1. Arithmetic decoding of the bitstream into quantized wavelet coefficients and the significance map for AC subbands.
1. Zero-tree decoding of the higher subband wavelet coefficients.
1. Inverse quantization of the wavelet coefficients.
1. Composition of the texture using inverse discrete wavelet transform (IDWT).

1. Decoding of the DC subband

The wavelet coefficients of DC band are decoded independently from the other bands. First the quantization step size decoded, then the magnitude of the minimum value of the differential quantization indices "band_offset" and the maximum value of the differential quantization indices "band_max_value" are decoded from bitstream. The parameter "band_offset" is negative or zero integer and the parameter "band_max" is a positive integer, so only the magnitude of these parameters are read from the bitstream.

The arithmetic model is initialized with a uniform distribution of band_max_value-band_offset+1. Then, the differential quantization indices are decoded using the arithmetic decoder in a raster scan order, starting from the upper left index and ending with the lowest right one. The model is updated with the decoding of each bits of the predicted wavelet quantization index to adopt the probability model to the statistics of DC band.

The "band_offset" is added to all the decoded quantization indices, and an inverse predictive scheme is applied. Each of the current indices w_X is predicted

from three quantization indices in its neighborhood, i.e. w_A, w_B, and w_C (see Figure 7-36), and the predicted value is added to the current decoded coefficient. That is,

if $(|w_A-w_B|) < | w_B-w_C|)$

$\hat{w_X} = w_C$

else

$\hat{w_X} = w_A$

$w_x = w_x + \hat{w_X}$

If any of nodes A, B or C is not in the image, its value is set to zero for the purpose of the inverse prediction. Finally, the inverse quantization scheme is applied to all decoded values to obtain the wavelet coefficients of DC band.

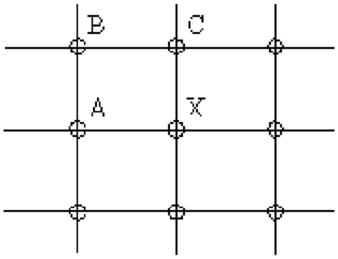


Figure -36 -- DPCM decoding of DC band coefficients

2. **ZeroTree Decoding of the Higher Bands**

The zero-tree algorithm is based on the strong correlation between the amplitudes of the wavelet coefficients across scales, and on the idea of partial ordering of the coefficients. The coefficient at the coarse scale is called the *parent*, and all coefficients at the same spatial location, and of similar orientation, at the next finer scale are that parent?s children. Figure 7-37 shows a wavelet tree where the parents and the children are indicated by squares and connected by lines. Since the DC subband (shown at the upper left in Figure 7-37) is coded separately using a DPCM scheme, the wavelet trees start from the adjacent higher bands.

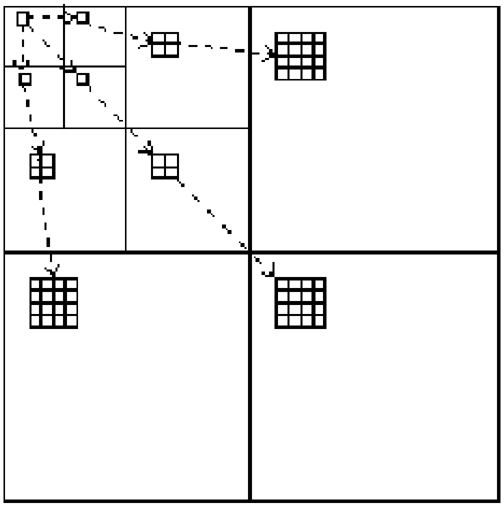


Figure -37 -- Parent-child relationship of wavelet coefficients

In transform-based coding, it is typically true that a large percentage of the transform coefficients are quantized to zero. A substantial number of bits must be spent either encoding these zero-valued quantized coefficients, or else encoding the location of the non-zero-valued quantized coefficients. ZeroTree Coding uses a data structure called a *zerotree*, built on the parent-child relationships described above, and used for encoding the location of non-zero quantized wavelet coefficients. The zerotree structure takes advantage of the principle that if a wavelet coefficient at a coarse scale is "insignificant" (quantized to zero), then all wavelet coefficients of the same orientation at the same spatial location at finer wavelet scales are also likely to be "insignificant". Zerotrees exist at any tree node where the coefficient is zero and all its descendents are also zero.

The wavelet trees are efficiently represented and coded by scanning each tree from the root in the 3 lowest AC bands through the children, and assigning one of four symbols to each node encountered: *zerotree root (ZTR)*, *value zerotree root (VZTR)*, *isolated zero (IZ)* or *value (VAL)*. A *ZTR* denotes a coefficient that is the root of a zerotree. Zerotrees do not need to be scanned further because it is known that all coefficients in such a tree have amplitude zero. A *VZTR* is a node where the coefficient has a nonzero amplitude, and all four children are zerotree roots. The scan of this tree can stop at this symbol. An *IZ* identifies a coefficient with amplitude zero, but also with some nonzero descendant. A *VAL* symbol identifies a coefficient with amplitude nonzero, and with some nonzero descendant. The symbols and quantized coefficients are losslessly encoded using an adaptive arithmetic coder. Table 7-15 shows the mapping of indices of the arithmetic decoding model into the zerotree symbols:

Table -15 -- The indexing of zerotree symbols

index	**Symbol**
0	IZ
1	VAL
2	ZTR
3	VZTR

In order to achieve a wide range of scalability levels efficiently as needed by different applications, three different zerotree scanning and associated inverse quantization methods are employed. The encoding mode is speficied in bitstream with quantization_type field as one of 1) single_quant, 2) multi_quant or

3) bilevel_quant:

Table -16 -- The quantization types

code	quantization_type
01	single_quant
10	multi _quant
11	bilevel_quant

In single_quant mode, the bitstream contains only one zero-tree map for the wavelet coefficients. After arithmetic decoding, the inverse quantization is applied to obtain the reconstructed wavelet coefficients and at the end, the inverse wavelet transform is applied to those coefficients.

In multi_quant mode, a multiscale zerotree decoding scheme is employed. Figure 7-38 shows the concept of this technique.

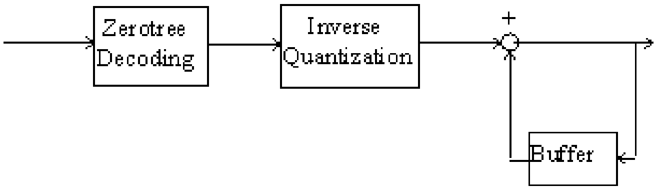


Figure -38 -- Multiscale zerotree decoding

The wavelet coefficients of the first spatial (and/or SNR) layer are read from the bitstream and decoded using the arithmetic decoder. Zerotree scanning is used for decoding the significant maps and quantized coefficients and locating them in their corresponding positions in trees.. These values are saved in the buffer to be used for quantization refinement at the next scalability layer. Then, an inverse quantization is applied to these indices to obtain the quantized wavelet coefficients. An inverse wavelet transform can also be applied to these coefficients to obtain the first decoded image. The above procedure is applied for the next spatial/SNR layers.

The bilevel_quant mode enables fine granular SNR scalability by encoding the wavelet coefficients in a bitplane by bitplane fashion. This mode uses the same zerotree symbols as the multi_quant mode. In this mode, a zero-tree map is decoded for each bitplane, indicating which wavelet coefficients are nonzero relative to that bitplane. The inverse quantization is also performed bitplane by bitplane. After the zero-tree map, additional bits are decoded to refine the accuracy of the previously decoded coefficients.

1. **Zerotree Scanning**

   In all the three quantization modes, the wavelet coefficients are scanned either in the tree-depth fashion or in the band-by-band fashion. In the tree-depth scanning order all coefficients of each tree are decoded before starting decoding of the next tree. In the band-by-band scanning order, all coefficients are decoded from the lowest to the highest frequency subbands.

   Figure 7-39 shows the scanning order for a 16x16 image, with 3 levels of decomposition. In this figure, the indices 0,1,2,3 represent the DC band coefficients which are decoded separately. The remaining coefficients are decoded in the order shown in this figure. As an example, indices 4,5,..., 24 represent one tree. At first, coefficients in this tree are decoded starting from index 4 and ending at index 24. Then, the coefficients in the second tree are decoded starting from index 25 and ending at 45. The third tree is decoded starting from index 46 and ending at index 66 and so on.

0	1	4	67	5	6	68	69	9	10	13	14	72
2	3	130	193	7	8	70	71	11	12	15	16	74
25	88	46	109	131	132	194	195	17	18	21	22	80
151	214	172	235	133	134	196	197	19	20	23	24	82
26	27	89	90	47	48	110	111	135	136	139	140	198
28	29	91	92	49	50	112	113	137	138	141	142	200
152	153	215	216	173	174	236	237	143	144	147	148	206
154	155	217	218	175	176	238	239	145	146	149	150	208
30	31	34	35	93	94	97	98	51	52	55	56	114
32	33	36	37	95	96	99	100	53	54	57	58	116
38	39	42	43	101	102	105	106	59	60	63	64	122
40	41	44	45	103	104	107	108	61	62	65	66	124
156	157	160	161	219	220	223	224	177	178	181	182	240
158	159	162	163	221	222	225	226	179	180	183	184	242
164	165	168	169	227	228	231	232	185	186	189	190	248
166	167	170	171	229	230	233	234	187	188	191	192	250

Figure -39 -- Tree depth scanning order of a wavelet block in the all three modes

Figure 7-40 shows that the wavelet coefficients are scanned in the subband by subband fashion, from the lowest to the highest frequency subbands. shows an example of decoding order for a 16x16 image with 3 levels of decomposition for the subband by subband scanning.The DC band is located at upper left corner (with indices 0, 1,2, 3) and is decoded separately as described in DC band decoding. The remaining coefficients are decoded in the order which is shown in the figure, starting from index 4 and ending at index 255. In multi_quant mode, at first scalability layer, the zerotree symbols and the corresponding values are decoded for the wavelet coefficients of that scalability layer. For the next scalability layers, the zerotree map is updated along with the corresponding value refinements. In each scalability layer, a new zerotree symbol is decoded for a coefficient only if it was decoded as ZTR or IZ in previous scalability layer or it is currently in SKIP mode. A node is said to be in SKIP mode when the number of quantization refinement levels for the current scalability layer is one. The detailed description of the refinement of quantization level is found in subclause 7.10.3. If the coefficient was decoded as VAL in previous layer and it is not currently in SKIP mode, a VAL symbol is also assigned to it at the current layer and only its refinement value is decoded from bitstream.

0	1	4	7	16	17	28	29	64	65	68	69	112
2	3	10	13	18	19	30	31	66	67	70	71	114
5	8	6	9	40	41	52	53	72	73	76	77	120
11	14	12	15	42	43	54	55	74	75	78	79	122
20	21	32	33	24	25	36	37	160	161	164	165	208
22	23	34	35	26	27	38	39	162	163	166	167	210
44	45	56	57	48	49	60	61	168	169	172	173	216
46	47	58	59	50	51	62	63	170	171	174	175	218
80	81	84	85	128	129	132	133	96	97	100	101	144
82	83	86	87	130	131	134	135	98	99	102	103	146
88	89	92	93	136	137	140	141	104	105	108	109	152
90	91	94	95	138	139	142	143	106	107	110	111	154
176	177	180	181	224	225	228	229	192	193	196	197	240
178	179	182	183	226	227	230	231	194	195	198	199	242
184	185	188	189	232	233	236	237	200	201	204	205	248
186	187	190	191	234	235	238	239	202	203	206	207	250

Figure -40 -- The band-by-band scanning order for all three modes

In bilevel_quant mode, the band by band scanning is also employed, similar to the multi_quant mode. When bi-level quantization is applied, the coefficients that are already found significant are replaced with zero symbols for the purpose of zero-tree forming in later scans.

2. Entropy Decoding

The zero-tree (or type) symbols, quantized coefficient values (magnitude and sign), and residual values (for the multi quant mode) are all decoded using an adaptive arithmetic decoder with a given symbol alphabet. The arithmetic decoder adaptively tracks the statistics of the zerotree symbols and decoded values. For both the single quant and multi quant modes the arithmetic decoder is initialized at the beginning of each color loop for band-by-band scanning and at the beginning of the tree-block loop for tree-depth scanning. In order to avoid start code emulation, the arithmetic encoder always starts with stuffing one bit ?1? at the beginning of the entropy encoding. It also stuffs one bit ?1? immediately after it encodes every 22 successive ?0?s. It stuffs one bit ?1? to the end of bitstream in the case in which the last output bit of arithmetic encoder is ?0?. Thus, the arithmetic decoder reads and discards one bit before starts entropy decoding. During the decoding, it also reads and discards one bit after receiving every 22 successive ?0?s. The arithmetic decoder reads one bit and discards it if the last input bit to the arithmetic decoder is ?0?.

The context models used for SQ and MQ are identical to the ones used in BQ mode.

For both scanning orders in the single quant and multi_quant modes separate probability models are kept for each color and wavelet decomposition layer for the type and sign symbols while separate probability models are kept for each color, wavelet decomposition layer, *and* bitplane for the magnitude and residual symbols. All the models are initialized with a uniform probability distribution.

The models and symbol sets for the non-zerotree type quantities to be decoded are as follows:

Model Possible Values

Sign POSITIVE (0), NEGATIVE (1)

Magnitude 0, 1

Residual 0, 1

The possible values of the magnitudes and residuals are only 0 or 1 because each bitplane is being decoded separately. The non-residual values are decoded in two steps. First, the absolute value is decoded in a bitplane fashion using the *magnitude* probability models, then its sign is decoded.

For the decoding of the type symbols different probability models are kept for the leaf and non-leaf coefficients. For the multi quant mode, context modeling, based on the zerotree type of the coefficient in the previous scalability layer, is used. The different zerotree type models and their possible values are as follows:

Context and Leaf/Non-Leaf Possible Values

INIT ZTR (2), IZ (0), VZTR (3), VAL (1)

ZTR ZTR (2), IZ (0), VZTR (3), VAL (1)

ZTR DESCENDENT ZTR (2), IZ (0), VZTR (3), VAL (1)

IZ IZ (0), VAL (1)

LEAF INIT ZTR (0), VZTR (1)

LEAF ZTR ZTR (0), VZTR (1)

LEAF ZTR DESCENDENT ZTR (0), VZTR (1)

For the single quant mode only the INIT and LEAF INIT models are used. For

the multi quant mode for the first scalability layer only the INIT and LEAF INIT models are used. Subsequent scalability layers in the multi quant mode use the context associated with the type. If a new spatial layer is added then the contexts of all previous leaf band coefficients are switched to the corresponding non-leaf contexts. The coefficients in the newly added bands use the LEAF INIT context. The residual models are used to decode the coefficient refinements if in the previous layer, a *VZTR or VAL* symbol was assigned. If a node is currently not in SKIP mode (meaning that no refinement is being done for the coefficient - see subclause 7.10.3 on inverse quantization for details) only the magnitude of the refinements are decoded as these values are always zero or positive integers.

If a node is in SKIP mode, then its new zerotree symbol is decoded from bitstream, but no value is decoded for the node and its value in the current scalability layer is assumed to be zero.

States in Previous Bitplane Possibilities in current bitplane

ZTR ZTR, VZTR, IZ, VAL

VZTR SKIP

IZ IZ, VAL

VAL SKIP

DZTR ZTR, VTRZ, IZ, VAL

For the bi-level quantization mode, the zero-tree map is decoded for each bitplane, indicating which wavelet coefficients are zeros relative to the current quantization step size. Different probability models for the arithmetic decoder are used and updated according to the local contexts. For decoding the zerotree symbols, five context models are used, which are dependent on the status of the current wavelet coefficients in the zerotree formed in the previous bitplane decoding. Specifically, the five models correspond to the following contexts of the current wavelet coefficient:

· IZ: the previous zerotree symbol is Isolated Zero.

· VAL: the previous zerotree symbol is Value.

· ZTR: the previous zerotree symbol is Zerotree Root.

· VZTR: the previous zerotree symbol is valued zerotree root.

. DZTR: in previous bitplane, the current coefficient is a descendant of a zerotree root

The additional symbol DZTR is used for switching the models only, where DZTR refers to the descendant of a ZTR symbol. The context symbols DZTR can be inferred from the decoding process and are not included in the bitstream. They are used for switching the models only. At the beginning of the decoding the first bitplane, the contexts of the coefficients are initialized to be DZ. For the highest subband, only IZ and VAL are possible (no ZTR and VZTR are possible). Therefore, we initialize the arithmetic model for the last band differently (with zero probablility for ZTR and VZTR symbols).

For decoding the sign information, another context model (the sign model) is used and updated. For decoding the refinement bits, another statistical model (the refinement model) is used.

Each decomposition levels have their own separate arithmetic models. Therefore, the above decoding process applies to each decomposition levels. All models are initialized at the beginning of coding each bitplane.

After the zero-tree map, additional bits are received to refine the accuracy of the coefficients that are already marked significant by previously received information at the decoder. For each significant coefficient, the 1-bit bi-level quantized refinement values are entropy coded using the arithmetic coder.

3. Inverse Quantization

Different quantization step sizes (one for each color component) are specified for each level of scalability. The quantizer of the DC band is a uniform mid-step quantizer with a dead zone equal to the quantization step size. The quantization index is a signed integer number and the quantization reconstructed value is obtained using the following equation:

$V = id * Qdc,$

where V is the reconstructed value, *id* is the decoded index and Qdc is the quantization step size.

All the quantizers of the higher bands (in all quantization modes) are uniform mid-step quantizer with a dead zone 2 times the quantization step size. For the single quantization mode, the quantization index is an signed integer. The reconstructed value is obtained using the following algorithm:

```
if ($id$ == 0)
V =0;
else if ( $id$ > 0 )
V = $id$*Q+Q/2;
else
V = $id$*Q-Q/2;
```

where V is the reconstructed value, id is the decoded index and Q is the quantization step size.

In the multi-quantization mode each SNR layer within each spatial layer has an associated quantization step-size value (Q value). These different Q Values are used for SNR scalability. A lower Q Value will result in a more accurate reconstruction.

If a coefficient is in a given spatial layer it is also in all higher numbered spatial layers. SNR scalability may be continued on these coefficients in the higher numbered spatial layers in the same way as is done in the spatial layer the coefficient first arises in. Thus, we can think of all the coefficients which first arise in a particular spatial layer as having a corresponding sequence of Q Values (call it a Q Sequence). The Q Sequence is made up of the quantization values for all SNR layers in the spatial layer the coefficient first arises in plus the quantization values in all SNR layers in all higher spatial layers. The order is from lower to higher numbered spatial layers and from lower to higher numbered SNR layers within each spatial layer.

EXAMPLE

Let the quantization value of the *i*-th spatial layer and the *j*-th SNR layer be denoted by *Q(i,j)*. Assume we have the following scenario:

Spatial SNR Layer

Layer 0 1 2 .

0 *Q(0,0) Q(0,1) Q(0,2)*

1 *Q(1,0) Q(1,1) Q(1,2)*

2 *Q(2,0) Q(2,1) Q(2,2)*

The Q Sequence which will be used to quantize all coefficients which first arise in spatial layer 0 is:

<Q(0,0) Q(0,1) Q(0,2) Q(1,0) Q(1,1) Q(1,2) Q(2,0) Q(2,1) Q(2,2)>

while the sequence for all coefficients first arising in spatial layer 1 is:

<Q(1,0) Q(1,1) Q(1,2) Q(2,0) Q(2,1) Q(2,2)>

and the sequence for all coefficients first arising in spatial layer 2:

<Q(2,0) Q(2,1) Q(2,2)>.

As in the single-quantization case we would like to have a uniform quantizer for all layers. Due to the manner in which the Q Values are used to achieve scalability (described below), in order to have a (approximately) uniform quantizer at each layer, we may have to revise the Q Values extracted from the bitstream before reconstruction. This revision is necessary if the Q Values within each Q Sequence are not integer multiples of one another or if Q Value is greater than a Q Value occurring earlier in the Q Sequence.

EXAMPLES

Q Sequences needing no revision: <*24 8 2*> and <*81 81 27*>.

Q Sequences needing revision: <*31 9 2*> (non-integer multiples) and <*81 162 4*> (increasing Q Value).

If a coefficient's quantization indices have been zero for all previous scalability layers (spatial and SNR) or if it is the first scalability layer, then the reconstruction is the similar to the single-quantization mode described above. The difference is in that the refined Q Values may be used instead of the ones extracted from the bitstream. The refinement process is described below in steps 1 and 2. If there has been a non-zero quantization index in a previous scalability layer then the quantization index specifies a refinement of the previous quantization. The indices are then called residuals. For every coefficient and scalability layer we know (1) the quantization interval where the coefficient occurred in the last scalability layer (both size and location), (2) the spatial layer the coefficient first arose in (and thus, which Q Sequence to use), (3) the current Q Value and the previous Q Value in the Q Sequence, and (4) the refinment (if any) of the previous Q Value.

The reconstruction of the residual is calculated in the following five steps.

Step 1: Calculation of the Number of Refinement Intervals

The quantization interval which was indexed in the previous layer is to be partitioned into disjoint intervals. The number of these "refinement" intervals is calculated based solely on the current Q Value (call it *curQ*) and the previous Q Value (call it *prevQ*). Note that *prevQ* may have been revised as mentioned above. Letting *m* be the number of refinement intervals we calculate

m = ROUND(prevQ¸ curQ)

where ROUND(x) = MAX(nearest integer of x, 1)).

If *m = 1*, no refinement is needed and no value will have been sent. If, at a certain scalability layer, a node has *m=1* then it is said to be in SKIP mode. Thus, steps 2, 3, and 4 need not be performed for the coefficient.

Step 2: Calculation of the Maximum Refinement Interval Size

Using the number of refinement intervals, the current Q Value, *curQ*, is revised (if necessary).

curQ = CEIL(prevQ ¸ m)

where CEIL rounds up to the nearest integer.

curQ represents the maximum size of the intervals in the partition. Since *prevQ* is the previous layer’s *curQ* (see step 5), we see that *prevQ* represents the maximum size of the intervals in the partition used in the previous scalability layer.

Step 3: Construction of Refinement Partition

Using the values *m* and *curQ* calculated above and the size of quantization interval where the coefficient occurred in the last scalability layer, we form the refinement partition.

The previous layer’s quantization interval is partitioned into *m* intervals which are of size *curQ* or *curQ-1*. The residual will be one of the values *{0, 1, ..., m-1}* which represent an index into this partition. A lower number index corresponds to an interval in the partition covering lower magnitude values. If the partition is made up of different size intervals (*curQ* and *curQ-1*) then the *curQ* size intervals correspond to the lower indices. Some combination of m *curQ* and *curQ-1* interval sizes are sufficient to cover the previous quantization interval. From step 2 we know that the previous quantization interval is of size *prevQ* or *prevQ-1*.

Step 4: Calculation of Reconstructed Value

The interval in the partition indexed by the residual is mapped to the reconstruction value. The reconstruction is just the middle point of the interval in the partition that the residual indexes. That is, if *PartStart* is the start of the interval in the partition which is indexed by the residual, *PartStartSize* is the size of the interval, *sign* is the corresponding sign (known from prior scalability layers), and // is integer division then the reconstructed value is:

PartStart + sign*(PartStartSize-1)//2

Step 5: Assignment of Maximum Size

If there is another scalability layer then *prevQ* is assigned the value of *curQ*.

Note that since steps 1, 2, and 5 depend entirely on the Q Values found in the Q Sequences they only need to be done once in each scalability layer for each Q Sequence being used in the current spatial layer.

FOUR EXAMPLES

In the examples:

1. 1. let the Q Values be Q1 , Q2 , and Q3,
1. 2. let two sample coefficients to be quantized be *C1* and *C2*,
1. 3. let *Cq1* and *Cq2* denote the corresponding quantized coefficients or residuals, and
1. 4. let *iC1* and *iC2* denote the corresponding reconstructed coefficient values.

1. Q Values not in need of revision

Q1 = 24 , Q2 = 8, Q3 = 2,

C1 = 16, and C2 = 28.

At first scalability layer we have

Cq1 = C1/Q1 = 0

iCq1 = 0

Cq2 = C2/Q1 = 1

iCq2 = 35

At second scalability layer we have

prevQ = Q1 = 24

curQ = Q2 = 8

m = ROUND(prevQ¸ curQ) = ROUND(24¸ 8) = 3

curQ = CEIL(prevQ¸ m) = CEIL(24¸ 3) = 8

partition sizes = {8, 8, 8}

Cq1 = C1/curQ = 2

iCq1 = 19

Cq2 = 0 (residual)

iCq2 = 27

At third scalability layer we have

prevQ = curQ = 8

curQ = Q3 = 2

m = ROUND(prevQ¸ curQ) = ROUND(8¸ 2) = 4

curQ = CEIL(prevQ¸ m) = CEIL(8¸ 2) = 4

partition sizes = {2, 2, 2, 2}

Cq1 = 0 (residual)

iCq1 = 16

Cq2 = 2 (residual)

iCq2 = 28

2. Q Values not in need of revision

Q1 = 81, Q2 = 81, Q3 = 27,

C1 = 115 , and C2 = 28.

At first scalability layer we have

Cq1 = C1/Q1 = 1

iCq1 = 121

Cq2 = C2/Q1 = 0

iCq2 = 0

At second scalability layer we have

prevQ = Q1 = 81

curQ = Q2 = 81

m = ROUND(prevQ¸ curQ) =ROUND(81¸ 81) = 1

curQ = CEIL(prevQ¸ m) = CEIL(81¸ 1) = 81

partition sizes = {81} (no refinement needed)

Cq1 = 0 (residual)

iCq1 = 121

Cq2 = C2/curQ = 0

iCq2 = 0

At third scalability layer we have

prevQ = curQ = 81

curQ = Q3 = 27

m = ROUND(prevQ¸ curQ) = ROUND(81¸ 27) = 3

curQ = CEIL(prevQ¸ m) = CEIL(81¸ 3) = 27

partition sizes = {27, 27, 27}

Cq1 = 1 (residual)

iCq1 = 121

Cq2 = C2/curQ = 1

iCq2 = 40

3. Q Values in need of revision

Q1 = 31 , Q2 = 9, Q3 = 2,

C1 = 115 , and C2 = 5.

At first scalability layer we have

Cq1 = C1/Q1 = 3

iCq1 = 108

Cq2 = C2/Q1 = 0

iCq2 = 0

At second scalability layer we have

prevQ = Q1 = 31

curQ = Q2 = 9

m = ROUND(prevQ¸ curQ) = ROUND(31¸ 9) = 3

curQ = CEIL(prevQ¸ m) = CEIL(31¸ 3) = 11

partition sizes = {11, 10, 10}

Cq1 = 2 (residual)

iCq1 = 118

Cq2 = C2/curQ = 0

iCq2 = 0

At third scalability layer we have

prevQ = curQ = 11

curQ = Q3 = 2

m = ROUND(prevQ¸ curQ) = ROUND(11¸ 2) = 6

curQ = CEIL(prevQ¸ m) = CEIL(11¸ 6) = 2

partition sizes = {2, 2, 2, 2, 2,1} if value occurs in level with size 11

partition sizes = {2, 2, 2, 2, 1, 1} if value occurs in level with size 10

Cq1 = 0 (residual)

iCq1 = 114

Cq2 = C2/curQ = 2

iCq2 = 4

4. Q Values in need of revision

Q1 = 81, Q2 = 162, Q3 = 4,

C1 = 115 , and C2 = 5.

At first scalability layer we have

Cq1 = C1/Q1 = 1

iCq1 = 121

Cq2 = C2/Q1 = 0

iCq2 = 0

At second scalability layer we have

prevQ = Q1 = 81

curQ = Q2 = 162

m = ROUND(prevQ¸ curQ) = ROUND(81¸ 162) = 1

curQ = CEIL(prevQ¸ m) = CEIL(81¸ 1) = 81

partition sizes = {81} (no refinement needed)

Cq1 = 0 (not used)

iCq1 = 121

Cq2 = C2/curQ = 0 (not used)

iCq2 = 0

At third scalability layer we have

prevQ = curQ = 81

curQ = Q3 = 4

m = ROUND(prevQ¸ curQ) = ROUND(81¸ 4) = 20

curQ = CEIL(prevQ¸ m) = CEIL(81¸ 20) = 5

partition sizes = {5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4}

Cq1 = 8 (residual)

iCq1 = 121

Cq2 = C2/curQ = 1

iCq2 = 6

It is important to note that coefficients which first arose in different spatial layers may use different *prevQ* and *curQ* values. They are basically being quantized from different lists. That is, they have different corresponding Q Sequences.

1. Shape adaptive zerotree decoding

When the texture_object_layer_shape is not rectangular or **texture_object_layer_width** !=

integer*2$^{\textbf{wavelet\_decomposition\_level}}$ or **texture_object_layer_height**!= integer*2 $^{\textbf{wavelet\_decomposition\_level}}$, the inverse shape adaptve wavelet transform is chosen to reconstructed the arbitrary-shaped image object or rectangular texture object. Decoding shape adaptive wavelet coefficients is the same as decoding regular wavelet coefficients except keep track of the locations of where to put the decoded wavelet coefficients according to the shape information. or a generated mask. The mask is generated with **texture_object_layer_width*****texture_object_layer_height** pixels of value 255 at the upper-left corner of a frame of size w*2$^{\text{wavelet\_decomposition\_level}}$ * h*2$^{\text{wavelet\_decomposition\_level}}$ and the rest of the pixels being value 0, where w is the smallest integer that makes w*2 $^{\text{wavelet\_decomposition\_level}}$ > **texture_object_layer_width** and h is the smallest integer that makes h*2 $^{\text{wavelet\_decomposition\_level}}$ > **texture_object_layer_height** . Similar to decoding of regular wavelet coefficients, the decoded zerotree symbols at a lower subband are used to determine whether decoding is needed at higher subbands. The difference is now that some zerotree nodes correspond to the pixel locations outside the shape boundary and no bits are to be decoded for these out_nodes. Root layer is defined as the lowest three AC subbands, leaf layer is defined as the highest three AC subbands. For decomposition level of one, the overlapped root layer and leaf laver shall be treated as leaf layer.

1. **DC layer**

The DC coefficient decoding is the same as that for rectangular image except the following,

1. Only those DC coefficients inside the shape boundary in the DC layer shall be traversed and decoded and DC coefficients outside the shape boundary may be set to zeros.
1. For the inverse DC prediction in the DC layer, if a reference coefficient (A, B, C in Fig. (DC prediction figure)) in the prediction context is outside the shape boundary, zero shall be used to form the prediction syntax.

1. Root layer

At the root layer (the lowest 3 AC bands), the shape information is examined for every node to determine whether a node is an out_node.

If it is an out_node,

- no bits are decoded for this node;
- the four children nodes of this node are marked "to_be_decoded" (TBD);

otherwise,

- a zerotree symbol is decoded for this node using an adaptive arithmetic decoder.

If the decoded symbol for the node is either isolated_zero (IZ) or value (VAL),

- the four children nodes of this node are marked TBD;

otherwise,

- the symbol is either zerotree_root (ZTR) or valued_zerotree_root (VZTR) and the four children nodes of this node are marked "no_code" (NC).

If the symbol is VAL or VZTR,

- a non-zero wavelet coefficient is decoded for this node by root model;

otherwise,

- the symbol is either IZ or ZTR and the wavelet coefficient is set to zero for this node.

1. **Between root and leaf layer**

At any layer between the root layer and the leaf layer, the shape information is examined for every node to determine whether a node is an out_node.

If it is an out_node,

- no bits are decoded for this node;
- the four children nodes of this node are marked as either TBD or NC depending on whether this node itself is marked TBD or NC respectively;

otherwise, if it is marked NC,

- no bits are decoded for this node;
- the wavelet coefficient is set to zero for this node;
- the four children nodes are marked NC;

otherwise,

- a zerotree symbol is decoded for this node using an adaptive arithmetic decoder.

If the decoded symbol for the node is either isolated_zero (IZ) or value (VAL),

- the four children nodes of this node are marked TBD;

otherwise,

- the symbol is either zerotree_root (ZTR) or valued_zerotree_root (VZTR) and the four nodes of this node are marked "no_code" (NC).

If the symbol is VAL or VZTR,

- a non-zero wavelet coefficient is decoded for this node by valnz model;

otherwise,

- the symbol is either IZ or ZTR and the wavelet coefficient is set to zero for this node.

1. **Leaf layer**

At the leaf layer, the shape information is examined for every node to determine whether a node is an out_node.

If it is an out_node,

- no bits are decoded for this node;

otherwise, if it is marked NC,

- no bits are decoded for this node;
- the wavelet coefficient is set to zero for this node;

otherwise,

- * a wavelet coefficient is decoded for this node by valz adaptive arithmetic model;

1. **Shape decomposition**

The shape information for both shape adaptive zerotree decoding and the inverse shape adaptive wavelet transform is obtained by decomposing the reconstructed shape from the shape decoder. Assuming binary shape with 0 or 1 indicating a pixel being outside or inside the arbitrarily shaped object, the shape decomposition procedure can be described as follows:

1. For each horizontal line, collect all even-indexed shape pixels together as the shape information for the horizontal low-pass band and collect all odd-indexed shape pixels together as the shape information for the horizontal high-pass band, except for the special case where the number of consecutive 1?s is one.
2. For an isolated 1 in a horizontal line, whether at an even-indexed location or at an odd-indexed location, it is always put together with the shape pixels for the low-pass band and a 0 is put at the corresponding position together with the shape pixels for the high-pass band.
3. Perform the above operations for each vertical line after finishing all horizontal lines.
4. Use the above operations to decompose the shape pixels for the horizontal and vertical low-pass band further until the number of decomposition levels is reached.

1. Mesh object decoding

An overview of the decoding process is show in Figure 7-41.

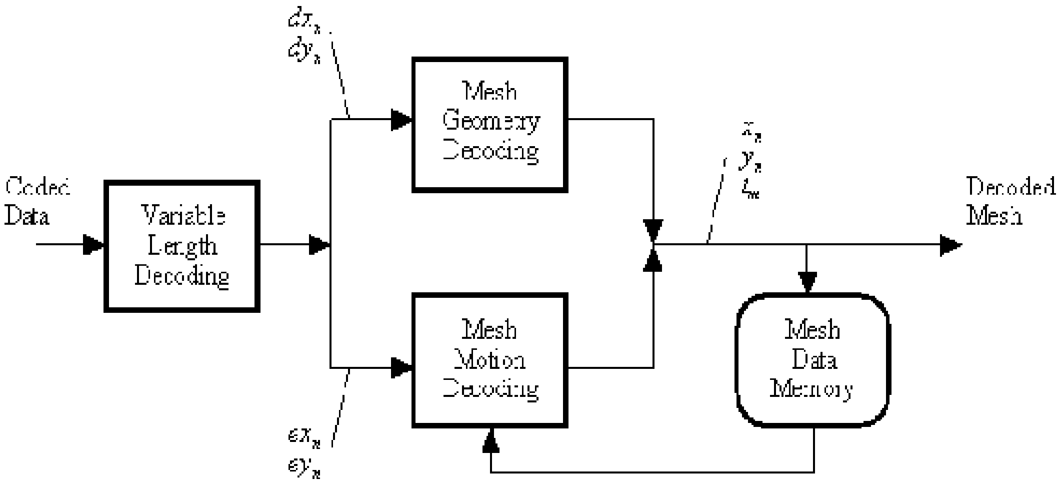


Figure -41 -- Simplified 2D Mesh Object Decoding Process

Variable length decoding takes the coded data and decodes either node point location data or node point motion data. Node point location data is denoted by dx_n, dy_n and node point motion data is denoted by ex_n, ey_n, where n is the node point index (n = 0, ..., N-1). Next, either mesh geometry decoding or mesh motion decoding is applied. Mesh geometry decoding computes the node point locations from the location data and reconstructs a triangular mesh from the node point locations. Mesh motion decoding computes the node point motion vectors from the motion data and applies these motion vectors to the node points of the previous mesh to reconstruct the current mesh.

The reconstructed mesh is stored in the mesh data memory, so that it may be used by the motion decoding process for the next mesh. Mesh data consists of node point locations (x_n, y_n) and triangles t_m, where m is the triangle index (m = 0, ..., M-1) and each triangle t_m contains a triplet <i, j, k> which stores the indices of the node points that form the three vertices of that triangle.

A mesh object consists of a sequence of mesh object planes. The is_intra flag of the mesh object plane class determines whether the data that follows specifies the initial geometry of a new dynamic mesh, or that it specifies the motion of the previous mesh to the current mesh, in a sequence of meshes. Firstly, the decoding of mesh geometry is described; then, the decoding of mesh motion is described. In this part of ISO/IEC 14496, a pixel-based coordinate system is assumed, where the x-axis points to the right from the origin, and the y-axis points down from the origin.

1. Mesh geometry decoding

Since the initial 2D triangular mesh is either a uniform mesh or a Delaunay mesh, the mesh triangular structure (i.e. the connections between node points) is not coded explicitly. Only a few parameters are coded for the uniform mesh; only the 2D node point coordinates $\vec{p}_n = (x_n, y_n)$ are coded for the Delaunay mesh. In each case, the coded information defines the triangular structure of the mesh implicitly, such that it can be computed uniquely by the decoder. The mesh_type_code specifies whether the initial mesh is uniform or Delaunay.

1. **Uniform mesh**

   A 2D uniform mesh subdivides a rectangular object plane area into a set of rectangles, where each rectangle in turn is subdivided into two triangles. Adjacent triangles share node points. The node points are spaced equidistant horizontally as well as vertically. An example of a uniform mesh is given in Figure 7-42.

   Five parameters are used to specify a uniform mesh. The first two parameters, nr_mesh_nodes_hor and nr_mesh_nodes_vert, specify the number of node points of the mesh in the horizontal, resp. vertical direction. In the example of Figure 7-42, nr_mesh_nodes_hor is equal to 5 and nr_mesh_nodes_vert is equal to 4. The next two parameters, mesh_rect_size_hor and mesh_rect_size_vert, specify the horizontal, resp. vertical size of each rectangle in half pixel units. The meaning of these parameters is indicated in Figure 7-42. The last parameter, triangle_split_code, specifies how each rectangle is split to form two triangles. The four methods of splitting that are allowed are indicated in Figure 7-43. The top-left node point of a uniform mesh coincides with the origin of a local coordinate system.

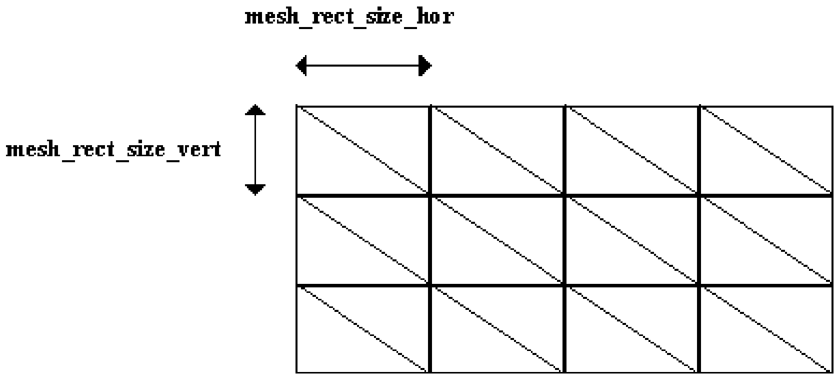
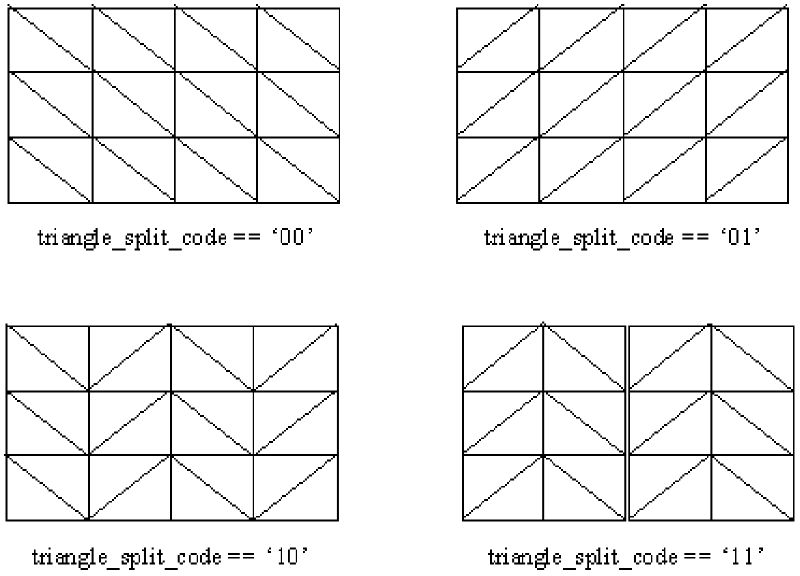


Figure -42 -- Specification of a uniform 2D mesh



triangle_split_code == '00'

triangle_split_code == '01'

triangle_split_code == '10'

triangle_split_code == '11'

Figure -43 -- Illustration of the types of uniform meshes defined

2. Delaunay mesh

First, the total number of node points in the mesh *N* is decoded; then, the number of node points that are on the boundary of the mesh N_b is decoded. Note that *N* is the sum of the number of nodes in the interior of the mesh, N_i and the number of nodes on the boundary, N_b,

$$N = N_i + N_b \,.$$

Now, the locations of boundary and interior node points are decoded, where we assume the origin of the local coordinate system is at the top left of the bounding rectangle surrounding the initial mesh. The x-, resp. y-coordinate of the first node point, $\vec{p}_0 = (x_0, y_0)$, is decoded directly, where x_0 and y_0 are specified w.r.t. to the origin of the local coordinate system All the other node point coordinates are computed by adding a dx_n, resp. dy_n value to, resp. the x- and y-coordinate of the previously decoded node point. Thus, the coordinates of the initial node point $\vec{p}_0 = (x_0, y_0)$ is decoded as is, whereas the coordinates of all other node points , $\vec{p}_n = (x_n, y_n)$, *n* = 1, ..., *N* - 1, are obtained by adding a decoded value to the previously decoded node point coordinates:

$$x_n = x_{n-1} + dx_n \text{ and } y_n = y_{n-1} + dy_n \,.$$

The ordering in the sequence of decoded locations is such that the first N_b locations correspond to boundary nodes. Thus, after receiving the first N_b locations, the decoder is able to reconstruct the boundary of the mesh by connecting each pair of successive boundary nodes, as well as the first and the last, by straight-line edge segments. The next *N* - N_b values in the sequence of decoded locations correspond to interior node points. Thus, after receiving *N* nodes, the locations of both the boundary and interior nodes can be reconstructed, in addition to the polygonal shape of the boundary. This is illustrated with an example in Figure 7-44.

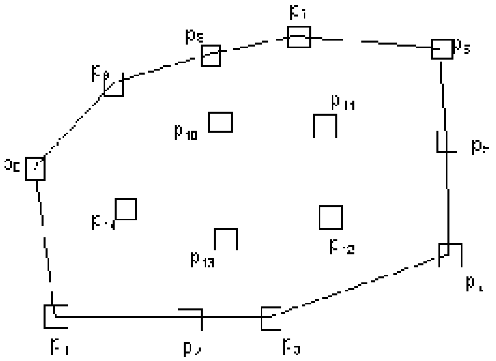


Figure -44 -- Decoded node points and mesh boundary edge

The mesh is finally obtained by applying constrained Delaunay triangulation to the set of decoded node points, where the polygonal mesh boundary is used as a constraint. A constrained triangulation of a set of node points $\vec{p}_n$ contains the line segments between successive node points on the boundary as edges and contains triangles only in the interior of the region defined by the boundary. Each triangle $t_k = \langle \vec{p}_l, \vec{p}_m, \vec{p}_n \rangle$ of a constrained Delaunay triangulation furthermore satisfies the property that the circumcircle of t_k does not contain in its interior any node point $\vec{p}_r$ visible from all three vertices of t_k. A node point is visible from another node point if a straight line drawn between them falls entirely inside or exactly on the constraining polygonal boundary. The Delaunay triangulation process is defined as any algorithm that is equivalent to the following.

a. Determine any triangulation of the given node points such that all triangles are contained in the interior of the polygonal boundary. The triangulation shall contain 2 $N_i + N_b$ - 2 triangles.

b. Inspect each interior edge, shared by two opposite triangles, of the triangulation and test if the edge is locally Delaunay. If there is an interior edge that is not locally Delaunay, the two opposite triangles <p_a, p_b, p_c> and <p_a, p_c, p_d> sharing this edge are replaced by triangles <p_a, p_b, p_d> and <p_b, p_c, p_d>. Continue until all interior edges of the triangulation are locally Delaunay.

An interior edge, shared by two opposite triangles <p_a, p_b, p_c> and <p_a, p_c, p_d>, is locally Delaunay if point p_d is outside the circumcircle of triangle <p_a, p_b, p_c>. If point p_d is inside the circumcircle of triangle <p_a, p_b, p_c>, then the edge is not locally Delaunay. If point p_d is exactly on the circumcircle of triangle <p_a, p_b, p_c>, then the edge between points p_a and p_c is deemed locally Delaunay only if point p_b or point p_d is the point (among these four points) with the maximum x-coordinate, or, in case there is more than one point with the same maximum x-coordinate, the point with the maximum y-coordinate among these points.An example of a mesh obtained by constrained triangulation of the node points of Figure 7-44 is shown in Figure 7-45.

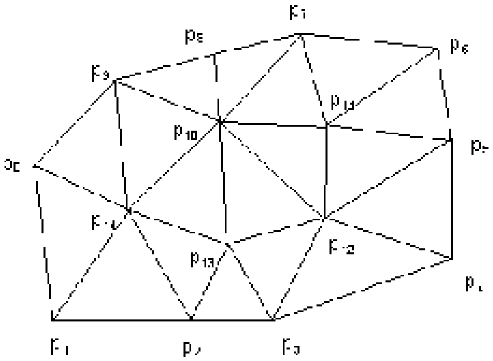


Figure -45 -- Decoded triangular mesh obtained by constrained Delaunay triangulation

1. Decoding of mesh motion vectors

Each node point $\vec{p}_n$ of a 2D Mesh Object Plane numbered *k* in the sequence of Mesh Object Planes has a 2D motion vector $\vec{v}_n = (vx_n, vy_n)$, defined from Mesh Object Plane k to $k+1$. By decoding these motion vectors, one is able to reconstruct the locations of node points in Mesh Object Plane numbered *k*+1. The triangular topology of the mesh remains the same throughout the sequence. Node point motion vectors are decoded according to a predictive method, i.e., the components of each motion vector are predicted using the components of already decoded motion vectors of other node points.

1. Motion vector prediction

To decode the motion vector of a node point $\vec{p}_n$ that is part of a triangle $t_k = \langle \vec{p}_l, \vec{p}_m, \vec{p}_n \rangle$, where the two motion vectors vectors $\vec{v}_l$ and $\vec{v}_m$ of the nodes $\vec{p}_l$ and $\vec{p}_m$ have already been decoded one can use the values of $\vec{v}_l$ and $\vec{v}_m$ to predict $\vec{v}_n$ and add the prediction vector to a decoded prediction error vector. Starting from an initial triangle t_k of which all three node motion vectors have been decoded, there must be at least one other, neighboring, triangle t_w that has two nodes in common with t_k. Since the motion vectors of the two nodes that t_k and t_w have in common have already been decoded, one can use these two motion vectors to predict the motion vector of the third node in t_w. The actual prediction vector $\vec{w}_n$ is computed by averaging of the two prediction motion vectors and the components of the prediction vector are rounded to half-pixel accuracy, as follows:

$$\vec{w}_n = 0.5 \bullet \left(\text{floor}(vx_m + vx_l + 0.5), \text{floor}(vy_m + vy_l + 0.5)\right),$$

$$\vec{v}_n = \vec{w}_n + \vec{e}_n$$

Here, $\vec{e}_n = (ex_n, ey_n)$ denotes the prediction error vector, the components of which are decoded from variable length codes. This procedure is repeated while traversing the triangles and nodes of the mesh, as explained below. While visiting all triangles of the mesh, the motion vector data of each node is decoded from the bitstream one by one. Note that no prediction is used to decode the first motion vector,

$$\vec{v}_{n_0} = \vec{e}_{n_0},$$

and that only the first decoded motion vector is used as a predictor to code the second motion vector,

$$\vec{v}_{n_1} = \vec{v}_{n_0} + \vec{e}_{n_1}.$$

Note further that the prediction error vector is specified only for node points with a nonzero motion vector. For all other node points, the motion vector is simply $\vec{v}_n = (0,0)$.

Finally, the horizontal and vertical components of mesh node motion vectors are processed to lie within a certain range, equivalent to the processing of video block motion vectors described in subclause 7.6.3.

2. **Mesh traversal**

We use a *breadth-first traversal* to order all the triangles and nodes in the mesh numbered k, and to decode the motion vectors defined from mesh k to k+1. The breadth-first traversal is determined uniquely by the topology and geometry of an intra-coded mesh. That is, the ordering of the triangles and nodes shall be computed on an intra-coded Mesh Object Plane and remains constant for the following predictive-coded Mesh Object Planes. The breadth-first traversal of the mesh triangles is defined as follows (see Figure 7-46 for an illustration).

First, define the *initial triangle* as follows. Define the top left mesh node as the node n with minimum $x_n + y_n$, assuming the origin of the local coordinate system is at the top left. If there is more than one node with the same value of $x_n + y_n$, then choose the node point among these with minimum y. The initial triangle is the triangle that contains the edge between the top-left node of the mesh and the next clockwise node on the boundary. Label the initial triangle with the number 0.

Next, all other triangles are iteratively labeled with numbers 1, 2, ..., M - 1, where M is the number of triangles in the mesh, as follows.

> Among all labeled triangles that have adjacent triangles which are not yet labeled, find the triangle with the lowest number label. This triangle is referred to in the following as the *current triangle*. Define the *base edge* of this triangle as the edge that connects this triangle to the already labeled neighboring triangle with the lowest number. In the case of the initial triangle, the base edge is defined as the edge between the top-left node and the next clockwise node on the boundary. Define the *right edge* of the current triangle as the next counterclockwise edge of the current triangle with respect to the base edge; and define the *left edge* as the next clockwise edge of the current triangle with respect to the base edge. That is, for a triangle $t_k = \langle \vec{p}_l, \vec{p}_m, \vec{p}_n \rangle$, where the vertices are in clockwise order, if $\langle \vec{p}_l \vec{p}_m \rangle$ is the base edge, then $\langle \vec{p}_l \vec{p}_n \rangle$ is the right edge and $\langle \vec{p}_m \vec{p}_n \rangle$ is the left edge.
>
> Now, check if there is an unlabeled triangle adjacent to the current triangle, sharing the right edge. If there is such a triangle, label it with the next available number. Then check if there is an unlabeled triangle adjacent to the current triangle, sharing the left edge. If there is such a triangle, label it with the next available number.

This process is continued iteratively until all triangles have been labeled with a unique number *m*.

The ordering of the triangles according to their assigned label numbers implicitly defines the order in which the motion vector data of each node point is decoded, as described in the following. Initially, motion vector data for the top-left node of the mesh is retrieved from the bitstream. No prediction is used for the motion vector of this node, hence this data specifies the motion vector itself. Then, motion vector data for the second node, which is the next clockwise node on the boundary w.r.t. the top-left node, is retrieved from the bitstream. This data contains the prediction error for the motion vector of this node, where the motion vector of the top-left node is used as a prediction. Mark these first two nodes (that form the base edge of the initial triangle) with the label ?done?.

Next, process each triangle as determined by the label numbers. For each triangle, the base edge is determined as defined above. The motion vectors of the two nodes of the base edge of a triangle are used to form a prediction for the motion vector of the third node of that triangle. If that third node is not yet labeled ?done?, motion vector data is retrieved and used as prediction error values, i.e. the decoded values are added to the prediction to obtain the actual motion vector. Then, that third node is labeled ?done?. If the third note is already labeled ?done?, then it is simply ignored and no data is retrieved. Note that due to the ordering of the triangles as defined above, the two nodes on the base edge of a triangle are guaranteed to be labeled ?done? when that triangle is processed, signifying that their motion vectors have already been decoded and may be used as predictors.

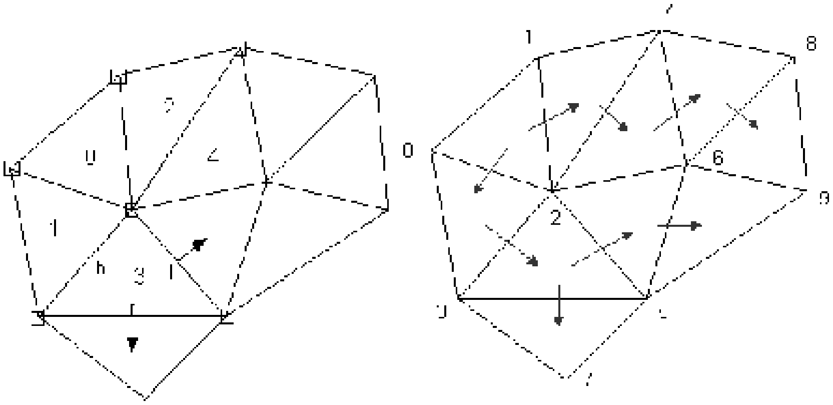


Figure -46 -- Breadth-first traversal of a 2D triangular example mesh

In Figure 7-46 an example is shown of breadth-first traversal. On the left, the traversal is halfway through the mesh - five triangles have been labeled (with numbers) and the motion vectors of six node points have been decoded (marked with a box symbol). The triangle which has been labeled ?3? is the ?current triangle?; the base edge is ?b?; the right and left edge are ?r? and ?l?. The triangles that will be labeled next are the triangles sharing the right, resp. left edge with the current triangle. After those triangles are labeled, the triangle which has been labeled ?4? will be the next ?current triangle? and another motion vector will be decoded. On the right, the traversed 2D triangular mesh is shown, illustrating the transitions between triangles and final order of node points according to which respective motion vectors are decoded.

1. Face object decoding

1. Frame based face object decoding

This subclause specifies the additional decoding process required for face object decoding.

The coded data is decoded by an arithmetic decoding process. The arithmetic decoding process is described in detail in annex B. Following the arithmetic decoding, the data is de-quantized by an inverse quantization process. The FAPs are obtained by a predictive decoding scheme as shown in Figure 7-47.

The base quantization step size QP for each FAP is listed in Table C-1. The quantization parameter fap_quant is applied uniformly to all FAPs. The magnitude of the quantization scaling factor ranges from 1 to 8. The value of fap_quant == 0 has a special meaning, it is used to indicate lossless coding mode, so no dequantization is applied. The quantization stepsize is obtained as follows:

```
if (fap_quant)
        qstep = QP * fap_quant
else
        qstep = 1
```

The dequantized FAP?(t) is obtained from the decoded coefficient FAP??(t) as follows:

FAP?(t) = qstep * FAP??(t)

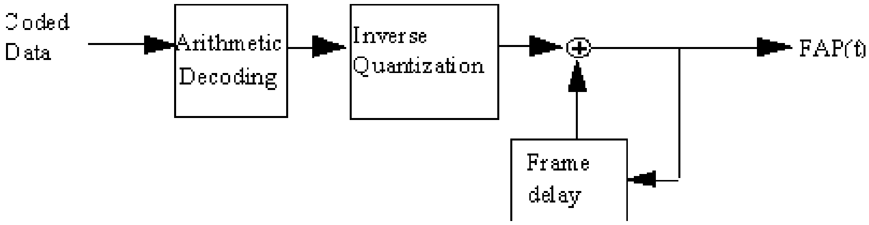


Figure -47 -- FAP decoding

1. Decoding of faps

For a given frame FAPs in the decoder assume one of three of the following states:

1. set by a value transmitted by the encoder
2. retain a value previously sent by the encoder
3. interpolated by the decoder

FAP values which have been initialized in an intra coded FAP set are assumed to retain those values if subsequently masked out unless a special mask mode is used to indicate interpolation by the decoder. FAP values which have never been initialized must be estimated by the decoder. For example, if only FAP group 2 (inner lip) is used and FAP group 8 (outer lip) is never used, the outer lip points must be estimated by the decoder. In a second example the FAP decoder is also expected to enforce symmetry when only the left or right portion of a symmetric FAP set is received (e.g. if the left eye is moved and the right eye is subject to interpolation, it is to be moved in the same way as the left eye).

1. DCT based face object decoding

The bitstream is decoded into segments of FAPs, where each segment is composed of a temporal sequence of 16 FAP object planes. The block diagram of the decoder is shown in Figure 7-48.

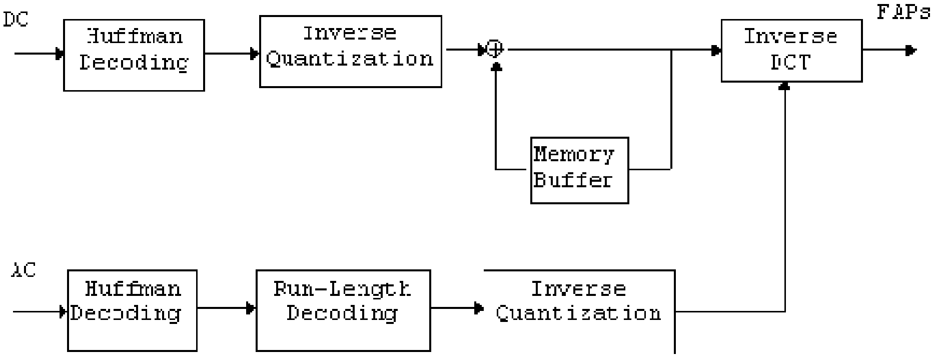


Figure -48 -- Block diagram of the DCT-based decoding process

The DCT-based decoding process consists of the following three basic steps:

1. Differential decoding the DC coefficient of a segment.
2. Decoding the AC coefficients of the segment
3. Determining the 16 FAP values of the segment using inverse discrete cosine transform (IDCT).

A uniform quantization step size is used for all AC coefficients. The quantization step size for AC coefficients is obtained as follows:

qstep[i] = fap_scale[fap_quant_inex] * DCTQP[i] where DCTQP[i] is the base quantization step size and its value is defined in subclause 6.3.10.10. The quantization step size of the DC coefficient is one-third of the AC coefficients. Different quantization step sizes are used for different FAPs.

The DCT-based decoding process is applied to all FAP segments except the viseme (FAP #1) and expression (FAP #2) parameters. The latter two parameters are differential decoded without transform. The decoding of viseme and expression segments are described at the end of this subclause.

For FAP #3 to FAP #68, the DC coefficient of an intra coded segment is stored as a 16-bit signed integer if its value is within the 16-bit range. Otherwise, it is stored as a 31-bit signed integer. For an inter coded segment, the DC coefficient of the previous segment is used as a prediction of the current DC coefficient. The prediction error is decoded using a Huffman table of 512 symbols. . An "ESC" symbol, if obtained, indicates that the prediction error is out of the range [-255, 255]. In this case, the next 16 bits extracted from the bitstream are represented as a signed 16-bit integer for the prediction error. If the value of the integer is equal to -256*128, it means that the value of the prediction error is over the 16-bit range. Then the following 32 bits from the bitstream are extracted as a signed 32-bit integer, in twos complement format and the most significant bit first

The AC coefficients, for both inter and intra coded segments, are decoded using Huffman tables. The run-length code indicates the number of leading zeros before each non-zero AC coefficient. The run-length ranges from 0 to 14 and proceeds the code for the AC coefficient. The symbol 15 in the run length table indicates the end of non-zero symbols in a segment. Therefore, the Huffman table of the run-length codes contains 16 symbols. The values of non-zero AC coefficients are decoded in a way similar to the decoding of DC prediction errors but with a different Huffman table.

The bitstreams corresponding to viseme and expression segments are basically differential decoded without IDCT. For an intra coded segment, the quantized values of the first viseme_select1, viseme_select2, viseme_blend, expression_select1, expression_select2, expression_intensity1, and expression_intensity2 within the segment are decoded using fixed length code. These first values are used as the prediction for the second viseme_select1, viseme_select2, ? etc of the segment and the prediction error are differential decoded using Huffman tables. For an inter coded segment, the last viseme_select1, for example, of the previous decoded segment is used to predict the first viseme_select1 of the current segment. In general, the decoded values (before inverse quantization) of differential coded viseme and expression parameter fields are obtained

byviseme_segment_select1q[k] = viseme_segment_select1q[k-1] +

viseme_segment_select1q_diff[k] - 14

viseme_segment_select2q[k] = viseme_segment_select2q[k-1] +

viseme_segment_select2q_diff[k] - 14

viseme_segment_blendq[k] = viseme_segment_blendq[k-1] +

viseme_segment_blendq_diff[k] - 63

expression_segment_select1q[k] = expression_segment_select1q[k-1] +

expression_segment_select1q_diff[k] - 6

expression_segment_select2q[k] = expression_segment_select2q[k-1] +

expression_segment_select2q_diff[k] - 6

expression_segment_intensity1q[k] = expression_segment_intensity1q[k-1] +

expression_segment_intensity1q_diff[k] - 63

expression_segment_intensity2q[k] = expression_segment_intensity2q[k-1] +

expression_segment_intensity2q_diff[k] - 63

1. **Decoding of the viseme parameter fap 1**

Fourteen visemes have been defined for selection by the Viseme Parameter FAP 1, the definition is given in annex C. The viseme parameter allows two visemes from a standard set to be blended together. The viseme parameter is composed of a set of values as follows.

Table -17 -- Viseme parameter range

viseme () {	**Range**
viseme_select1	0-14
viseme_select2	0-14
viseme_blend	0-63
viseme_def	0-1
}	

Viseme_blend is quantized (step size = 1) and defines the blending of viseme1 and viseme2 in the decoder by the following symbolic expression where viseme1 and 2 are graphical interpretations of the given visemes as suggested in the non-normative annex.

final viseme = (viseme 1) * (viseme_blend / 63) + (viseme 2) * (1 - viseme_blend / 63)

The viseme can only have impact on FAPs that are currently allowed to be interpolated.

If the viseme_def bit is set, the current mouth FAPs can be used by the decoder to define the selected viseme in terms of a table of FAPs. This FAP table can be used when the same viseme is invoked again later for FAPs which must be interpolated.

2. **Decoding of the viseme parameter fap 2**

The expression parameter allows two expressions from a standard set to be blended together.The expression parameter is composed of a set of values as follows.

Table -18 -- Expression parameter range

expression () {	**Range**
expression_select1	0-6
expression_intensity1	0-63
expression_select2	0-6
expression_intensity2	0-63
init_face	0-1
expression_def	0-1
}	

Expression_intensity1 and expression_intensity2 are quantized (step size = 1) and define excitation of expressions 1 and 2 in the decoder by the following equations where expressions 1 and 2 are graphical interpretations of the given expression as suggested by the non-normative reference:

final expression = expression1 * (expression_intensity1 / 63)+ expression2 * (expression_intensity2 / 63)

The decoder displays the expressions according to the above fomula as a superposition of the 2 expressions.

The expression can only have impact on FAPs that are currently allowed to be interpolated. If the init_face bit is set, the neutral face may be modified within the neutral face constraints of mouth closure, eye opening, gaze direction, and head orientation before FAPs 3-68 are applied. If the expression_def bit is set, the current FAPs can be used to define the selected expression in terms of a table of FAPs. This FAP table can then be used when the same expression is invoked again later.

3. Fap masking

The face is animated by sending a stream of facial animation parameters. FAP masking, as indicated in the bitstream, is used to select FAPs. FAPs are selected by using a two level mask hierarchy. The first level contains two bit code for each group indicating the following options:

1. no FAPs are sent in the group.
2. a mask is sent indicating which FAPs in the group are sent. FAPs not selected by the group mask retain their previous value if any previously set value (not interpolated by decoder if previously set)
3. a mask is sent indicating which FAPs in the group are sent. FAPs not selected by the group mask retain must be interpolated by the decoder.
4. all FAPs in the group are sent.

1. Output of the decoding process

This subclause describes the output of the theoretical model of the decoding process that decodes bitstreams conforming to this part of ISO/IEC 14496.

The visual decoding process input is one or more coded visual bitstreams (one for each of the layers). The visual layers are generally multiplexed by the means of a system stream that also contains timing information.

1. Video data

The output of the video decoding process is a series of VOPs that are normally the input of a display process. The order in which fields or VOPs are output by the decoding process is called the display order, and may be different from the coded order (when B-VOPs are used).

2. 2D Mesh data

The output of the decoding process is a series of one or more mesh object planes. The mesh object planes are normally input to a compositor that maps the texture of a related video object or still texture object onto each mesh. The coded order and the composited order of the mesh object planes are identical.

3. Face animation parameter data

The output of the decoding process is a sequence of facial animation parameters. They are input to a display process that uses the parameters to animate a face object.

1. Visual-Systems Composition Issues

1. Temporal Scalability Composition

Background composition is used in forming the background region for objects at the enhancement layer of temporal scalability when the value of both enhancement_type and background_composition is one. This process is useful when the enhancement VOP corresponds to the partial region of the VOP belonging to the reference layer. In this process, the background of a current enhancement VOP is composed using the previous and the next VOPs in display order belonging to the reference layer.

Figure 8-1 shows the background composition for the current frame at the enhancement layer. The dotted line represents the shape of the selected object at the previous VOP in the reference layer (called "forward shape"). As the object moves, its shape at the next VOP in the reference layer is represented by a broken line (called "backward shape").

For the region outside these shapes, the pixel value from the nearest VOP at the reference layer is used for the composed background. For the region occupied only by the forward shape, the pixel value from the next VOP at the reference layer is used for the composed frame. This area is shown as lightly shaded in Figure 8-1. On the other hand, for the region occupied only by the backward shape, pixel values from the previous VOP in the reference layer are used. This is the area shaded dark in Figure 8-1. For the region where the areas enclosed by these shapes overlap, the pixel value is given by padding from the surrounding area. The pixel value which is outside of the overlapped area should be filled before the padding operation.

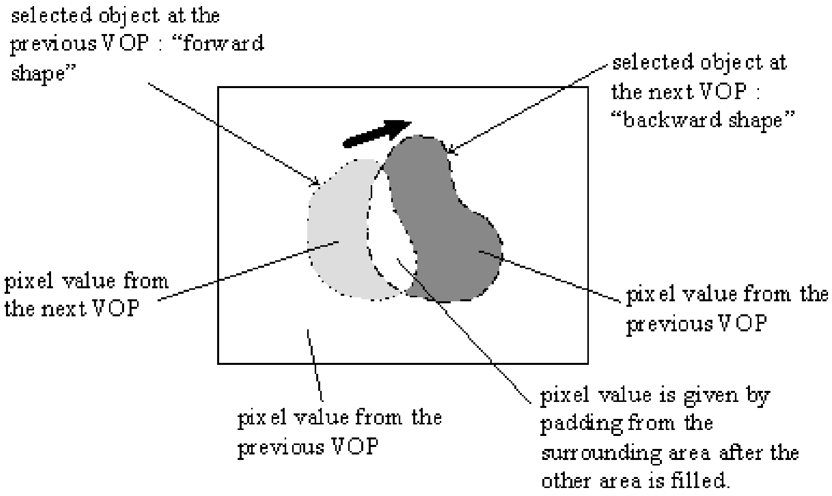

selected object at the previous VOP : "forward shape"
selected object at the next VOP : "backward shape"
pixel value from the next VOP
pixel value from the previous VOP
pixel value from the previous VOP
pixel value is given by padding from the surrounding area after the other area is filled.

Figure -1 -- Background composition

The following process is a mathematical description of the background composition method.

If s(x,y,ta)=0 and s(x,y,td)=0

fc(x,y,t) = f(x,y,td) (|t-ta|>|t-td|)

fc(x,y,t) = f(x,y,ta) (otherwise),

if s(x,y,ta)=1 and s(x,y,td)=0

fc(x,y,t) = f(x,y,td)

if s(x,y,ta)=0 and s(x,y,td)=1

fc(x,y,t) = f(x,y,ta)

if s(x,y,ta)=1 and s(x,y,td)=1

The pixel value of fc(x,y,t) is given by repetitive padding from the surrounding area.

where

fc composed background

f decoded VOP at the reference layer

s shape information (alpha plane) , 0: transparent, 1: opaque

(x,y) the spatial coordinate

t time of the current VOP

ta time of the previous VOP

td time of the next VOP

Two types of shape information, $s(x, y, ta)$ and $s(x, y, td)$, are necessary for the background composition. $s(x, y, ta)$ is called a "forward shape" and $s(x, y, td)$ is called a "backward shape". If $f(x, y, td)$ is the last VOP in the bitstream of the reference layer, it should be made by copying $f(x, y, ta)$. In this case, two shapes $s(x, y, ta)$ and $s(x, y, td)$ should be identical to the previous backward shape.

2. **Sprite Composition**

   The static sprite technology enables to encode very efficiently video objects which content is expected not to vary in time along a video sequence. For example, it is particularly well suited to represent backgrounds of scenes (decor, landscapes) or logos.

   A static sprite (sometimes referred as mosaic in the literature) is a frame containing spatial information for a single object, obtained by gathering information for this object throughout the sequence in which it appears. A static sprite can be a very large frame: it can correspond for instance to a wide angle view of a panorama.

   The ISO/IEC 14496-2 syntax defines a dedicated coding mode to obtain VOPs from static sprites: the so-called "S-VOPs". S-VOPs are extracted from a static sprite using a warping operation consisting in a global spatial transformation driven by few motion parameters (0,2,4, 6 or 8).

   For composition with other VOPs, there are no special rules for S-VOPs. However, it is classical to use S-VOPs as background objects over which "classical" objects are superimposed.

3. **Mesh Object Composition**

A Mesh Object represents the geometry of a sequence of 2D triangular meshes. This data can be used along with separately coded image texture data to render texture-mapped images, e.g., by the composition process as defined in ISO/IEC 14496-1. A Mesh Object stream may be contained in part of a BIFS animation stream, as defined in ISO/IEC 14496-1. In terminals implementing mesh animation functionality using both ISO/IEC 14496-1 and this part of ISO/IEC 14496, the decoded mesh data is used to update the appropriate fields of a BIFS IndexedFaceSet2D node, defined in ISO/IEC 14496-1, for composition purposes. In this case, the appropriate fields of the IndexedFaceSet2D BIFS node are updated as described in the following.

a) The coordinates of the mesh points (vertices) are obtained from the output of the Mesh Object decoder. The Mesh Object uses a pixel-based local coordinate system with *x*-axis pointing to the right and *y*-axis pointing down. However, ISO/IEC 14496-1 specifies a coordinate system with *y*-axis pointing up. Therefore, a simple coordinate transform shall be applied to the *y*-coordinates of mesh points to ensure the proper orientation of the object after composition. The *y*-coordinate y_n of a decoded mesh node point *n* shall be transformed as follows:

$$Y_n = -y_n ,$$

where Y_n is the *y*-coordinate of this mesh node point in the coordinate system as specified in ISO/IEC 14496-1. The origin of this object is at the top-left point. The same transform shall be applied to the coordinates of node points of each Mesh Object Plane (MOP).

b) The coordinate indices are the indices of the mesh points forming faces (triangles) obtained from the output of the Mesh Object decoder. All decoded faces are triangles. The topology of a Mesh Object is constant starting from an intra-coded MOP, throughout a sequence of predictive-coded MOPs (until the next intra-coded MOP); therefore, the coordinate indices shall be updated only for intra-coded MOPs.

c) Texture coordinates for mapping textures onto the mesh geometry are computed from the decoded node point locations of an intra-coded Mesh Object Plane and its bounding rectangle. Let $x_{\min}$, $y_{\min}$ and $x_{\max}$, $y_{\max}$ define the bounding rectangle of all node points of an intra-coded MOP. Then the width w and height h of the texture map shall be:

$$w = \mathrm{ceil}(x_{\max}) - \mathrm{floor}(x_{\min}) \,,$$

$$h = \mathrm{ceil}(y_{\max}) - \mathrm{floor}(y_{\min}) \,.$$

A texture coordinate pair (s_n, t_n) is computed for each node point $p_n = (x_n,y_n)$ as follows:

$$s_n = (x_n - \mathrm{floor}(x_{\min}))/w \,,$$

$$t_n = 1.0 - (y_n - \mathrm{floor}(y_{\min}))/h \,.$$

The topology of a Mesh Object is constant starting from an intra-coded MOP, throughout a sequence of predictive-coded MOPs (until the next intra-coded MOP); therefore, the texture coordinates shall be updated only for intra-coded MOPs.

d) The texture coordinate indices are identical to the coordinate indices.

2. Profiles and Levels

NOTE In this part of ISO/IEC 14496 the word "profile" is used as defined below. It should not be confused with other definitions of "profile" and in particular it does not have the meaning that is defined by ISO/IEC JTC1/SGFS.

Profiles and levels provide a means of defining subsets of the syntax and semantics of this part of ISO/IEC 14496 and thereby the decoder capabilities required to decode a particular bitstream. A profile is a defined sub-set of the entire bitstream syntax that is defined by this part of ISO/IEC 14496. A level is a defined set of constraints imposed on parameters in the bitstream. Conformance tests will be carried out against defined profiles at defined levels.

The purpose of defining conformance points in the form of profiles and levels is to facilitate bitstream interchange among different applications. Implementers of this part of ISO/IEC 14496 are encouraged to produce decoders and bitstreams which correspond to those defined conformance regions. The discretely defined profiles and levels are the means of bitstream interchange between applications of this part of ISO/IEC 14496.

In this clause the constrained parts of the defined profiles and levels are described. All syntactic elements and parameter values which are not explicitly constrained may take any of the possible values that are allowed by this part of ISO/IEC 14496. In general, a decoder shall be deemed to be conformant to a given profile at a given level if it is able to properly decode all allowed values of all syntactic elements as specified by that profile at that level.

1. Visual Object Types

The following table lists the tools included in each of the Object Types. Bitstreams that represent a particular object corresponding to an Object Type shall not use any of the tools for which the table does not have an ?X?.

Table -1 -- Tools and Visual Object Types

	Visual Object Types								
Visual Tools	**Simple**	**Core**	**Main**	**Simple Scalable**	**N-bit**	**Animated 2D Mesh**	**Basic Animated Texture**	**Still Scalable Texture**	**Simple Face**
Basic ● I-VOP, P-VOP ● AC/DC Prediction ● 4-MV, Unrestricted MV	X	X	X	X	X	X			
Error resilience ● Slice Resynchronization ● Data Partitioning ● Reversible VLC	X	X	X	X	X	X			
Short Header	X	X	X		X	X			
B-VOP		X	X	X	X	X			
P-VOP with OBMC (Texture)									
Method 1/Method 2 Quantization		X	X		X	X			
P-VOP based temporal scalability ● Rectangular ● Arbitrary Shape		X	X		X	X			
Binary Shape		X	X		X	X	X		
Grey Shape			X						
Interlace			X						
Sprite			X						
Temporal Scalability (Rectangular)				X					
Spatial Scalability (Rectangular)				X					
N-Bit					X				
Scalable Still Texture						X	X	X	
2D Dynamic Mesh with uniform topology						X	X		
2D Dynamic Mesh with Delaunay topology						X			
Facial Animation Parameters									X

NOTE 1 "Binary Shape Coding" includes constant alpha.

NOTE 2 The parameters are restricted as follows for the tool "P-VOP based temporal scalability Arbitrary Shape":

- ref_select_code shall be either ?00? or ?01?.
- reference layer shall be either I-VOP or P-VOP.
- load_backward_shape shall be ?0? and background composition is not performed.

1. **Visual Profiles**

   Decoders that conform to a Profile shall be able to decode all objects that comply to the Object Types for which the table lists an ?X?.

Table -2 -- Visual Profiles

	Object Types **Profiles**	**Simple**	**Core**	**Main**	**Simple Scalable**	**N-Bit**	**Animated 2D Mesh**	**Basic Animated Texture**	**Scalable Texture**	**Simple Face**
1.	Simple	X								
2.	Simple Scaleable	X			X					
3.	Core	X	X							
4.	Main	X	X	X					X	
5.	N-Bit	X	X			X				
6.	Hybrid	X	X				X	X	X	X
7.	Basic Animated Texture							X	X	X
8.	Scaleable Texture								X	
9.	Simple FA									X

Note that the Profiles can be grouped into three categories: Natural Visual (Profile numbers 1-5), Synthetic Visual (Profile numbers 8 and 9), and Synthetic/Natural Hybrid Visual (Profile numbers 6 and 7).

2. **Visual Profiles@Levels**
   1. **Natural Visual**

      The table that describes the natural visual profiles is given in annex N.

   2. **Synthetic Visual**
      1. **Scalable Texture Profile**

Table -3 -- Scalable texture profile levels

Profile	**Levels**	**Default Wavelet Filter**	**Download Filter, length**	**Maximum number of Decomposition Levels**	**Typical Visual Session Size**[1]	**Maximum Qp value**	**M n p S**
Scalable Texture	L3	Float, Integer	ON, 24	10	8192x8192	12 bits	6
Scalable Texture	L2	Integer	ON, 18	8	2048x2048	10 bits	4
Scalable Texture	L1	Integer	OFF	5	704x576	8 bits	4

(1) This column is for informative use only. It provides an example configuration of the Maximum number of pixels/Session.

2. Simple Face Animation Profile

All ISO/IEC 14496-2 facial animation decoders (for all object types) are required to generate at their output a facial model including all the feature points defined in this part of ISO/IEC 14496, even if some of the features points will not be affected by any information received from the encoder.

The Simple Face object is not required to implement the viseme_def/expression_def functionality.

Level 1:

- number of objects: 1,
- The total FAP decode frame-rate in the bitstream shall not exceed 72 Hz ,
- The decoder shall be capable of a face model rendering update of at least 15 Hz, and
- Maximum bitrate 16 kbit/s.

Level 2:

- maximum number of objects: 4,
- The FAP decode frame-rate in the bitstream shall not exceed 72 Hz (this means that the FAP decode framerate is to be shared among the objects),
- The decoder shall be capable of rendering the face models with the update rate of at least 60Hz, sharable between faces, with the constraint that the update rate for each individual face is not required to exceed 30Hz, and
- Maximum bitrate 32 kbit/s.

1. Synthetic/Natural Hybrid Visual

The *Levels* of the Profiles which support both Natural Visual Object Types and Synthetic Visual Object Types are specified by giving bounds for the natural objects and for the synthetic objects. Parameters like bitrate can be combined across natural and synthetic objects.

1. Basic Animated Texture Profile

Level 1 = Simple Facial Animation Profile @ Level 1 + Scalable Texture @ Level 1 + the following restrictions on Basic Animated Texture object types:

- Maximum number of Mesh objects (with uniform topology): 4,
- Maximum total number of nodes (vertices) in Mesh objects: 480,
  (= 4 x nr. of nodes of a uniform mesh covering a QCIF image with 16x16 pixel elements),
- Maximum frame-rate of a Mesh object: 30 Hz, and
- Maximum bitrate of Mesh objects: 128 kbit/sec.

Level 2 = Simple Facial Animation Profile @ Level 2 + Scalable Texture @ Level 2 + the following restrictions on Basic Animated Texture object types:

- Maximum number of Mesh objects (with uniform topology): 8,
- Maximum total number of nodes (vertices) in Mesh objects: 1748,
  (= 4 x nr. of nodes of a uniform mesh covering a CIF image with 16x16 pixel elements),
- Maximum frame-rate of a Mesh object: 60 Hz, and
- Maximum bitrate of Mesh objects: 128 kbit/sec.

1. Hybrid Profile

Level 1 = Core Visual Profile @ Level 1 + Basic Animated Texture Profile @ Level 1 + the following restrictions on Animated 2D Mesh object types:

- Maximum number of Mesh objects (with uniform or Delaunay topology): 4
  (= maximum number of objects in visual session)
- Maximum total number of nodes (vertices) in Mesh objects: 480
  (= 4 x nr. of nodes of a uniform mesh covering a QCIF image with 16x16 pixel elements)
- Maximum frame-rate of a Mesh object: 30 Hz
  (= maximum frame-rate of video object)
- Maximum bitrate of Mesh objects: 64 kbit/sec.

Level 2 = Core Visual Profile @ Level 2 + Basic Animated Texture Profile @ Level 2 + the following restrictions on Animated 2D Mesh object types:

- Maximum number of Mesh objects(with uniform or Delaunay topology): 8
  (= maximum number of objects in visual session)
- Maximum total number of nodes (vertices) in Mesh objects: 1748
  (= 4 x nr. of nodes of a uniform mesh covering a CIF image with 16x16 pixel elements)
- Maximum frame-rate of a Mesh object: 60 Hz
  (= 2 x the maximum frame-rate of video object)
- Maximum bitrate of Mesh objects: 128 kbit/sec.

A. (normative)

Coding transforms

1. Discrete cosine transform for video texture

The NxN two dimensional DCT is defined as:

$$F(u,v) = \frac{2}{N} C(u) C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N}$$

with u, v, x, y = 0, 1, 2, ¼ N-1

where x, y are spatial coordinates in the sample domain

u, v are coordinates in the transform domain

$$C(u), C(v) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } u, v = 0 \\ 1 & \text{otherwise} \end{cases}$$

The inverse DCT (IDCT) is defined as:

$$f(x, y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u) C(v) F(u, v) \cos \frac{(2x+1)u\pi}{2N} \cos \frac{(2y+1)v\pi}{2N}$$

If each pixel is represented by n bits per pixel, the input to the forward transform and output from the inverse transform is represented with (n+1) bits. The coefficients are represented in (n+4) bits. The dynamic range of the DCT coefficients is $[-2^{n+3}:+2^{n+3}-1]$.

The N by N inverse discrete transform shall conform to IEEE Standard Specification for the Implementations of 8 by 8 Inverse Discrete Cosine Transform, Std 1180-1990, December 6, 1990.

NOTE 1 lause 2.3 Std 1180-1990 "Considerations of Specifying IDCT Mismatch Errors" requires the specification of periodic intra-picture coding in order to control the accumulation of mismatch errors. Every macroblock is required to be refreshed before it is coded 132 times as predictive macroblocks. Macroblocks in B-pictures (and skipped macroblocks in P-pictures) are excluded from the counting because they do not lead to the accumulation of mismatch errors. This requirement is the same as indicated in 1180-1990 for visual telephony according to ITU-T Recommendation H.261.

NOTE 2 Whilst the IEEE IDCT standard mentioned above is a necessary condition for the satisfactory implementation of the IDCT function it should be understood that this is not sufficient. In particular attention is drawn to the following sentence from subclause 5.4: "Where arithmetic precision is not specified, such as the calculation of the IDCT, the precision shall be sufficient so that significant errors do not occur in the final integer values."

2. Discrete wavelet transform for still texture

1. Adding the mean

Before applying the inverse wavelet transform, the mean of each color component ("mean_y", "mean_u", and "mean_v") is added to the all wavelet coefficients of dc band.

2. wavelet filter

A 2-D separable inverse wavelet transfrom is used to synthesize the still texture. The default wavelet composition is performed using Daubechies (9,3) tap biorthogonal filter bank. The inverse DWT is performed either in floating or integer operations depending on the field "wavelet_filter_type", defined in the syntax.

The floating filter coefficients are:

Lowpass	**g[ ] =**	
[0.35355339059327	0.70710678118655	0.35355339059327]

Highpas	**h[ ] =**	
[0.03314563036812	0.06629126073624	-0.17677669529665
-0.41984465132952	0.99436891104360	-0.41984465132952
-0.17677669529665	0.06629126073624	0.03314563036812]

The integer filter coefficients are:

Lowpass	**g[ ] =**	
32	64	32

Highpass	**h[ ] =**	
3	6	-16
-38	90	-38
-16	6	3

The synthesis filtering operation is defined as follows:

1 4

y[n] = å L[n+i]*g[i+1] + å H[n+i]*h[i+4]

i=-1 i=-4

where

- n = 0, 1, ... N-1, and N is the number of output points;
- L[2*i] = xl[i] and L[2*i+1] = 0 for i=0,1,...,N/2-1, and {xl[i]} are the N/2 input wavelet coefficients in the low-pass band;
- H[2*i+1] = xh[i] and H[2*i] = 0 for i=0,1,...,N/2-1, and {xh[i]} are the N/2 input wavelet coefficients in the high-pass band.

NOTE 1 the index range for h[] is from 0 to 8;

NOTE 2 the index range for g[] is from 0 to 2;

NOTE 3 the index range for L[] is from -1 to N;

NOTE 4 the index range for H[] is from -4 to N+3; and

NOTE 5 the values of L[] and H[] for indexes less than 0 or greater than N-1 are obtained by symmetric extension described in the following subclause.

In the case of integer wavelet, the outputs at each composition level are scaled down with dividing by 8096 with rounding to the nearest integer.

1. **Symmetric extension**

A symmetric extension of the input wavelet coefficients is performed and the up-sampled and extended wavelet coefficients are generated. Note that the extension process shown below is an example when the extension is performed before up-sampling and that only the generated coefficients are specified. Two types of symmetric extensions are needed, both mirror the boundary pixels. Type A replicates the edge pixel and Type B does not replicate the edge pixel. This is illustrated in Figure A-1 and Figure A-2, where the edge pixel is indicated by z. The types of extension for the input data to the wavelet filters are shown in Table A-1.

Type A **?v w x y z** | z y x w v?

Type B **?...v w x y** | z y x w v?

Figure -1 -- Symmetrical extensions at leading boundary

Type A ?v w x y z | **z y x w v?**

Type B .?v w x y z | **y x w v?.**

Figure -2 -- Symmetrical extensions at the trailing boundary

Table -1 -- Extension method for the input data to the synthesis filters

	boundary	**Extension**
lowpass input xl[]	leading	TypeB
to 3-tap filter g[]	trailing	TypeA
highpass input xh[]	leading	TypeA
to 9-tap filter h[]	trailing	TypeB

The generated up-sampled and extended wavelet coefficients L[] and H[] are eventually specified as follows:

low-pass band: ? 0 L[2] 0 | **L[0] 0 L[2] 0 ? L[N-4] 0 L[N-2] 0 |** L[N-2] 0 L[N-4] 0 ?

high-pass band: ? H[3] 0 H[1] | **0 H[1] 0 H[3] ? 0 H[N-3] 0 H[N-1]** | 0 H[N-1] 0 H[N-3]?

2. **Decomposition level**

The number of decomposition levels of the luminance component is defined in the input bitstream. The number of decompostion levels for the chrominance components is one level less than that of the luminance components. If texture_object_layer width or texture_object_layer height cannot be divisible by (2 ^ decomposition_levels), then shape adaptive wavelet is applied.

3. **Shape adaptive wavelet filtering and symmetric extension**
    1. **Shape adaptive wavelet**

The 2-D inverse shape adaptive wavelet transform uses the same wavelet filter as specified in Table A-1. According to the shape information, segments of consecutive output points are reconstructed and put into the correct locations. The filtering operation of shape adaptive wavelet is a generalization of that for the regular wavelet. The generalization allows the number of output points to be an odd number as well as an even number. Relative to the bounding rectangle, the starting point of the output is also allowed to be an odd number as well as an even number according to the shape information. Within the generalized wavelet filtering, the regular wavelet filtering is a special case where the number of output points is an even number and the starting point is an even number (0) too. Another special case is for reconstruction of rectangular textures with an arbitrary size where the number of output points may be even or odd and the starting point is always even (0).

The same synthesis filtering is applied for shape-adaptive wavelet composition, i.e:

1 4

y[n] = å L[n+i]*g[i+1] + å H[n+i]*h[i+4]

i=-1 i=-4

where

- n = 0, 1, ... N-1, and N is the number of output points;
- L[2*i+s] = xl[i] and L[2*i+1-s] = 0 for i=0,1,...,(N+1-s)/2-1, and {xl[i]} are the (N+1-s)/2 input wavelet coefficients in the low-pass band;
- H[2*i+1-s] = xh[i] and H[2*i+s] = 0 for i=0,1,...,(N+s)/2-1, and {xh[i]} are the (N+s)/2 input wavelet coefficients in the high-pass band.

The only difference from the regular synthesis filtering is to introduce a binary parameter s in up-sampling, where s = 0 if the starting point of the output is an even number and s = 1 if the starting point of the output is an odd number.

The symmetric extension for the generalized synthesis filtering is specified in Table A-2 if N is an even number and in Table A-3 if N is an odd number.

Table -2 -- Extension method for the data to the synthesis wavelet filters if N is even

	Boundary	**extension (s=0)**	**extension(s=1)**
lowpass input xl[]	Leading	TypeB	TypeA
to 3-tap filter g[]	Trailing	TypeA	TypeB
highpass input xh[]	Leading	TypeA	TypeB
to 9-tap filter h[]	Trailing	TypeB	TypeA

Table -3 -- Extension method for the data to the synthesis wavelet filters if N is odd

	Boundary	**extension(s=0)**	**extension(s=1)**
lowpass input xl[]	Leading	TypeB	TypeA
to 3-tap filter g[]	Trailing	TypeB	TypeA
highpass input xh[]	Leading	TypeA	TypeB
to 9-tap filter h[]	Trailing	TypeA	TypeB

A. (normative)

Variable length codes and arithmetic decoding

1. Variable length codes

1. Macroblock type

Table -1 -- Macroblock types and included data elements for I- and P-VOPs in combined motion-shape-texture coding

VOP type	**mb type**	**Name**	**not_coded**	**mcbpc**	**cbpy**	**dquant**	**mvd**	**mvd$_{2-4}$**
P	not coded	-	1					
P	0	inter	1	1	1		1	
P	1	inter+q	1	1	1	1	1	
P	2	inter4v	1	1	1		1	1
P	3	intra	1	1	1			
P	4	intra+q	1	1	1	1		
P	stuffing	-	1	1				
I	3	intra		1	1			
I	4	intra+q		1	1	1		
I	stuffing	-		1				
S (update)	not_coded	-	1					
S (update)	0	inter	1	1	1			
S (update)	1	inter+q	1	1	1	1		
S (update)	3	intra	1	1	1			
S (update)	4	intra+q	1	1	1	1		
S (update)	stuffing	-	1	1				
S (piece)	3	intra		1	1			
S (piece)	4	intra+q		1	1	1		
S (piece)	stuffing	-		1				

NOTE "1" means that the item is present in the macroblock
S (piece) indicates S-VOPs with low_latency_sprite_enable == 1 and sprite_transmit_mode == "piece"
S (update) indicates S-VOPs with low_latency_sprite_enable == 1 and sprite_transmit_mode == "update"

Table -2 -- Macroblock types and included data elements for a P-VOP (scalability && ref_select_code == ?11?)

VOP Type	mb_type	Name	not_coded	mcbpc	cbpy	dquant	MVD	MVI
P	not coded	-	1					
P	0	INTER	1	1	1			
P	1	INTER+Q	1	1	1	1		
P	3	INTRA	1	1	1			
P	4	INTRA+Q	1	1	1	1		
P	stuffing	-	1	1				
NOTE "1" means that the item is present in the macroblock								

Table -3 -- VLC table for modb in combined motion-shape-texture coding

Code	cbpb	mb_type
1		
01		1
00	1	1

Table -4 -- mb_type and included data elements in coded macroblocks in B-VOPs (ref_select_code != ?00?||scalability==?0?) for combined motion-shape-texture coding

Code	dquant	mvd_f	mvd_b	mvdb	mb_type
1				1	direct
01	1	1	1		interpolate mc+q
001	1		1		backward mc+q
0001	1	1			forward mc+q

Table -5 -- mb_type and included data elements in coded macroblocks in B-VOPs (ref_select_code == ?00?&&scalability!=?0?) for combined motion-shape-texture coding

Code	dquant	mvd_f	mvd_b	mb_type
01	1	1		interpolate mc+q
001	1			backward mc+q
1	1	1		forward mc+q

2. **Macroblock pattern**

Table -6 -- VLC table for mcbpc for I-VOPs in combined-motion-shape-texture coding and S-VOPs with low_latence_sprite_enable==1 and sprite_transmit_mode=="piece"

Code	mbtype	cbpc (56)
1	3	00
001	3	01
010	3	10
011	3	11
0001	4	00
0000 01	4	01
0000 10	4	10
0000 11	4	11
0000 0000 1	Stuffing	--

Table -7 -- VLC table for mcbpc for P-VOPs in combined-motion-shape-texture and S-VOPs with low_latence_sprite_enable==1 and sprite_transmit_mode=="update"

Code	MB type	cbpc (56)
1	0	00
0011	0	01
0010	0	10
0001 01	0	11
011	1	00
0000 111	1	01
0000 110	1	10
0000 0010 1	1	11
010	2	00
0000 101	2	01
0000 100	2	10
0000 0101	2	11
0001 1	3	00
0000 0100	3	01
0000 0011	3	10
0000 011	3	11
0001 00	4	00
0000 0010 0	4	01
0000 0001 1	4	10
0000 0001 0	4	11
0000 0000 1	Stuffing	--

Table -8 -- VLC table for cbpy in the case of four non-transparent macroblocks

Code	cbpy(intra-MB) (12 34)	cbpy(inter-MB), (12 34)
0011	00 00	11 11
0010 1	00 01	11 10

0010 0	00 10	11 01
1001	00 11	11 00
0001 1	01 00	10 11
0111	01 01	10 10
0000 10	01 10	10 01
1011	01 11	10 00
0001 0	10 00	01 11
0000 11	10 01	01 10
0101	10 10	01 01
1010	10 11	01 00
0100	11 00	00 11
1000	11 01	00 10
0110	11 10	00 01
11	11 11	00 00

Table -9 -- VLC table for cbpy in the case of three non transparent blocks

Code	cbpy (intra-MB)	cbpy (inter-MB)
011	000	111
000001	001	110
00001	010	101
010	011	100
00010	100	011
00011	101	010
001	110	001
1	111	000

Table -10 -- VLC table for cbpy in the case of two non transparent blocks

Code	cbpy (intra-MB)	cbpy (inter-MB)
0001	00	11
001	01	10
01	10	01
1	11	00

Table -11 -- VLC table for cbpy in the case of one non transparent block

Code	cbpy (intra-MB)	cbpy (inter-MB)
01	0	1
1	1	0

3. **Motion vector**

Table -12 -- VLC table for MVD

Codes	Vector differences
0000 0000 0010 1	-16
0000 0000 0011 1	-15.5
0000 0000 0101	-15

0000 0000 0111	-14.5
0000 0000 1001	-14
0000 0000 1011	-13.5
0000 0000 1101	-13
0000 0000 1111	-12.5
0000 0001 001	-12
0000 0001 011	-11.5
0000 0001 101	-11
0000 0001 111	-10.5
0000 0010 001	-10
0000 0010 011	-9.5
0000 0010 101	-9
0000 0010 111	-8.5
0000 0011 001	-8
0000 0011 011	-7.5
0000 0011 101	-7
0000 0011 111	-6.5
0000 0100 001	-6
0000 0100 011	-5.5
0000 0100 11	-5
0000 0101 01	-4.5
0000 0101 11	-4
0000 0111	-3.5
0000 1001	-3
0000 1011	-2.5
0000 111	-2
0001 1	-1.5
0011	-1
011	-0.5
1	0
010	0.5
0010	1

0001 0	1.5
0000 110	2
0000 1010	2.5
0000 1000	3
0000 0110	3.5
0000 0101 10	4
0000 0101 00	4.5
0000 0100 10	5
0000 0100 010	5.5
0000 0100 000	6
0000 0011 110	6.5
0000 0011 100	7
0000 0011 010	7.5
0000 0011 000	8
0000 0010 110	8.5
0000 0010 100	9
0000 0010 010	9.5
0000 0010 000	10
0000 0001 110	10.5
0000 0001 100	11
0000 0001 010	11.5
0000 0001 000	12
0000 0000 1110	12.5
0000 0000 1100	13
0000 0000 1010	13.5
0000 0000 1000	14
0000 0000 0110	14.5
0000 0000 0100	15
0000 0000 0011 0	15.5
0000 0000 0010 0	16

4. DCT coefficients

Table -13 -- Variable length codes for dct_dc_size_luminance

Variable length code	**dct_dc_size_luminance**
011	0
11	1
10	2
010	3
001	4
0001	5
0000 1	6
0000 01	7
0000 001	8
0000 0001	9
0000 0000 1	10
0000 0000 01	11
0000 0000 001	12

Table -14 -- Variable length codes for dct_dc_size_chrominance

Variable length code	**dct_dc_size_chrominance**
11	0
10	1
01	2
001	3
0001	4
0000 1	5
0000 01	6
0000 001	7
0000 0001	8
0000 0000 1	9
0000 0000 01	10
0000 0000 001	11
0000 0000 0001	12

Table -15 -- Differential DC additional codes

Additional code	Differential DC	Size
000000000000 to 011111111111 *	-2048 to -4095	12
00000000000 to 01111111111 *	-1024 to -2047	11
0000000000 to 0111111111 *	-512 to -1023	10
000000000 to 011111111 *	-256 to -511	9
00000000 to 01111111	-255 to -128	8
0000000 to 0111111	-127 to -64	7
000000 to 011111	-63 to -32	6
00000 to 01111	-31 to -16	5
0000 to 0111	-15 to -8	4
000 to 011	-7 to -4	3
00 to 01	-3 to -2	2
0	-1	1
	0	0
1	1	1
10 to 11	2 to 3	2
100 to 111	4 to 7	3
1000 to 1111	8 to 15	4
10000 to 11111	16 to 31	5
100000 to 111111	32 to 63	6
1000000 to 1111111	64 to 127	7
10000000 to 11111111	128 to 255	8
100000000 to 111111111 *	256 to 511	9
1000000000 to 1111111111 *	512 to 1023	10
10000000000 to 11111111111 *	1024 to 2047	11
100000000000 to 111111111111 *	2048 to 4095	12

In cases where dct_dc_size is greater than 8, marked ?*? in , a marker bit is inserted after the dct_dc_additional_code to prevent start code emulations.

Table -16 -- VLC Table for Intra Luminance and Chrominance TCOEF

VLC CODE	LAST	RUN	LEVEL		VLC CODE	LAST	RUN	LEVEL
10s	0	0	1		0111 s	1	0	1

1111 s	0	0	3		0000 1100 1s	0	11	1
0101 01s	0	0	6		0000 0000 101s	1	0	6
0010 111s	0	0	9		0011 11s	1	1	1
0001 1111 s	0	0	10		0000 0000 100s	1	0	7
0001 0010 1s	0	0	13		0011 10s	1	2	1
0001 0010 0s	0	0	14		0011 01s	0	5	1
0000 1000 01s	0	0	17		0011 00s	1	0	2
0000 1000 00s	0	0	18		0010 011s	1	5	1
0000 0000 111s	0	0	21		0010 010s	0	6	1
0000 0000 110s	0	0	22		0010 001s	1	3	1
0000 0100 000s	0	0	23		0010 000s	1	4	1
110s	0	0	2		0001 1010 s	1	9	1
0101 00s	0	1	2		0001 1001 s	0	8	1
0001 1110 s	0	0	11		0001 1000 s	0	9	1
0000 0011 11s	0	0	19		0001 0111 s	0	10	1
0000 0100 001s	0	0	24		0001 0110 s	1	0	3
0000 0101 0000s	0	0	25		0001 0101 s	1	6	1
1110 s	0	1	1		0001 0100 s	1	7	1
0001 1101 s	0	0	12		0001 0011 s	1	8	1
0000 0011 10s	0	0	20		0000 1100 0s	0	12	1
0000 0101 0001s	0	0	26		0000 1011 1s	1	0	4
0110 1s	0	0	4		0000 1011 0s	1	1	2
0001 0001 1s	0	0	15		0000 1010 1s	1	10	1
0000 0011 01s	0	1	7		0000 1010 0s	1	11	1
0110 0s	0	0	5		0000 1001 1s	1	12	1

0001 0001 0s	0	4	2		0000 1001 0s	1	13	1
0000 0101 0010s	0	0	27		0000 1000 1s	1	14	1
0101 1s	0	2	1		0000 0001 11s	0	13	1
0000 0011 00s	0	2	4		0000 0001 10s	1	0	5
0000 0101 0011s	0	1	9		0000 0001 01s	1	1	3
0100 11s	0	0	7		0000 0001 00s	1	2	2
0000 0010 11s	0	3	4		0000 0100 100s	1	3	2

VLC CODE	LAST	RUN	LEVEL		VLC CODE	LAST	RUN	LEVEL
0000 0101 0100s	0	6	3		0000 0100 101s	1	4	2
0100 10s	0	0	8		0000 0100 110s	1	15	1
0000 0010 10s	0	4	3		0000 0100 111s	1	16	1
0100 01s	0	3	1		0000 0101 1000s	0	14	1
0000 0010 01s	0	8	2		0000 0101 1001s	1	0	8
0100 00s	0	4	1		0000 0101 1010s	1	5	2
0000 0010 00s	0	5	3		0000 0101 1011s	1	6	2
0010 110s	0	1	3		0000 0101 1100s	1	17	1
0000 0101 0101s	0	1	10		0000 0101 1101s	1	18	1
0010 101s	0	2	2		0000 0101 1110s	1	19	1
0010 100s	0	7	1		0000 0101 1111s	1	20	1

0001 1100 s	0	1	4		0000 011	escape		
0001 1011 s	0	3	2					
0001 0000 1s	0	0	16					
0001 0000 0s	0	1	5					
0000 1111 1s	0	1	6					
0000 1111 0s	0	2	3					
0000 1110 1s	0	3	3					
0000 1110 0s	0	5	2					
0000 1101 1s	0	6	2					
0000 1101 0s	0	7	2					
0000 0100 010s	0	1	8					
0000 0100 011s	0	9	2					
0000 0101 0110s	0	2	5					
0000 0101 0111s	0	7	3					

Table -17 -- VLC table for Inter Lumimance and Chrominance TCOEF

VLC CODE	LAST	RUN	LEVEL		VLC CODE	LAST	RUN	LEVEL
10s	0	0	1		0111 s	1	0	1
1111 s	0	0	2		0000 1100 1s	1	0	2
0101 01s	0	0	3		0000 0000 101s	1	0	3
0010 111s	0	0	4		0011 11s	1	1	1
0001 1111 s	0	0	5		0000 0000 100s	1	1	2
0001 0010 1s	0	0	6		0011 10s	1	2	1
0001 0010 0s	0	0	7		0011 01s	1	3	1

0000 1000 01s	0	0	8		0011 00s	1	4	1
0000 1000 00s	0	0	9		0010 011s	1	5	1
0000 0000 111s	0	0	10		0010 010s	1	6	1
0000 0000 110s	0	0	11		0010 001s	1	7	1
0000 0100 000s	0	0	12		0010 000s	1	8	1
110s	0	1	1		0001 1010 s	1	9	1
0101 00s	0	1	2		0001 1001 s	1	10	1
0001 1110 s	0	1	3		0001 1000 s	1	11	1
0000 0011 11s	0	1	4		0001 0111 s	1	12	1
0000 0100 001s	0	1	5		0001 0110 s	1	13	1
0000 0101 0000s	0	1	6		0001 0101 s	1	14	1
1110 s	0	2	1		0001 0100 s	1	15	1
0001 1101 s	0	2	2		0001 0011 s	1	16	1
0000 0011 10s	0	2	3		0000 1100 0s	1	17	1
0000 0101 0001s	0	2	4		0000 1011 1s	1	18	1
0110 1s	0	3	1		0000 1011 0s	1	19	1
0001 0001 1s	0	3	2		0000 1010 1s	1	20	1
0000 0011 01s	0	3	3		0000 1010 0s	1	21	1
0110 0s	0	4	1		0000 1001 1s	1	22	1
0001 0001 0s	0	4	2		0000 1001 0s	1	23	1
0000 0101 0010s	0	4	3		0000 1000 1s	1	24	1
0101 1s	0	5	1		0000 0001 11s	1	25	1
0000 0011 00s	0	5	2		0000 0001 10s	1	26	1
0000 0101 0011s	0	5	3		0000 0001 01s	1	27	1

0100 11s	0	6	1		0000 0001 00s	1	28	1
0000 0010 11s	0	6	2		0000 0100 100s	1	29	1
0000 0101 0100s	0	6	3		0000 0100 101s	1	30	1
0100 10s	0	7	1		0000 0100 110s	1	31	1
0000 0010 10s	0	7	2		0000 0100 111s	1	32	1
0100 01s	0	8	1		0000 0101 1000s	1	33	1
0000 0010 01s	0	8	2		0000 0101 1001s	1	34	1
0100 00s	0	9	1		0000 0101 1010s	1	35	1
0000 0010 00s	0	9	2		0000 0101 1011s	1	36	1
0010 110s	0	10	1		0000 0101 1100s	1	37	1
0000 0101 0101s	0	10	2		0000 0101 1101s	1	38	1
0010 101s	0	11	1		0000 0101 1110s	1	39	1
0010 100s	0	12	1		0000 0101 1111s	1	40	1
0001 1100 s	0	13	1		0000 011	escape		
0001 1011 s	0	14	1					
0001 0000 1s	0	15	1					
0001 0000 0s	0	16	1					
0000 1111 1s	0	17	1					
0000 1111 0s	0	18	1					
0000 1110 1s	0	19	1					
0000 1110 0s	0	20	1					
0000 1101 1s	0	21	1					

0000 1101 0s	0	22	1					
0000 0100 010s	0	23	1					
0000 0100 011s	0	24	1					
0000 0101 0110s	0	25	1					
0000 0101 0111s	0	26	1					

Table -18 -- FLC table for RUNS and LEVELS

Code	Run		Code	Level
000 000	0		forbidden	-2048
000 001	1		1000 0000 0001	-2047
000 010	2		.	.
.	.		1111 1111 1110	-2
.	.		1111 1111 1111	-1
111 111	63		forbidden	0
			0000 0000 0001	1
			0000 0000 0010	2
			.	.
			0111 1111 1111	2047

Table -19 -- ESCL(a), LMAX values of intra macroblocks

LAST	RUN	LMAX		LAST	RUN	LMAX
0	0	27		1	0	8
0	1	10		1	1	3
0	2	5		1	2-6	2
0	3	4		1	7-20	1
0	4-7	3		1	others	N/A
0	8-9	2				
0	10-14	1				
0	others	N/A				

Table -20 -- ESCL(b), LMAX values of inter macroblocks

LAST	RUN	LMAX		LAST	RUN	LMAX
0	0	12		1	0	3
0	1	6		1	1	2
0	2	4		1	2-40	1
0	3-6	3		1	others	N/A
0	7-10	2				
0	11-26	1				
0	others	N/A				

Table -21 -- ESCR(a), RMAX values of intra macroblocks

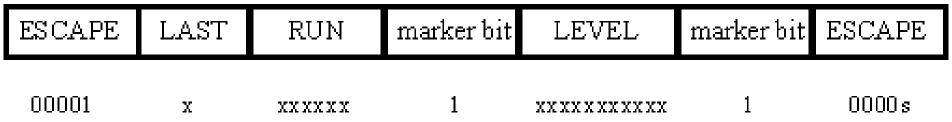
LAST	LEVEL	RMAX		LAST	LEVEL	RMAX
0	1	14		1	1	20
0	2	9		1	2	6
0	3	7		1	3	1
0	4	3		1	4-8	0
0	5	2		1	others	N/A
0	6-10	1				
0	11-27	0				
0	others	N/A				

Table -22 -- ESCR(b), RMAX values of inter macroblocks

LAST	LEVEL	RMAX		LAST	LEVEL	RMAX
0	1	26		1	1	40
0	2	10		1	2	1
0	3	6		1	3	0
0	4	2		1	others	N/A
0	5-6	1				
0	7-12	0				
0	others	N/A				

Table -23 -- RVLC table for TCOEF

ESCAPE code is added at the beginning and the end of these fixed-length codes for realizing two-way decode as shown below. A marker bit is inserted before and after the 11-bit-LEVEL in order to avoid the resync_marker emulation.



ESCAPE	LAST	RUN	marker bit	LEVEL	marker bit	ESCAPE
00001	x	xxxxxx	1	xxxxxxxxxxx	1	0000s

NOTE There are two types for ESCAPE added at the end of these fixed-length codes, and codewords are "0000s". Also, S=0 : LEVEL is positive and S=1 : LEVEL is negative.

		intra			inter			
INDEX	**LAST**	**RUN**	**LEVEL**	**LAST**	**RUN**	**LEVEL**	**BITS**	**VLC_CODE**
0	0	0	1	0	0	1	4	110s
1	0	0	2	0	1	1	4	111s
2	0	1	1	0	0	2	5	0001s
3	0	0	3	0	2	1	5	1010s
4	1	0	1	1	0	1	5	1011s
5	0	2	1	0	0	3	6	00100s
6	0	3	1	0	3	1	6	00101s
7	0	1	2	0	4	1	6	01000s
8	0	0	4	0	5	1	6	01001s
9	1	1	1	1	1	1	6	10010s
10	1	2	1	1	2	1	6	10011s
11	0	4	1	0	1	2	7	001100s
12	0	5	1	0	6	1	7	001101s
13	0	0	5	0	7	1	7	010100s
14	0	0	6	0	8	1	7	010101s
15	1	3	1	1	3	1	7	011000s
16	1	4	1	1	4	1	7	011001s
17	1	5	1	1	5	1	7	100010s
18	1	6	1	1	6	1	7	100011s
19	0	6	1	0	0	4	8	0011100s
20	0	7	1	0	2	2	8	0011101s
21	0	2	2	0	9	1	8	0101100s
22	0	1	3	0	10	1	8	0101101s
23	0	0	7	0	11	1	8	0110100s
24	1	7	1	1	7	1	8	0110101s
25	1	8	1	1	8	1	8	0111000s
26	1	9	1	1	9	1	8	0111001s
27	1	10	1	1	10	1	8	1000010s
28	1	11	1	1	11	1	8	1000011s
29	0	8	1	0	0	5	9	00111100s
30	0	9	1	0	0	6	9	00111101s
31	0	3	2	0	1	3	9	01011100s
32	0	4	2	0	3	2	9	01011101s
33	0	1	4	0	4	2	9	01101100s

34	0	1	5	0	12	1	9	01101101s
35	0	0	8	0	13	1	9	01110100s
36	0	0	9	0	14	1	9	01110101s
37	1	0	2	1	0	2	9	01111000s
38	1	12	1	1	12	1	9	01111001s
39	1	13	1	1	13	1	9	10000010s
40	1	14	1	1	14	1	9	10000011s
41	0	10	1	0	0	7	10	001111100s
42	0	5	2	0	1	4	10	001111101s
43	0	2	3	0	2	3	10	010111100s
44	0	3	3	0	5	2	10	010111101s
45	0	1	6	0	15	1	10	011011100s
46	0	0	10	0	16	1	10	011011101s
47	0	0	11	0	17	1	10	011101100s
48	1	1	2	1	1	2	10	011101101s
49	1	15	1	1	15	1	10	011110100s
50	1	16	1	1	16	1	10	011110101s
51	1	17	1	1	17	1	10	011111000s
52	1	18	1	1	18	1	10	011111001s
53	1	19	1	1	19	1	10	100000010s
54	1	20	1	1	20	1	10	100000011s
55	0	11	1	0	0	8	11	0011111100s
56	0	12	1	0	0	9	11	0011111101s
57	0	6	2	0	1	5	11	0101111100s
58	0	7	2	0	3	3	11	0101111101s
59	0	8	2	0	6	2	11	0110111100s
60	0	4	3	0	7	2	11	0110111101s
61	0	2	4	0	8	2	11	0111011100s
62	0	1	7	0	9	2	11	0111011101s
63	0	0	12	0	18	1	11	0111101100s
64	0	0	13	0	19	1	11	0111101101s
65	0	0	14	0	20	1	11	0111110100s
66	1	21	1	1	21	1	11	0111110101s
67	1	22	1	1	22	1	11	0111111000s
68	1	23	1	1	23	1	11	0111111001s
69	1	24	1	1	24	1	11	1000000010s
70	1	25	1	1	25	1	11	1000000011s
71	0	13	1	0	0	10	12	00111111100s
72	0	9	2	0	0	11	12	00111111101s

73	0	5	3	0	1	6	12	01011111100s
74	0	6	3	0	2	4	12	01011111101s
75	0	7	3	0	4	3	12	01101111100s
76	0	3	4	0	5	3	12	01101111101s
77	0	2	5	0	10	2	12	01110111100s
78	0	2	6	0	21	1	12	01110111101s
79	0	1	8	0	22	1	12	01111011100s
80	0	1	9	0	23	1	12	01111011101s
81	0	0	15	0	24	1	12	01111101100s
82	0	0	16	0	25	1	12	01111101101s
83	0	0	17	0	26	1	12	01111110100s
84	1	0	3	1	0	3	12	01111110101s
85	1	2	2	1	2	2	12	01111111000s
86	1	26	1	1	26	1	12	01111111001s
87	1	27	1	1	27	1	12	10000000010s
88	1	28	1	1	28	1	12	10000000011s
89	0	10	2	0	0	12	13	001111111100s
90	0	4	4	0	1	7	13	001111111101s
91	0	5	4	0	2	5	13	010111111100s
92	0	6	4	0	3	4	13	010111111101s
93	0	3	5	0	6	3	13	011011111100s
94	0	4	5	0	7	3	13	011011111101s
95	0	1	10	0	11	2	13	011101111100s
96	0	0	18	0	27	1	13	011101111101s
97	0	0	19	0	28	1	13	011110111100s
98	0	0	22	0	29	1	13	011110111101s
99	1	1	3	1	1	3	13	011111011100s
100	1	3	2	1	3	2	13	011111011101s
101	1	4	2	1	4	2	13	011111101100s
102	1	29	1	1	29	1	13	011111101101s
103	1	30	1	1	30	1	13	011111110100s
104	1	31	1	1	31	1	13	011111110101s
105	1	32	1	1	32	1	13	011111111000s
106	1	33	1	1	33	1	13	011111111001s
107	1	34	1	1	34	1	13	100000000010s
108	1	35	1	1	35	1	13	100000000011s
109	0	14	1	0	0	13	14	0011111111100s
110	0	15	1	0	0	14	14	0011111111101s
111	0	11	2	0	0	15	14	0101111111100s

112	0	8	3	0	0	16	14	0101111111101s
113	0	9	3	0	1	8	14	0110111111100s
114	0	7	4	0	3	5	14	0110111111101s
115	0	3	6	0	4	4	14	0111011111100s
116	0	2	7	0	5	4	14	0111011111101s
117	0	2	8	0	8	3	14	0111101111100s
118	0	2	9	0	12	2	14	0111101111101s
119	0	1	11	0	30	1	14	0111110111100s
120	0	0	20	0	31	1	14	0111110111101s
121	0	0	21	0	32	1	14	0111111011100s
122	0	0	23	0	33	1	14	0111111011101s
123	1	0	4	1	0	4	14	0111111101100s
124	1	5	2	1	5	2	14	0111111101101s
125	1	6	2	1	6	2	14	0111111110100s
126	1	7	2	1	7	2	14	0111111110101s
127	1	8	2	1	8	2	14	0111111111000s
128	1	9	2	1	9	2	14	0111111111001s
129	1	36	1	1	36	1	14	1000000000010s
130	1	37	1	1	37	1	14	1000000000011s
131	0	16	1	0	0	17	15	00111111111100s
132	0	17	1	0	0	18	15	00111111111101s
133	0	18	1	0	1	9	15	01011111111100s
134	0	8	4	0	1	10	15	01011111111101s
135	0	5	5	0	2	6	15	01101111111100s
136	0	4	6	0	2	7	15	01101111111101s
137	0	5	6	0	3	6	15	01110111111100s
138	0	3	7	0	6	4	15	01110111111101s
139	0	3	8	0	9	3	15	01111011111100s
140	0	2	10	0	13	2	15	01111011111101s
141	0	2	11	0	14	2	15	01111101111100s
142	0	1	12	0	15	2	15	01111101111101s
143	0	1	13	0	16	2	15	01111110111100s
144	0	0	24	0	34	1	15	01111110111101s
145	0	0	25	0	35	1	15	01111111011100s
146	0	0	26	0	36	1	15	01111111011101s
147	1	0	5	1	0	5	15	01111111101100s
148	1	1	4	1	1	4	15	01111111101101s
149	1	10	2	1	10	2	15	01111111110100s
150	1	11	2	1	11	2	15	01111111110101s

151	1	12	2	1	12	2	15	01111111111000s
152	1	38	1	1	38	1	15	01111111111001s
153	1	39	1	1	39	1	15	10000000000010s
154	1	40	1	1	40	1	15	10000000000011s
155	0	0	27	0	0	19	16	001111111111100s
156	0	3	9	0	3	7	16	001111111111101s
157	0	6	5	0	4	5	16	010111111111100s
158	0	7	5	0	7	4	16	010111111111101s
159	0	9	4	0	17	2	16	011011111111100s
160	0	12	2	0	37	1	16	011011111111101s
161	0	19	1	0	38	1	16	011101111111100s
162	1	1	5	1	1	5	16	011101111111101s
163	1	2	3	1	2	3	16	011110111111100s
164	1	13	2	1	13	2	16	011110111111101s
165	1	41	1	1	41	1	16	011111011111100s
166	1	42	1	1	42	1	16	011111011111101s
167	1	43	1	1	43	1	16	011111101111100s
168	1	44	1	1	44	1	16	011111101111101s
169	ESCAPE						5	0000s

Table -24 -- FLC table for RUN

RUN	CODE
0	000000
1	000001
2	000010
:	:
63	111111

Table -25 -- FLC table for LEVEL

LEVEL	CODE
0	FORBIDDEN
1	00000000001
2	00000000010
:	:
2047	11111111111

5. Shape Coding

Table -26 -- Meaning of shape mode

Index	**Shape mode**
0	= "MVDs==0 && No Update"
1	= "MVDs!=0 && No Update"
2	transparent
3	opaque
4	"intraCAE"
5	"interCAE && MVDs==0"
6	"interCAE && MVDs!=0"

Table -27 -- bab_type for I-VOP

Index	**(2)**	**(3)**	**(4)**	**Index**	**(2)**	**(3)**	**(4)**
0	1	001	01	41	001	01	1
1	001	01	1	42	1	01	001
2	01	001	1	43	001	1	01
3	1	001	01	44	001	01	1
4	1	01	001	45	1	01	001
5	1	01	001	46	001	01	1
6	1	001	01	47	01	001	1
7	1	01	001	48	1	01	001
8	01	001	1	49	001	01	1
9	001	01	1	50	01	001	1
10	1	01	001	51	1	001	01
11	1	01	001	52	001	1	01
12	001	01	1	53	01	001	1
13	1	01	001	54	1	001	01
14	01	1	001	55	01	001	1

15	001	01	1	56	01	001	1
16	1	01	001	57	1	01	001
17	1	01	001	58	1	01	001
18	01	001	1	59	1	01	001
19	1	01	001	60	1	01	001
20	001	01	1	61	1	01	001
21	01	001	1	62	01	001	1
22	1	01	001	63	1	01	001
23	001	01	1	64	001	01	1
24	01	001	1	65	001	01	1
25	001	01	1	66	01	001	1
26	001	01	1	67	001	1	01
27	1	01	001	68	001	1	01
28	1	01	001	69	01	001	1
29	1	01	001	70	001	1	01
30	1	01	001	71	001	01	1
31	1	01	001	72	1	001	01
32	1	01	001	73	001	01	1
33	1	01	001	74	01	001	1
34	1	01	001	75	01	001	1
35	001	01	1	76	001	1	01
36	1	01	001	77	001	01	1
37	001	01	1	78	1	001	01
38	001	01	1	79	001	1	01
39	1	01	001	80	001	01	1
40	001	1	01				

Table -28 -- bab_type for P-VOP and B-VOP

		bab_type in current VOP (n)						
		0	**1**	**2**	**3**	**4**	**5**	**6**
	0	1	01	00001	000001	0001	0010	0011
bab_type	**1**	01	1	00001	000001	001	0000001	0001
in previous	**2**	0001	001	1	000001	01	0000001	00001
VOP(n-1)	**3**	1	0001	000001	001	01	0000001	00001
	4	011	001	0001	00001	1	000001	010
	5	01	0001	00001	000001	001	11	10
	6	001	0001	00001	000001	01	10	11

Table -29 -- VLC table for MVDs

MVDs	**Codes**
0	0
± 1	10s
± 2	110s
± 3	1110s
± 4	11110s
± 5	111110s
± 6	1111110s
± 7	11111110s
± 8	111111110s
± 9	1111111110s
± 10	11111111110s
± 11	111111111110s
± 12	1111111111110s
± 13	11111111111110s
± 14	111111111111110s
± 15	1111111111111110s
± 16	11111111111111110s

Table -30 -- VLC table for MVDs (Horizontal element is 0)

MVDs	Codes
± 1	0s
± 2	10s
± 3	110s
± 4	1110s
± 5	11110s
± 6	111110s
± 7	1111110s
± 8	11111110s
± 9	111111110s
± 10	1111111110s
± 11	11111111110s
± 12	111111111110s
± 13	1111111111110s
± 14	11111111111110s
± 15	111111111111110s
± 16	1111111111111110s
s: sign bit (if MVDs is positive s="1", otherwise s="0").	

Table -31 -- VLC for conv_ratio

conv_ratio	Code
1	0
2	10
4	11

These tables contain the probabilities for a binary alpha pixel being equal to 0 for intra and inter shape coding using CAE. All probabilities are normalised to the range [1,65535].

As an example, given an INTRA context number C, the probability that the pixel is zero is given by intra_prob[C].

Table -32 -- Probabilities for arithmetic decoding of shape

```
USInt intra_prob[1024] = {
65267,16468,65003,17912,64573,8556,64252,5653,40174,3932,29789,277,45152,1140,32768,2043,
4499,80,6554,1144,21065,465,32768,799,5482,183,7282,264,5336,99,6554,563,
54784,30201,58254,9879,54613,3069,32768,58495,32768,32768,32768,2849,58982,54613,32768,128
31006,1332,49152,3287,60075,350,32768,712,39322,760,32768,354,52659,432,61854,150,
64999,28362,65323,42521,63572,32768,63677,18319,4910,32768,64238,434,53248,32768,61865,135
16384,32768,13107,333,32768,32768,32768,32768,32768,32768,1074,780,25058,5461,6697,233,
62949,30247,63702,24638,59578,32768,32768,42257,32768,32768,49152,546,62557,32768,54613,19
62405,32569,64600,865,60495,10923,32768,898,34193,24576,64111,341,47492,5231,55474,591,
65114,60075,64080,5334,65448,61882,64543,13209,54906,16384,35289,4933,48645,9614,55351,731
49807,54613,32768,32768,50972,32768,32768,32768,15159,1928,2048,171,3093,8,6096,74,
32768,60855,32768,32768,32768,32768,32768,32768,32768,32768,32768,32768,32768,55454,32768,
32768,16384,32768,21845,32768,32768,32768,32768,32768,32768,32768,5041,28440,91,32768,45,
65124,10923,64874,5041,65429,57344,63435,48060,61440,32768,63488,24887,59688,3277,63918,14
32768,32768,32768,32768,32768,32768,32768,32768,690,32768,32768,1456,32768,32768,8192,728,
32768,32768,58982,17944,65237,54613,32768,2242,32768,32768,32768,42130,49152,57344,58254,1
32768,10923,54613,182,32768,32768,32768,7282,49152,32768,32768,5041,63295,1394,55188,77,
63672,6554,54613,49152,64558,32768,32768,5461,64142,32768,32768,32768,62415,32768,32768,16
1481,438,19661,840,33654,3121,64425,6554,4178,2048,32768,2260,5226,1680,32768,565,
60075,32768,32768,32768,32768,32768,32768,32768,32768,32768,32768,32768,32768,32768,32768,
16384,261,32768,412,16384,636,32768,4369,23406,4328,32768,524,15604,560,32768,676,
49152,32768,49152,32768,32768,32768,64572,32768,32768,32768,54613,32768,32768,32768,32768,
4681,32768,5617,851,32768,32768,59578,32768,32768,32768,3121,3121,49152,32768,6554,10923,
32768,32768,54613,14043,32768,32768,32768,3449,32768,32768,32768,32768,32768,32768,32768,3
57344,32768,57344,3449,32768,32768,32768,3855,58982,10923,32768,239,62259,32768,49152,85,
58778,23831,62888,20922,64311,8192,60075,575,59714,32768,57344,40960,62107,4096,61943,3921
39862,15338,32768,1524,45123,5958,32768,58982,6669,930,1170,1043,7385,44,8813,5011,
```

59578,29789,54613,32768,32768,32768,32768,32768,32768,32768,32768,32768,58254,56174,32768,
64080,25891,49152,22528,32768,2731,32768,10923,10923,3283,32768,1748,17827,77,32768,108,
62805,32768,62013,42612,32768,32768,61681,16384,58982,60075,62313,58982,65279,58982,62694,
32768,32768,10923,950,32768,32768,32768,32768,5958,32768,38551,1092,11012,39322,13705,2072
54613,32768,32768,11398,32768,32768,32768,145,32768,32768,32768,29789,60855,32768,61681,54
32768,32768,32768,17348,32768,32768,32768,8192,57344,16384,32768,3582,52581,580,24030,303,
62673,37266,65374,6197,62017,32768,49152,299,54613,32768,32768,32768,35234,119,32768,3855,
31949,32768,32768,49152,16384,32768,32768,32768,24576,32768,49152,32768,17476,32768,32768,
51200,50864,54613,27949,60075,20480,32768,57344,32768,32768,32768,32768,32768,45875,32768,
11498,3244,24576,482,16384,1150,32768,16384,7992,215,32768,1150,23593,927,32768,993,
65353,32768,65465,46741,41870,32768,64596,59578,62087,32768,12619,23406,11833,32768,47720,
32768,32768,2621,6554,32768,32768,32768,32768,32768,32768,5041,32768,16384,32768,4096,2731
63212,43526,65442,47124,65410,35747,60304,55858,60855,58982,60075,19859,35747,63015,64470,
58689,1118,64717,1339,24576,32768,32768,1257,53297,1928,32768,33,52067,3511,62861,453,
64613,32768,32768,32768,64558,32768,32768,2731,49152,32768,32768,32768,61534,32768,32768,3
32768,32768,32768,32768,13107,32768,32768,32768,32768,32768,32768,32768,20480,32768,32768,
32768,32768,32768,54613,40960,5041,32768,32768,32768,32768,32768,3277,64263,57592,32768,31
32768,32768,32768,32768,32768,10923,32768,32768,32768,8192,32768,32768,5461,6899,32768,172
63351,3855,63608,29127,62415,7282,64626,60855,32768,32768,60075,5958,44961,32768,61866,537
32768,32768,32768,32768,32768,32768,6554,32768,32768,32768,32768,32768,2521,978,32768,1489
58254,32768,58982,61745,21845,32768,54613,58655,60075,32768,49152,16274,50412,64344,61643,
32768,32768,32768,1638,32768,32768,32768,24966,54613,32768,32768,2427,46951,32768,17970,65
65385,27307,60075,26472,64479,32768,32768,4681,61895,32768,32768,16384,58254,32768,32768,6
37630,3277,54613,6554,4965,5958,4681,32768,42765,16384,32768,21845,22827,16384,32768,6554,
65297,64769,60855,12743,63195,16384,32768,37942,32768,32768,32768,32768,60075,32768,62087,
41764,2161,21845,1836,17284,5424,10923,1680,11019,555,32768,431,39819,907,32768,171,
65480,32768,64435,33803,2595,32768,57041,32768,61167,32768,32768,32768,32768,32768,32768,1

```
60855,32768,17246,978,32768,32768,8192,32768,32768,32768,14043,2849,32768,2979,6554,6554,
65507,62415,65384,61891,65273,58982,65461,55097,32768,32768,32768,55606,32768,2979,3745,16
61885,13827,60893,12196,60855,53248,51493,11243,56656,783,55563,143,63432,7106,52429,445,
65485,1031,65020,1380,65180,57344,65162,36536,61154,6554,26569,2341,63593,3449,65102,533,
47827,2913,57344,3449,35688,1337,32768,22938,25012,910,7944,1008,29319,607,64466,4202,
64549,57301,49152,20025,63351,61167,32768,45542,58982,14564,32768,9362,61895,44840,32768,2
59664,17135,60855,13291,40050,12252,32768,7816,25798,1850,60495,2662,18707,122,52538,231,
65332,32768,65210,21693,65113,6554,65141,39667,62259,32768,22258,1337,63636,32768,64255,52
60362,32768,6780,819,16384,32768,16384,4681,49152,32768,8985,2521,24410,683,21535,16585,
65416,46091,65292,58328,64626,32768,65016,39897,62687,47332,62805,28948,64284,53620,52870,
65032,31174,63022,28312,64299,46811,48009,31453,61207,7077,50299,1514,60047,2634,46488,235
};

USInt inter_prob[512] = {
65532,62970,65148,54613,62470,8192,62577,8937,65480,64335,65195,53248,65322,62518,62891,38
65075,53405,63980,58982,32768,32768,54613,32768,65238,60009,60075,32768,59294,19661,61203,
63000,9830,62566,58982,11565,32768,25215,3277,53620,50972,63109,43691,54613,32768,39671,17
59788,6068,43336,27913,6554,32768,12178,1771,56174,49152,60075,43691,58254,16384,49152,993
23130,7282,40960,32768,10923,32768,32768,32768,27307,32768,32768,32768,32768,32768,32768,3
36285,12511,10923,32768,45875,16384,32768,32768,16384,23831,4369,32768,8192,10923,32768,32
10175,2979,18978,10923,54613,32768,6242,6554,1820,10923,32768,32768,32768,32768,32768,5461
28459,593,11886,2030,3121,4681,1292,112,42130,23831,49152,29127,32768,6554,5461,2048,
65331,64600,63811,63314,42130,19661,49152,32768,65417,64609,62415,64617,64276,44256,61068,
64887,57525,53620,61375,32768,8192,57344,6554,63608,49809,49152,62623,32768,15851,58982,34
55454,51739,64406,64047,32768,32768,7282,32768,49152,58756,62805,64990,32768,14895,16384,1
57929,24966,58689,31832,32768,16384,10923,6554,54613,42882,57344,64238,58982,10082,20165,2
62687,15061,32768,10923,32768,10923,32768,16384,59578,34427,32768,16384,32768,7825,32768,7
```

58052,23400,32768,5041,32768,2849,32768,32768,47663,15073,57344,4096,32768,1176,32768,1320
24858,410,24576,923,32768,16384,16384,5461,16384,1365,32768,5461,32768,5699,8192,13107,
46884,2361,23559,424,19661,712,655,182,58637,2094,49152,9362,8192,85,32768,1228,
65486,49152,65186,49152,61320,32768,57088,25206,65352,63047,62623,49152,64641,62165,58986,
64171,16384,60855,54613,42130,32768,61335,32768,58254,58982,49152,32768,60985,35289,64520,
51067,32768,64074,32768,40330,32768,34526,4096,60855,32768,63109,58254,57672,16384,31009,2
23406,32768,44620,10923,32768,32768,32099,10923,49152,49152,54613,60075,63422,54613,46388,
58982,32768,54613,32768,14247,32768,22938,5041,32768,49152,32768,32768,25321,6144,29127,10
41263,32768,46811,32768,267,4096,426,16384,32768,19275,49152,32768,1008,1437,5767,11275,
5595,5461,37493,6554,4681,32768,6147,1560,38229,10923,32768,40960,35747,2521,5999,312,
17052,2521,18808,3641,213,2427,574,32,51493,42130,42130,53053,11155,312,2069,106,
64406,45197,58982,32768,32768,16384,40960,36864,65336,64244,60075,61681,65269,50748,60340,
58982,23406,57344,32768,6554,16384,19661,61564,60855,47480,32768,54613,46811,21701,54909,3
32768,58982,60855,60855,32768,32768,39322,49152,57344,45875,60855,55706,32768,24576,62313,
54613,8192,49152,10923,32768,32768,32768,32768,32768,19661,16384,51493,32768,14043,40050,4
59578,5174,32768,6554,32768,5461,23593,5461,63608,51825,32768,23831,58887,24032,57170,3298
39322,12971,16384,49152,1872,618,13107,2114,58982,25705,32768,60075,28913,949,18312,1815,
48188,114,51493,1542,5461,3855,11360,1163,58982,7215,54613,21487,49152,4590,48430,1421,
28944,1319,6868,324,1456,232,820,7,61681,1864,60855,9922,4369,315,6589,14
};

6. Sprite Coding

Table -33 -- Code table for the first trajectory point

dmv value	SSS	VLC	dmv_code
-16383 ? -8192, 8192 ? 16383	14	111111111110	00000000000000...01111111111111 10000000000000...1111111111111
-8191 ? -4096, 4096 ? 8191	13	11111111110	0000000000000...0111111111111, 1000000000000...1111111111111
-4095 ? -2048, 2048 ? 4095	12	1111111110	000000000000...011111111111, 100000000000...111111111111
-2047...-1024, 1024...2047	11	111111110	00000000000...01111111111, 10000000000...11111111111
-1023...-512, 512...1023	10	11111110	0000000000...0111111111, 1000000000...1111111111
-511...-256, 256...511	9	1111110	000000000...011111111, 100000000...111111111
-255...-128, 128...255	8	111110	00000000...01111111, 10000000...1111
-127...-64, 64...127	7	11110	0000000...0111111, 1000000...11111
-63...-32, 32...63	6	1110	000000...011111, 100000...111111
-31...-16, 16...31	5	110	00000...01111, 10000...1111
-15...-8, 8...15	4	101	0000...0111, 1000...1111
-7...-4, 4...7	3	100	000...011, 100...111
-3...-2, 2...3	2	011	00...01, 10...11
-1, 1	1	010	0, 1
0	0	00	-

Table -34 -- Code table for scaled brightness change factor

brightness_change_factor value	brightness_change_factor_length value	brightness_change_factor_length VLC	brightness_change_factor
-16...-1, 1...16	1	0	00000...01111, 10000...11111
-48...-17, 17...48	2	10	000000...011111, 100000...11111
112...-49, 49...112	3	110	0000000...0111111, 1000000...111111
113?624	4	1110	000000000...111111111
625...1648	4	1111	0000000000?1111111111

7. **DCT based facial object decoding**

Table -35 -- Viseme_select_table, 29 symbols

symbol	bits	code	symbol	bits	code	symbol	bits	code
0	6	001000	10	6	010001	20	6	010000
1	6	001001	11	6	011001	21	6	010010
2	6	001011	12	5	00001	22	6	011010
3	6	001101	13	6	011101	23	5	00010
4	6	001111	14	1	1	24	6	011110
5	6	010111	15	6	010101	25	6	010110
6	6	011111	16	6	010100	26	6	001110
7	5	00011	17	6	011100	27	6	001100
8	6	011011	18	5	00000	28	6	001010
9	6	010011	19	6	011000			

Table -36 --Expression_select_table, 13 symbols

symbol	bits	code	symbol	bits	code	symbol	bits	code
0	5	01000	5	4	0011	10	5	01110
1	5	01001	6	1	1	11	5	01100
2	5	01011	7	4	0001	12	5	01010
3	5	01101	8	4	0000			
4	5	01111	9	4	0010			

Table -37 -- Viseme and Expression intensity_table, 127 symbols

symbol	bits	code	symbol	bits	code	symbol	bits	code
0	17	10010001101010010	43	16	1001000110100111	86	16	100100011010
1	17	10010001101010011	44	8	10011100	87	16	100100011010
2	17	10010001101010101	45	11	10010001111	88	16	100100011010
3	17	10010001101010111	46	9	100100010	89	16	100100011010
4	17	10010001101011001	47	10	1110001011	90	16	100100011001
5	17	10010001101011011	48	9	100011011	91	16	100100011001
6	17	10010001101011101	49	10	1110001001	92	16	100100011001
7	17	10010001101011111	50	9	100011010	93	16	100100011001
8	17	10010001101100001	51	9	100111010	94	16	100100011001
9	17	10010001101100011	52	10	1110001000	95	16	100100011001
10	17	10010001101100101	53	7	1000111	96	16	100100011001
11	17	10010001101100111	54	7	1000010	97	16	100100011001
12	17	10010001101101001	55	8	10010000	98	16	100100011000
13	17	10010001101101011	56	7	1001111	99	16	100100011000
14	17	10010001101101101	57	7	1110000	100	16	100100011000
15	17	10010001101101111	58	6	100000	101	16	100100011000
16	17	10010001101110001	59	6	100101	102	16	100100011000
17	17	10010001101110011	60	6	111010	103	16	100100011000
18	17	10010001101110111	61	5	11111	104	16	100100011000
19	17	10010001101111001	62	3	101	105	16	100100011000
20	17	10010001101111011	63	1	0	106	17	100100011011
21	17	10010001101111101	64	3	110	107	17	100100011011

22	17	10010001101111111	65	5	11110	108	17	1001000110111
23	16	1001000110000001	66	6	111001	109	17	1001000110111
24	16	1001000110000011	67	6	111011	110	17	1001000110111
25	16	1001000110000101	68	6	100010	111	17	1001000110111
26	16	1001000110000111	69	7	1001100	112	17	1001000110111
27	16	1001000110001001	70	7	1001001	113	17	1001000110111
28	16	1001000110001011	71	7	1001101	114	17	1001000110111
29	16	1001000110001101	72	8	10001100	115	17	1001000110111
30	16	1001000110001111	73	8	10000111	116	17	1001000110111
31	16	1001000110010001	74	8	10000110	117	17	1001000110111
32	16	1001000110010011	75	17	10010001101110100	118	17	1001000110111
33	16	1001000110010101	76	9	111000110	119	17	1001000110111
34	16	1001000110010111	77	11	11100010100	120	17	1001000110111
35	16	1001000110011001	78	11	10011101111	121	17	1001000110100
36	16	1001000110011011	79	17	10010001101110101	122	17	1001000110100
37	16	1001000110011101	80	10	1001110110	123	17	1001000110100
38	16	1001000110011111	81	16	1001000110101000	124	17	1001000110100
39	16	1001000110100001	82	11	10010001110	125	17	1001000110100
40	16	1001000110100011	83	10	1110001111	126	17	1001000110100
41	11	11100010101	84	11	10011101110			
42	16	1001000110100101	85	10	1110001110			

Table -38 -- Runlength_table, 16 symbols

symbol	bits	code	symbol	bits	code	symbol	bits	code
0	1	1	6	9	000001011	12	8	00000000
1	2	01	7	9	000001101	13	8	00000010
2	3	001	8	9	000001111	14	9	000001110
3	4	0001	9	8	00000011	15	9	000001100
4	5	00001	10	8	00000001			
5	9	000001010	11	8	00000100			

Table -39 -- DC_table, 512 symbols

symbol	bits	code	symbol	bits	code	symbol	bits	code
0	17	11010111001101010	171	17	11010111001111001	342	17	110101110011110
1	17	11010111001101011	172	17	11010111010000001	343	17	110101110011100
2	17	11010111001101101	173	17	11010111010001001	344	17	110101110011100
3	17	11010111001101111	174	17	11010111010010001	345	17	110101110011110
4	17	11010111001110101	175	17	11010111010011001	346	17	110101110100000
5	17	11010111001110111	176	17	11010111010101001	347	17	110101110100010
6	17	11010111001111101	177	17	11010111010110001	348	17	110101110100100
7	17	11010111001111111	178	17	11010111010111001	349	17	110101110100110
8	17	11010111010000101	179	17	11010111011000001	350	17	110101110101010
9	17	11010111010000111	180	17	11010111011001001	351	17	110101110101100
10	17	11010111010001101	181	17	11010111011011001	352	17	110101110101110
11	17	11010111010001111	182	17	11010111011111001	353	17	110101110110000
12	17	11010111010010101	183	17	11010111100000001	354	17	110101110110010
13	17	11010111010010111	184	17	11010111100001001	355	17	110101110110110
14	17	11010111010011101	185	17	11010111100011001	356	17	110101110111110
15	17	11010111010011111	186	17	11010111100100001	357	17	110101111000000
16	17	11010111010101101	187	17	11010111100101001	358	17	110101111000010
17	17	11010111010101111	188	17	11010111100111001	359	17	110101111000110
18	17	11010111010110111	189	17	11010111101000001	360	17	110101111001000
19	17	11010111010111101	190	17	11010111101001001	361	17	110101111001010
20	17	11010111010111111	191	17	11010111101011001	362	17	110101111001110
21	17	11010111011000111	192	17	11010111101111001	363	17	110101111010000
22	17	11010111011001101	193	17	11010111110000001	364	17	110101111010010
23	17	11010111011001111	194	17	11010111110001001	365	17	110101111010110
24	17	11010111011011101	195	17	11010111110011001	366	17	110101111011110
25	17	11010111011011111	196	17	11010111110111001	367	17	110101111100000
26	17	11010111011111101	197	17	11010111111100001	368	17	110101111100010
27	17	11010111011111111	198	17	11010111111101001	369	17	110101111100110
28	17	11010111100000111	199	17	11010111111111001	370	17	110101111101110
29	17	11010111100001101	200	16	1101011100000001	371	17	110101111111000
30	17	11010111100001111	201	16	1101011100001001	372	17	110101111111010
31	17	11010111100011101	202	16	1101011100011001	373	17	110101111111110
32	17	11010111100011111	203	17	11010111111001001	374	16	1101011100000001
33	17	11010111100100101	204	17	11010111111010001	375	16	1101011100000101
34	17	11010111100100111	205	17	11010111111011001	376	16	1101011100011101
35	17	11010111100101101	206	16	1101011100101001	377	17	110101111110010
36	17	11010111100101111	207	17	11010111110100001	378	17	110101111110100
37	17	11010111100111101	208	17	11010111110101001	379	17	110101111110110
38	17	11010111100111111	209	17	11010111101101001	380	16	1101011100101011

39	17	11010111101000101	210	17	11010111011100001	381	17	110101111101000
40	17	11010111101000111	211	16	1101011100100000	382	17	110101111101010
41	17	11010111101001101	212	16	1101011100100001	383	17	110101111011010
42	17	11010111101001111	213	17	11010111111000001	384	17	110101110111000
43	17	11010111101011101	214	16	1101011100010001	385	17	110101110111010
44	17	11010111101011111	215	17	11010111111110001	386	17	110101110111010
45	17	11010111101111101	216	17	11010111110110001	387	16	110101110010001
46	17	11010111101111111	217	17	11010111110010001	388	17	110101111110000
47	17	11010111110000101	218	11	11101100101	389	16	110101110001001
48	17	11010111110000111	219	11	11011111011	390	17	110101111111100
49	17	11010111110001101	220	11	11011110001	391	17	110101111101100
50	17	11010111110001111	221	10	1101110011	392	17	110101111100100
51	17	11010111110011101	222	17	11010111101110001	393	17	110101111011100
52	17	11010111110011111	223	17	11010111010100000	394	17	110101111010100
53	17	11010111110111101	224	17	11010111010100001	395	17	110101111010100
54	17	11010111110111111	225	17	11010111011110100	396	17	110101111000100
55	17	11010111111100101	226	17	11010111011110101	397	17	110101111000100
56	17	11010111111100111	227	17	11010111011110001	398	17	110101110110100
57	17	11010111111101101	228	17	11010111100010101	399	17	110101110110100
58	17	11010111111101111	229	17	11010111100110000	400	16	110101110011001
59	17	11010111111111101	230	17	11010111100110001	401	16	110101110011000
60	17	11010111111111111	231	17	11010111101010101	402	17	110101110101001
61	16	1101011100000101	232	11	11101100111	403	17	110101110101001
62	16	1101011100000111	233	17	11010111101110101	404	17	110101110101000
63	16	1101011100001101	234	11	11101100110	405	17	110101110110101
64	16	1101011100001111	235	17	11010111110110101	406	17	110101110110101
65	16	1101011100011101	236	17	11010111111000100	407	17	110101110111101
66	16	1101011100011111	237	8	11010110	408	17	110101110111100
67	17	11010111111001101	238	11	11011110010	409	17	110101111000101
68	17	11010111111001111	239	9	110010100	410	17	110101111001101
69	17	11010111111010101	240	10	1101110001	411	17	110101111001101
70	17	11010111111010111	241	9	110001111	412	17	110101111001100
71	17	11010111111011101	242	10	1101111100	413	17	110101111010101
72	17	11010111111011111	243	9	110010101	414	17	110101111011101
73	16	1101011100101101	244	9	110111111	415	17	110101111100101
74	16	1101011100101111	245	10	1101110100	416	17	110101111100101
75	17	11010111110100101	246	7	1100100	417	17	110101111101101
76	17	11010111110100111	247	8	11101101	418	17	110101111111101
77	17	11010111110101101	248	8	11001011	419	17	110101111111101
78	17	11010111110101111	249	7	1101100	420	16	110101110001011
79	17	11010111101101101	250	7	1101101	421	16	110101110001010
80	17	11010111101101111	251	7	1110111	422	17	110101111110001

81	17	11010111011100101	252	6	110100	423	16	110101110010011
82	17	11010111011100111	253	6	111001	424	16	110101110010010
83	17	11010111011101101	254	5	11111	425	17	110101111011001
84	17	11010111011101111	255	3	100	426	17	110101111011001
85	17	11010111101100001	256	1	0	427	17	110101111011000
86	17	11010111101100011	257	3	101	428	17	110101111011000
87	17	11010111101100101	258	5	11110	429	17	110101110111011
88	17	11010111101100111	259	6	111000	430	17	110101110111011
89	16	1101011100100101	260	6	111010	431	17	110101110111001
90	16	1101011100100111	261	6	110000	432	17	110101110111001
91	17	11010111111000111	262	7	1100111	433	17	110101111011011
92	16	1101011100010101	263	7	1100110	434	17	110101111011011
93	16	1101011100010111	264	7	1101010	435	17	110101111101011
94	17	11010111111110101	265	8	11000101	436	17	110101111101011
95	17	11010111111110111	266	8	11000110	437	17	110101111101001
96	17	11010111110110111	267	8	11000100	438	17	110101111101001
97	17	11010111110010101	268	17	11010111111000101	439	16	110101110010111
98	17	11010111110010111	269	9	111011000	440	16	110101110010110
99	17	11010111101110111	270	11	11011111010	441	17	110101111110111
100	17	11010111101010111	271	11	11011110101	442	17	110101111110111
101	17	11010111100110011	272	17	11010111100000101	443	17	110101111110101
102	17	11010111100110101	273	10	1101111011	444	17	110101111110101
103	17	11010111100110111	274	17	11010111011000101	445	17	110101111110011
104	17	11010111100010111	275	11	11011110011	446	17	110101111110011
105	17	11010111011110011	276	9	110001110	447	16	110101110001111
106	17	11010111011110111	277	11	11011110000	448	16	110101110001110
107	17	11010111011010101	278	10	1101110111	449	16	110101110000111
108	17	11010111011010111	279	17	11010111010110101	450	16	110101110000110
109	17	11010111010100011	280	16	1101011100110100	451	16	110101110000011
110	17	11010111010100101	281	10	1101110010	452	16	110101110000010
111	17	11010111010100111	282	10	1101110000	453	17	110101111111111
112	16	1101011100110001	283	11	11011101010	454	17	110101111111111
113	16	1101011100110011	284	17	11010111010110100	455	17	110101111111011
114	17	11010111011010001	285	17	11010111011000100	456	17	110101111111011
115	17	11010111011010011	286	17	11010111100000100	457	17	110101111111001
116	17	11010111100010001	287	11	11011101100	458	17	110101111111001
117	17	11010111100010011	288	17	11010111110110100	459	17	110101111101111
118	17	11010111101010001	289	17	11010111101110100	460	17	110101111101111
119	17	11010111101010011	290	17	11010111101010100	461	17	110101111100111
120	17	11010111101110011	291	11	11101100100	462	17	110101111100111
121	17	11010111110010011	292	17	11010111100010100	463	17	110101111100011
122	17	11010111110110011	293	17	11010111011110000	464	17	110101111100011

123	17	11010111111110011	294	11	11011110100	465	17	110101111100001
124	16	1101011100010011	295	11	11011101011	466	17	110101111100001
125	17	11010111111000011	296	17	11010111101110000	467	17	110101111011111
126	16	1101011100100011	297	17	11010111110010000	468	17	110101111011111
127	17	11010111011101001	298	17	11010111110110000	469	17	110101111010111
128	17	11010111011101011	299	17	11010111111110000	470	17	110101111010111
129	17	11010111011100011	300	16	1101011100010000	471	17	110101111010011
130	17	11010111101101011	301	17	11010111111000000	472	17	110101111010011
131	17	11010111110101011	302	11	11011101101	473	17	110101111010001
132	17	11010111110100011	303	17	11010111011100000	474	17	110101111010001
133	16	1101011100101011	304	17	11010111101101000	475	17	110101111001111
134	17	11010111111011011	305	17	11010111110101000	476	17	110101111001111
135	17	11010111111010011	306	17	11010111110100000	477	17	110101111001011
136	17	11010111111001011	307	16	1101011100101000	478	17	110101111001011
137	16	1101011100011011	308	17	11010111111011000	479	17	110101111001001
138	16	1101011100001011	309	17	11010111111010000	480	17	110101111001001
139	16	1101011100000011	310	17	11010111111001000	481	17	110101111000111
140	17	11010111111111011	311	16	1101011100011000	482	17	110101111000111
141	17	11010111111101011	312	16	1101011100001000	483	17	110101111000011
142	17	11010111111100011	313	16	1101011100000000	484	17	110101111000011
143	17	11010111110111011	314	17	11010111111111000	485	17	110101111000001
144	17	11010111110011011	315	17	11010111111101000	486	17	110101110111111
145	17	11010111110001011	316	17	11010111111100000	487	17	110101110111111
146	17	11010111110000011	317	17	11010111110111000	488	17	110101110110111
147	17	11010111101111011	318	17	11010111110011000	489	17	110101110110111
148	17	11010111101011011	319	17	11010111110001000	490	17	110101110110011
149	17	11010111101001011	320	17	11010111110000000	491	17	110101110110011
150	17	11010111101000011	321	17	11010111101111000	492	17	110101110110001
151	17	11010111100111011	322	17	11010111101011000	493	17	110101110101111
152	17	11010111100101011	323	17	11010111101001000	494	17	110101110101111
153	17	11010111100100011	324	17	11010111101000000	495	17	110101110101101
154	17	11010111100011011	325	17	11010111100111000	496	17	110101110101011
155	17	11010111100001011	326	17	11010111100101000	497	17	110101110101011
156	17	11010111100000011	327	17	11010111100100000	498	17	110101110100111
157	17	11010111011111011	328	17	11010111100011000	499	17	110101110100111
158	17	11010111011011011	329	17	11010111100001000	500	17	110101110100101
159	17	11010111011001011	330	17	11010111100000000	501	17	110101110100101
160	17	11010111011000011	331	17	11010111011111000	502	17	110101110100011
161	17	11010111010111011	332	17	11010111011011000	503	17	110101110100011
162	17	11010111010110011	333	17	11010111011001000	504	17	110101110100001
163	17	11010111010101011	334	17	11010111011000000	505	17	110101110100001
164	17	11010111010011011	335	17	11010111010111000	506	17	110101110011111

165	17	11010111010010011	336	17	11010111010110000	507	17	110101110011111
166	17	11010111010001011	337	17	11010111010101000	508	17	110101110011101
167	17	11010111010000011	338	17	11010111010011000	509	17	110101110011101
168	17	11010111001111011	339	17	11010111010010000	510	17	110101110011011
169	17	11010111001110011	340	17	11010111010001000	511	17	110101110011011
170	17	11010111001110001	341	17	11010111010000000			

Table -40 -- AC_table, 512 symbols

symbol	no_ of_ bits	code	symbol	no_ of_ bits	code	symbol	no_ of_ bits	code
0	16	1000011100011000	171	16	1000011101100001	342	16	1000011101
1	16	1000011100011001	172	16	1000011110100001	343	15	1000011100
2	16	1000011100011011	173	16	1000011111000001	344	16	1000011101
3	16	1000011100011101	174	16	1000011111100001	345	16	1000011110
4	16	1000011100011111	175	15	100001000100001	346	16	1000011111
5	16	1000011100100101	176	15	100001001100001	347	16	1000011111
6	16	1000011100100111	177	15	100001011000001	348	15	1000010001
7	16	1000011100101101	178	15	100001011100001	349	15	1000010011
8	16	1000011100101111	179	15	100001010100001	350	15	1000010110
9	16	1000011100111101	180	15	100001010000001	351	15	1000010111
10	16	1000011100111111	181	15	100001001000001	352	15	1000010101
11	16	1000011101111101	182	15	100001000000001	353	15	1000010100
12	16	1000011101111111	183	16	1000011110000001	354	15	1000010010
13	16	1000011110111111	184	16	1000011101000001	355	15	1000010000
14	16	1000011111011101	185	16	1000011101010001	356	16	1000011110
15	16	1000011111011111	186	16	1000011110010001	357	16	1000011101
16	16	1000011111111101	187	15	100001000010001	358	16	1000011101
17	16	1000011111111111	188	15	100001001010001	359	16	1000011110
18	15	100001000111101	189	15	100001010010001	360	15	1000010000
19	15	100001000111111	190	15	100001010110001	361	15	1000010010
20	15	100001001111101	191	15	100001011110001	362	15	1000010100
21	15	100001001111111	192	15	100001011010001	363	15	1000010101
22	15	100001011011101	193	15	100001001110001	364	15	1000010111
23	15	100001011011111	194	15	100001000110001	365	15	1000010110
24	15	100001011111101	195	16	1000011111110001	366	15	1000010011

25	15	100001011111111	196	16	1000011111010001	367	15	1000010001
26	15	100001010111111	197	16	1000011110110001	368	16	1000011111
27	15	100001010011101	198	16	1000011101110001	369	16	10000111110
28	15	100001010011111	199	16	1000011100110001	370	16	1000011110
29	15	100001001011111	200	15	100001110001001	371	16	1000011101
30	15	100001000011111	201	16	1000011100110101	372	16	1000011100
31	16	1000011110011111	202	16	1000011101110101	373	16	1000011100
32	16	1000011101011111	203	16	1000011110110101	374	16	1000011100
33	16	1000011101001111	204	16	1000011111010101	375	16	1000011100
34	16	1000011110001111	205	16	1000011111110101	376	16	1000011100
35	15	100001000001111	206	15	100001000110101	377	16	1000011100
36	15	100001001001111	207	15	100001001110101	378	16	1000011101
37	15	100001010001111	208	15	100001011010101	379	16	1000011110
38	15	100001010101111	209	15	100001011110101	380	16	10000111110
39	15	100001011101111	210	15	100001010110101	381	16	1000011111
40	15	100001011001111	211	15	100001010010101	382	15	1000010001
41	15	100001001101111	212	15	100001001010101	383	15	1000010011
42	15	100001000101111	213	15	100001000010101	384	15	1000010110
43	16	1000011111101111	214	16	1000011110010101	385	15	1000010111
44	16	1000011111001111	215	16	1000011101010101	386	15	1000010101
45	16	1000011110101111	216	16	1000011101000101	387	15	1000010100
46	16	1000011101101111	217	16	1000011110000101	388	15	1000010010
47	15	100001110000111	218	15	100001000000101	389	15	1000010000
48	16	1000011101100111	219	15	100001001000101	390	16	10000111100
49	16	1000011110100111	220	15	100001010000101	391	16	10000111010
50	16	1000011111000111	221	15	100001010100101	392	16	10000111010
51	16	1000011111100111	222	15	100001011100101	393	16	10000111100
52	15	100001000100111	223	15	100001011000101	394	15	10000100000
53	15	100001001100111	224	15	100001001100101	395	15	10000100100
54	15	100001011000111	225	15	100001000100101	396	15	10000101000
55	15	100001011100111	226	16	1000011111100101	397	15	10000101010
56	15	100001010100111	227	16	1000011111000101	398	15	10000101110
57	15	100001010000111	228	16	1000011110100101	399	15	10000101100
58	15	100001001000111	229	16	1000011101100101	400	15	10000100110
59	15	100001000000111	230	15	100001110000101	401	15	10000100010
60	16	1000011110000111	231	16	1000011101101101	402	16	1000011111

61	16	1000011101000111	232	16	1000011110101101	403	16	10000111110
62	16	1000011101010111	233	16	1000011111001101	404	16	1000011110
63	16	1000011110010111	234	16	1000011111101101	405	16	1000011101
64	15	100001000010111	235	15	100001000101101	406	15	10000111000
65	15	100001001010111	236	15	100001001101101	407	16	1000011101
66	15	100001010010111	237	15	100001011001101	408	16	1000011110
67	15	100001010110111	238	15	100001011101101	409	16	10000111110
68	15	100001011110111	239	15	100001010101101	410	16	1000011111
69	15	100001011010111	240	15	100001010001101	411	15	10000100010
70	15	100001001110111	241	15	100001001001101	412	15	10000100110
71	15	100001000110111	242	15	100001000001101	413	15	10000101100
72	16	1000011111110111	243	16	1000011110001101	414	15	10000101110
73	16	1000011111010111	244	16	1000011101001101	415	15	10000101010
74	16	1000011110110111	245	16	1000011101011101	416	15	10000101000
75	16	1000011101110111	246	16	1000011110011101	417	15	10000100100
76	16	1000011100110111	247	15	100001000011101	418	15	10000100000
77	15	100001110001011	248	6	100000	419	16	10000111100
78	16	1000011100110011	249	15	100001001011101	420	16	10000111010
79	16	1000011101110011	250	15	100001010111101	421	16	10000111010
80	16	1000011110110011	251	7	1001110	422	16	10000111100
81	16	1000011111010011	252	6	100110	423	15	1000010000
82	16	1000011111110011	253	5	10010	424	15	1000010010
83	15	100001000110011	254	4	1010	425	15	1000010100
84	15	100001001110011	255	2	11	426	15	1000010101
85	15	100001011010011	256	16	1000011110111100	427	15	1000010111
86	15	100001011110011	257	1	0	428	15	1000010110
87	15	100001010110011	258	4	1011	429	15	1000010011
88	15	100001010010011	259	6	100011	430	15	1000010001
89	15	100001001010011	260	6	100010	431	16	1000011111
90	15	100001000010011	261	7	1001111	432	16	10000111110
91	16	1000011110010011	262	16	1000011110111101	433	16	1000011110
92	16	1000011101010011	263	8	10000110	434	16	1000011101
93	16	1000011101000011	264	15	100001010111100	435	16	1000011100
94	16	1000011110000011	265	15	100001001011100	436	15	10000111000
95	15	100001000000011	266	15	100001000011100	437	16	1000011100
96	15	100001001000011	267	16	1000011110011100	438	16	1000011101

97	15	100001010000011	268	16	1000011101011100	439	16	1000011110
98	15	100001010100011	269	16	1000011101001100	440	16	10000111110
99	15	100001011100011	270	16	1000011110001100	441	16	1000011111
100	15	100001011000011	271	15	100001000001100	442	15	1000010001
101	15	100001001100011	272	15	100001001001100	443	15	1000010011
102	15	100001000100011	273	15	100001010001100	444	15	1000010110
103	16	1000011111100011	274	15	100001010101100	445	15	1000010111
104	16	1000011111000011	275	15	100001011101100	446	15	1000010101
105	16	1000011110100011	276	15	100001011001100	447	15	1000010100
106	16	1000011101100011	277	15	100001001101100	448	15	1000010010
107	15	100001110000011	278	15	100001000101100	449	15	1000010000
108	16	1000011101101011	279	16	1000011111101100	450	16	10000111100
109	16	1000011110101011	280	16	1000011111001100	451	16	10000111010
110	16	1000011111001011	281	16	1000011110101100	452	16	10000111010
111	16	1000011111101011	282	16	1000011101101100	453	16	10000111100
112	15	100001000101011	283	15	100001110000100	454	15	10000100000
113	15	100001001101011	284	16	1000011101100100	455	15	10000100100
114	15	100001011001011	285	16	1000011110100100	456	15	10000101000
115	15	100001011101011	286	16	1000011111000100	457	15	10000101010
116	15	100001010101011	287	16	1000011111100100	458	15	10000101110
117	15	100001010001011	288	15	100001000100100	459	15	10000101100
118	15	100001001001011	289	15	100001001100100	460	15	10000100110
119	15	100001000001011	290	15	100001011000100	461	15	10000100010
120	16	1000011110001011	291	15	100001011100100	462	16	1000011111
121	16	1000011101001011	292	15	100001010100100	463	16	10000111110
122	16	1000011101011011	293	15	100001010000100	464	16	1000011110
123	16	1000011110011011	294	15	100001001000100	465	16	1000011101
124	15	100001000011011	295	15	100001000000100	466	15	10000111000
125	15	100001001011011	296	16	1000011110000100	467	16	1000011101
126	15	100001010011011	297	16	1000011101000100	468	16	1000011110
127	15	100001010111011	298	16	1000011101010100	469	16	10000111110
128	15	100001011111011	299	16	1000011110010100	470	16	1000011111
129	15	100001011011011	300	15	100001000010100	471	15	10000100010
130	15	100001001111011	301	15	100001001010100	472	15	10000100110
131	15	100001000111011	302	15	100001010010100	473	15	10000101100
132	16	1000011111111011	303	15	100001010110100	474	15	10000101110

133	16	1000011111011011	304	15	100001011110100	475	15	10000101010
134	16	1000011110111011	305	15	100001011010100	476	15	10000101000
135	16	1000011101111011	306	15	100001001110100	477	15	10000100100
136	16	1000011100111011	307	15	100001000110100	478	15	10000100000
137	16	1000011100101011	308	16	1000011111110100	479	16	10000111100
138	16	1000011100100011	309	16	1000011111010100	480	16	10000111010
139	16	1000011100100001	310	16	1000011110110100	481	16	10000111010
140	16	1000011100101001	311	16	1000011101110100	482	16	10000111100
141	16	1000011100111001	312	16	1000011100110100	483	15	1000010000
142	16	1000011101111001	313	15	100001110001000	484	15	1000010010
143	16	1000011110111001	314	16	1000011100110000	485	15	1000010100
144	16	1000011111011001	315	16	1000011101110000	486	15	1000010100
145	16	1000011111111001	316	16	1000011110110000	487	15	1000010101
146	15	100001000111001	317	16	1000011111010000	488	15	1000010111
147	15	100001001111001	318	16	1000011111110000	489	15	1000010111
148	15	100001011011001	319	15	100001000110000	490	15	1000010110
149	15	100001011111001	320	15	100001001110000	491	15	1000010110
150	15	100001010111001	321	15	100001011010000	492	15	1000010011
151	15	100001010011001	322	15	100001011110000	493	15	1000010011
152	15	100001001011001	323	15	100001010110000	494	15	1000010001
153	15	100001000011001	324	15	100001010010000	495	15	1000010001
154	16	1000011110011001	325	15	100001001010000	496	16	1000011111
155	16	1000011101011001	326	15	100001000010000	497	16	1000011111
156	16	1000011101001001	327	16	1000011110010000	498	16	10000111110
157	16	1000011110001001	328	16	1000011101010000	499	16	10000111110
158	15	100001000001001	329	16	1000011101000000	500	16	1000011110
159	15	100001001001001	330	16	1000011110000000	501	16	1000011101
160	15	100001010001001	331	15	100001000000000	502	16	1000011101
161	15	100001010101001	332	15	100001001000000	503	16	1000011100
162	15	100001011101001	333	15	100001010000000	504	16	1000011100
163	15	100001011001001	334	15	100001010100000	505	16	1000011100
164	15	100001001101001	335	15	100001011100000	506	16	1000011100
165	15	100001000101001	336	15	100001011000000	507	16	1000011100
166	16	1000011111101001	337	15	100001001100000	508	16	1000011100
167	16	1000011111001001	338	15	100001000100000	509	16	10000111000
168	16	1000011110101001	339	16	1000011111100000	510	16	10000111000

169	16	1000011101101001	340	16	1000011111000000	511	16	1000011100
170	15	100001110000001	341	16	1000011110100000			

2. **Arithmetic Decoding**

   1. **Aritmetic decoding for still texture object**

To fully initialize the decoder, the function ac_decoder_init is called followed by ac_model_init respectively:

```
void ac_decoder_init (ac_decoder *acd) {

int i, t;

acd->bits_to_go = 0;

acd->total_bits = 0;

acd->value = 0;

for (i=1; i<=Code_value_bits; i++) {

acd->value = 2*acd->value + input_bit(acd);

}

acd->low = 0;

acd->high = Top_value;

return;

}

void ac_model_init (ac_model *acm, int nsym) {

int i;

acm->nsym = nsym;


acm->freq = (unsigned short *) malloc (nsym*sizeof (unsigned short));

check (!acm->freq, "arithmetic coder model allocation failure");

acm->cfreq = (unsigned short *) calloc (nsym+1, sizeof (unsigned short));

check (!acm->cfreq, "arithmetic coder model allocation failure");


for (i=0; i<acm->nsym; i++) {

acm->freq[i] = 1;

acm->cfreq[i] = acm->nsym - i;
}

acm->cfreq[acm->nsym] = 0;


return;

}
```

The acd is structures which contains the decoding variables and whose addresses act as handles for the decoded symbol/bitstreams. The fields bits_to_go, buffer, bitstream, and bitstream_len are used to manage the bits in memory. The low, high, and fbits fields describe the scaled range corresponding to the symbols which have been decoded. The value field contains the currently seen code value inside the range. The total_bits field contains the total number of bits encoded or used for decoding so far. The values Code_value_bits and Top_value describe the maximum number of bits and the maximum size of a coded value respectively. The ac_model structure contains the variables used for that particular probability model and it’s address acts as a handle. The nsym field contains the number of symbols in the symbol set, the freq field contains the table of frequency counts for each of the nsym symbols, and the cfreq field contains the cumulative frequency count derived from freq.

The bits are read from the bitstream using the function:

```
static int input_bit (ac_decoder *acd) {

int t;

unsigned int tmp;



if (acd->bits_to_go==0) {

acd->buffer = ace->bitstream[ace->bitstream_len++];

acd->bits_to_go = 8;

}



t = acd->buffer & 0x080;

acd->buffer <<= 1;

acd->buffer &= 0x0ff;

acd->total_bits += 1;

acd->bits_to_go -= 1;

t = t >> 7;



return t;

}
```

The decoding process has four main steps. The first step is to decode the symbol based on the current state of the probability model (frequency counts) and the current code value (value) which is used to represent (and is a member of) the current range. The second step is to get the new range. The third step is to rescale the range and simultaneously load in new code value bits. The fourth step is to update the model. To decode symbols, the following function is called:

```
int ac_decode_symbol (ac_decoder *acd, ac_model *acm) {
long range;
int cum;
int sym;

range = (long)(acd->high-acd->low)+1;

/*--- decode symbol ---*/
cum = (((long)(acd->value-acd->low)+1)*(int)(acm->cfreq[0])-1)/range;
for (sym = 0; (int)acm->cfreq[sym+1]>cum; sym++)
/* do nothing */ ;

check (sym<0||sym>=acm->nsym, "symbol out of range");

/*--- Get new range ---*/
acd->high = acd->low + (range*(int)(acm->cfreq[sym]))/(int)(acm->cfreq[0])-1;
acd->low = acd->low + (range*(int)(acm->cfreq[sym+1]))/(int)(acm->cfreq[0]);

/*--- rescale and load new code value bits ---*/
for (;;) {
if (acd->high<Half) {
/* do nothing */
} else if (acd->low>=Half) {
acd->value -= Half;
acd->low -= Half;
acd->high -= Half;
} else if (acd->low>=First_qtr && acd->high<Third_qtr) {
acd->value -= First_qtr;
acd->low -= First_qtr;
acd->high -= First_qtr;
} else
break;
acd->low = 2*acd->low;
acd->high = 2*acd->high+1;
acd->value = 2*acd->value + input_bit(acd);
}


/*--- Update probability model ---*/
update_model (acm, sym);


return sym;
}
```

The bits_plus_follow function mentioned above calls another function, output_bit. They are:

```
static void output_bit (ac_encoder *ace, int bit) {
ace->buffer <<= 1;
if (bit)
ace->buffer |= 0x01;

ace->bits_to_go -= 1;
ace->total_bits += 1;
if (ace->bits_to_go==0) {

if (ace->bitstream) {
if (ace->bitstream_len >= MAX_BUFFER)
if ((ace->bitstream = (uChar *)realloc(ace->bitstream, sizeof(uChar)*
(ace->bitstream_len/MAX_BUFFER+1)*MAX_BUFFER))==NULL) {
fprintf(stderr, "Couldn’t reallocate memory for ace->bitstream in output_bit.\n");
exit(-1);
}
ace->bitstream[ace->bitstream_len++] = ace->buffer;
}
ace->bits_to_go = 8;
}

return;
}

static void bit_plus_follow (ac_encoder *ace, int bit) {
output_bit (ace, bit);
while (ace->fbits > 0) {
output_bit (ace, !bit);
ace->fbits -= 1;
}


return;
}
```

The update of the probability model used in the decoding of the symbols is shown in the following function:

```
static void update_model (ac_model *acm, int sym)
{
int i;


if (acm->cfreq[0]==Max_frequency) {
int cum = 0;
acm->cfreq[acm->nsym] = 0;
for (i = acm->nsym-1; i>=0; i--) {
acm->freq[i] = ((int)acm->freq[i] + 1) / 2;
cum += acm->freq[i];
acm->cfreq[i] = cum;
}
}

acm->freq[sym] += 1;
for (i=sym; i>=0; i--)
acm->cfreq[i] += 1;

return;
}
```

This function simply updates the frequency counts based on the symbol just decoded. It also makes sure that the maximum frequency allowed is not exceeded. This is done by rescaling all frequency counts by 2.

2. Arithmetic decoding for shape decoding

1. Structures and Typedefs

```
typedef void Void;
typedef int Int;
typedef unsigned short int USInt;
#define CODE_BIT 32
#define HALF ((unsigned) 1 << (CODE_BITS-1))
#define QUARTER (1 << (CODE_BITS-2))
struct arcodec {
UInt L; /* lower bound */
UInt R; /* code range */
UInt V; /* current code value */
UInt arpipe;
Int bits_to_follow; /* follow bit count */
Int first_bit;
Int nzeros;
Int nonzero;
Int nzerosf;
Int extrabits;
};
```

```
typedef struct arcodec ArCoder;
typedef struct arcodec ArDecoder;
#define MAXHEADING 3
#define MAXMIDDLE 10
#define MAXTRAILING 2
```

2. **Decoder Source**

```
Void StartArDecoder(ArDecoder *decoder, Bitstream *bitstream) {
Int i,j;
decoder->V = 0;
decoder->nzerosf = MAXHEADING;
decoder->extrabits = 0;
for (i = 1; i<CODE_BITS; i++) {
j=BitstreamLookBit(bitstream,i+decoder->extrabits);
decoder->V += decoder->V + j;
if (j == 0) {
decoder->nzerosf--;
if (decoder->nzerosf == 0) {
decoder->extrabits++;
decoder->nzerosf = MAXMIDDLE;
}
}
else
decoder->nzerosf = MAXMIDDLE;
}
decoder->L = 0;
decoder->R = HALF - 1;
decoder->bits_to_follow = 0;
decoder->arpipe = decoder->V;
decoder->nzeros = MAXHEADING;
```

```
decoder->nonzero = 0;
}
Void StopArDecoder(ArDecoder *decoder, Bitstream *bitstream) {
Int a = decoder->L >> (CODE_BITS-3);
Int b = (decoder->R + decoder->L) >> (CODE_BITS-3);
Int nbits,i;
if (b == 0)
b = 8;
if (b-a >= 4 || (b-a == 3 && a&1))
nbits = 2;
else
nbits = 3;
for (i = 1; i <= nbits-1; i++)
AddNextInputBit(bitstream, decoder);
if (decoder->nzeros < MAXMIDDLE-MAXTRAILING || decoder->nonzero == 0)
BitstreamFlushBits(bitstream,1);
}
Void AddNextInputBit(Bitstream *bitstream, ArDecoder *decoder) {
Int i;
if (((decoder->arpipe >> (CODE_BITS-2))&1) == 0) {
decoder->nzeros--;
if (decoder->nzeros == 0) {
BitstreamFlushBits(bitstream,1);
decoder->extrabits--;
decoder->nzeros = MAXMIDDLE;
decoder->nonzero = 1;
}
}
else {
```

```
decoder->nzeros = MAXMIDDLE;
decoder->nonzero = 1;
}
BitstreamFlushBits(bitstream,1);
i = (Int)BitstreamLookBit(bitstream, CODE_BITS-1+decoder->extrabits);
decoder->V += decoder->V + i;
decoder->arpipe += decoder->arpipe + i;
if (i == 0) {
decoder->nzerosf--;
if (decoder->nzerosf == 0) {
decoder->nzerosf = MAXMIDDLE;
decoder->extrabits++;
}
}
else
decoder->nzerosf = MAXMIDDLE;
}
Int ArDecodeSymbol(USInt c0, ArDecoder *decoder, Bitstream *bitstream ) {
Int bit;
Int c1 = (1<<16) - c0;
Int LPS = c0 > c1;
Int cLPS = LPS ? c1 : c0;
unsigned long rLPS;
rLPS = ((decoder->R) >> 16) * cLPS;
if ((decoder->V - decoder->L) >= (decoder->R - rLPS)) {
bit = LPS;
decoder->L += decoder->R - rLPS;
decoder->R = rLPS;
}
```

```
else {

bit = (1-LPS);

decoder->R -= rLPS;

}

DECODE_RENORMALISE(decoder,bitstream);

return(bit);

}

Void DECODE_RENORMALISE(ArDecoder *decoder, Bitstream *bitstream) {

while (decoder->R < QUARTER) {

if (decoder->L >= HALF) {

decoder->V -= HALF;

decoder->L -= HALF;

decoder->bits_to_follow = 0;

}

else

if (decoder->L + decoder->R <= HALF)

decoder->bits_to_follow = 0;

else{

decoder->V -= QUARTER;

decoder->L -= QUARTER;

(decoder->bits_to_follow)++;

}

decoder->L += decoder->L;

decoder->R += decoder->R;

AddNextInputBit(bitstream, decoder);

}

}
```

1. BitstreamLookBit(bitstream,nbits) : Looks nbits ahead in the bitstream beginning from the current position in the bitstream and returns the bit.

1. BitstreamFlushBits(bitstream,nbits) : Moves the current bitstream position forward by nbits.

The parameter c0 (used in ArDecodeSymbol()) is taken directly from the probability tables of USint inter_prob or Usint intra_prob in Table B-32. That is, for the pixel to be coded/decoded, c0 is the probability than this pixel is equal to zero. The value of c0 depends on the context number of the given pixel to be decoded.

1. Face Object Decoding

In FAP decoder, a symbol is decoded by using a specific model based on the syntax and by calling the following procedure which is specified in C.

```
static long low, high, code_value, bit, length, sacindex, cum, zerorun=0;


int aa_decode(int cumul_freq[ ])
{
length = high - low + 1;
cum = (-1 + (code_value - low + 1) * cumul_freq[0]) / length;
for (sacindex = 1; cumul_freq[sacindex] > cum; sacindex++);
high = low - 1 + (length * cumul_freq[sacindex-1]) / cumul_freq[0];
low += (length * cumul_freq[sacindex]) / cumul_freq[0];


for ( ; ; ) {
if (high < q2) ;
else if (low >= q2) {
code_value -= q2;
low -= q2;
high -= q2;
}
else if (low >= q1 && high < q3) {
code_value -= q1;
low -= q1;
high -= q1;
}
else {
break;
}
low *= 2;

high = 2*high + 1;

bit_out_psc_layer();

code_value = 2*code_value + bit;

used_bits++;

}

return (sacindex-1);

}


void bit_out_psc_layer()

{

bit = getbits(1);

}
```

Again the model is specified through cumul_freq[ ]. The decoded symbol is returned through its index in the model. The decoder is initialized to start decoding an arithmetic coded bitstream by calling the following procedure.

```
void decoder_reset( )

{

int i;

zerorun = 0; /* clear consecutive zero's counter */

code_value = 0;

low = 0;

high = top;

for (i = 1; i <= 16; i++) {

bit_out_psc_layer();

code_value = 2 * code_value + bit;

}

used_bits = 0;

}
```

A.

(normative)

Face object decoding tables and definitions

FAPs names may contain letters with the following meaning: l = left, r = right, t = top, b = bottom, i = inner, o = outer, m = middle. The sum of two corresponding top and bottom eyelid FAPs must equal 1024 when the eyelids are closed. Inner lips are closed when the sum of two corresponding top and bottom lip FAPs equals zero. For example: (lower_t_midlip + raise_b_midlip) = 0 when the lips are closed. All directions are defined with respect to the face and not the image of the face.

Table -1 -- FAP definitions, group assignments and step sizes

#	FAP name	FAP description	units	Uni-orBidir	Pos motion	Grp	FDP subgrp num	Quant step size	Min/Max I-Frame quantized values	Min quan
1	viseme	Set of values determining the mixture of two visemes for this frame (e.g. pbm, fv, th)	na	na	na	1	na	1	viseme_blend: +63	visen
2	expression	A set of values determining the mixture of two facial expression	na	na	na	1	na	1	expression_intensity1, expression_intensity2: +63	expr expr +-63
3	open_jaw	Vertical jaw displacement (does not affect mouth opening)	MNS	U	down	2	1	4	+1080	+360
4	lower_t_midlip	Vertical top middle inner lip displacement	MNS	B	down	2	2	2	+-600	+-18
5	raise_b_midlip	Vertical bottom middle inner lip displacement	MNS	B	up	2	3	2	+-1860	+-60
6	stretch_l_cornerlip	Horizontal displacement of left inner lip corner	MW	B	left	2	4	2	+-600	+-18
7	stretch_r_cornerlip	Horizontal displacement of right inner lip corner	MW	B	right	2	5	2	+-600	+-18

8	lower_t_lip_lm	Vertical displacement of midpoint between left corner and middle of top inner lip	MNS	B	down	2	6	2	+-600	+-18
9	lower_t_lip_rm	Vertical displacement of midpoint between right corner and middle of top inner lip	MNS	B	down	2	7	2	+-600	+-18
10	raise_b_lip_lm	Vertical displacement of midpoint between left corner and middle of bottom inner lip	MNS	B	up	2	8	2	+-1860	+-60
11	raise_b_lip_rm	Vertical displacement of midpoint between right corner and middle of bottom inner lip	MNS	B	up	2	9	2	+-1860	+-60
12	raise_l_cornerlip	Vertical displacement of left inner lip corner	MNS	B	up	2	4	2	+-600	+-18
13	raise_r_cornerlip	Vertical displacement of right inner lip corner	MNS	B	up	2	5	2	+-600	+-18
14	thrust_jaw	Depth displacement of jaw	MNS	U	forward	2	1	1	+600	+180
15	shift_jaw	Side to side displacement of jaw	MW	B	right	2	1	1	+-1080	+-36
16	push_b_lip	Depth displacement of bottom middle lip	MNS	B	forward	2	3	1	+-1080	+-36
17	push_t_lip	Depth displacement of top middle lip	MNS	B	forward	2	2	1	+-1080	+-36

18	depress_chin	Upward and compressing movement of the chin (like in sadness)	MNS	B	up	2	10	1	+-420	+-18
19	close_t_l_eyelid	Vertical displacement of top left eyelid	IRISD	B	down	3	1	1	+-1080	+-60
20	close_t_r_eyelid	Vertical displacement of top right eyelid	IRISD	B	down	3	2	1	+-1080	+-60
21	close_b_l_eyelid	Vertical displacement of bottom left eyelid	IRISD	B	up	3	3	1	+-600	+-24
22	close_b_r_eyelid	Vertical displacement of bottom right eyelid	IRISD	B	up	3	4	1	+-600	+-24
23	yaw_l_eyeball	Horizontal orientation of left eyeball	AU	B	left	3	na	128	+-1200	+-42
24	yaw_r_eyeball	Horizontal orientation of right eyeball	AU	B	left	3	na	128	+-1200	+-42
25	pitch_l_eyeball	Vertical orientation of left eyeball	AU	B	down	3	na	128	+-900	+-30
26	pitch_r_eyeball	Vertical orientation of right eyeball	AU	B	down	3	na	128	+-900	+-30
27	thrust_l_eyeball	Depth displacement of left eyeball	ES	B	forward	3	na	1	+-600	+-18
28	thrust_r_eyeball	Depth displacement of right eyeball	ES	B	forward	3	na	1	+-600	+-18
29	dilate_l_pupil	Dilation of left pupil	IRISD	B	growing	3	5	1	+-420	+-12
30	dilate_r_pupil	Dilation of right pupil	IRISD	B	growing	3	6	1	+-420	+-12

31	raise_l_i_eyebrow	Vertical displacement of left inner eyebrow	ENS	B	up	4	1	2	+-900	+-36
32	raise_r_i_eyebrow	Vertical displacement of right inner eyebrow	ENS	B	up	4	2	2	+-900	+-36
33	raise_l_m_eyebrow	Vertical displacement of left middle eyebrow	ENS	B	up	4	3	2	+-900	+-36
34	raise_r_m_eyebrow	Vertical displacement of right middle eyebrow	ENS	B	up	4	4	2	+-900	+-36
35	raise_l_o_eyebrow	Vertical displacement of left outer eyebrow	ENS	B	up	4	5	2	+-900	+-36
36	raise_r_o_eyebrow	Vertical displacement of right outer eyebrow	ENS	B	up	4	6	2	+-900	+-36
37	squeeze_l_eyebrow	Horizontal displacement of left eyebrow	ES	B	right	4	1	1	+-900	+-30
38	squeeze_r_eyebrow	Horizontal displacement of right eyebrow	ES	B	left	4	2	1	+-900	+-30
39	puff_l_cheek	Horizontal displacement of left cheeck	ES	B	left	5	1	2	+-900	+-30
40	puff_r_cheek	Horizontal displacement of right cheeck	ES	B	right	5	2	2	+-900	+-30
41	lift_l_cheek	Vertical displacement of left cheek	ENS	U	up	5	3	2	+-600	+-18
42	lift_r_cheek	Vertical displacement of right cheek	ENS	U	up	5	4	2	+-600	+-18
43	shift_tongue_tip	Horizontal displacement of tongue tip	MW	B	right	6	1	1	+-1080	+-42

44	raise_tongue_tip	Vertical displacement of tongue tip	MNS	B	up	6	1	1	+-1080	+-42
45	thrust_tongue_tip	Depth displacement of tongue tip	MW	B	forward	6	1	1	+-1080	+-42
46	raise_tongue	Vertical displacement of tongue	MNS	B	up	6	2	1	+-1080	+-42
47	tongue_roll	Rolling of the tongue into U shape	AU	U	concave upward	6	3, 4	512	+300	+60
48	head_pitch	Head pitch angle from top of spine	AU	B	down	7	na	170	+-1860	+-60
49	head_yaw	Head yaw angle from top of spine	AU	B	left	7	na	170	+-1860	+-60
50	head_roll	Head roll angle from top of spine	AU	B	right	7	na	170	+-1860	+-60
51	lower_t_midlip _o	Vertical top middle outer lip displacement	MNS	B	down	8	1	2	+-600	+-18
52	raise_b_midlip_o	Vertical bottom middle outer lip displacement	MNS	B	up	8	2	2	+-1860	+-60
53	stretch_l_cornerlip_o	Horizontal displacement of left outer lip corner	MW	B	left	8	3	2	+-600	+-18
54	stretch_r_cornerlip_o	Horizontal displacement of right outer lip corner	MW	B	right	8	4	2	+-600	+-18
55	lower_t_lip_lm _o	Vertical displacement of midpoint between left corner and middle of top outer lip	MNS	B	down	8	5	2	+-600	+-18

56	lower_t_lip_rm _o	Vertical displacement of midpoint between right corner and middle of top outer lip	MNS	B	down	8	6	2	+-600	+-18
57	raise_b_lip_lm_o	Vertical displacement of midpoint between left corner and middle of bottom outer lip	MNS	B	up	8	7	2	+-1860	+-60
58	raise_b_lip_rm_o	Vertical displacement of midpoint between right corner and middle of bottom outer lip	MNS	B	up	8	8	2	+-1860	+-60
59	raise_l_cornerlip_o	Vertical displacement of left outer lip corner	MNS	B	up	8	3	2	+-600	+-18
60	raise_r_cornerlip _o	Vertical displacement of right outer lip corner	MNS	B	up	8	4	2	+-600	+-18
61	stretch_l_nose	Horizontal displacement of left side of nose	ENS	B	left	9	1	1	+-540	+-12
62	stretch_r_nose	Horizontal displacement of right side of nose	ENS	B	right	9	2	1	+-540	+-12
63	raise_nose	Vertical displacement of nose tip	ENS	B	up	9	3	1	+-680	+-18
64	bend_nose	Horizontal displacement of nose tip	ENS	B	right	9	3	1	+-900	+-18
65	raise_l_ear	Vertical displacement of left ear	ENS	B	up	10	1	1	+-900	+-24
66	raise_r_ear	Vertical displacement of right ear	ENS	B	up	10	2	1	+-900	+-24

67	pull_l_ear	Horizontal displacement of left ear	ENS	B	left	10	3	1	+-900	+-30
68	pull_r_ear	Horizontal displacement of right ear	ENS	B	right	10	4	1	+-900	+-30

Table -2 -- FAP grouping

Group	**Number of FAPs**
1: visemes and expressions	2
2: jaw, chin, inner lowerlip, cornerlips, midlip	16
3: eyeballs, pupils, eyelids	12
4: eyebrow	8
5: cheeks	4
6: tongue	5
7: head rotation	3
8: outer lip positions	10
9: nose	4
10: ears	4

In the following, each facial expression is defined by a textual description and a pictorial example. (reference [10], page 114.) This reference was also used for the characteristics of the described expressions.

Table -3 -- Values for expression_select

expression_select	**expression name**	**textual description**
0	na	na
1	joy	The eyebrows are relaxed. The mouth is open and the mouth corners pulled back toward the ears.
2	sadness	The inner eyebrows are bent upward. The eyes are slightly closed. The mouth is relaxed.
3	anger	The inner eyebrows are pulled downward and together. The eyes are wide open. The lips are pressed against each other or opened to expose the teeth.
4	fear	The eyebrows are raised and pulled together. The inner eyebrows are bent upward. The eyes are tense and alert.
5	disgust	The eyebrows and eyelids are relaxed. The upper lip is raised and curled, often asymmetrically.
6	surprise	The eyebrows are raised. The upper eyelids are wide open, the lower relaxed. The jaw is opened.

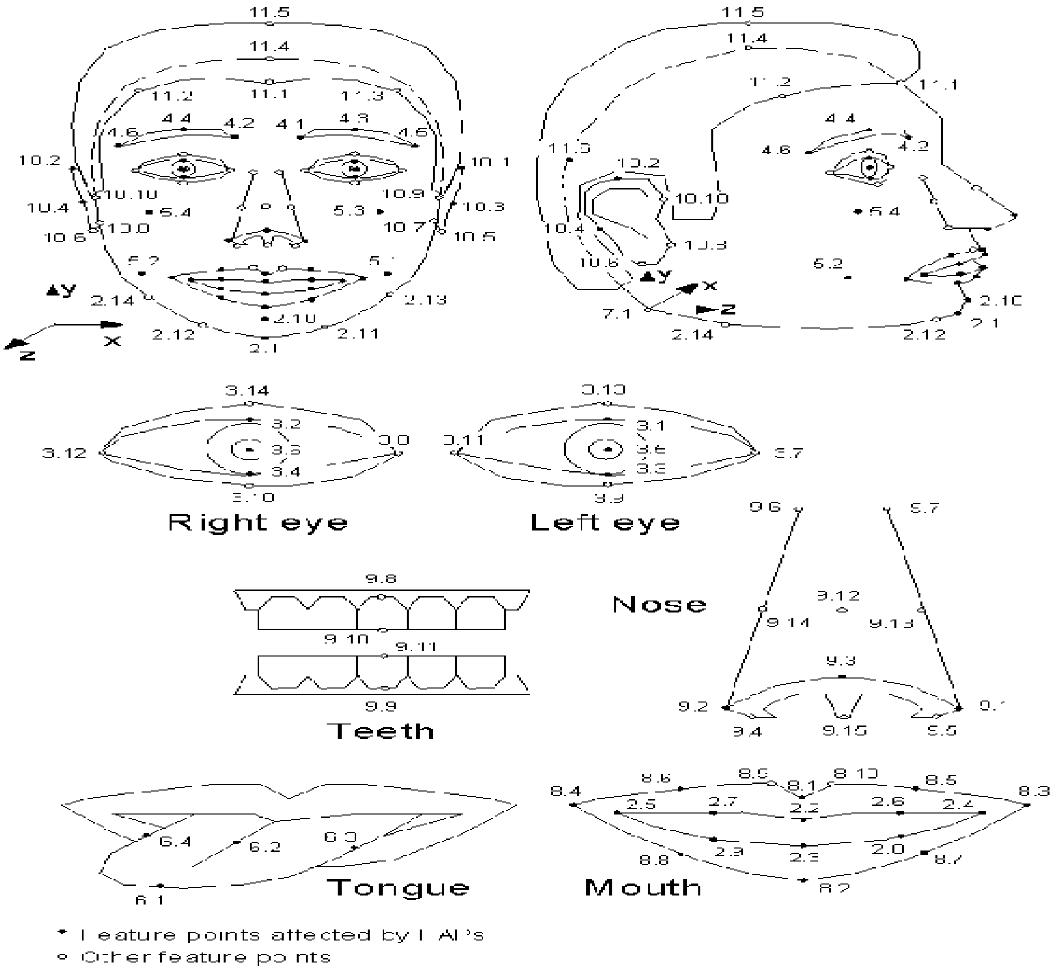


Figure -1 -- FDP feature point set

In the following, the notation 2.1.x indicates the x coordinate of feature point 2.1.

Feature points		**Recommended location constraints**		
#	Text description	x	y	z
2.1	Bottom of the chin	7.1.x		
2.2	Middle point of inner upper lip contour	7.1.x		
2.3	Middle point of inner lower lip contour	7.1.x		
2.4	Left corner of inner lip contour			
2.5	Right corner of inner lip contour			
2.6	Midpoint between f.p. 2.2 and 2.4 in the inner upper lip contour	(2.2.x+2.4.x)/2		
2.7	Midpoint between f.p. 2.2 and 2.5 in the inner upper lip contour	(2.2.x+2.5.x)/2		
2.8	Midpoint between f.p. 2.3 and 2.4 in the inner lower lip contour	(2.3.x+2.4.x)/2		
2.9	Midpoint between f.p. 2.3 and 2.5 in the inner lower lip contour	(2.3.x+2.5.x)/2		

2.10	Chin boss	7.1.x		
2.11	Chin left corner	> 8.7.x and < 8.3.x		
2.12	Chin right corner	> 8.4.x and < 8.8.x		
2.13	Left corner of jaw bone			
2.14	Right corner of jaw bone			
3.1	Center of upper inner left eyelid	(3.7.x+3.11.x)/2		
3.2	Center of upper inner right eyelid	(3.8.x+3.12.x)/2		
3.3	Center of lower inner left eyelid	(3.7.x+3.11.x)/2		
3.4	Center of lower inner right eyelid	(3.8.x+3.12.x)/2		
3.5	Center of the pupil of left eye			
3.6	Center of the pupil of right eye			
3.7	Left corner of left eye			
3.8	Left corner of right eye			
3.9	Center of lower outer left eyelid	(3.7.x+3.11.x)/2		
3.10	Center of lower outer right eyelid	(3.7.x+3.11.x)/2		
3.11	Right corner of left eye			
3.12	Right corner of right eye			
3.13	Center of upper outer left eyelid	(3.8.x+3.12.x)/2		
3.14	Center of upper outer right eyelid	(3.8.x+3.12.x)/2		
4.1	Right corner of left eyebrow			
4.2	Left corner of right eyebrow			
4.3	Uppermost point of the left eyebrow	(4.1.x+4.5.x)/2 or x coord of the uppermost point of the contour		

4.4	Uppermost point of the right eyebrow	(4.2.x+4.6.x)/2 or x coord of the uppermost point of the contour		
4.5	Left corner of left eyebrow			
4.6	Right corner of right eyebrow			
5.1	Center of the left cheek		8.3.y	
5.2	Center of the right cheek		8.4.y	
5.3	Left cheek bone	> 3.5.x and < 3.7.x	> 9.15.y and < 9.12.y	
5.4	Right cheek bone	> 3.6.x and < 3.12.x	> 9.15.y and < 9.12.y	
6.1	Tip of the tongue	7.1.x		
6.2	Center of the tongue body	7.1.x		
6.3	Left border of the tongue			6.2.z
6.4	Right border of the tongue			6.2.z
7.1	top of spine (center of head rotation)			
8.1	Middle point of outer upper lip contour	7.1.x		
8.2	Middle point of outer lower lip contour	7.1.x		
8.3	Left corner of outer lip contour			
8.4	Right corner of outer lip contour			
8.5	Midpoint between f.p. 8.3 and 8.1 in outer upper lip contour	(8.3.x+8.1.x)/2		
8.6	Midpoint between f.p. 8.4 and 8.1 in outer upper lip contour	(8.4.x+8.1.x)/2		
8.7	Midpoint between f.p. 8.3 and 8.2 in outer lower lip contour	(8.3.x+8.2.x)/2		
8.8	Midpoint between f.p. 8.4 and 8.2 in outer lower lip contour	(8.4.x+8.2.x)/2		
8.9	Right hiph point of Cupid?s bow			

8.10	Left hiph point of Cupid?s bow			
9.1	Left nostril border			
9.2	Right nostril border			
9.3	Nose tip	7.1.x		
9.4	Bottom right edge of nose			
9.5	Bottom left edge of nose			
9.6	Right upper edge of nose bone			
9.7	Left upper edge of nose bone			
9.8	Top of the upper teeth	7.1.x		
9.9	Bottom of the lower teeth	7.1.x		
9.10	Bottom of the upper teeth	7.1.x		
9.11	Top of the lower teeth	7.1.x		
9.12	Middle lower edge of nose bone (or nose bump)	7.1.x	(9.6.y + 9.3.y)/2 or nose bump	
9.13	Left lower edge of nose bone		(9.6.y +9.3.y)/2	
9.14	Right lower edge of nose bone		(9.6.y +9.3.y)/2	
9.15	Bottom middle edge of nose	7.1.x		
10.1	Top of left ear			
10.2	Top of right ear			
10.3	Back of left ear		(10.1.y+10.5.y)/2	
10.4	Back of right ear		(10.2.y+10.6.y)/2	
10.5	Bottom of left ear lobe			
10.6	Bottom of right ear lobe			
10.7	Lower contact point between left lobe and face			
10.8	Lower contact point between right lobe and face			

10.9	Upper contact point between left ear and face			
10.10	Upper contact point between right ear and face			
11.1	Middle border between hair and forehead	7.1.x		
11.2	Right border between hair and forehead	< 4.4.x		
11.3	Left border between hair and forehead	> 4.3.x		
11.4	Top of skull	7.1.x		> 10.4.z and < 10.2.z
11.5	Hair thickness over f.p. 11.4	11.4.x		11.4.z
11.6	Back of skull	7.1.x	3.5.y	

Table -4 -- FDP fields

FDP field	Description
featurePointsCoord	contains a **Coordinate** node. Specifies feature points for the calibration of the proprietary face. The coordinates are listed in the ?point? field in the **Coordinate** node in the prescribed order, that a feature point with a lower label is listed before a feature point with a higher label (e.g. feature point 3.14 before feature point 4.1).
textureCoords	contains a **Coordinate** node. Specifies texture coordinates for the feature points. The coordinates are listed in the **point** field in the **Coordinate** node in the prescribed order, that a feature point with a lower label is listed before a feature point with a higher label (e.g. feature point 3.14 before feature point 4.1).
textureType	contains a hint to the decoder on the type of texture image, in order to allow better interpolation of texture coordinates for the vertices that are not feature points. If **textureType** is 0, the decoder should assume that the texture image is obtained by cylindrical projection of the face. If **textureType** is 1, the decoder should assume that the texture image is obtained by orthographic projection of the face.
faceDefTables	contains **faceDefTables** nodes. The behavior of FAPs is defined in this field for the face in **faceSceneGraph**.
faceSceneGraph	contains a **Group** node. In case of option 1, this can be used to contain a texture image as explained above. In case of option 2, this is the grouping node for face model rendered in the compositor and has to contain the face model. In this case, the effect of Facial Animation Parameters is defined in the **faceDefTablesfield** .

Table -5 -- Values for viseme_select

viseme_select	phonemes	example
0	none	na
1	p, b, m	p ut, b ed, m ill
2	f, v	far, voice
3	T,D	think, that
4	t, d	tip, doll
5	k, g	call, gas
6	tS, dZ, S	ch air, j oin, sh e
7	s, z	sir, zeal
8	n, l	lot, not
9	r	red
10	A:	car
11	e	bed
12	I	tip
13	Q	top
14	U	book

Integration of Facial Animation with TTS

The following figure shows the complete block diagram describing the integration of a proprietary TTS Synthesizer into an ISO/IEC 14496 face animation system. The FAP bookmarks defined by the user in the input text of the TTS Stream are identified by the speech synthesizer and transmitted in ASCII format to the Phoneme to FAP converter using the TtsFAPInterface as defined in ISO/IEC 14496-3.

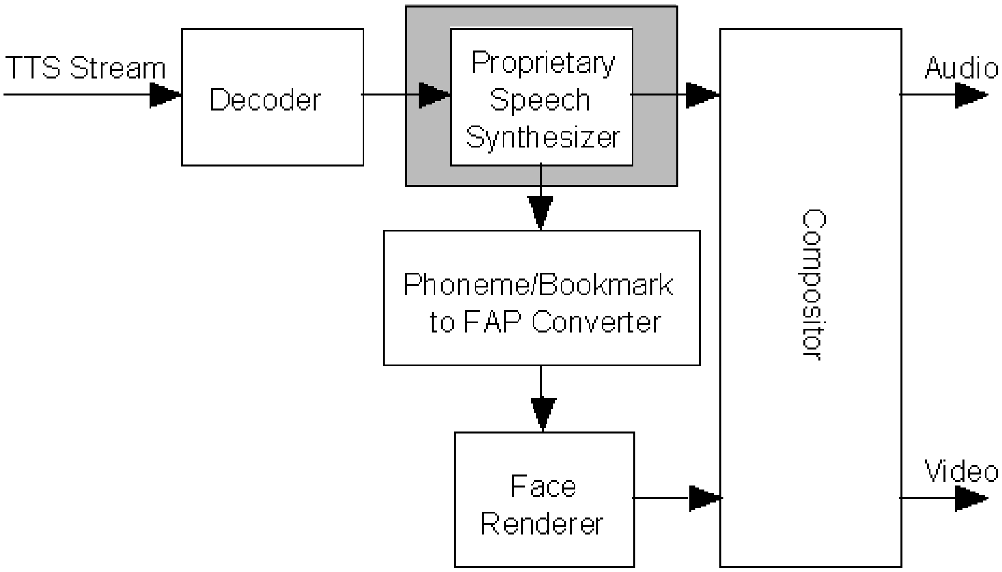

TTS Stream
Decoder
Proprietary Speech Synthesizer
Phoneme/Bookmark to FAP Converter
Face Renderer
Compositor
Audio
Video

Figure -2: Blockdiagram showing the integration of a proprietary TTS into an ISO/IEC 14496 face animation system with Phoneme/Bookmark to FAP Converter.

The syntax of the bookmark sequences used to convey commands to the TTS system is the following:

<FAP *n* FAPfields *T C* >

n: FAP number defined according to annex C, Table C-1 with 2<=n<=68.

FAPfields := expression_select1* expression_intensity1 expression_select2* expression_intensity2, in case n == 2

FAPfields := *a***, in case 2 < n < =68

T: transition time defined in ms

C: time curve for computation of *a* during transition time*T*

where ":=" indicates a production rule

* defined according to Table C-3 (expression select)

** defined in units according to Table C-1

Phoneme/Bookmark to FAP Converter

In addition to its function to convert phonemes into visemes, the Phoneme/Bookmark to FAP Converter (Fig. 1) is responsible for translating the FAP bookmarks defined by the user and made available by the bookmark field of the TtsFAPInterface as defined in ISO/IEC 14496-1 into a sequence of FAPs that can be interpreted by the Face Renderer, which then has to use them. An interpolation function is used for computing the FAP amplitudes. The converter merges the FAPs it derives from phonemes and bookmarks into one set of FAP parameters for each time instant using the following steps:

1. Set all low-level FAPs to interpolate.

2. Set specified FAPs to values computed using the interpolation function and phonemes.

3. init_face is set according to the following suggestion: In case of FAP 1 viseme_select1/2 != 0 and FAP 2 expression_select1/2 !=0, init_face=1; In case of FAP 2 expression_select1/2 !=0 and using FAP 3-68 for visemes,

init_face=0.

Facial expressions and visemes are combined such that mouth closures are achieved for the relevant visemes while maintaining the overall facial expression. In case that the bookmarks contain visemes, they are ignored.

Transitions between facial expressions are achieved by computing the interpolation function for expression_intensity1 and expression_intensity2. Smooth transitions have to be achieved if expression_select1 and expression_select2 are the same for consecutive bookmarks.

Face Renderer:

In the case that the face model is also animated from an FAP parameter stream, the face renderer has to animate the face using the input from the TtsFAPInterface which is processed by the Phoneme/Bookmark to FAP converter and the input from the FAP decoder. In case that the renderer receives values for the same FAP from both sources, the values derived from the FAP stream take precedence.

Interpolation Functions

For simplicity, the following description on how to compute the amplitude of an FAP is explained using the amplitude *a*. The same method is applied to expression_intensity1 and expression_intensity2.

The FAP amplitude *a* defines the amplitude to be applied at the end of the transition time T. The amplitude a_s of the FAP at the beginning of the transition depends on previous bookmarks and can be equal to:

- 0 if the FAP bookmark is the first one with this FAP *n* made available through the TtsFAPInterface.
- *a* of the previous FAP bookmark with the same FAP *n* if a time longer than the previous transition time *T* has elapsed between these two FAP bookmarks.
- The actual reached amplitude due to the previous FAP definition if a time shorter than the previous transition time *T* has elapsed between the two FAP bookmarks.

At the end of the transition time *T*, *a* is maintained until another FAP bookmark gives a new value to reach. To reset an FAP, a bookmark for FAP *n* with *a*=0 is transmitted in the text.

To avoid too many parameters for defining the evolution of the amplitude during the transition time, the functions that compute for each frame the amplitude of the FAP to be sent to the face renderer are predefined. Assuming that the transition time *T* is always 1, the following 3 functions *f(t)* are selected according the value of *C*:

$$f(t) = a_s + (a - a_s)t \quad \text{(linear) (1)}$$

$$f(t) = a_s + \begin{cases} (a - a_s)2t & for\ t \le 0.5 \\ (a - a_s)2(1 - t) & for\ 0.5 < t \le 1 \end{cases} \quad \text{(triangle) (2)}$$

$$f(t) = (2t^3 - 3t^2 + 1)a_s + (-2t^3 + 3t^2)a + (t^3 - 2t^2 + t)g_s \quad \text{(Hermite function) (3)}$$

with time *t*Î [0,1], the amplitude a_s at the beginning of the FAP at *t*=0, and the gradient g_s of $f(0)$ which is the FAP amplitude over time at *t*=0. If the transition time *T*¹ 1, the time axis of the functions (1) to (4) has to be scaled. These functions depend on a_s, g_s, a and T, and thus they are completely determined as soon as the FAP bookmark is known.

The Hermite function of third order enables one to match the tangent at the beginning of a segment with the tangent at the end of the previous segment, so that a smooth curve can be guarantied. Usually, the computation of the Hermite function requires 4 parameters as input, which are a_s, g_s, a and the gradient of $f(t)$ at t=1. Here a horizontal gradient at the end of the transition time is assumed. The gradient $g_s(t)$ at time *t* is computed according to

$$g_s(t) = (6t^2 - 6t)(a_s - a) + (3t^2 - 4t + 1)g_s \quad (4)$$

with a_s, g_s ,and *a* defined by the amplitude prior to the current bookmark.

A. (normative)

Video buffering verifier

1. Introduction

The video verifier comprises the following models, as shown in Figure D-1:

1. A video rate buffer model
2. A video complexity model, and
3. A video reference memory model

A video rate buffer model is required in order to bound the memory requirements for the bitstream buffer needed by a video decoder. With a rate buffer model, the video encoder can be constrained to make bitstreams which are decodable with a predetermined buffer memory size.

A video complexity model is required in order to bound the processing speed requirements needed by a compliant video decoder. With a video complexity model, the video encoder can be constrained to make bitstreams which are decodable with a predetermined decoder processor capability.

A video reference memory model is required in order to bound the macroblock (pixel) memory requirements needed by a video decoder. With a video reference memory model, a video encoder can be constrained to make bitstreams which are decodable with a predetermined reference memory size.

Figure -1 -- Video Verifier Model

1. Video Rate Buffer Model Definition

The ISO/IEC 14496-2 video buffering verifier (VBV) is an algorithm for checking a bitstream with its delivery rate function, R(t), to verify that the amount of rate buffer memory required in a decoder is less than the stated buffer size. If a visual bitstream is composed of multiple VOs each with one or more VOLs, the rate buffer model is applied independently to each VOL (using buffer size and rate functions particular to that VOL).

The VBV applies to natural video coded as a combination of I, P, B and S-VOPs; face animation, still textures, and mesh objects are not constrained by this model.

The coded video bitstream shall be constrained to comply with the requirements of the VBV defined as follows:

1. When the vbv_buffer_size and vbv_occupancy parameters are specified by systems-level configuration information, the bitstream shall be constrained according to the specified values. When the vbv_buffer_size and vbv_occupancy parameters are not specified (except in the short video header case as described below), this indicates that the bitstream should be constrained according to the default values of vbv_buffer_size and vbv_occupancy. The default value of vbv_buffer_size is the maximum value of vbv_buffer_size allowed within the profile and level. The default value of vbv_occupancy is 170 ´ vbv_buffer_size, where vbv_occupancy is in 64-bit units and vbv_buffer_size is in 16384-bit units. This corresponds to an initial occupancy of approximately two-thirds of the full buffer size.

1. The VBV buffer size is specified by the vbv_buffer_size field in the VOL header in units of 16384 bits. A vbv_buffer_size of 0 is forbidden. Define B = 16384 ´ vbv_buffer_size to be the VBV buffer size in bits.

2. The instantaneous video object layer channel bit rate seen by the encoder is denoted by $R_{vol}(t)$ in bits per second. If the bit_rate field in the VOL header is present, it defines a peak rate (in units of 400 bits per second; a value of 0 is forbidden) such that $R_{vol}(t)$ <= 400 ´ bit_rate The bits related to the initial I-VOP in the elementary stream for basic sprite sequences are ignored for the calculation of the peak rate. Note that $R_{vol}(t)$ counts only visual syntax for the current VOL (refer to the

definition of d_i below). If the channel is a serial time mutiplex containing other VOLs or as defined by ISO/IEC 14496-1 with a total instantaneous channel rate seen by the encoder of R(t), then

$$R_{vol}(t) = \begin{cases} R(t) & \text{if } t \in \{\text{channel bit duration of a bit from VOL } vol\} \\ 0 & \text{otherwise} \end{cases}$$

3. The VBV buffer is initially empty. The vbv_occupancy field specifies the initial occupancy of the VBV buffer in 64-bit units before decoding the initial VOP. The first bit in the VBV buffer is the first bit of the elementary stream, except for basic sprite sequences. The first bit in the VBV buffer for the basic sprite sequence is the first bit for first S-VOP in the elementary stream. For basic sprint sequences, the first S-VOP in the elementary stream is regarded as the first decoded VOP.

4. Define d_i to be size in bits of the i-th VOP plus any immediately preceding GOV header, where i is the VOP index which increments by 1 in decoding order. A VOP includes any trailing stuffing code words before the next start code and the size of a coded VOP (d_i) is always a multiple of 8 bits due to start code alignment.

5. Let t_i be the decoding time associated with VOP i in decoding order. All bits (d_i) of VOP i are removed from the VBV buffer instantaneously at t_i. This instantaneous removal property distinguishes the VBV buffer model from a real rate buffer. The method of determining the value of t_i is defined in item 7 below.

6. t_i is the composition time (or presentation time in a no-compositor decoder) of VOP *i*. For a video object plane, t_i defined by vop_time_increment (in units of 1/vop_time_increment_resolution seconds) plus the cumulative number of whole seconds specified by module_time_base In the case of interlaced video, a VOP consists of lines from two fields and t_i is the composition time of the first field. The relationship between the composition time and the decoding time for a VOP is given by:

   $t_i = t_i$ *if ((vop_coding_type of VOP i == B-VOP) || low_delay || scalability || sprite_enable)*

   $t_i = t_i - m_i$ *otherwise*

   Low_delay, scalability and sprite_enable are defined in section 6.3.3. If low_delay,scalability and sprite_enable are all ?0?, then the composition time of I and P VOP?s is delayed until all immediately temporally-previous B-VOPs have been composed. This delay period is $m_i = \tau_f - \tau_p$, *where f, for an I or P VOP is the index of the VOP itself, and for a B-VOP is the index of the nearest temporally-future non-B VOP relative to VOP i, and p is the index of the nearest temporally-previous non-B VOP relative to VOP i.*

In order to initialize the model decoder when m_i is needed for the first VOP, it is necessary to define an initial decoding time t_0 for the first VOP (since the timing structure is locked to the B-VOP times and the first decoded VOP would not be a B-VOP). This defined decoding timing shall be that $t_0 = 2t_1 - t_2$ (i.e., assuming that $t_1 - t_0 = t_2 - t_1$), since t_p is not defined in the case.

The following example demonstrates how m_i is determined for a sequence with variable numbers of consecutive B-VOPs:

Decoding order : $I_0 P_1 P_2 P_3 B_4 P_5 B_6 P_7 B_8 B_9 P_{10} B_{11} B_{12}$

Presentation order : $I_0 P_1 P_2 B_4 P_3 B_6 P_5 B_8 B_9 P_7 B_{11} B_{12} P_{10}$

Assume that vop_time_increment=1 and modulo_time_base=0 in this example. The sub-index i is in decoding order.

Table -1 -- An example that demonstrates how m_i is determined

i	τ_i	t_i	m_i
0	0	0-1=-1	1
1	1	1-1=0	1
2	2	2-1=1	1
3	4	4-2=2	2
4	3	3	2
5	6	6-2=4	2
6	5	5	2
7	9	9-3=6	3
8	7	7	3
9	8	8	3
10	12	12-3=9	3
11	10	10	3
12	11	11	3

8. Define b_i as the buffer occupancy in bits immediately following the removal of VOP i from the rate buffer. Using the above definitions, b_i can be iteratively defined

$$b_o \;= 64 \times vbv\_occupancy - d_o$$

$$b_{i+1} = b_i + \int_{t_i}^{t_{i+1}} R_{vol}(t)dt - d_{i+1} \quad \text{for } i \geq 0$$

9. The rate buffer model requires that the VBV buffer never overflow or underflow, that is

$$0 <= b_i \quad and \quad b_i + d_i <= B \quad for\ all\ i$$

Real-valued arithmetic is used to compute b_i so that errors are not accumulated.

A coded VOP size must always be less than the VBV buffer size, i.e., d_i < B for all i.

10. If the short video header is in use (i.e., when short_video_header = 1), then the parameter vbv_buffer_size is not present and the following conditions are required for VBV operation. The buffer is initially empty at the start of encoder operation (i.e., *t*=0 being at the time of the generation of the first video plane with short header), and its fullness is subsequently checked after each time interval of 1001/30000 seconds (i.e., at *t*=1001/30000, 2002/30000, etc.). If a complete video plane with short header is in the buffer at the checking time, it is removed. The buffer fullness after the removal of a VOP, b_i, shall be greater than or equal to zero and less than (4 · *R*max · 1001) / 30000 bits, where *R*max is the maximum bit rate in bits per second allowed within the profile and level. The number of bits used for coding any single VOP, d_i, shall not exceed *k* · 16384 bits, where *k* = 4 for QCIF and Sub-QCIF, *k* = 16 for CIF, *k* = 32 for 4CIF, and *k* = 64 for 16CIF, unless a larger value of *k* is specified in the profile and level definition. Furthermore, the total buffer fullness at any time shall not exceed a value of *B* = *k* · 16384 + (4 · *R*max · 1001) / 30000.

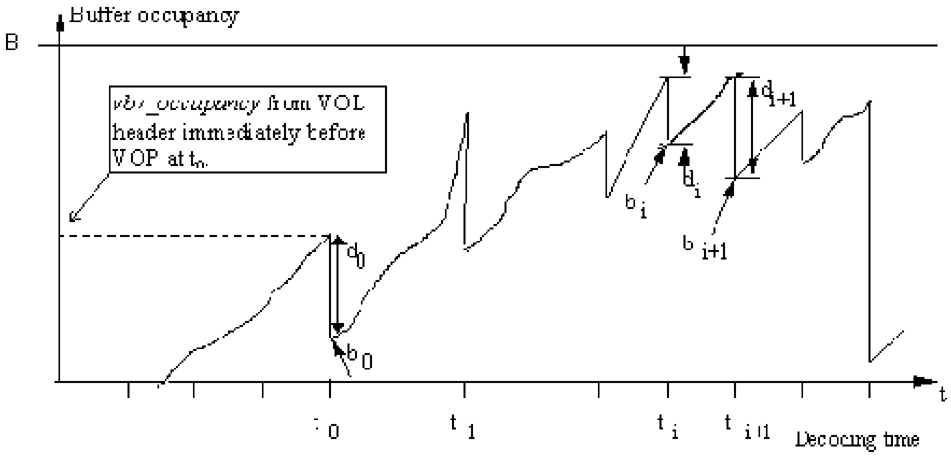


Figure -2 -- VBV Buffer occupancy

It is a requirement on the encoder to produce a bitstream which does not overflow or underflow the VBV buffer. This means the encoder must be designed to provide correct VBV operation for the range of values of $R_{vol,decoder}$(t) over which the system will operate for delivery of the bitstream. A channel has constant delay if the encoder bitrate at time t when particular bit enters the channel, $R_{vol,encoder}$(t) is equal to $R_{vol,decoder}$(t + L), where the bit is received at (t + L) and L is constant. In the case of constant delay channels, the encoder can use its locally estimated $R_{vol,encoder}$(t) to simulate the VBV occupancy and control the number of bits per VOP, d_i, in order to prevent overflow or underflow.

The VBV model assumes a constant delay channel. This allows the encoder to produce an elementary bitstream which does not overflow or underflow the buffer using $R_{vol,encoder}(t)$ - note that $R_{vol}(t)$ is defined as $R_{vol,encoder}(t)$ in item 2 above.

1. **Comparison between ISO/IEC 14496-2 VBV and the ISO/IEC 13818-2 VBV (Informative)**

   The ISO/IEC 13818-2 and ISO/IEC 14496-2 VBV models both specify that the rate buffer may not overflow or underflow and that coded pictures (VOPs) are removed from the buffer instantaneously. In both models a coded picture/VOP is defined to include all higher-level syntax immediately preceding the picture/VOP.

   ISO/IEC 13818-2 video has a constant frame period (although the bitstream can contain both frame and field pictures and frame pictures can use explicit 2:3 pulldown via the repeat_first_field flag). In ISO/IEC 14496 terms, this frame rate would be the output of the compositor (the ISO/IEC 13818 terminology is the output of the display process that is not defined normatively by ISO/IEC 13818). This output frame rate together with the ISO/IEC 13818-2 picture_structure and repeat_first_field flag precisely defines the time intervals between consecutive decoded picture (either frames or fields) passed between the decoding process and the display process.

   In general, the ISO/IEC 13818-2 bitstream contains B pictures (we assume ISO/IEC 13818-2 low_delay = 0). This means the coding order and display order of pictures is different (since both reference pictures used by a B picture must precede the B picture in coding order). The ISO/IEC 13818-2 VBV (and ISO/IEC 13818-1 systems T-STD) specifies that a B picture is decoded and presented (instantaneously) at the same time and the anchor pictures are re-ordered to make this possible. This is the same reordering model specified in item 7 above.

   A model ISO/IEC 14496-2 decoder using its VBV buffer model emulates a model decoder using the ISO/IEC 13838-2 VBV buffer model if the VOP time stamps given by vop_time_increment and the cumulative modulo_time_base agree with the sequence of ISO/IEC 13818-2 picture presentation times. We assume here that both coded picture/VOPs use the common subset of both standards (frame structured pictures and no 3:2 pulldown on the decoder, i.e., repeat_first_field = 0). For example, if the ISO/IEC 14496-2 sequence is coded at 29.97Hz (the NTSC picture rate), vop_time_increment_resolution will be 30000 and the change in vop_time_increment between consecutive VOPs in presentation order will be 1001 because picture skipping is not permitted in ISO/IEC 13818-2 (when low_delay = 0).

2. **Video Complexity Model Definition**

The ISO/IEC 14496-2 video complexity verifier (VCV) is an algorithm for checking a bitstream to verify that the amount of processing capability required in a decoder is less than the stated complexity measure in macroblocks per second. This model applies to all macroblocks in a MPEG-4 visual bitstream.

A separate VCV buffer accumulates the boundary macroblocks of all VOLs in a combined MPEG-4 visual bitstream. The boundary MB VCV has a separate decoding rate (specified in Annex N) but it is subject to the same maximum value used in the VCV accumulating the total number of macroblocks. Boundary macroblocks (containing coded shape information which is not totally transparent or totally opaque) are included in both the VCV model and in the boundary VCV model. For S-VOPs in low-latency sprite sequences, the number of boundary macroblocks used in the boundary VCV model is defined as the number of boundary macroblocks with coded shape information included in sprite_shape_texture() (see subclauses 6.2.5.4 and 7.8.3).

The VCV applies to natural video coded as a combination of I, P and B-VOPs; face animation, still textures, and mesh objects are not constrained by this model. For sprites, an equivalent number of macroblocks is defined for each S-VOP, however initial loading of the sprite for basic sprite sequences is not covered by the VCV model.

The coded video bitstream shall be constrained to comply with the requirements of the VCV defined as follows:

1. A vcv_buffer_size is defined as the maximum number of MBs which can be contained in the VCV-buffer and is specified in Annex N. These MBs are consumed by the decoder at vcv_decoder_rate or H (in MB/s), as specified in Annex N for each profile and level. A vcv_decoder_latency or L is defined as the time needed for the decoder to

decode a full VCV-buffer (vcv_buffer_size MBs) with the vcv_decoder_rate (MB/s). Thus the relation vcv_buffer_size = H*L holds. The VCV-buffer is initially empty at the start of decoding.

2. Let M_i be number of macroblocks in each VOP.

For S-VOPs, M_i is $MB_{S\text{-}VOP}$. The hypothetical number of macroblocks in an S-VOP, $MB_{S\text{-}VOP}$, for the usage in the VCV model is defined as follows:

$MB_{S\text{-}VOP} = (MB_{rcn} > MB_{ref}\ ?\ MB_{rcn} : MB_{ref}) + MB_{lls}$,

where MB_{rcn} is the number of macroblocks (including opaque, boundary, and transparent macroblocks) fully or partially included in the bounding rectangle of the reconstructed VOP, MB_{ref} is he number macroblocks included in the support area in the sprite, and MB_{lls} is the number of updated macroblocks in the sprite. MB_{ref} is defined as follows:

$MB_{ref} = MB_{rcn}$, when no_of_sprite_warping_points == 0 or 1,

$= (MB_{rcn}\ \text{´}\ ((i_1?\text{-}\ i_0?)^2 + (j_1?\text{-}\ j_0?)^2)))/(W\ ^2\ s^2)$,when no_of_sprite_warping_points == 2,

$= (MB_{rcn}\ \text{´}\ |(i_1? - i_0?)\ (j_2? - j_0?) - (i_2? - i_0?)\ (j_1? - j_0?)|\ )\ /\ (W\ H\ s^2)$,

when no_of_sprite_warping_points == 3, and

$= (MB_{rcn}\ \text{´}\ (|(i_1? - i_0?)\ (j_2? - j_0?) - (i_2? - i_0?)\ (j_1? - j_0?)| +$

$|(i_1? - i_3?)\ (j_2? - j_3?) - (i_2? - i_3?)\ (j_1? - j_3?)|)/(2\ W\ H\ s^2)$,

when no_of_sprite_warping_points == 4,

where the definition of W, H, s, $i_0?$, $j_0?$, $i_1?$, $j_1?$, $i_2?$, $j_2?$, $i_3?$, and $j_3?$ is described in subclause 7.8.4. MB_{lls} denotes the number of macroblocks with at least one coded DCT coefficient included in sprite_shape_texture() (see subclauses 6.2.5.4 and 7.8.3). The value of MB_{lls} is 0 when low_latency_sprite_enable == 0.

3. At time t_i, M_i is added to the VCV-buffer occupancy, v(t), where t_i is the decode time calculated according to clause D.2 for each VOL. The occupancy of the VCV buffer decreases linearly at rate H (defined in Annex N) until the occupancy is zero or until the time reaches t_j where t_j is the earliest VOP decoding time greater than t_i for some VOP in the visual bitstream. If the VCV occupancy becomes zero (the VCV decoder is idle), then the VCV remains idle until t_j. See Figure D-3.

4. The interval of time where VOP i is being decoded extends from s_i to e_i which are defined as

   $s_i = t_i + v(t_i)/H$

   $e_i = t_i + (v(t_i) + M_i)/H$

   $v(t_i)$ is the VCV occupancy before the macroblocks representing VOP i are added to v(t).

5. To decode a VOL, the decoding of each VOP i must be complete by t_i + L (composition time plus the latency of the VCV decoding process). The latency L of the VCV decoding process is constant for all VOPs. This requirement is equivalent to the statement that v(t) < H*L for all times t.

6. The complexity model allows the VCV buffer to underflow, in which case the decoder simply remains idle, and the VCV buffer occupancy, v(t), remains during the idle period.

1. Video Reference Memory Model Definition

The ISO/IEC 14496-2 video memory verifier (VMV) is an algorithm for checking a bitstream to verify that the amount of pixel memory required in a decoder is less than the maximum VMV buffer size (in units of MBs) specified in Annex N for each profile and level. If a visual bitstream is composed of multiple VOs each with one or more VOLs, the VMV models the memory requirements of all VOLs (this model assumes a shared memory space shared by all VOLs of all VOs).

The VMV applies to natural video coded as a combination of I, P and B-VOPs; face animation, still textures, sprites, and mesh objects are not constrained by this model.

The coded video bitstream shall be constrained to comply with the requirements of the VMV defined as follows:

1. The reference memory is initially empty. It is filled with the decoded data as each macroblock is decoded.

2. The amount of reference memory required for the decoding of the ith VOP is defined as the number of macroblocks in the VOP, M_i. This memory is consumed at the same constant rate specified in the VCV (i.e., H MB/s) as the decoding process occurs. The decoding duration of VOP i, T_i , is identical in the VCV and VMV models and starts at s_i and ends at e_i as defined in D.4 above.

3. At the composition time (or presentation time in a no-compositor decoder) plus VCV latency, t_i + L, of an I- or P-VOP the total memory allocated to the previous I- or P-VOP in the decoding order is released instantaneously.

4. At the composition time (or presentation time in a no-compositor decoder) plus VCV latency, t_i + L, of a B-VOP the total memory allocated to that B-VOP is released instantaneously.

5. The reference memory model requires that the VMV buffer never exceeds the vmv_buffer_size obtained from the Table N-1. The buffer occupancy of the video memory verifier is shown in Figure D-3.

1. Interaction between VBV, VCV and VMV (informative)

The VBV model defines when the bitstream is available for the decoding and is removed from the buffer. The VCV model defines the speed at which the macroblocks are decoded. The reference memory model defines the amount of reference memory that is consumed and released. Obviously, it is advantageous for the video decoder to decode as far in advance as possible. However, this is constrained by the VBV and the VMV. The decoder can only start decoding if the bits are available for decoding. At the same time as the decoder decodes the bitstream, it generates macroblocks which consumes the reference memory. So if it decodes too fast it will overflow the reference memory.

On the other hand if the decoder start decoding too late, then it will not be able to complete the decoding in time and the bitstream will be removed from the VBV before it could be processed. Similarly the reference memory required for the prediction of the current VOP may also be removed from the VMV.

Therefore, the encoder will have to simulate the VBV, VCV and VMV in order to produce a bitstream compliant with the intended profile and level. If the simulated decoder VBV becomes too full, the encoder should produce more bits (decrease the coarseness of quantization) or generate stuffing bits to prevent overflow. If the VBV occupancy becomes low, fewer bits are to produced. If the VCV approaches its limit, the encoder must allow more time for the decoder to produce its output. This can be done, for example, by increasing the interval of time between VOPs If the VMV becomes too full, decoder memory usage must be decreased. Using smaller VOPs (fewer macroblocks) and avoiding B-VOPs reduce the decoder?s memory requirements.

2. Video Presentation Model Definition (informative)

The video presentation verifier (VPV) is not part of this specification. It is an algorithm for checking a bitstream to verify that the amount of presentation buffer required in a decoder is less than a given amount of memory in units of MB. It is also used to constraint the speed of the compositor in terms of maximum number of MB/sec.

The VPV operates in the same manner as the VCV:

1. At the composition time plus VCV decoder latency of the i^{th} VOP, t_i, + L, the VOP is placed in the presentation buffer.

2. The data in the presentation buffer is consumed by the compositor at a rate equivalent to the maximum number of MB/sec.

3. At t_i + L + compositor_latency the VOP should be composited.

4. The presentation memory model requires that the VPV buffer never overflows

Figure -3 -- VCV, VMV and VPV Buffer Occupancy

A. (informative)

Features supported by the algorithm

1. Error resilience

1. Resynchronization

Resynchronization tools, as the name implies, attempt to enable resynchronization between the decoder and the bitstream after a residual error or errors have been detected. Generally, the data between the synchronization point prior to the error and the first point where synchronization is reestablished, is discarded. If the resynchronization approach is effective at localizing the amount of data discarded by the decoder, then the ability of other types of tools that recover data and/or conceal the effects of errors is greatly enhanced.

The resynchronization approach adopted by ISO/IEC 14496, referred to as a packet approach, is similar to the Group of Blocks (GOBs) structure utilized by the ITU-T recommendations H.261 and H.263. In these standards a GOB is defined as one or more rows of macroblocks (MB). At the start of a new GOB, information called a GOB header is placed within the bitstream. This header information contains a GOB start code, which is different from a picture start code, and allows the decoder to locate this GOB. Furthermore, the GOB header contains information which allows the decoding process to be restarted (i.e., resynchronize the decoder to the bitstream and reset all coded data that is predicted).

The GOB approach to resynchronization is based on spatial resynchronization. That is, once a particular macroblock location is reached in the encoding process, a resynchronization marker is inserted into the bitstream. A potential problem with this approach is that since the encoding process is variable rate, these resynchronization markers will most likely be unevenly spaced throughout the bitstream. Therefore, certain portions of the scene, such as high motion areas, will be more susceptible to errors, which will also be more difficult to conceal.

The video packet approach adopted by ISO/IEC 14496, is based on providing periodic resynchronization markers throughout the bitstream. In other words, the length of the video packets are not based on the number of macroblocks, but instead on the number of bits contained in that packet. If the number of bits contained in the current video packet exceeds a predetermined threshold, then a new video packet is created at the start of the next macroblock.

Resync Marker	**macroblock_number**	**quant_scale**	**HEC**	**Macroblock Data**	**Resync Marker**

Figure -1 -- Error Resilient Video Packet

In Figure E-1, a typical video packet is described. A resynchronization marker is used to distinguish the start of a new video packet. This marker is distinguishable from all possible VLC code words as well as the VOP start code. Header information is also provided at the start of a video packet. Contained in this header is the information necessary to restart the decoding process and includes: the macroblock address (number) of the first macroblock contained in this packet and the quantization parameter (quant_scale) necessary to decode that first macroblock. The macroblock number provides the necessary spatial resynchronization while the quantization parameter allows the differential decoding process to be resynchronized. Following the quant_scale is the Header Extension Code (HEC). As the name implies, HEC is a single bit used to indicate whether additional information will be available in this header. If the HEC is equal to one then the following additional information is available in this packet header: modulo time base, vop_time_increment, vop_coding_type, intra_dc_vlc_thr, vop_fcode_forward, vop_fcode_backward.

The Header Extension Code makes each video packet (VP) possible to be decoded independently, when its value is equal to 1. The necessary information to decode the VP is included in the header extension code field, if the HEC is equal to 1.

If the VOP header information is corrupted by the transmission error, they can be corrected by the HEC information. The decoder can detect the error in the VOP header, if the decoded information is inconsistent with its semantics. For example, because it is prohibited that the values of the vop_fcode_forward and vop_fcode_backward are set to "0", if they are 0, the decoder can detect the error in the fcode information. In such a case, the decoder can correct the value by using the HEC information of the next VP.

When utilizing the error resilience tools within ISO/IEC 14496, some of the compression efficiency tools are modified. For example, all predictively encoded information must be confined within a video packet so as to prevent the

propagation of errors. In other words, when predicting (i.e., AC/DC prediction and motion vector prediction) a video packet boundary is treated like a VOP boundary.

In conjunction with the video packet approach to resynchronization, a second method called fixed interval synchronization has also been adopted by ISO/IEC 14496. This method requires that VOP start codes and resynchronization markers (i.e., the start of a video packet) appear only at legal fixed interval locations in the bitstream. This helps to avoid the problems associated with start codes emulations. That is, when errors are present in a bitstream it is possible for these errors to emulate a VOP start code. In this case, when fixed interval synchronization is utilized the decoder is only required to search for a VOP start code at the beginning of each fixed interval. The fixed interval synchronization method extends this approach to be any predetermined interval.

The fixed interval synchronization is achieved by first inserting a bit with the value 0 and then, if necessary, inserting bits with the value 1 before the start code and the resync marker. The video decoder can determine if errors are injured in a video packet by detecting the incorrect number of the stuffing bits. (e.g. eight or more 1?s are followed after 0 at the last part of a video packet, or the remaining bit pattern is not "011...")

2. Data Partitioning

Error concealment is an extremely important component of any error robust video codec. Similar to the error resilience tools discussed above, the effectiveness of an error concealment strategy is highly dependent on the performance of the resynchronization scheme. Basically, if the resynchronization method can effectively localize the error then the error concealment problem becomes much more tractable. For low bitrate, low delay applications the current resynchronization scheme provides very acceptable results with a simple concealment strategy, such as copying blocks from the previous frame.

In recognizing the need to provide enhanced concealment capabilities, the Video Group has developed an additional error resilient mode that further improves the ability of the decoder to localize an error. Specifically, this approach utilizes data partitioning. This data partitioning is achieved by separating the motion and macroblock header information away from the texture information. This approach requires that a second resynchronization marker be inserted between motion and texture information. Data partitioning, like the use of RVLCs, is signaled to the decoder in the VOL. Figure E-2 illustrates the syntactic structure of the data partitioning mode. If the texture information is lost, this approach utilizes the motion information to conceal these errors. That is, due to the errors the texture information is discarded, while the motion is used to motion compensate the previously decoded VOP.

Resync Marker	macroblock_number	quant_scale	HEC	Motion &Header Information	Motion Marker	Texture Information

Figure -2 -- Data Partitioning

3. **Reversible VLC**

Reversible Variable Length Codes (RVLC) are designed such that they can be instantaneously decoded both in forward and reverse directions. A part of a bitstream which cannot be decoded in the forward direction due to the presence of errors can often be decoded in the backward direction. This is illustrated in Figure E-3. Therefore number of discarded bits can be reduced. RVLC is applied only to TCOEF coding

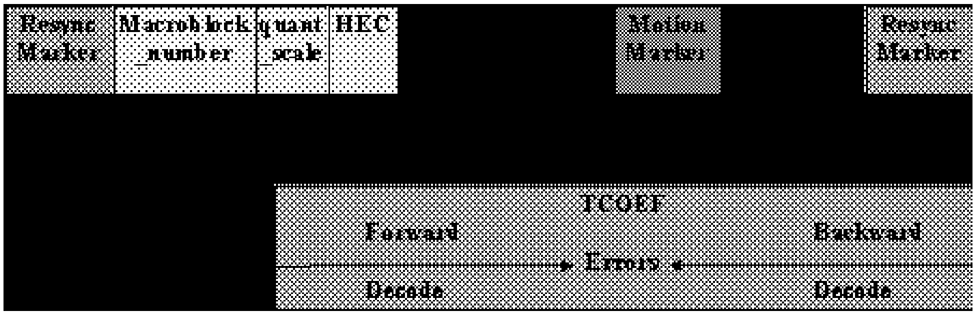


Figure -3 -- Reversible VLC

4. **Decoder Operation**
    1. **General Error Detection**

1. **An illegal VLC is received.**
1. **A semantic error is detected.**

1. **More than 64 DCT coefficients are decoded in a block.**

1. **Inconsistent resyncronization header information (i.e., QP out of range, MBA(k)<MBA(k-1),etc.)**

    1. **Resynchronization**

When an error is detected in the bitstream, the decoder should resynchronize at the next suitable resynchronization point(vop_start_code or resync_marker).

After that, it can be determined by detecting the incorrect number of the stuffing bits whether or not errors are injured in a video packet. If eight or more 1?s are followed after 0 at the last part of a video packet, or the

remaining bit pattern is not "011?", it means there is any error in this video packet.

If the VOP start code is corrupted by the transmission error and the frame synchronization is lost, the decoder may establish the resynchronization by using the HEC information. The decoder compares the vop_time_increment in the VOP header with one in the HEC field. If they are not same, the decoder may find that the current VOP start code is corrupted by the error. In this case, there must not be the error in the both vop_time_increments. The simple method is to check whether the vop_time_increment is mutilple of frame interval of the original source format (NTSC, PAL and so on). Therefore, it is expected that the number of the vop_time_increment is as many as possible. As this check method does not always detect the error, this is the auxiliary technique.

Missing blocks may be replaced with the same block from the previous frame.

2. **Data Partitioning**
3. **Reversible VLC**

This subclause describes a decoding methodology for Reversible Variable Length Codes (RVLC) when errors in the video bitstream are detected during the decoding process. This particular decoding methodology was developed during the RVLC core experiment process.

1. **Process for detecting errors for both forward and backward decoding**

Errors are present in the following cases:

(1) An illegal RVLC is found, where an illegal RVLC is defined as follows:

1. **A codeword whose pattern is not listed in the RVLC table (e.g. 169 codeword patterns and escape codes).**

1. **Escape coding is used (i.e., a legal codeword is not available in the RVLC table) and the decoded value for LEVEL is zero.**

1. **The second escape code is incorrect (e.g. codeword is not "00000" or "00001" for forward decoding, and/or is not "00001" for backward decoding).**

1. **There is a decoded value of FLC part using escape codes (e.g. LAST, RUN, LEVEL) in the RVLC table.**

1. **An incorrect number of stuffing bits for byte alignment (e.g. eight or more 1s follow 0 at the last part of a Video packet (VP), or the remaining bit pattern is not "0111..."**

after decoding process is finished).

(2) More than 64 DCT coefficients are decoded in a block.

1. **Decoding information**

   The bitstream is decoded in the forward direction first. If no errors are detected, the bitstream is assumed to be valid and the decoding process is finished for that video packet. If an error is detected however, two-way decoding is applied. The following strategies for determining which bits to discard are used. These strategies are described using the figures given below along with the following definitions:

   L : Total number of bits for DCT coefficients part in a VP.

   N : Total number of macroblocks (MBs) in a VP.

   L1 : Number of bits which can be decoded in a forward decoding.

   L2 : Number of bits which can be decoded in a backward decoding.

   N1 : Number of MBs which can be completely decoded in a forward decoding.

   (0 <= N1 <= (N-1))

   N2 : Number of MBs which can be completely decoded in a backward decoding.

   (0 <= N2 <= (N-1))

   f_mb(S) : Number of decoded MBs when S bits can be decoded in a forward direction.

   (Equal to or more than one bit can be decoded in a MB, f_mb(S) counter is up.)

   b_mb(S) : Number of decoded MBs when S bits can be decoded in a backward direction.

   T : Threshold (90 is used now).

   1. **Strategies for decoding RVLC**

      (1) Strategy 1 : $L1+L2 < L$ **and** $N1+N2 < N$

MBs of *f_mb(L1-T)* from the beginning and MBs of *b_mb(L2-T)* from the end are used. In the figure below, the MBs of the dark part are discarded.

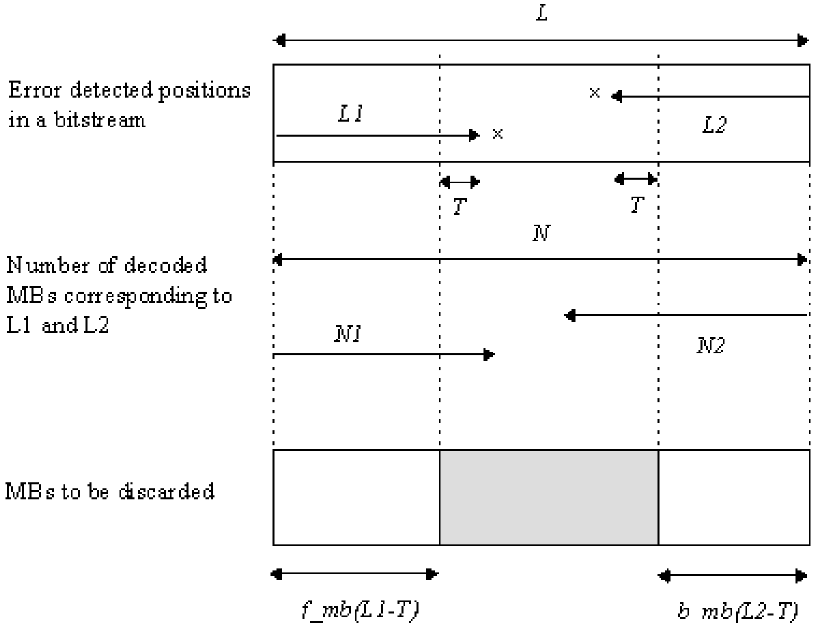

L
Error detected positions in a bitstream
L1
L2
T
T
N
Number of decoded MBs corresponding to L1 and L2
N1
N2
MBs to be discarded
f_mb(L1-T)
b mb(L2-T)

Figure -4 -- RVLC decoding; Strategy 1

(2) Strategy 2 : *L1+L2 < L* and *N1+N2 >= N*

MBs of *N-N2-1* from the beginning and MBs of *N-N1-1* from the end are used. MBs of the dark part are discarded.

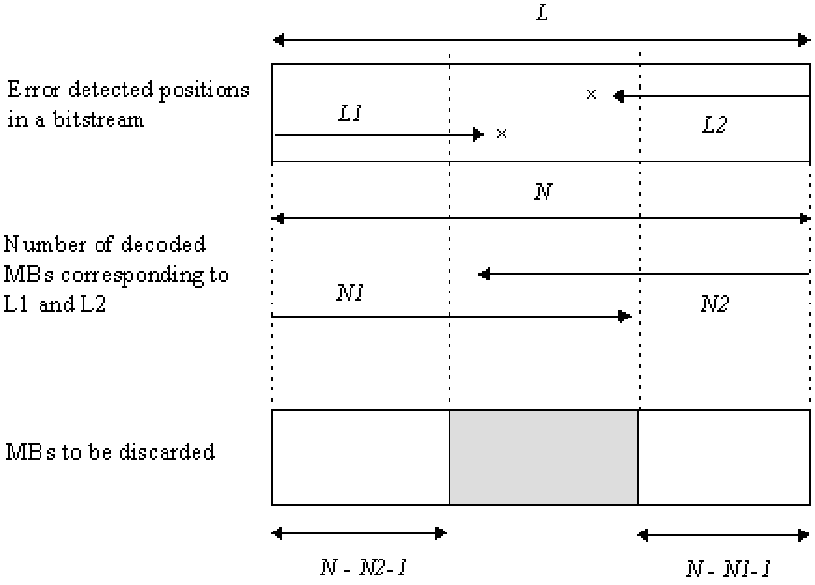

L
Error detected positions in a bitstream
L1
L2
N
Number of decoded MBs corresponding to L1 and L2
N1
N2
MBs to be discarded
N - N2-1
N - N1-1

Figure -5 -- RVLC decoding; Strategy 2

(3) Strategy 3 : *L1+L2 >= L* and *N1+N2 < N*

MBs of *N-b_mb(L2)* from the beginning and MBs of *N-f_mb(L1)* from the end are used. MBs of the dark part are discarded.

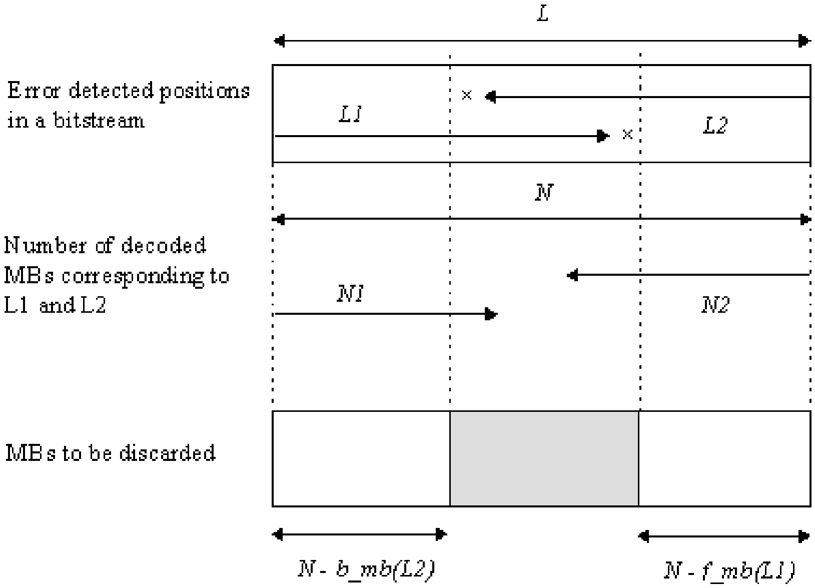

L
Error detected positions in a bitstream
L1
L2
N
Number of decoded MBs corresponding to L1 and L2
N1
N2
MBs to be discarded
N - b_mb(L2)
N - f_mb(L1)

Figure -6 -- RVLC decoding; Strategy 3

(4) Strategy 4 : *L1+L2 >= L* and *N1+N2 >= N*

MBs of *min{N-b_mb(L2), N-N2-1}* from the beginning and MBs of *min{N-f_mb(L1), N-N1-1}* from the end are used. MBs of the dark part are discarded.

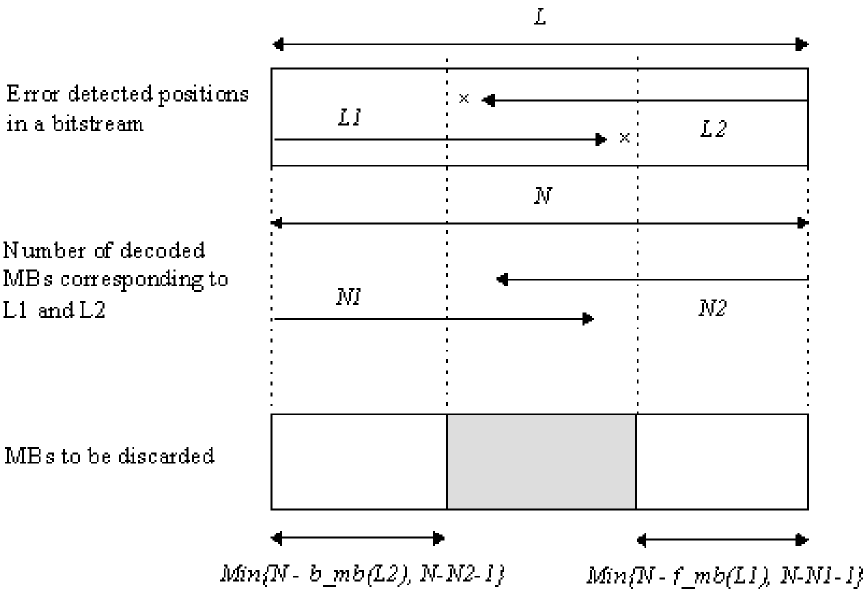

L
Error detected positions in a bitstream
L1
L2
N
Number of decoded MBs corresponding to L1 and L2
N1
N2
MBs to be discarded
Min{N - b_mb(L2), N-N2-1}
Min{N - f_mb(L1), N-N1-1}

Figure -7 -- RVLC decoding; Strategy 4

2. INTRA MBs within a bitstream

In the above strategies (Strategy 1 - Strategy 4), INTRA MBs are discarded even though they could have been decoded. An example of such a case is shown below.

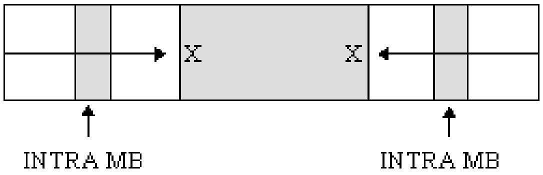


Figure -8 -- Intra MB discarding

Although these intra MBs are thought to be correct, the result of displaying an Intra MB that does contain an error can substantially degrade the quality of the video. Therefore, when a video packet is determined to contain errors, all Intra MBs are not displayed, but instead concealed.

1. Adaptive Intra Refresh (AIR) Method

The AIR is the technique of the intra refresh method for the error resilience. In the AIR, motion area is encoded frequently in Intra mode. Therefore, it is possible to recover the corrupted motion area quickly.

NOTE This informative technique is used for the rectangular VOP. It was designed in order to extract the motion area from the rectangular VOP at the encoder and encode it in intra mode frequently. Therefore, the quality of this area that is corrupted by the transmission error can be recovered quickly. As the objects are extracted in advance in the arbitrary shape coding mode, it

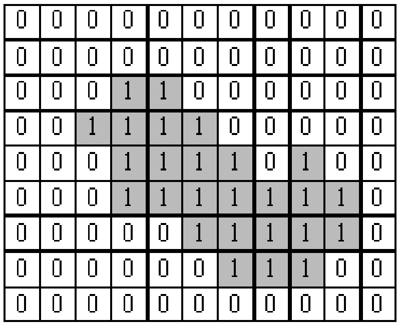
is possible to obtain the same performance by using the conventional cyclic intra refresh within the arbitrary shape.

The method of the "AIR"

The number of Intra MBs in a VOP is fixed and pre-determined. It depends on bitrates and frame rate and so on.

The encoder estimates motion of each MB and the only motion area is encoded in Intra mode. The results of the estimation are recorded to the Refresh Map MB by MB. The encoder refers to the Refresh Map and decides to encode current MB in Intra mode or not. The estimation of motion is performed by the comparison between SAD and SAD_th. SAD is the Sum of the Absolute Differential value between the current MB and the MB in same location of the previous VOP. The SAD has been already calculated in the Motion Estimation part. Therefore, additional calculation for the AIR is not needed. SAD_th is the threshold value. If the SAD of the current MB is larger than the SAD_th, this MB is regarded as motion area. Once the MB is regarded as motion area, it is regarded as motion area until it is encoded in Intra mode predetermined times. The predetermined value is recorded to the Refresh Map. (*See* Figure E-9. *In this figure, predetermined value is "1" as an example*)

The holizontal scan is used to determine the MBs to be encoded in Intra mode within the moving area (*see*Figure E-10).



0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0	0
0	0	0	1	1	1	1	0	1	0	0
0	0	0	1	1	1	1	1	1	1	0
0	0	0	0	0	1	1	1	1	1	0
0	0	0	0	0	0	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0

Figure -9 -- Refresh Map for QCIF

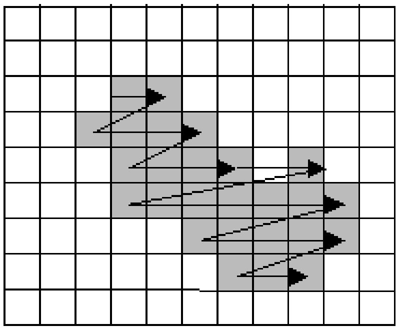


Figure -10 -- Scan order for the Adaptive Refresh

The processing of the "AIR"

The following is the explanation of the processing of AIR (seeFigure E-11). The fixed number of the Intra MB in a VOP should be determined in advance. Here, it is set to "2" as an example.

[1] 1st VOP ([a]~[b] in Figure E-11)
The all MBs in the 1st VOP are encoded in Intra mode [a]. The Refresh Map is set to "0", because there is no previous VOP [b].

[2] 2nd VOP ([c] ~[f])
The 2nd VOP is encoded as P-VOP. Intra refresh is not performed in this VOP, because all values in the Refresh Map is zero yet ([c] and [d]). The encoder estimates motion of each MB. If the SAD for current MB is larger than the SAD_th, it is regarded as motion area (hatched area in Figure E-11 [e]). And the Refresh Map is updated [f].

[3] 3rd VOP ([g] ~[k])
When the 3rd VOP is encoded, the encoder refers to the Refresh Map [g]. If the current MB is the target of the Intra refresh, it is encoded in Intra mode [h]. The value of the MB in Refresh Map is decreased by 1 [i]. If the decreased value is 0, this MB is not regarded as motion area. After this, the processing is as same as the 2nd VOP [j]~[k].

[4] 4th VOP ([l]~[p])
It is as same as 3rd VOP

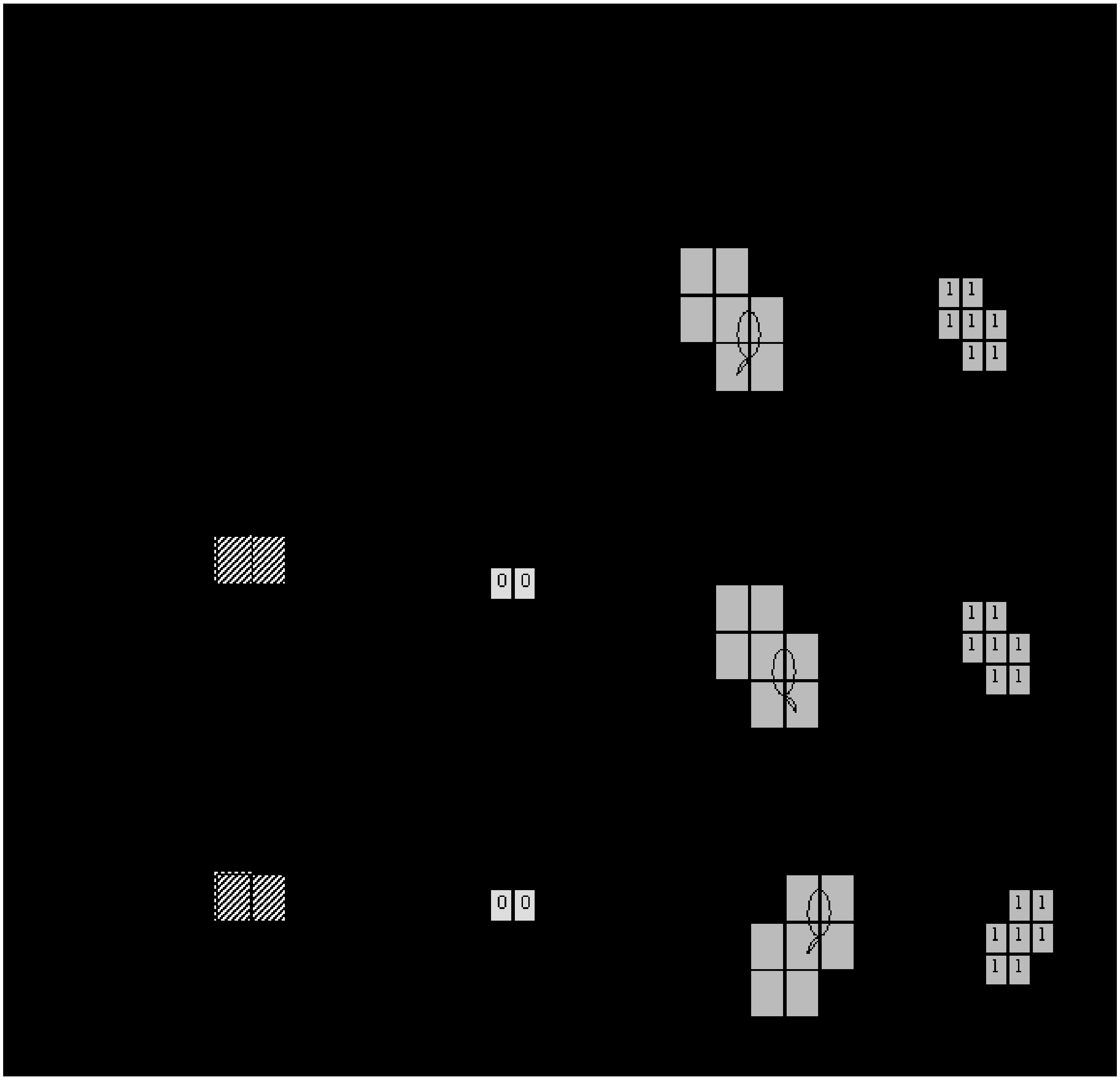


Figure -11 -- Explanation of AIR

1. Complexity Estimation

The Complexity Estimation Tool enables the estimation of decoding complexity without the need of the actual decoding of the incoming VOPs. The tool is based on the trasmission of the statistic of the actual encoding algorithms, modes and parameters used to encode the incoming VOP. The ?cost? in complexity for the execution of each algorithm is measured or eastimated on each decoder platform. The actual statistic of the decoding algorithms is transmitted in the video bitstream and can be converted by means of the mentioned ?costs? into the VOP actual decoding cost for the specific decoder.

The tool is flexible since it enables, for each VO, the definition of the set of used statistics. Such definition is done in the VO header. The actual values of the defined statistics is then inserted into each VOP header according to the ?complexity estimation syntax?.

2. **Resynchronization in Case of Unknown Video Header Format**

Two video object layer starting indicators are supported:

1. **video_object_layer_start_code, and**
2. **short_video_start_marker**

The automatic detection of which of the these byte aligned codes is present is unambiguous. The short_video_start_marker will never emulate a video_object_layer_start_code, since 23 byte-aligned zeros cannot occur in any video stream using the short_video_start_marker. The video_object_layer_start_code will also never emulate a short_video_start_marker, because its first non-zero bit is in a different location (provided byte alignment is not lost).

However, special attention needs to be paid if some application requires starting at any arbitrary point in a bitstream for which there is no prior knowledge of the format type. Although unlikely, a resync_marker can emulate a short_video_start_marker (for certain macroblock_number field lengths and macroblock_number values and vop_fcode_forward values).

Although the behavior of the decoder in these circumstances is not specified, it is suggested to perform validity checks on the first few bits beyond the short_video_start_marker if the video_object_layer_start_code is not the first starting indicator found. Numerous checks are possible, for example, checking the values of the bits 9, 10, 18-21 and 27 beyond the short_video_start_marker. The decoder may also choose to delay acquisition until an "I" vop-type is indicated in bit 17. Even simply discarding some data while searching for a video_object_layer_start_code prior to "timing out" with a decision to seek the short_video_start_marker may be acceptable for some applications.

A. (informative)

Preprocessing and postprocessing

1. Segmentation for VOP Generation

1. Introduction

The video coding scheme defined by this part of ISO/IEC 14496 offers several content-based functionalities, demanding the description of the scene in terms of so-called video-objects (VOs). The separate coding of the video objects may enrich the user interaction in several multimedia services due to flexible access to the bitstream and an easy manipulation of the video information. In this framework, the coder may perform a locally defined pre-processing, aimed at the automatic identification of the objects appearing in the sequence. Hence, segmentation is a key issue in efficiently applying the ISO/IEC 14496-2 coding scheme, although not affecting at all the bitstream syntax and thus not being a normative part of the standard.

Usually, the term segmentation denotes the operation aimed at partitioning an image or a video sequence into regions extracted according to a given criterion. In the case of video sequences, this partition should achieve the temporal coherence of the resulting sequence of object masks representing the video object. In the recent literature, different methods have been proposed for segmentation of video sequences, based on either a spatial homogeneity, a motion coherence criterion [[8]] or on joint processing of spatio-temporal information [4][[9]][[11]][[15]]. These algorithms are expected to identify classes of moving objects, according to some luminance homogeneity and motion coherence criterion.

In this annex, a framework aiming at an appropriate combination of temporal and spatial segmentation strategies, developed throughout the standardisation phase of ISO/IEC 14496 Version 1, is described. The description is given only for informative purposes as the technique to extract objects from the scene is not standardised. The classification of the pels in a video sequence is performed into two classes, namely moving objects (foreground) and background. This framework will continue to be investigated throughout the standardisation phase of ISO/IEC 14496 Version 2, leading to improved segmentation results. Only the general principles are shortly described, however, if more detail is required a number of references containing much more detailed descriptions are given.

2. Description of a combined temporal and spatial segmentation framework

Throughout the work on automatic segmentation of moving objects, different proposals for temporal and spatial segmentation algorithms have been proposed and investigated. This resulted at the end in a combined temporal and spatial segmentation framework [[6]] which is shown in a high level block diagram in Figure F-1.

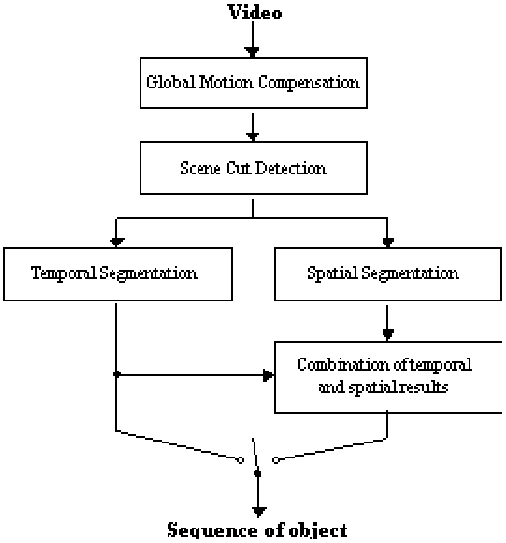


Figure -1 -- Block diagram of combined temporal and spatial segmentation framework

The combined scheme applies in a first step the general blocks of camera motion estimation and compensation [[17]][[18]] and scene cut detection [[13]] which can be seen as a kind of pre-processing in order to eliminate the influence of a moving camera.

In a second step, either temporal or combined spatio-temporal segmentation of each image are carried out, depending on the requirements. The reason for this is, that in general only performing temporal segmentation requires less computational complexity. On the other hand, taking into account also spatial segmentation leads to more accurate segmentation results, but increases the computational complexity of the segmentation.

For temporal segmentation, two possible algorithms are under consideration. It will be one main task for the group which will be working on segmentation for ISO/IEC 14496 Version 2, to decide which of these algorithms performs better. For spatial segmentation, only one algorithm is considered.

Finally, if temporal and spatial segmentation is performed, both temporal and spatial segmentation results are combined. It will be the second main task of the group to work out an appropriate algorithm for combining the temporal and spatial segmentation results.

The three algorithms for temporal and spatial segmentation will be shortly described in the following. For more details on them as well as on the possible combination approaches [[15]][[10]][[7]], the reader is referred to the given references, where more detailed descriptions can be found.

Temporal segmentation based on change detection: this segmentation algorithm [[16]][[17]][[18]], which is mainly based on a change detection, can be subdivided into two main steps, assuming that a possible camera motion has already been compensated: by the first step, a change detection mask between two successive frames is estimated. In this mask, pels for which the image luminance has changed due to a moving object are labelled as changed. For that, first an initial change detection mask between the two successive frames is generated by global thresholding the frame difference. After that, boundaries of changed image areas are smoothed by a relaxation technique using local adaptive thresholds [[1]][[2]]. Thereby, the algorithm adapts frame-wise automatically to camera noise. In order to finally get temporal stable object regions, an object mask memory with scene adaptive memory length is applied. Finally, the mask is simplified and small regions are eliminated, resulting in the final change detection mask.

In the second step, an object mask is calculated by eliminating the uncovered background areas from the change detection mask as in [[12]]. Therefore, displacement information for pels within the changed regions is used. The displacement is estimated by a hierarchical blockmatcher (HBM) [[3]]. For a higher accuracy of the calculated displacement vector field (DVF), the change detection mask from the first step is considered by the HBM. Pels are set to foreground in the object mask, if foot- and top-point of the corresponding displacement vector are both inside the changed area in the current CDM. If not, these pels are set to background. Results for the described method can be found in [[14]][[16]][[18]].

Temporal segmentation using higher order moments and motion tracking: The algorithm [8][[9]][[19]][[20]] produces the segmentation map of each frame f_{k} of the sequence by processing a group of frames $\{ f_{\mathrm{k-i}},\ i=0,..n \}$. The number of frames n varies on the basis of the estimated object speed [8]. For each frame f_{k}, the algorithm splits in three steps. First, the differences $\{ d_{\mathrm{k-j}}(x,y)=f_{\mathrm{k-j}}(x,y)-f_{\mathrm{k-n}}(x,y),\ j=0,..n\text{-}1 \}$ of each frame of the group with respect to the first frame $f_{\mathrm{k-n}}$ are evaluated in order to *detect the changed areas*, due to object motion, uncovered background and noise. In order to reject the luminance variations due to noise, an Higher Order Statistic test is performed. Namely, for each pixel (*x*,*y*) the fourth-order moment $\hat{m}_d^{(4)}(x,y)$of each inter-frame difference *d*(*x*,*y*) is estimated on a *3*x*3* window, it is compared with a threshold adaptively set on the basis of the estimated background activity [8], and set to zero if it is below the threshold. Then, on the sequence of the thresholded fourth-order moment maps $\tilde{m}_{d_{i-j}}^{(4)}(x,y)$, a *motion detection* procedure is performed. This step aims at distinguish changed areas representing uncovered background (which stands still in the HOS maps) and moving objects (moving in the HOS maps). At the *j*-th iteration, the pair of thresholded HOS maps $\tilde{m}_{d_{i-j}}^{(4)}(x,y), \tilde{m}_{d_{i-j-1}}^{(4)}(x,y)$ is examined. For each pixel (*x*,*y*) the displacement of is evaluated on a 3x3 window, adopting a SAD criterion, and if the displacement is not null the pixel is classified as moving. Then, the lag *j* is increased (i.e. the pair of maps slides) and the motion analysis is repeated, until *j* = *n-2* . Pixels presenting null displacements on all the observed pairs are classified as still. Finally, a *regularization* algorithm re-assigns still regions, internal to moving regions, to foreground and refines the segmentation results imposing a priori topological constraints on the size of objects irregularities such as holes, isthmi, gulfs and isles by morphological filtering. A post-processing operation refines the results on the basis of spatial edges.

Spatial segmentation based on watershed algorithm : In the spatial segmentation, images are first simplified to make easier the image segmentation [[21]]. Morphological filters are used for the purpose of image simplification. These filters remove regions that are smaller than a given size but preserve the contours of the remaining objects. By the second step, the spatial gradient of the simplified image is approximated by the use of a morphological gradient operator [[21]]. The spatial gradient can be used as an input of watershed algorithm to partition an image into homogeneous intensity regions. For the problem of ambiguous boundaries by spatial gradient, we incorporate color information into gradient computation in which the largest values among the weighed gradients obtained in $\omega Y, \beta C_r, \gamma C_b$are chosen [[5]]. In the boundary decision step, the boundary decision is taken through the use of a watershed algorithm that assigns pixels in the uncertainty area to the most similar region with some segmentation criterion such as difference of intensity values [[22]]. To merge into semantic regions the genetically over-segmented regions from watershedding, a region merging algorithm is then incorporated [[5]]. The final output of the spatial segmentation is the images that are composed of semantically meaningful regions with precise boundaries. Moving objects are therefore represented with semantic regions with precise boundaries and can be segmented in conjunction with temporal information that localizes the moving objects.

3. References

1. T. Aach, A. Kaup, R. Mester, ?Statistical model-based change detection in moving video", *Signal Processing*, Vol. 31, No. 2, pp. 165-180, March 1993.

2. T. Aach, A. Kaup, R. Mester, ?Change detection in image sequences using Gibbs random fields: a Bayesian approach", *Proceedings Int. Workshop on Intelligent Signal Processing and Communication Systems*, Sendai, Japan, pp. 56-61, October 1993.

3. M. Bierling, ?Displacement estimation by hierarchical blockmatching", *3rd SPIE Symposium on Visual Communications and Image Processing*, Cambridge, USA, pp. 942-951, November 1988.

4. P. Bouthemy, E. François, ?Motion segmentation and qualitative dynamic scene analysis from an image sequence" in Int. Journal of Computer Vision vol.10, no.2, pp157-182, 1993.

5. J. G. Choi, M. Kim, M. H. Lee, C. Ahn, **?**Automatic segmentation based on spatio-temporal information", Doc. ISO/IEC JTC1/SC29/WG11 MPEG97/2091, April 1997.

6. J. G. Choi, M. Kim, M. H. Lee, C. Ahn (ETRI); S. Colonnese, U. Mascia, G. Russo, P. Talone (FUB); Roland Mech, Michael Wollborn (UH), **?**Merging of temporal and spatial segmentation", Doc. ISO/IEC JTC1/SC29/WG11 MPEG97/2383, July 1997.

7. J. G. Choi, M. Kim, M. H. Lee, C. Ahn, ?New ETRI results on core experiment N2 on automatic segmentation techniques", Doc. ISO/IEC JTC1/SC29/WG11 MPEG97/2641, October 1997.

8. S. Colonnese, A. Neri, G. Russo, C. Tabacco, ?Adaptive Segmentation of Moving Object versus Background for Video Coding", Proceedings of SPIE Annual Symposium, Vol. 3164, San Diego, August 1997.

9. S. Colonnese, U. Mascia, G. Russo, P. Talone, ?Core Experiment N2: Preliminary FUB results on combination of automatic segmentation techniques", Doc. ISO/IEC JTC1/SC29/WG11 MPEG97/2365, July 1997.

10. S. Colonnese, U. Mascia, G. Russo, ?Automatic segmentation techniques: updated FUB results on core experiment N2", Doc. ISO/IEC JTC1/SC29/WG11 MPEG97/2664, October 1997.

11. J. Dugelay, H. Sanson, ?Differential methods for the identification of 2D and 3D motion models in image sequences", *Signal Processing*, Vol.7, pp. 105-127, Sept. 1995.

12. M. Hötter, R. Thoma, ?Image Segmentation based on object oriented mapping parameter estimation", *Signal Processing*, Vol. 15, No. 3, pp. 315-334, October 1988.

13. M. Kim, J. G. Choi, M. H. Lee, C. Ahn; **?**Performance analysis of an ETRI?s global motion compensation and scene cut detection algorithms for automatic segmentation", Doc. ISO/IEC JTC1/SC29/WG11 MPEG97/2387, July 1997.

14. R. Mech, P. Gerken, ?Automatic segmentation of moving objects (Partial results of core experiment N2), Doc. ISO/IEC JTC1/SC29/WG11 MPEG97/1949, April 1997.

15. R. Mech, M. Wollborn, ?Automatic segmentation of moving objects (Partial results of core experiment N2)", Doc. ISO/IEC JTC1/SC29/WG11 MPEG97/2703, October 1997.

16. R. Mech, M. Wollborn, ?A Noise Robust Method for Segmentation of Moving Objects in Video Sequences", *International Conference on Acoustic*, *Speech and Signal*, Munich, Germany, April 1997.

17. R. Mech, M. Wollborn, ?A Noise Robust Method for 2D Shape Estimation of Moving Objects in Video Sequences Considering a Moving Camera", *Workshop on Image Analysis for Multimedia Interactive Services*, Louvain-la-Neuve, Belgium, June 1997.

18. R. Mech, M. Wollborn, ?A Noise Robust Method for 2D Shape Estimation of Moving Objects in Video Sequences Considering a Moving Camera", accepted for publication in *Signal Processing: Special Issue on Video Sequence Segmentation for Content-based Processing and Manipulation*, to be published in the beginning of 1998.

19. A. Neri, S. Colonnese, G. Russo, ?Video Sequence Segmentation for Object-based Coders using Higher Order Statistics", ISCAS ?97, Hongkong, June 1997.

20. A. Neri, S. Colonnese, G. Russo, ?Automatic Moving Objects and Background Segmentation by means of Higher Order Statistics", IS&T Electronic Imaging ?97 Conference: Visual Communication and Image Processing, San Jose?, 8-14 February 1997, SPIE Vol. 3024.

21. P. Salembier, M. Pardàs, ?Hierarchical Morphological Segmentation for Image Sequence Coding", IEEE Transactions on Image Processing, Vol.3, No.5, pp. 639-651, September 1994.

22. Luc Vincent, Pierre Soille, ?Watersheds in digital spaces: an efficient algorithm based on immersion simulations", *IEEE Transactions on PAMI*, Vol.13, No. 6, pp. 583-598, June 1991.

1. Bounding Rectangle of VOP Formation

This clause describes the bounding rectangle of VOP formation process. The formation of the bounding rectangle of VOP is based on the segmented shape information. The following explains the process to achieve the bounding rectangle of a VOP in such a way that the minimum number of macroblocks containing the object will be attained in order to achieve a higher coding efficiency.

1. Generate the tightest rectangle whose vertical and horizontal positions of the top-left point are even numbered. In case of interlaced mode, the vertical position of the top-left point of the rectangle is a multiple of 4.

2. If the top left position of this rectangle is not the same as the origin of the image, the following steps have to be performed. Otherwise no further processing is necessary.

3. Form a control macroblock at the top left corner of the tightest rectangle as shown in Figure F-2.

4. Count the number of macroblocks that completely contain the VOP for all the points of which vertical and horizonal positions are even numbered (in case of interlaced mode, all the points of which horizontal positions are even numbered and vertical positions are multiple of 4) of the control macroblock using the following procedure:

- Generate a bounding rectangle from the control point to the right bottom side of the VOP which consists of multiples of 16x16 blocks.

- Count the number of macroblocks in this bounding rectangle, which contain at least one object pel. To do so, it would suffice to take into account the boundary pels of a macroblock only.

1. Select the control point that results in the smallest number of non transparent macroblocks for the given object.

2. Extend the top left coordinate of the tightest rectangle generated in Figure F-2. to the selected control coordinate. This will create a rectangle that completely contains the object but with the minimum number of non transparent macroblocks in it. The VOP horizontal and vertical spatial references are taken directly from the modified top-left coordinate.

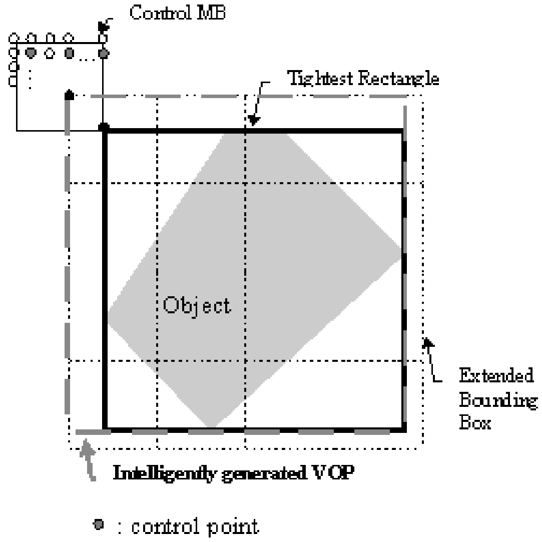


Figure -2 -- Intelligent VOP Formation

1. Postprocessing for Coding Noise Reduction

The post-filter consists of deblocking filter and deringing filter. Either one or both of them can be turned on as needed.

1. Deblocking filter

The filter operations are performed along the 8x8 block edges at the decoder as a post-processing operation. Luminance as well as chrominace data is filtered. Figure F-3 shows the block boundaries.

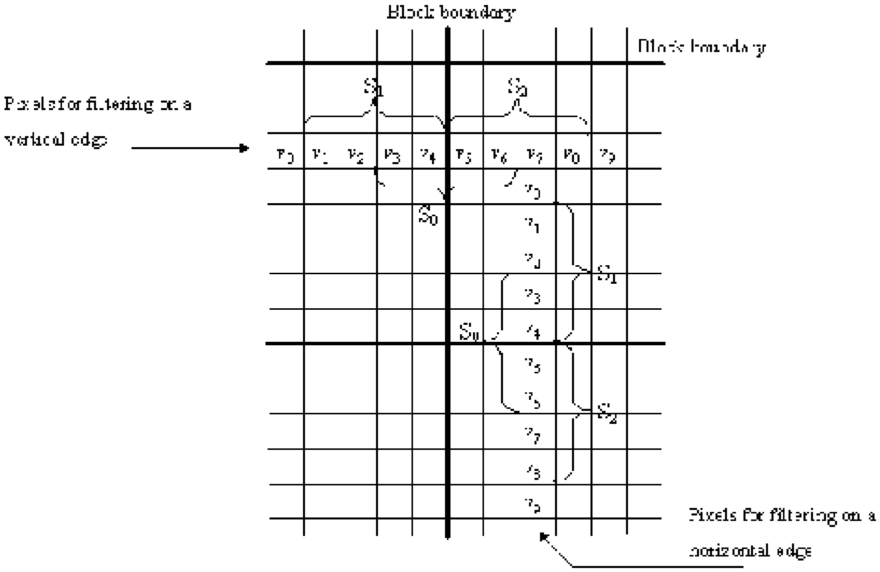

Block boundary
Block boundary
Pixels for filtering on a vertical edge
S_1
S_2
v_0 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9
S_0
v_0
v_1
v_2
v_3
S_0
v_4
v_5
v_6
v_7
v_8
v_9
S_1
S_2
Pixels for filtering on a horizontal edge

Figure -3 -- Boundary area around block of interest

In the filter operations, two modes are used separately depending on the pixel conditions around a boundary. The following procedure is used to find a very smooth region with blocking artifacts due to small dc offset and to assign it a DC offset mode. In the other case, default mode operations are applied.

eq_cnt = f (v0- v1) + f (v1- v2) + f (v2- v3) + f (v3- v4) + f (v4- v5) + f (v5- v6) + f (v6- v7)

+ f (v7- v8) + f (v8- v9),

where f (g) = 1 if |g | £ THR1 and 0 otherwise.

If (eq_cnt ³ THR2)

DC offset mode is applied,

else

Default mode is applied.

For the simulation, threshold values of THR1 = 2 and THR2 = 6 are used.

In the default mode, a signal adaptive smoothing scheme is applied by differentiating image details at the block discontinuities using the frequency information of neighbor pixel arrays, S_0, S_1, and S_2,. The filtering scheme in default mode is executed by replacing the boundary pixel values v_4 and v_5 with $v_{4¢}$ and $v_{5¢}$ as follows:

$v_{4¢} = v_{4-}\, d,$

$v_{5¢} = v_5 + d,$

and d = CLIP(5× ($a_{3,0¢\ -}\ a_{3,0}$)//8, 0, ($v_{4-}\ v_5$)/2) × d (|$a_{3,0}$| < QP)

where $a_{3,0¢}$ = SIGN($a_{3,0}$) × MIN(|$a_{3,0}$|, |$a_{3,1}$|, |$a_{3,2}$|).

Frequency components $a_{3,0}$, $a_{3,1}$, and $a_{3,2}$ can be evaluated from the simple inner product of the approximated DCT kernel [2 -5 5 -2] with the pixel vectors, i.e.,

a3,0 = ([2 -5 5 -2] · [v3 v4 v5 v6]T) // 8,

a3,1 = ([2 -5 5 -2] · [v1 v2 v3 v4]T) // 8,

a3,2 = ([2 -5 5 -2] · [v5 v6 v7 v8]T) // 8.

Here CLIP(x,p,q) clips x to a value between p and q; and QP denotes the quantization parameter of the macroblock where pixel v_5 belongs. d(*condition*)=1 if the "*condition*" is true and 0 otherwise..

In very smooth region, the filtering in the default mode is not good enough to reduce the blocking artifact due to dc offset. So we treat this case in the DC offset mode and apply a stronger smoothing filter as follows :

max = MAX (v1, v2, v3, v4, v5, v6, v7, v8),

min = MIN (v1, v2, v3, v4, v5, v6, v7, v8),

if (|max- min| < 2× QP) {

$$v'_n = \sum_{k=-4}^{4} b_k \cdot p_{n+k}, 1 \le n \le 8$$

$$p_m = \begin{cases} (|v_1 - v_0| < QP) ? v_0 : v_1, if \quad m < 1 \\ v_m, \quad if 1 \le m \le 8 \\ (|v_8 - v_9| < QP) ? v_9 : v_8, if \quad m > 8 \end{cases}$$

$$\{b_k : -4 \le k \le 4\} = \{1,1,2,2,4,2,2,1,1\} // 16$$

}

else

No change will be done.

The above filter operations are applied for all the block boundaries first along the horizontal edges followed by the vertical edges. If a pixel value is changed by the previous filtering operation, the updated pixel value is used for the next filtering.

2. **Deringing filter**

This filter comprises three subprocesses; threshold determination, index acquisition and adaptive smoothing. This filter is applied to the pixels on 8x8 block basis. More specifically 8x8 pixels are processed by referencing 10x10 pixels at each block. The following notation is used to specify the six blocks in a macroblock. For instance, block[5] corresponds to the *Cb* block whereas block[*k*] is used as a general representation in the following subclauses.

1. **Threshold determination**

   Firstly, calculate maximum and minimum gray value within a block in the decoded image. Secondary, the threshold denoted by *thr[k]* and the dynamic range of gray scale denoted by *range[k]* are set:

   $$thr[k] = \left(\max imum[k] + \min imum[k] + 1\right) / 2$$

   $$range[k] = \max imum[k] - \min imum[k]$$

   An additional process is done only for the luminance blocks. Let *max_range* be the maximum value of the dynamic range among four luminance blocks.

   $$\max\_range = range[k_{\max}]$$

   Then apply the rearrangement as follows.

```
for( k=1 ; k<5 ; k++ ){

if( range[k] < 32 && max_range > =64 )

thr[k] = thr[kmax];

if( max_range<16 )

thr[k] = 0;

}
```

2. **Index acquisition**

   Once the threshold value is determined, the remaining operations are purely 8x8 block basis. Let *rec(h,v)* and *bin(h,v)* be the gray value at coordinates *(h,v)* where *h,v=0,1,2,...,7* , and the corresponding binary index, respectively. Then *bin(h,v)* can be obtained by:

   $$bin(h,v) = \begin{cases} 1 & if\ rec(h,v) \geq thr \\ 0 & otherwise \end{cases}$$

   Note that *(h,v)* is use to address a pixel in a block, while *(i,j)* is for accessing a pixel in a 3x3 window.

3. **Adaptive smoothing**
   1. **Adaptive filtering**

      The figure below is the binary indices in 8x8 block level, whereas practically 10x10 binary indices are calculated to process one 8x8 block.

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1	1
1	1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	0	0

Figure -4 -- Example of adaptive filtering and binary index

The filter is applied only if the binary indices in a 3x3 window are all the same, i.e., all "0" indices or all "1" indices. Note 10x10 binary indices are obtained with a single threshold which corresponds to the 8x8 block shown in the above figure, where the shaded region represents the pixels to be filtered.

The filter coefficients used for both intra and non-intra blocks denoted by *coef(i,j)*, where *i,j=-1,0,1*, are:

1	2	1
2	4	2
1	2	1

Figure -5 -- Filter mask for adaptive smoothing

Here the coefficient at the center pixel, i.e., *coef(0,0)*, corresponds to the pixel to be filtered. The filter output *flt?(i,j)* is obtained by:

$$flt'(h,v) = \{8 + \sum_{i=-1}^{1} \sum_{j=-1}^{1} coef(i,j) \cdot rec(h+i, v+j)\} // 16$$

2. Clipping

The maximum gray level change between the reconstructed pixel and the filtered one is limited according to the quantization parameter, i.e., QP. Let *flt(h,v)* and *flt?(h,v)* be the filtered pixel value and the pixel value before limitation, respectively.

```
if( flt?(h,v) - rec(h,v) > max_diff )

flt(h,v) = rec(h,v) + max_diff

else if( flt?(h,v) - rec(h,v) < -max_diff )

flt(h,v) = rec(h,v) - max_diff

else

flt(h,v) = flt?(h,v)
```

where max_diff=QP/2 for both intra and inetr macroblocks.

3. Further issues

In order to reduce the number of computations in post-filtering, two kinds of semaphores can be defined: the blocking semaphores and the ringing semaphore. Depending on the blocking semaphores, the horizontal and the vertical deblocking filtering is applied strongly and weakly on the block boundary. If the ringing semaphore (RS) of a current block is "1", deringing filtering is applied. For extracting the semaphores in intra-frame, when only a DC component in the 8x8 inverse quantized coefficients (IQC), the DCT coefficients after inverse quantization, has a non-zero value, both the horizontal blocking semaphore (HBS) and the vertical blocking semaphore (VBS) of the block are set to "1". When only the coefficients in the top row of the IQC have non-zero values, the VBS is set to "1". When only the coefficients in the far left column of the IQC have non-zero values, the HBS is set to "1". The RS is set to "1" if any non-zero coefficient exists in positions other than a DC component, the first horizontal AC component, and the first vertical AC comonent. Also the semaphores of the inter-frame are calculated from both the residual signal and the semaphores of the reference frame by using the motion vector.

[1] M2723 Y.L. Lee et al.

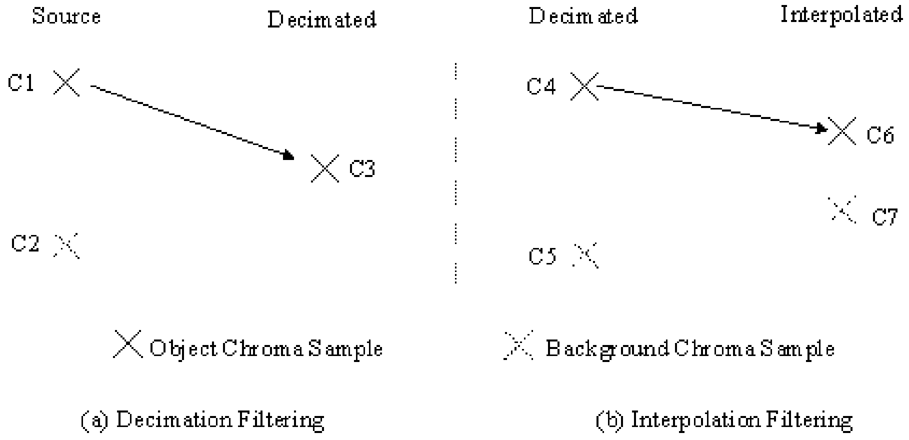
2. Chrominance Decimation and Interpolation Filtering for Interlaced Object Coding

The chrominance decimation filtering and the chrominance interpolation filtering are performed on the basis of the same field in case of interlaced object-based coding during the image format conversion process from 4:2:2 to 4:2:0 and from 4:2:0 to 4:2:2, respectively. In case that filter taps contain only object samples, conventional filtering methods are applied. When filter taps contain both object and background samples, the following decimation and interpolation filtering methods

are applied:

Decimation filtering: The source object and background chroma samples are identified from the luminance shape information. Among the decimated chroma samples, the object chroma samples are identified by the field-based chroma subsampling method described in subclause 7.6.1.5. When filter taps contain both of object and background samples at the object boundary, only chroma sample values of object are used for obtaining its decimated chroma sample value. For example, in case of a two-tap filter, one tap is on an object sample and the other is on a background sample, the resulting chroma sample is obtained using only the object chroma sample, not counting on the background sample. In Figure F-6 (a), the source chroma sample C1 is object and C2 is background. The value of the resulting decimated chroma sample C3 is the value of C1.

Interpolation filtering: In the interpolation filtering process which is the inverse of the decimation filtering, object chroma samples are identified among the interpolated chroma samples from the luminance shape information and the values of the interpolated object chroma samples are obtained applying an interpolation filter on input decimated chroma samples. When filter taps include both of decimated object and background chroma samples, only the decimated object chroma samples are counted. For example, in case of a two-tap filter in Figure F-6 (b), two decimated chroma samples, C4 and C5, are used for obtaining the interpolated chroma samples C6 and C7. Suppose that C6 is identified as object and C7 is identified as background from the luminance shape information and the chroma sample C4 is object and C5 is background, then the interpolated chroma sample C6 is obtained as the value of the chroma sample C4. The interpolation for C7 is not necessary because it is background.



(a) Decimation Filtering

(b) Interpolation Filtering

Figure -6 -- Chrominance Decimation and Interpolation Filtering at the Object Boundaries

A.

(normative)

Profile and level indication and restrictions

This annex specifies the syntax element restrictions and permissible layer combinations.

Table -1 -- FLC table for profile_and_level_indication

Profile/Level	**Code**

Reserved	00000000
Simple Profile/Level 1	00000001
Simple Profile/Level 2	00000010
Simple Profile/Level 3	00000011
Reserved	00000100 - 00010000
Simple Scalable Profile/Level 1	00010001
Simple Scalable Profile/Level 2	00010010
Reserved	00010011 - 00100000
Core Profile/Level 1	00100001
Core Profile/Level 2	00100010
Reserved	00100011 - 00110001
Main Profile/Level 2	00110010
Main Profile/Level 3	00110011
Main Profile/Level 4	00110100
Reserved	00110101 - 01000001
N-bit Profile/Level 2	01000010
Reserved	01000011 - 01010000
Scalable Texture Profile/Level 1	01010001
Reserved	01010010 - 01100000
Simple Face Animation Profile/Level 1	01100001
Simple Face Animation Profile/Level 2	01100010
Reserved	01100011 - 01110000
Basic Animated Texture Profile/Level 1	01110001
Basic Animated Texture Profile/Level 2	01110010
Reserved	01110011 - 10000000
Hybrid Profile/Level 1	10000001
Hybrid Profile/Level 2	10000010
Reserved	10000011 - 11111111

Table -2 -- possible combination of two tools

Tools		**BA**	**B-VOP**	**OBMC**	**Q**	**ER**	**SH**	**BS**	**GS**	**IN**	**NB**	**TS(B)**	**TS(E**
Basic · I-VOP · P-VOP · AC/DC Prediction · 4-MV, Unrestricted MV	BA		4	4	4	4	4 a)	4	4	4	4	4	4
B-VOP	BV			4	4	4	8	4	4	4	4	4	4
P-VOP with OBMC (Texture)	OBMC				4	4	8	4	4	4	4	4	4
Method 1/Method 2 Quantization	Q					4	8	4	4	4	4	4	4
Error resilience	ER						8	4	4 d)	4 e)	4	4	8
Short Header	SH							8	8	8	8	8	8
Binary Shape (progressive)	BS								8	4	4	4	4
Greyscale Shape	GS									4	D	D	D
Interlace	IN										4	D	D
N-Bit	NB											4 c)	4 c)
Temporal Scalability (Base)	TS(B)												
Temporal Scalability (Enhancement)	TS(E)												

Spatial Scalability (Base)	SS(B)												
Spatial Scalability (Enhancement)	SS(E)												
Sprite	SP												
Still Texture	ST												

4 - This combination is supported by the syntax. (Subject to further restrictions imposed by the definition of visual object types).

8 - This combination is not supported by the syntax.

D - This combination is supported by the syntax but not allowed within one Visual Object(). This combination may be supported in the future, if requirements arise for the corresponding combination.

H - supported at a higher syntactic level.

a. Short Header does not support DC/AC Prediction, 4MV nor Unrestricted MV
b. Sprite does not support 4MV nor Unrestricted MV.
c. P-VOP Temporal Scalability only.
d. Grey Scale does not support Data Partitioning nor RVLC
e. Interlace does not support Data Partitioning nor RVLC

A. (informative)

Patent statements

The user’s attention is called to the possibility that, for some of the processes specified in this part of ISO/IEC 14496, conformance with this specification may require use of an invention covered by patent rights.

By publication of this part of ISO/IEC 14496, no position is taken with respect to the validity of this claim or of any patent rights in connection therewith. Information regarding such patents can be obtained from the following organisations.

1. Patent statements

The table summarises the formal patent statements received and indicates the parts of the standard to which the statement applies. (S: Systems, V: Visual, A: Audio, R: Reference Software, D: DMIF) The list includes all organisations that have submitted informal patent statements. However, if no "X" is present, no formal patent statement has yet been received from that organisation.

		S	**V**	**A**	**R**	**D**
	Alcatel	x	x	x	x	x
	AT&T					
	BBC	x	x	x	x	x

	Bosch		x	x	x	
	British Telecommunications	x	x	x	x	x
	Canon	x	x	x	x	x
	CCETT	x	x	x	x	x
	Columbia University	x	x	x	x	x
	Creative	x		x	x	
	CSELT			x		
	DEmoGraFX		x		x	
	DirecTV	x	x	x		
	Dolby	x	x	x	x	x
	EPFL	x	x	x		
	ETRI	x	x	x	x	x
	FhG	x	x	x	x	x
	France Telecom	x	x	x	x	x
	Fujitsu Limited	x	x	x	x	x
	GC Technology Corporation	x	x	x		
	General Instrument		x		x	
	Hitachi	x	x	x	x	x
	Hyundai	x	x	x	x	x
	IBM					
	Institut für Rundfunktechnik	x	x	x		x
	InterTrust					
	JVC	x	x	x	x	x
	KDD Corporation	x	x			
	KPN	x	x	x	x	x

	LG Semicon					
	Lucent					
	Matsushita	x	x	x	x	x
	Microsoft	x	x	x	x	x
	MIT					
	Mitsubishi	x	x	x	x	
	Motorola		x		x	
	NEC Corporation	x	x	x	x	x
	NHK	x	x	x	x	x
	Nokia		x	x	x	
	NTT	x	x	x	x	x
	OKI	x	x	x	x	x
	Philips	x	x	x	x	x
	PictureTel Corporation		x		x	
	Rockwell	x	x	x	x	x
	Samsung	x	x	x		
	Sarnoff	x	x	x	x	x
	Scientific Atlanta	x	x	x	x	x
	Sharp	x	x	x	x	x
	Siemens	x	x	x		
	Sony	x	x	x	x	x
	Telenor	x	x	x	x	x
	Teltec DCU		x		x	
	Texas Instruments					
	Thomson	x	x	x		

	Toshiba		x			
	Unisearch Ltd.		x		x	
	Vector Vision		x			

A.

(informative)

Bibliography

1. Arun N. Netravali and Barry G. Haskell "*Digital Pictures, representation and compression*" Plenum Press, 1988
2. See the Normative Reference for Recommendation ITU-R BT.601
3. See the Normative Reference for IEC Standard Publication 461
4. See the Normative Reference for Recommendation ITU-T H.263
5. See the Normative reference for IEEE Standard Specification P1180-1990
6. ISO/IEC 10918-1:1994 | ITU-T T.81, *Information technology - Digital compression and coding of continuous-tone still images: Requirements and guidelines.*
7. Barry G. Haskell, Atul Puri, Arun N. Netravali, "*Digital Video: An Introduction to MPEG-2*," Chapman & Hall, ISBN 0-412-08411-2, 1997.
8. F. I. Parke, K. Waters, *Computer Facial Animation*, A K Peters, Wellesley, MA, USA, 1996

A. (normative)

View dependent object scalability

1. Introduction

Coding of View-Dependent Scalability (VDS) parameters for texture can provide for efficient incremental decoding of 3D images (e.g. 2D texture mapped onto a gridded 3D mesh such as terrain). Corresponding tools from ISO/IEC 14496-1 and ISO/IEC 14496-2 are used in conjunction with downstream and upstream channels of a decoding terminal. The combined capabilities provide the means for an encoder to react to a stream of viewpoint information received from a terminal. The encoder transmits a series of coded textures optimized for the viewing conditions which can be applied in the rendering of textured 3D meshes by the receiving terminal. Each encoded view-dependent texture (initial texture and incremental updates) typically corresponds to a specific 3D view in the user?s viewpoint that is first transmitted from the receiving terminal.

A tool defined in ISO/IEC 14496-1 transmits 3D viewpoint parameters in the upstream channel back to the encoder. The encoder’s response is a frequency-selective, view-dependent update of DCT coefficients for the 2D texture (based upon view-dependent projection of the 2D texture in 3D) back to the receiving terminal, along the downstream channel, for decoding by a Visual DCT tool at the receiving terminal. This bilateral communication supports interactive server-based refinement of texture for low-bandwidth transmissions to a decoding terminal that renders the texture in 3D for a user controlling the viewpoint movement. A gain in texture transmission efficiency is traded for longer closed-loop latency in the rendering of the textures in 3D. The terminal coordinates inbound texture updates with local 3D renderings, accounting for network delays so that texture cached in the terminal matches each rendered 3D view.

A method to obtain an optimal coding of 3D data is to take into account the viewing position in order to transmit only the most visible information. This approach reduces greatly the transmission delay, in comparison to transmitting all scene texture that might be viewable in 3D from the encoding database server to the decoder. At a given time, only the most important information is sent, depending on object geometry and viewpoint displacement. This technique allows the data to be streamed across a network, given that a upstream channel is available for sending the new viewing conditions to the remote database. This principle is applied to the texture data to be mapped on a 3D grid mesh. The mesh is first downloaded into the memory of the decoder using the appropriate BIFS node, and then the DCT coefficients of the texture image are updated by taking into account the viewing parameters, i.e. the field of view, the distance and the direction to the viewpoint.

2. Decoding Process of a View-Dependent Object

This clause explains the process for decoding the texture data using the VDS parameters. In order to determine which of the DCT coefficients are to be updated, a "mask", which is a simple binary image, shall be computed. The first step is to determine the viewing parameters obtained from the texture-mesh composition procedure that drives 3D rendering in the user’s decoding terminal. These parameters are used to construct the DCT mask corresponding to the first viewpoint of the session (VD mask). This mask is then updated with differential masks, built with the new viewing parameters that allow the texture image to be streamed. The bitstream syntax for view parameters and incremental transmission of DCT coefficients is given elsewhere in ISO/IEC 14496-1 and ISO/IEC 14496-2.

1. **General Decoding Scheme**

   The following subclauses outline the overall process for the decoder and encoder to accomplish the VDS functionalities.

   1. **View-dependent parameters computation**

      The VDS parameters (a and b angles, distance d for each cell) shall be computed using the geometrical parameters (Mesh, Viewpoint, Aimpoint, Rendering window). These parameters shall be computed for each cell of the grid mesh.

   2. **VD mask computation**

      For each 8x8 block of texture elements within a 3D mesh cell, the locations of the visible DCT coefficients inside the DCT block shall be computed using a and b angles, and the distance d defined for each cell relative to the viewpoint. The result shall be put in a binary mask image.

   3. **Differential mask computation**

      With the knowledge of which DCT coefficients have already been received (Binary mask buffered image) and which DCT coefficients are necessary for the current viewing conditions (Binary VD mask image), the new DCT coefficients shall be determined (Binary Differential mask image) as described in subclause J.2.4.

   4. **DCT coefficients decoding**

      The Video Intra bitstream, in the downstream channel, shall be decoded by the receiver terminal to obtain the DCT coefficients (DCT image). The decoding procedure is described in subclause J.2.5

.

5. **Texture update**

   The current DCT buffer in the receiver terminal shall be updated according to the Differential mask, using the received DCT image. The new received DCT coefficients shall be added to the buffered DCT image.

6. IDCT

   The Inverse DCT of the updated DCT image shall computed, as specified in subclause J.2.7 , to obtain the final texture.

7. **Rendering**

The texture is mapped onto the 3D mesh and the rendering of the scene is done, taking into account the mesh and the viewing conditions. This part of the procedure is outside the scope of this part of ISO/IEC 14496.

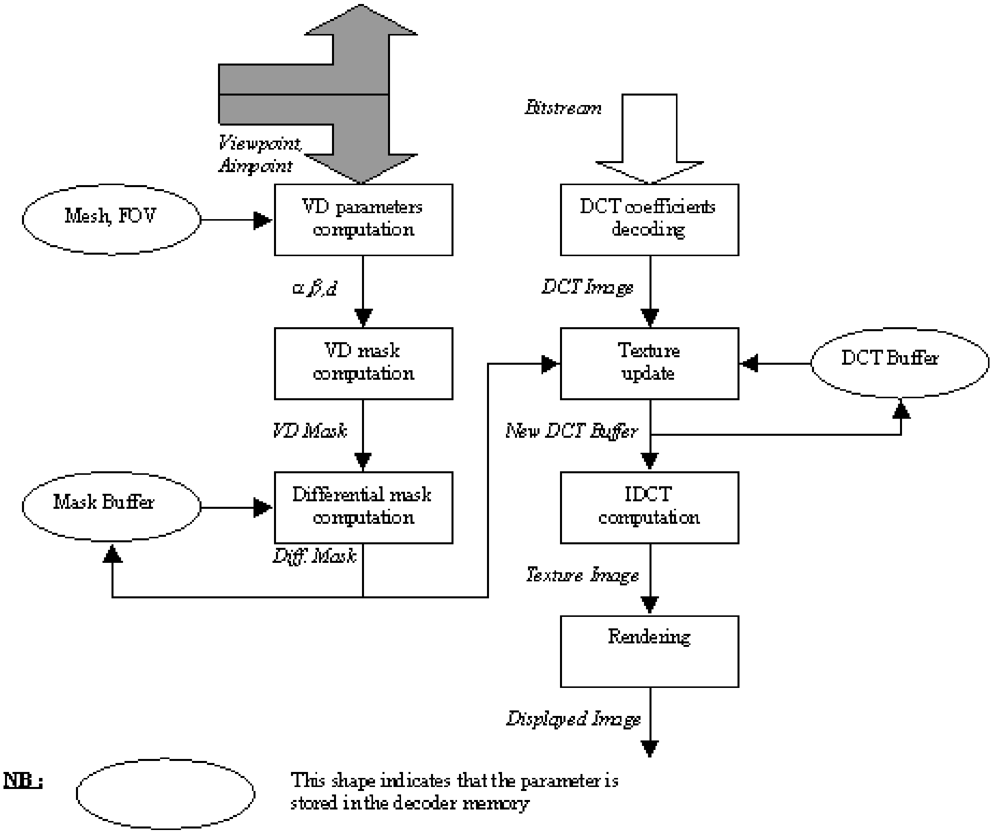


Figure -1 -- General Decoding Scheme of a View-Dependent Object

2. Computation of the View-Dependent Scalability parameters

The VDS parameters shall be computed for each cell of the grid mesh. The mesh may either be a quadrilateral or a triangular mesh. The number of cells in each dimension shall be equal to the texture size divided by 8.

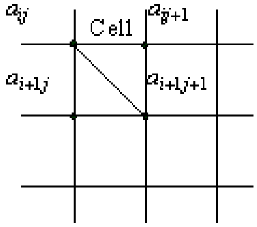


Figure -2 -- Mesh cell

1. Distance criterion:

$$Rd = \frac{1}{u}$$

u is the distance between viewpoint and Cell center: $u = \|\vec{v} - \vec{c}\|$ with $\vec{c} = \frac{1}{4}(\vec{a}_{ij} + \vec{a}_{i+1j} + \vec{a}_{ij+1} + \vec{a}_{i+1j+1})$ and $\vec{v}$ is the viewpoint vector.

2. **Rendering criterion:**

$$R_r = \frac{p}{q}$$

p is the distance between viewpoint and projection plane normalized to window width. *p* may be computed using:

$$p = \frac{1}{2\tan(FOV/2)}$$

$$p = \frac{1}{2\tan(FOV/2)}, \text{FOV is the Field Of View}$$

where FOV is the Field of View specified in radians, and q = <TextureWidth>/<WindowWidth> where the texture width is the width of the full texture (1024 for instance) and the WindowWidth is the width of the rendering window.

3. **Orientation criteria:**

$$Ra = \cos(\alpha)$$
$$Rb = \cos(\beta)$$

The angle between the aiming direction and the normal of the current cell center shall be projected into two planes. These two planes are spans of normal vector $\vec{n}$ of the cell and the cell edges in x and y directions, respectively. Then the angles (a , b) between projected vectors and the normal $\vec{n}$ shall be calculated, respectively.

The angle *b* is specified as the projection of the angle between $\vec{n}$, the normal of the quad cell, and $\vec{u}$, the aiming direction, onto the plane Π_x that passes through $\vec{g}_x$ and is parallel to $\vec{n}$. Similarly, the angle *a* is specified as the projection of the same angle onto the plane Π_y that passes through $\vec{g}_y$ and its parallel to $\vec{n}$.

This is illustrated in Figure J-3

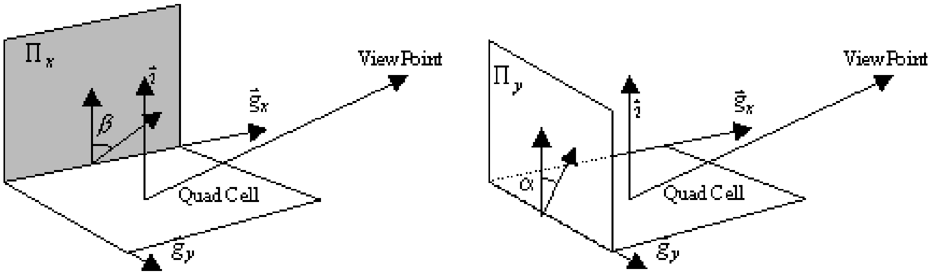


Figure -3 -- Definition of a and b angles

4. **Cropping criterion:**

Cells that are out of the field of view shall not be transmitted/received: that is, at least one of the 4 vertices which define the cell should all be inside the horizontal and vertical Field Of View (FOV).

The horizontal FOV shall be deduced from the vertical FOV using the screen geometry. The vertical FOV is equal to the FOV. Then the following shall be calculated

$$HFOV = A\tan\left(\frac{w}{h}\cdot\tan(FOV/2)\right)$$ where *w* and *h* are the width and height, respectively, of the rendered image.

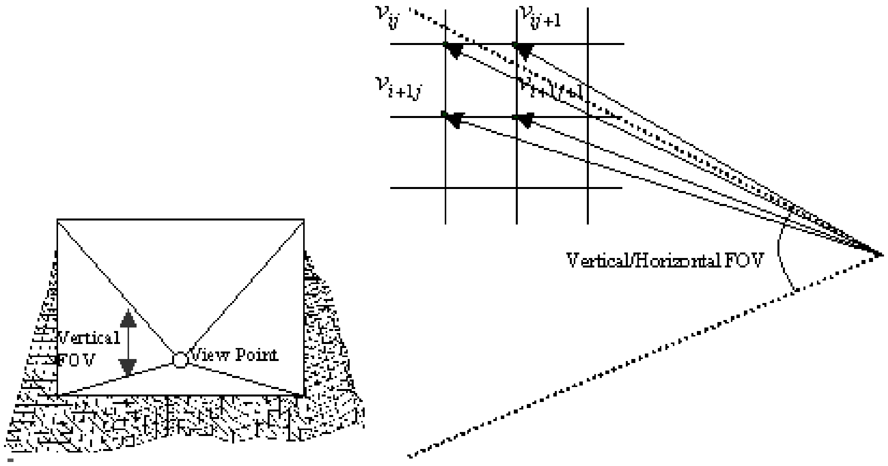


Figure -4 -- Definition of Out of Field of View cells

3. VD mask computation

The VD mask is a binary image of the same size as the texture image. Each value in the mask shall indicate if the corresponding DCT coefficient is needed (1) or not (0), given the VDS parameters.

For each cell, the following rules shall be applied to fill the corresponding 8x8 block of the VD mask:

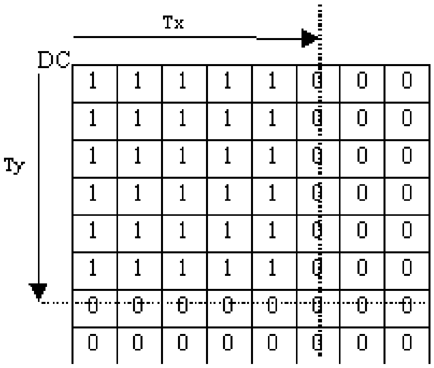
- **Use of cropping criterion**: If all the vertices of the cell are out of the field of view, the corresponding 8x8 block of the mask image shall be set to 0.

- **Use of rendering, distance, tilting and rotation criteria**: For each 8x8 block of the mask (corresponding to a quad cell), the 4 criteria mentioned above shall be computed. Two values of the rotation and tilting criteria shall be obtained for a quad cell, but only the higher value of each criterion shall be kept.

Two thresholds, T_X and $T_{Y,}$ shall be calculated as the product of the three VDS parameters *Rr, Rd, Rb,* and *Rr, Rd, Ra*, respectively, and the value 8. The results shall be bounded to 8. This procedure may be indicated symbolically as follows

$$Tx = Min(8, 8 \cdot R_r \cdot R_d \cdot R_b)$$
$$Ty = Min(8, 8 \cdot R_r \cdot R_d \cdot R_a)$$

The flag (i,j) of the 8x8 block corresponding to the current cell shall be set to 1 if i < T_X and j < T_Y. The flag shall be set to 0 in all other cases, as illustrated in the figure below.



1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure -5 -- VD mask of an 8x8 block using VD parameters

1. Differential mask computation

Once the first image has been received using the previously described filter, less data is necessary to update the texture data for the following frames. (assuming there is a correlation between viewpoint positions). Since this computation is exactly the same for each flag of each cell, it shall be performed directly on the full mask images and not on a cell by cell basis.

If the coefficient has not been already transmitted (buffer mask set to 0) and is needed according to VDS visibility criteria (VD mask set to 1), then the corresponding pixel of the differential mask shall be set to 1. This implies that the texture shall be updated, according to the procedure described in subclause J.2.6.

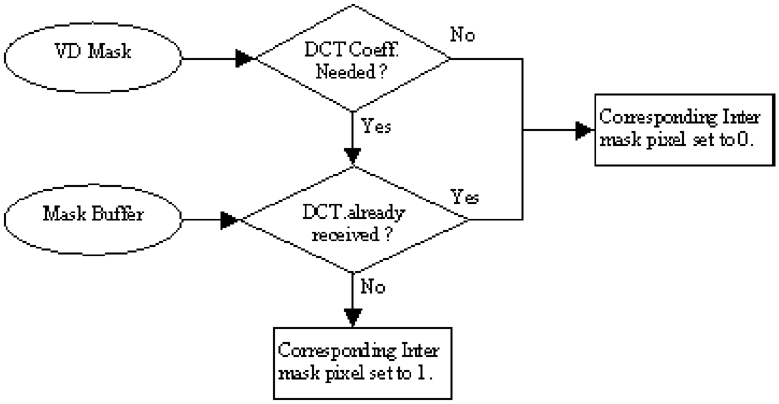

VD Mask
DCT Coeff. Needed ?
No
Yes
Corresponding Inter mask pixel set to 0.
Mask Buffer
DCT.already received ?
Yes
No
Corresponding Inter mask pixel set to 1.

Figure -6 -- Differential mask computation scheme

2. DCT coefficients decoding

The DCT coefficients shall be decoded using the Video Intra mode, Separated Texture/Motion mode as described in the subclause 7.4 .

3. Texture update

The Differential mask image shall be used to select which DCT coefficients of the buffered texture should be updated using the decoded DCT coefficients.

- Y component

If the Differential mask is set to 0, the corresponding DCT value of the buffer shall be left unchanged, otherwise the value shall be updated with the previously decoded DCT coefficient.

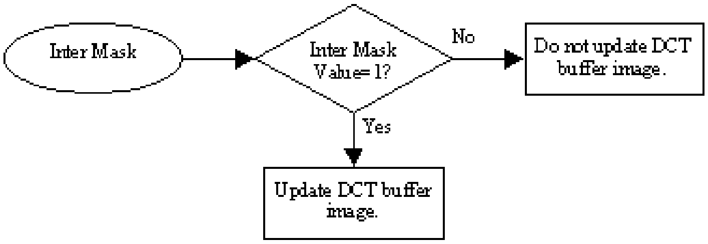


Figure -7 -- Texture update scheme

- U and V component

The texture is coded in 4:2:0 format, as specified in subclause 6.1.3.6, which shall imply that for each chrominance DCT coefficient, 4 Differential mask flags shall be available. The chrominance coefficients shall be received/transmitted if at least 1 of these 4 flags is set to 1.

1. **IDCT**

The IDCT and de-quantization shall be performed using the same process as in the Video Intra mode as described in subclause 7.4.

A. (normative)

Decoder Configuration Information

1. Introduction

This annex identifies the syntax elements defined in the main body of this part of ISO/IEC 14496 that describe the configuration of the visual decoder. This Annex amplifies the information provided in subclause 6.2.1. These elements shall be processed differently if this part of the specification is used jointly with the Systems part, ISO/IEC 14496-1. Instead of conveying the configuration at the beginning of a visual elementary bitstream, it shall be conveyed as part of an Elementary Stream Descriptor that is itself included in an Object Descriptor describing the visual object. This descriptor framework is specified in ISO/IEC 14496-1.

2. Description of the set up of a visual decoder (informative)

The process of accessing ISO/IEC 14496 content is specified in ISO/IEC 14496-1. It is summarized here and visualized in Fig. L.1 to outline the processing of decoder configuration information in an ISO/IEC 14496 terminal. This description assumes that all elementary streams are accessible and solely the problem of identifying them is to be solved.

The content access procedure starts from an Initial Object Descriptor that may be made available through means that are not defined in this part of ISO/IEC 14496, like a http or ftp link on an HTML page, a descriptor in an ISO/IEC 13818-1 Transport Stream, or some H.245 signaling, etc. Note that this may need standardisation by the responsible group.

This Initial Object Descriptor contains pointers at least to a scene description stream and an object descriptor stream. The scene description stream conveys the time variant spatiotemporal layout of the scene. For each streaming media object incorporated in the scene, an Object Descriptor exists that describes the set of streams associated to this media object. The set of Object Descriptors is conveyed in a separate stream, in order to distinguish scene description from the description of the streaming resources.

Both the scene description stream and the object descriptor stream allow for time stamped updates of the scene description and the object descriptors, respectively. Due to the time stamps it is always known at which point in time, or from which point in time onwards a data item, called Access Unit, is valid.

The Object Descriptor associated to a given visual object is identified by its ObjectDescriptor_ID. Each visual object may require more than one elementary stream (ES) that convey its coded representation, especially if any form of scaleability is used. Each of these streams is described by an ES_Descriptor. This description contains a unique label for the stream, the ES_Id, and, among others, the DecoderSpecificInfo structure that is of concern for the purpose of this Annex.

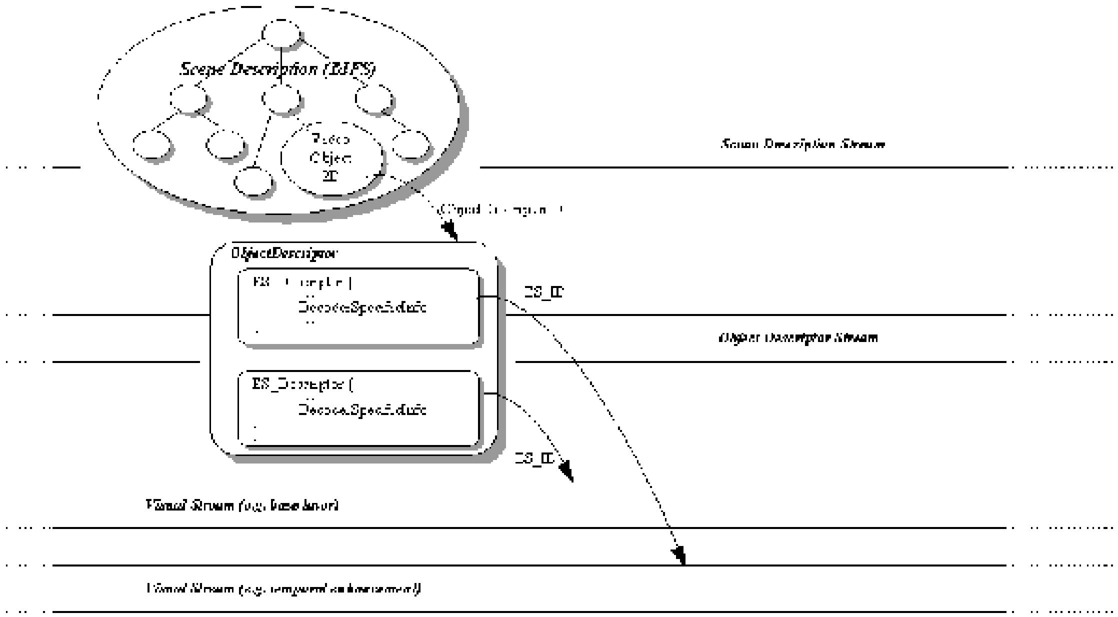


Figure -1 -- Visual decoder setup

1. Processing of decoder configuration information

After the retrieval of the Object Descriptor for a media object a decoder for the visual stream(s) is instantiated, connected to the stream(s) and initialised with the data found in ES_Descriptor.DecoderSpecificInfo.specificInfo[] for each stream. Subsequently, in a random access scenario, the decoder is expected to search forward to the next random access point, while in a client-server or local storage scenario, data in the visual stream may already be aligned in a way that the first data arriving in the visual stream corresponds to a random access point.

The difference between a visual-only scenario, as specified in the main body of this part of ISO/IEC 14496, and an integrated application using both ISO/IEC 14496-1 and this part of ISO/IEC 14496 is visualized in a figure. Figure K-2 shows the Visual-only approach with configuration information at the beginning of a bitstream and optionally repeated thereafter to enable random access. Figure K-3 with the integrated Systems and Visual approach shows the plain bitstreams with the configuration information extracted into the object descriptors. In this case the object descriptors will be repeated if random access is desired. The Access Units shown in the figure correspond to VOPs in the case of visual media streams.

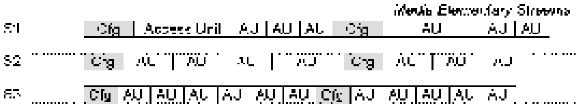


Figure -2 -- Visual-only scenario

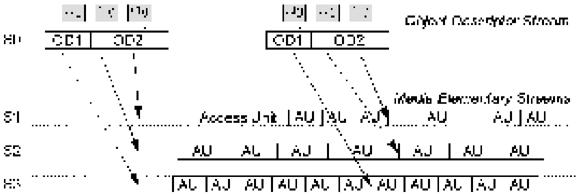


Figure -3 -- Integrated Systems and Visual approach

3. Specification of decoder configuration information

The decoder configuration information for a visual elementary stream is given by a concatenation of those syntax elements that precede the actual encoded data, i. e., that form the ‚stream header? according to the syntax specification in subclause 6.2. Those syntax elements are identified separately for each type of visual object in the subsequent subclauses. The syntax elements that are conveyed as decoder configuration information shall not be present in the visual elementary stream itself. Furthermore, the generic syntax definition in subclause 6.2 is constrained to clarify that an elementary stream may not contain a concatenation of, for example, multiple VisualObject() structures or multiple VideoObjectLayer() structures.

The decoder configuration information shall be conveyed in the DecoderSpecificInfo.specificInfo[] field of the respective ES_Descriptor and passed to the decoder before any data of the visual elementary stream itself.

VisualObjectSequence() is not transmitted explicitly. The value present in the Initial Object Descriptor for this presentation notifies the value of VisualObjectSequence.profile_and_level_indication.

Identification and priority of objects is signaled generically for all kinds of audiovisual objects in the ES_Descriptor. This signalling overrides the value of VisualObject.is_visual_object_identifier and
VideoObjectLayer.is_object_layer_identifier.

VisualObject() carries information describing all the elementary streams associated with an object and is carried in a container associated with all the layers of this object.

VideoObjectLayer() carries information describing a single elementary stream, i.e. a single-layer object or one layer of a multi-layer object. This information is carried in a container associated with the corresponding elementary stream.

1. **VideoObject**

The decoder configuration information for a visual object of type VideoObject consists of the following elements

- All syntax elements of VisualObjectSequence()
- including all syntax elements of one VisualObject()
- including all syntax elements of one VideoObject()
  - including all syntax elements of one VideoObjectLayer() excluding the trailing Group_of_VideoObjectPlane() and VideoObjectPlane() that convey the coded data.

The elementary stream consists of the Group_of_VideoObjectPlane(), VideoObjectPlane(), video_plane_with_short_header and all lower layers.

1. **StillTextureObject**

The decoder configuration information for a visual object of type StillTextureObject consists of the following elements

- All syntax elements of VisualObjectSequence()
- including all syntax elements of one VisualObject()
- including all syntax elements of one StillTextureObject() up to but not including the first call to wavelet_dc_decode().

1. MeshObject

The decoder configuration information for a visual object of type MeshObject consists of the following elements

- All syntax elements of VisualObjectSequence()
- including all syntax elements of one VisualObject()

1. FaceObject

The decoder configuration information for a visual object of type FaceObject consists of the following elements

- All syntax elements of VisualObjectSequence()
- including all syntax elements of one VisualObject()

A. (informative)

Rate control

1. Frame Rate Control

1. Introduction

Rate control and buffer regulation is an important issue for both VBR and CBR applications. In the case of VBR encoding, the rate controller attempts to achieve optimum quality for a given target rate. In the case of CBR encoding and real-time application, the rate control scheme has to satisfy the low-latency and VBV buffer constraints. In addition, the rate control scheme has to be applicable to a wide variety of sequences and bit rates.

The scalable rate control (SRC) scheme is designed to meet both VBR without delay constraints and CBR with low-latency and buffer constraints. The SRC scheme is scalable for various bit rates (e.g. 10kbps to 1Mbps), various spatial resolutions (e.g. qcif to cif) and various temporal resolutions (e.g. 7.5fps to 30fps) and various coders (e.g. DCT and wavelet). The technique can handle I, P, and B pictures. The SRC is for single VO, and extensions to M-VOs and other improvements are being addressed in clause L.2 The current description in clause L.1 only handles I and P pictures.

2. **Description**

The SRC scheme assumes that the encoder rate distortion function can be modeled as

R=X1*S*Q**(-1)+X2*S*Q**(-2)

The encoding bit count is denoted as R. The encoding complexity which is mean absolute difference (MAD) is denoted as S. The quantization parameter is denoted as Q. The modeling parameters are denoted as X1 and X2. Because of the generality of the assumption, the scalable rate control scheme is applicable to a variety of bit rates, spatial resolutions, temporal resolutions, buffer constraints and types of coders.

There are four steps in the SRC scheme:

1). Initialization

- X1 and X2 are the first and second order coefficients.

2). Computation of the target bit rate before encoding.

- The target bit rate is computed based on the bits available and the last encoded frame bits. If the last frame is complex and uses excessive bits, more bits should be assigned to this frame. However, there are fewer bits left for encoding. Thus, fewer bits can be assigned to this frame. A weighted average reflects a compromise of these two factors.

- A lower bound of target rate (R/30) is used so that minimal quality is guaranteed.

- The target rate is adjusted according to the buffer status to prevent both overflow and underflow.

3). Computation of the quantization parameters (QP) before encoding.

- QP is solved based on the model parameters X1 and X2.

- QP is clipped between 1 and 31.
- QP is limited to vary within 25 percent of the previous QP to maintain a VBR quality.

  4). After encoding model parameters is updated based on the encoding results of the current frame.

- The rate distortion model is updated based on the encoding results of the current frame. The bits used for the header and the motion vectors are deducted since they are not related to QP.
- The data points are selected using a window whose size depends on the change in complexity. If the complexity changes significantly, a smaller window with more recent data points is used.
- The model is calibrated again by rejecting the outlier data points. The rejection criteria are that data point is discarded when the prediction error is more than one standard deviation.
- The next frame is skipped if the current buffer status is above 80 percents.

The algorithm is described as follows:

Step 1: Initialization after the first frame is coded

Rs: bit rate for the sequence (or segment). /* e.g., 24000 bits/sec */

Rf: bits used for the first frame. /* e.g., 10000 bits */

Rc: bits used for the current frame. It is the bit count obtained after encoding.

Rp: bits to be removed from the buffer per picture.

Ts: number of seconds for the sequence (or segment). /* e.g., 10 sec */

Ec: mean absolute difference for the current frame after motion compensation.

If the macroblock is intra coded, the original spatial pixel values are summed.

Qc: quantization level used for the current frame.

Nr: number of P frames remaining for encoding.

Ns: distance between encoded frames. /* e.g., 4 for 7.5 fps */

Rr: number of bits remaining for encoding this sequence (or segment).

T: target bit to be used for the current frame.

S: number of bits used for encoding the previous frame.

Hc: header and motion vector bits used in the current frame. It includes all the information except to the residual information.

Hp: header and motion vector bits used in the previous frame. It includes all the information except to the residual information.

Ql: quantization level used in the previous frame

Bs; buffer size e.g., R/2

B; current buffer level e.g., R/4 - start from the middle of the buffer

```
Rr = Ts*Rs-Rf; /* total number of bits available for this segment */

Rp = Rr/Nr; /* the average bits to be removed from the buffer */

if(first frame){

Qc=15;

goto step 4

}
```

Step 2: Target Bit Calculation (Before Encoding the Current Frame)

```
T=Max(Rs/30,Rr/Nr*0.95+S*0.05);/*each frame is assigned a minimum of R/30 */

T=T*(B+2*(Bs-B))/(2*B+(Bs-B)); /* increase if less than half */
```

```
/* decrease if more than half, don?t change if half */
if (B+T > 0.9*Bs)
T = Max(Rs/30, 0.9*Bs-B); /* to avoid overflow*/
else if (B-Rp+T < 0.1*Bs)
T = Rp-B+0.1*Bs; /* to avoid underflow*/
```

Step 3: Quantization Level Calculation (Before Encoding the Current Frame)

```
T= Max(Rp/3+Hp, T);
if ((X2==0) || (((X1*Ec)**2+4*X2*Ec*(T-Hp))<0))
Qc = X11*Ec/(T-Hp); /* fall back 1st order mode */
else Qc= (2*X2*Ec)/(sqrt((X1*Ec)**2+4*X2*Ec*(T-Hp))-X1*Ec) /* 2nd order mode */
Qc = Min (ceil(Ql*1.25), Qc, 31); /* clipping*/
Qc = Max (ceil(Ql*0.75), Qc, 1); /* clipping */
Quantization();
Encoding();
```

Step 4: After encoding the current frame

```
B += Rc - Rp; /* update buffer fullness */
Rr -= Rc; /* update the remaining bit counts */
S = Rc; /* update the previous bit counts */
Hp = Hc; /* update the previous header and motion bit counts */
Ql = Qc; /* update the previous quantization level */
```

```
Nr--; /* update the frame counter */

UpdateRDModel(Qc,Rc,Ec,Hc,X1,X2);

/* estimation of a new model */

while (B > 0.8 * Bs) {

skip_next_frame();

Nr--;

B -= Rp;

}
```

If the buffer has reached 80% of the buffer size, the encoder will skip the upcoming frame to be encoded. Thus the buffer is reduced to prevent from the buffer overflow.

```
UpdateRDModel (Qc, Rc, Ec, Hc, X1, X2) {

    Qall[n]: quantization levels for the past frames

    Rall[n]: scaled encoding complexity used for the past frames

    n: number of encoded past frames;

    x: matrix contains Q;

    y: matrix contains Q*(R-H)/E;

    Ep: mean absolute difference for the previous frame. This is computed
    after motion compensation for the Y component only.

    Rall[n] = (Rc-Hc)/Ec;

    Qall[n] = Qc;

    w = Min(total_data_number, 20) /* Maximum data number set to 20 */

    if(Ep>Ec)
    w = ceil(Ec/Ep*w); /* sliding window for scene change */
    else
    w = ceil(Ep/Ec*w); /* sliding window for scene change */
    Ep = Ec; /* update mad */
    for(i=n; i>n-w; i++){
    Qp[i]=Qall[i]
    Rp[i]=Rall[i]
    }
    Estimator(Qp,Rp,w,b);
    RemoveOutlier(X1,X2,Qp,Rp,Ec,w);
    Estimator(Qp,Rp,w,b);
}
Estimator(Qp, Rp,w,b) {
    X11: First order coefficient.
    x = [1, Qp[i]**(-1)( i=1,2,...w)] /* Dimension wx2 */
    y = [Qp[i]*Rp[i] (i=1,2,.....w)] /* Dimension wx1 */
    for (i=1; i<=w; i++) X11 += y[i]/w;
    if (all Qp[i] are not all the same) {
    b = (x_Transpose*x)**(-1)*x_Transpose*y;
    /* Dimension of the matrix */
    /* 2x1 = (2xw * wx2)**(-1)*(2xw)*(w*1) */
```

```
        X1 = b(1,1);
        X2 = b(2,1);
        }
}
RemoveOutlier(X1,X2,Qp,Rp,Ec,w) {
        error[w]; Estimation error
        new_w;
        for (i=1; i<=w; i++) {
        std += ((X1*Ec*Qp[i]**(-1)+X2*Ec*Qp[i]**(-2)-Rp[i]*Ec))**2;
        error[i] = X1*Ec*Qp[i]**(-1)+X2*Ec*Qp[i]**(-2)-Rp[i]*Ec;
        }
        /* When w is 2 the threshold should be 0, some implementation may
        resolve a small number due to precision problem. */
        threshold = sqrt(std/w); /* Setup rejection threshold */
        new_w = 0
        for (i=1; i<=w; i++){
        if(abs(error[i]) <= threshold){
        new_w++;
        Qp[new_w] = Qp[i];
        Rp[new_w] = Rp[i];
        }
        }

        w = new_w;

    }
```

1. **Summary**

The scalable rate control scheme achieves frame level rate control for both VBR and CBR cases. It assumes a simple quadratic rate distortion function of the video encoder. In the case of CBR encoding, a variable frame rate approach is used to achieve the target rate. If a tighter rate control is desired, the same technique is applicable at either slice layer or macroblock layer. Because of the generality of this scheme, extension of such a scheme to shape coding and Multiple-VO coding can be easily made on the basis of the existing rate control framework.

1. **Multiple Video Object Rate Control**

This clause describes an algorithm which achieves a constant bit rate when coding multiple video objects. The scheme assumes the following relationship between the bits and quantization parameter for the i-th object:

R[i]=X1[i]*S[i]*Q[i]**(-1)+X2[i]*S[i]*Q[i]**(-2)

The above is a simple extension of the relationship used for SVOP rate control.

As shown in Figure L-1, the algorithm for performing the joint rate control can be decomposed into a number of steps, which may be grouped into a pre-encoding and post-encoding stage. Each block is briefly described below.

1. **Initialization**

The initialization process is very similar to the SVOP process. Since a single buffer is used, the buffer drain rate is the same. The major difference is that most parameters are extended to vector quantities.

2. **Quantization Level Calculation for I-frame and first P-frame**

To calculate the quantization parameters which are used for the I-frame and first P-frame, a bit allocation table based on Human Visual Sensitivity (HVS) of color tolerance is used. This scheme assigns a target bit rate which is predicted by histogram of variance and HVS classification.

Color/Variance classification: A macro block is classified into one of 32 macro block classes based on HVS and one of 16 classes based on the variance. A reference quantization step size is assigned to the macro block according to its macro block class. A macro block is classified into a sensitive category if the number of pixels in the macro block which satisfy LUT (Look-Up Table) is larger than a threshold value. Otherwise the macro block is classified into insensitive category.

$$texture\_class = H_n + V_n$$

$$H_n = \sum_{y=1}^{16}\sum_{x=1}^{16}\left(\left|P_{x,y} - P_{x-1,y}\right| > th\right)$$

$$V_n = \sum_{y=1}^{16}\sum_{x=1}^{16}\left(\left|P_{x,y} - P_{x,y-1}\right| > th\right)$$

$$th = f\left(\sum_{y=1}^{16}\sum_{x=1}^{16} P_{x,y}\right)$$

$$color\_class = \min\left(color\_tolerance\right)$$

$$color\_tolerance = LUT\left(color_{mean}\right)$$

$$color_{mean} = \frac{1}{64}\sum_{y=1}^{16}\sum_{x=1}^{16}\left(P_{x,y}\right)$$

The block variance classification allows us to estimate the number of coded bits in a block. Generally, the number of coded bits generated in a block is proportional to the block variance when the quantization step size is assumed to be constant. We divide the magnitude of block variance into 16 intervals. Threshold values in each interval can be found in Table L-1. Table L-2 illustrates the reference quantization step size based on HVS for each macro block class and picture type.

Table -1 -- Block variance classes

Bv **VOP**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
I	16	32	64	128	256	384	512	768	1024	1280	1792	2048	3072	4096
P	8	16	32	48	64	96	128	160	192	256	320	448	640	896
Bv : Block Variance Class Number, I : INTRA FRAME TYPE, P : INTER FR														

Table -2 -- Reference quantization step size

Mc	0	1	2	3	4	5	6	7
Qss	-4	-2	-1	0	1	2	3	4

Initial QP setting method

```
SetInitialQP()
{
PreDiff = 1000000;
Classification();
if(Flat Image) TargetBit = AverageBitPerFrame * 9;
else TargetBit = AverageBitPerFrame * 4;
/* Find Optimal Qp value */
for(i=1; i<31; i++){
SetMB_QP(i); /* Allocate HVS optimal QP to each MB */
EstimatedBit = Bitestimation(); /* Pre-bit-estimation */
Diff = abs(TargetBit - EstimatedBit);
if(Diff<PreDiff){
PreDiff = Diff;
MinErrorQ = i;
}
} /* MinErrorQ is the final average Qp value */
SetMB_QP(MinErrorQ); /* Allocate HVS Optimal Qp to each MB */
}
```

Frame target bit allocation

$$\mathrm{FrameTargetBit} = \begin{Bmatrix} \frac{bitrate}{fps} * (4 \sim 9) ---- \mathrm{Intraframe} \\ \frac{bitrate}{fps} ----- \mathrm{First\ Inter\ Fr.} \end{Bmatrix}$$

Frame coded-bit estimation: There are 512 classes which are combinations of 32 macro block color classes(Mc) and 16 block variance classes(Bv). We can estimate the total number of coded bits in a frame with the histogram and the bit model table. **FrameEstimatedBit** is the estimated number of coded bit in the current frame

$$\mathrm{FrameEstimateBit} = \sum_{Mc=0}^{31} \sum_{Bv=0}^{15} \{ if(Coded) B[R[Mc]][Bv] * H[Mc][Bv] \} + \mathrm{ShapeBit}$$

In the above equation, B[Mc][Bv] is the number of bits in the bit table and H[Mc][Bv] is the computation result of the histogram when a block is classified into Bv. The histogram can be computed with the information of macro block classes and block classes of the previous frame because the current frame to be coded and the previous frame are very highly correlated. Therefore the real-time coding process is possible.

Bit Table

The number of macro block classes and block classes in a frame depends on the statistical characteristics of the frame. However, the target number of bits which is assigned to the frame by bit allocation is not related with statistical characteristics of the frame. Hence the target number of bits for frame has much difference from the total number of coded bits in the frame when all macro blocks in the frame are quantized with their reference quantization step sizes. So reference quantization step sizes should be updated in order that the total number of coded bit in the frame becomes almost the same as the number of target bits for the frame. To determine the ratio of change of reference quantization step size, we have to estimate the total number of coded bits in a frame when all macro blocks in the frame are quantized with their reference quantization step sizes. The bit table shows the average value of the estimated number of bits in a block according to macro block classification and block classification in Intra/Inter frame. It is possible to estimate the total number of coded bits in a frame with the bit table.

(INTRA FRAME Bit Table)

```
int BitTableI[32][16] =

{
65,145,174,208,242,278,298,317,341,352,365,386,400,419,435,438,
24, 79,102,127,153,179,194,209,232,243,255,266,272,279,286,292,
16, 52, 70, 93,114,136,148,161,180,190,201,214,220,226,231,234,
12, 38, 53, 73, 91,110,122,132,150,159,166,179,185,188,193,195,
11, 29, 42, 59, 75, 92,104,112,131,138,144,154,158,160,163,168,
10, 24, 35, 49, 64, 79, 91, 98,116,122,127,135,137,142,148,159,
9, 19, 29, 42, 55, 69, 80, 87,103,109,116,122,126,130,137,147,
9, 17, 25, 37, 48, 61, 72, 79, 95,100,105,111,114,119,125,135,
9, 15, 22, 33, 43, 55, 65, 71, 86, 91, 97,103,106,110,115,123,
8, 13, 19, 29, 39, 50, 59, 65, 79, 84, 89, 95, 98,100,103,114,
8, 12, 18, 26, 35, 45, 54, 60, 73, 78, 84, 88, 89, 95, 99,104,
8, 11, 16, 24, 32, 42, 51, 55, 68, 74, 79, 83, 85, 91, 94, 98,
8, 11, 15, 22, 30, 38, 47, 51, 63, 69, 74, 78, 81, 86, 88, 90,
8, 10, 14, 21, 28, 35, 44, 48, 59, 64, 69, 75, 77, 79, 80, 83,
8, 10, 13, 19, 26, 33, 41, 45, 55, 61, 65, 71, 71, 74, 76, 78,
8, 9, 12, 18, 24, 31, 39, 42, 52, 58, 61, 66, 66, 70, 72, 75,
8, 9, 12, 17, 23, 29, 37, 40, 48, 55, 57, 62, 63, 65, 69, 73,
8, 9, 11, 16, 21, 27, 35, 37, 46, 51, 55, 60, 60, 64, 66, 69,
8, 9, 11, 14, 20, 26, 33, 36, 43, 48, 52, 56, 59, 63, 65, 66,
8, 9, 10, 14, 19, 25, 31, 34, 41, 45, 50, 55, 57, 61, 64, 66,
8, 9, 10, 13, 18, 23, 30, 33, 40, 44, 48, 53, 54, 59, 62, 63,
8, 8, 10, 12, 17, 22, 28, 31, 38, 42, 46, 50, 52, 57, 60, 62,
8, 8, 9, 12, 16, 21, 27, 30, 36, 40, 44, 49, 51, 55, 57, 60,
8, 8, 9, 11, 16, 21, 26, 29, 35, 39, 43, 46, 48, 54, 55, 58,
8, 8, 9, 11, 15, 20, 25, 28, 33, 38, 41, 44, 45, 51, 54, 57,
8, 8, 9, 11, 14, 19, 24, 27, 32, 36, 40, 43, 44, 48, 50, 55,
8, 8, 9, 10, 13, 18, 24, 26, 31, 35, 38, 42, 43, 46, 48, 55,
8, 8, 8, 10, 13, 17, 23, 25, 29, 33, 37, 41, 41, 43, 46, 55,
8, 8, 8, 10, 13, 17, 22, 24, 29, 32, 36, 39, 40, 42, 41, 53,
8, 8, 8, 9, 12, 16, 22, 23, 28, 31, 35, 38, 39, 40, 43, 52,
8, 8, 8, 9, 12, 16, 21, 22, 27, 30, 34, 36, 37, 40, 43, 52,

};
```

3. **Update Rate-Distortion Model**

   After the encoding stage, the parameters for the RD model are updated. Given the number of texture bits, R[i], the quantization level, Q[i], and the mean absolute difference, S[i], for the current and a specified number of past frames, the model parameters, X1[i] and X2[i] can be calculated using a least-squares estimation.

4. **Post-Frameskip Control**

The post-frameskip control determines the value of *N_skip_post.* It is invoked after the buffer has been updated, i.e., the sum of bits used to code every VO is added to the current buffer level and the number of bits to be removed per picture is subtracted. Essentially, if the "virtual buffer" has exceeded a specified level (e.g., 80% of the buffer size), the encoder will skip the upcoming frames so as to reduce the current buffer level. The virtual buffer level (VB) is given by:

$$VB = B + Bp - (N\_skip\_post+1)*Rp$$

where the new variable, Bp, denotes the number of bits used in the previous frame. As before, B is the current buffer level and Rp is the number of bits removed from the buffer per picture. After the value of *N_skip_post* has been found, the value of *N_skip_pre* is added to it, where *N_skip_pre* is calculated in the pre-frameskip control. The time instant is updated based on the total frames to be skipped, *N_skip* = *N_skip_pre* + *N_skip_post*.

Mode of Operation: Since coding decisions may change according to the environment, it is desirable to have a mechanism to detect and keep an update of such changes.

```
if(N_skip > SkipTH)
    Operate in LowMode
else
    Operate in HighMode
```

The mode of operation can be used to impose constraints on the coding parameters, assist in setting the weights for target distribution, and influence the shape rate control mechanism.

Target Estimation: The initial target bit rate can be calculated based on the number of available bits for the segment and the number of bits used in the previous corresponding VO. A similar lower bound to the SVOP simulation can be used to target minimum quality.

Joint Buffer Control: The joint buffer control symbolizes the same operation as buffer control in SVOP, however, the target is a sum of individual targets, i.e., T = sum of T[i].

Pre-FrameSkip Control: Often, in low bit rate coding conditions, the target which emerges from the joint buffer control may not be enough to even code the motion, shape, and header information, let alone the texture. In that case, there should be a mechanism to alert other parts of the system (e.g., target distribution, QP calculation, time instant update) that there is a deficiency in the number of allocated bits. Let s = T - Hp be the difference between the target and the amount of bits used in the previous frame for shape, motion, and header, and Beta be a bit threshold greater than zero. N_skip_pre is determined by:

```
while(s < Beta)
    N_skip_pre = N_skip_pre + 1
    s = s + Rd
```

Target Distribution: The joint buffer control provides a new total target to be distributed among each VO. The distribution can be based on the size, the motion, and the variance (MAD^2) of each object:

T[i] = (MOTION[i]*wm + SIZE[i]*ws + MAD2[i]*wv)*T

This type of distribution is only executed when the target is valid. The target is valid when it is greater than the sum of header bits used for coding the previous frame. If it is not valid, all the targets are made negative. The negative targets can serve as a flag for clipping the quantization parameters to lie in a specified range [Qc 31].

Shape Rate Control: The value of AlphaTH has considerable effect over the number of bits which will be spent on the shape information. Using the current mode of operation, an appropriate value can be determined.

- If the mode of operation is LowMode, then increment the current AlphaTH by AlphaINC;
- If the mode of operation is HighMode, then decrement the current AlphaTH by AlphaDEC.

The minimum value of AlphaTH is 0. This implies lossless shape encoding. The largest value that AlphaTH may take is 255, however, it is recommended that an AlphaMAX in the range [32, 64] be specified.

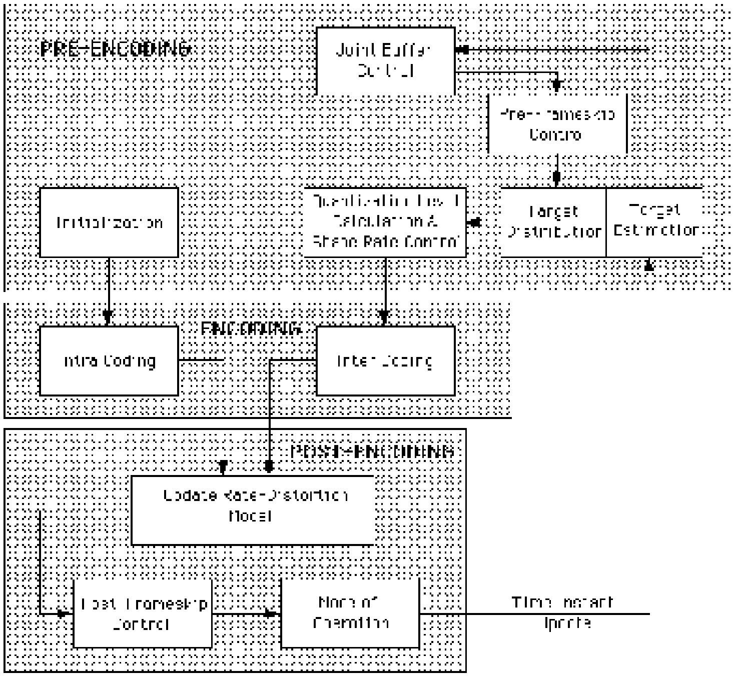


Figure -1 -- Block diagram for multi-VO rate control

1. Macroblock Rate Control

The rate control method described in clause L.1 used a fixed quantization parameter QP for all the macroblocks in a VOP, where the value of QP was chosen to encode the VOP with a target number of bits. In this clause we describe a method for adapting the value of QP at each macroblock within the VOP. This technique achieves the VOP target more accurately, which is useful for low-delay video applications with small buffer constraints, and allows adapting QP with the macroblock energy and other perceptually relevant measures. The method attempts to combine the strengths of the following contributions:

- A model of the human visual system as described in clause L.2 can be used to compute W_j.
- The model with two parameters maintains a near-constant value of QP for all the macroblocks in the VOP. This approach is more appropriate at lower bit rates, since it requires a small number of bits for encoding the QP?s (using DQUANT).
- The model with single parameter adapts QP with the energy of the block by using finer quantization (smaller QP) for macroblocks of flatter image regions. It requires a larger QP bit

overhead and hence is more appropriate at higher bit rates.

1. **Rate-Distortion Model**

We model the number of bits B_i produced for the i-th macroblock as follows:

$$B_i = \begin{cases} \dfrac{A_1}{QP_i^2} MAD_i^2; & \text{Bitrate} > R \\ \dfrac{A_2}{QP_i^2} MAD_i + \dfrac{A_3}{QP_i} MAD_i; & \text{Bitrate} \leq R \end{cases} \quad (1)$$

- A1 , A2 and A3 are the model parameters.
- Bitrate is the rate available (in bits per pixel) for the texture of the VOP.
- R is a threshold. By default, we set R=0.085.

1. **Target Number of Bits for Each Macroblock**

Let N be the number of macroblocks in a VOP. The target number of bits, T_i, for the i-th macroblock is:

$$T_i = \frac{W_i MAD_i}{\sum_{j=1}^{N} W_j MAD_j} T_{TEX} \quad , (2)$$

where:

- W_i is a weight that indicates the perceptual importance of the i-th macroblock. By default, we set $W_1 = W_2 =$?$= W_N = 1$, but they could be obtained from similar techniques as described in clause L.2.
- T_{TEX} is the target number of bits for the texture or DCT coefficients (luminance and chrominance components) of the VOP, which is the target frame bits T from frame-layer rate control [see clause L.1] minus the bits for motion, shape, and syntax overhead. If the latter number of bits is not available for the current VOP, one can subtract those from the previous VOP. Bitrate in (1) is T_{TEX} divided by the number of pixels in the VOP.

Observe that if we combine (1) and (2), we obtain the following expression:

$$QP_i^2 = \begin{cases} MAD_i C_1; & \text{Bitrate} > R \\ C_2; & \text{Bitrate} \leq R \end{cases} \quad (3)$$

where the values of C_1 and C_2 depend on { A_1, MAD_1, ?, MAD_N, T } and {A_2, A_3, MAD_1, ?, MAD_N, T}, respectively. The model parameters A_1, A_2, and A_3 are estimated while encoding and in practice they are nearly constant within a VOP, and hence C_1 and C_2 are also approximately constant. As a result, observe that for large Bitrate, the quantization parameter QP will increase with the MAD energy of the macroblock, and for low Bitrate, QP will be nearly constant for all the macroblocks in the VOP.

1. **Macroblock Rate Control**

Step 1 Initialization.

N - number of macroblocks in the VOP,

T_{TEX} - number of bits available for encoding the VOP texture

If first frame, let A1= A1_prev= 100, A2= A2_prev= 400, A3= A3_prev= 0

Otherwise, let A1= A1_prev , A2= A2_prev, and A3= A3_prev.

If Bitrate > R = 0.085, let K=1, otherwise let K=2.

Set macroblock index i=1 and skip= 0.

Finally, let $S_1 = \sum_{j=1}^{N} W_j \, MAD_j$. (By default, set W1= W2= ?=WN = 1.)

Step 2 Compute QP for i_th macroblock

Let $T_i = \frac{W_i \, MAD_i}{S_i} T_{TEX}$, the target number of bits per macroblock and:

$$QP = \begin{cases} MAD_i \sqrt{\frac{A_1}{T_i}}; & K = 1 \\ Q: \frac{A_2}{Q^2} MAD_i + \frac{A_3}{Q} MAD_i = T_i; & K = 2 \end{cases}$$

If $(A_3MAD_i)^2 + 4\, T_iA_2MAD_i$ is negative, there is no value of Q that solves the second-order equation for K=2. In that case, the term with A_2 is dropped and the following is used instead:

$$QP = \frac{MAD_i A_{3\_X11}}{T_i},$$

where A_{3_X11} is similar to the X11 parameter in the frame-based rate control in clause L.1.2.

Round QP to value in 1, ?, 31, and indicate the difference between the current value of QP and that of the previous macroblock in DQUANT. (Recall that DQUANT can only take values within {-2, -1, 0, 1, 2 }, so clipping may be needed.)

Step 3 Encode Macroblock with QP

Step 4 Update Counters

Let B_i' be the actual number of texture (DCT) bits produced when encoding the i_th macroblock. Compute:

$T_{TEX} = T_{TEX} - B_i'$, and $S_{i+1} = S_i - W_i MAD_i$

Step 5 Update A_1, or A_2, A_3, A_{3_X11}

If B $_i$? is zero (the DCT produced no bits), then skip= skip +1, and go to Step 6.

Otherwise, do the updating:

If K = 1, $\hat{A}_1 = \frac{B_i'}{MAD_i^2} QP^2$.

Let $A_1' = \hat{A}_1 \frac{1}{i - skip} + A_1' \frac{i - skip - 1}{i}$, $\quad A_1 = A_1' \frac{i}{N} + A_{1\_prev} \frac{N - i}{N}$.

If K = 2, use linear regression to estimate A2, A3, and A3_X11, similarly as in subclause L.1.2:

We order the values of B?i, QP, and MADi, for the last W encoded macroblocks (whose B?i > 0) into three sets: { b1, b2, b3, ?, bW }, { q1, q2, ?, qW }, and { m1, m2, ?, mW }, where the bj?s, qj?s and mj?s correspond to the values of B?i, QP, and MADi, respectively, for the j-th macroblock in the set.

Let NC be the number of macroblocks (whose B?i > 0) and set W= min{20, NC }. Define:

P1= [1, 1, ?, 1]T,

P2= [1/q1, 1/q2, ?, 1/qW]T,

P3= [b1q1/m1, b2q2/m2, ?, bWqW/mW]T,

P4= [P1 P2],

where the superscript "T" denotes the transpose of a vector or matrix. P1, P2 and P3 are vectors of dimension W x 1, and P4 is a matrix of W x 2 elements.

Compute:

$$\begin{bmatrix} A_2 \\ A_3 \end{bmatrix} = \left(P4^T \cdot P4\right)^{-1} P4^T \cdot P3$$

,

and $A_{3\_xm1} = \frac{P3^{I} \cdot P2}{W}$,

where the superscript "-1" denotes the inverse of the matrix.

Step 6. If i = N, Stop (all macroblocks are encoded), let A_f_prev = A_f for f =1,2,3.

Otherwise, i = i+1, and go to Step 2.

A.

B. (informative)

Binary shape coding

1. Introduction

This part of ISO/IEC 14496 will be the first international standard allowing the transmission of arbitrarily shaped video objects (VO). A time instant of a VO is called Video Object Plane (vop). A vop is a rectangular video frame or a part thereof. Following an object-based approach, the ISO/IEC 14496-2 video encoder applies the motion, texture and shape coding tools to the vop using I, P, and B modes similar to the modes of MPEG-2. The encoder transmits texture, motion, and shape information of one VO within one bit stream. The bit streams of several VOs and accompanying composition information can be multiplexed such that the decoder receives all the information to decode the VOs and arrange them into a video scene (Figure M-1).

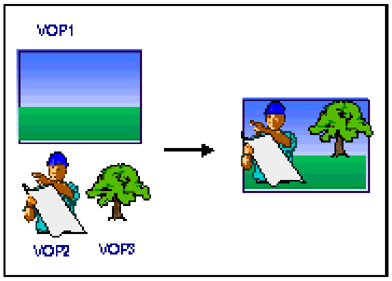


Figure -1 -- Object-based coding requires the decoder to compose different Video Object Planes (vop) into a scene.

An arbitrarily shaped VO can have binary shape information only or binary shape and texture information. Several shape coding algorithms were evaluated [1][2]. Binary shape information is transmitted as a bitmap using a context-based arithmetic coder described in clause M.2. Coding of texture for arbitrarily shaped VOs is described in clause M.3. This part of ISO/IEC 14496 leaves it up mainly to the encoder to achieve efficient coding of texture at object boundaries by employing an appropriate texture

extrapolation algorithm. The efficient coding of a VO with lossily encoded shapes requires an encoder architecture as described in clause M.4. In clause M.5, guidelines for coding arbitrarily shaped VOs will be presented. Different shape encoding modes are investigated in order to achieve a bit efficient representation of a VO or in order to allow for fast encoding that is important for real-time applications.

2. Context-Based Arithmetic Shape Coding

In order to enable content based access to video objects, this part of ISO/IEC 14496 codes the shape of video objects. This part of ISO/IEC 14496 chose to code the shape as a bitmap [2]. For binary shape coding, a rectangular bounding rectangle enclosing the arbitrarily shaped vop is formed such that its horizontal and vertical dimensions are multiples of 16 pels (macroblock size).

Each block of size 16x16 pels within this bounding rectangle is called binary alpha block (bab). Each bab is associated with the co-located macroblock. Three types of babs are distinguished and signaled to the decoder: Transparent blocks do not contain information about the object, opaque blocks are located entirely inside the object and boundary blocks cover part of the object as well as part of the background. For boundary blocks a context-based shape coder was developed. This coder exploits the spatial redundancy of the binary shape information to be coded. Pels are coded in scan-line order and row by row. In the following paragraphs, shape encoding in intra mode is described first. Then, this technique is extended to include an inter mode.

1. Intra Mode

In intra mode, three different types of macroblocks are distinguished: Transparent and opaque blocks are signaled as macroblock type. The macroblocks on the object boundary containing transparent as well as opaque pels belong to the third type. For these boundary macroblocks, a template of 10 pels is used to define the causal context for predicting the shape value of the current pel (Figure M-2a). For encoding the state transition, a context-based arithmetic encoder corresponding to the arithmetic decoder as defined in subclause 7.5.3 is used. The probability table of the arithmetic encoder for the 1024 contexts was derived from sequences that are outside of the test set used for comparing different shape coders. With two bytes allocated to describe the symbol probability for each context, the table size is 2048 bytes. In order to avoid emulation of start codes like vop start code, the arithmetic coder stuffs one ?1? into the bitstream whenever a long sequence of ?0? is sent.

The template extends up to 2 pels to the left, to the right and to the top of the pel to be coded (Figure M-2a). Hence, for encoding the pels in the 2 top and left rows of a bab, parts of the template are defined by the shape information of the already transmitted babs on the top and on the left side of the current bab. For the 2 right-most columns, each undefined pel of the context is set to the value of its closest neighbor inside the bab. In case the top and left babs are transparent, the top and left 2 rows of the current bab are duplicated in order to define the context for the template for intra mode.

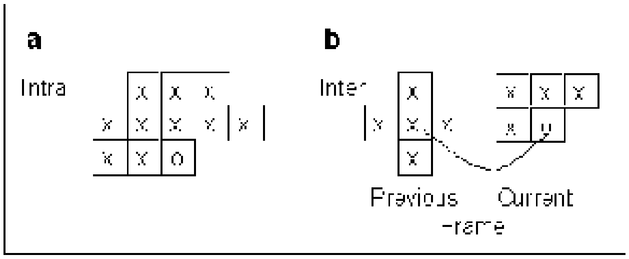


Figure -2 -- Templates for defining the context of the pel to be coded (o), a) defines the intra mode context, b) the context when coding in inter mode. The alignment is done after motion compensating the previous VOP [1].

In order to increase coding efficiency as well as to allow lossy shape coding, a bab can be subsampled by a factor of 2 or 4 resulting in a sub-block of size 8*8 pels or 4*4 pels, respectively. The sub-block is encoded using the encoder as described above. The encoder transmits to the decoder the subsampling factor such that the decoder decodes the shape data and then upsamples the decoded sub-block to macroblock size. Obviously, encoding the shape using a high subsampling factor is more efficient, but the decoded shape after upsampling may or may not be the same as the original shape. Hence, this subsampling is mostly used for lossy shape coding. In order to achieve smooth object boundaries after upsampling, an adaptive non-linear upsampling filter is used. The context of this upsampling filter is shown in Figure M-3.

The efficiency of the shape coder differs depending on the orientation of the shape data. Therefore the encoder can choose to code the bab using horizontal scanning as described above or transpose the macroblock using vertical scanning prior to arithmetic coding.

2. Inter Mode

In order to exploit temporal redundancy in the shape information, the coder described above is extended by an inter mode requiring motion compensation and a different template for defining the context.

For motion compensation, a 2D integer pel motion vector is estimated using full search for each bab in order to minimize the prediction error between the previous coded vop shape M?k-1 and the current shape Mk. The shape motion vectors are predictively encoded with respect to the shape motion vectors of neighboring macroblocks. If no shape motion vector is available, texture motion vectors are used as predictors. The shape motion vector of the current block is used to align a new template designed for coding shape in inter mode (Figure M-2b). The template defines a context of 9 pels resulting in 512 contexts. The probability for one symbol is described by 2 bytes giving a probability table size of 1024 bytes. Four pels of the context are neighbors of the pel to be coded, 5 pels are located at the motion compensated location in the previous vop. Assuming that the motion vector (dx,dy)T points from the current vopk to the previous coded vop?k-1, the part of the template located in the previously coded shape is centered at m?(x-dx,y-dy) with (x,y)T being the location of the current pel to be coded.

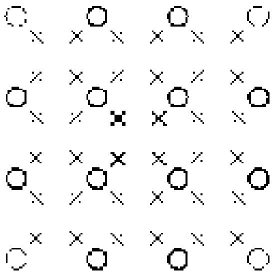


Figure -3 -- For shape upsampling, the upsampled pels (x) lie between the location of the subsampled pels (o). Neighboring pels (bold o) defining the values (transparent or opaque) of the pels to be upsampled (bold x).

In inter mode, the same options as in intra mode like subsampling and transposing are available. For lossy shape coding, the encoder may also decide that the shape representation achieved by just carrying out motion compensation is sufficient thus saving bits by avoiding the coding of the prediction error. The encoder can select one of 7 modes for the shape information of each bab: transparent, opaque, intra, inter with and without shape motion vectors, and inter with/without shape motion vectors and prediction error coding. These different options with optional subsampling and transposition allow for encoder implementations of different coding efficiency and implementation complexity.

3. Texture Coding of Boundary Blocks

For motion compensated prediction of the texture of the current vop, the reference vop is motion compensated using block motion compensation. In order to guarantee that every pel of the current vop has a value to be predicted from, all of the boundary blocks and some of the transparent blocks of the reference vop have to be padded using motion compensation (MC) padding. Boundary blocks are padded using repetitive padding: First boundary pels are replicated in horizontal direction, then in vertical direction making sure that if a value can be assigned to a pel by both padding directions an average value is assigned to the pel. Since this repetitive padding puts a significant computational burden on the decoder, a simpler mean padding is used in a second step. Transparent macroblocks bordering boundary blocks are assigned to an average value determined by the pels of its neighboring padded blocks.

In order to encode the texture of a boundary block, this part of ISO/IEC 14496 treats the macroblock as a regular macroblock and encodes each block using an 8*8 DCT. The decoder decodes the texture and discards all information that falls outside of the decoded shape. In order to increase coding efficiency, the encoder can choose the

texture of pels outside of the object such that the bitrate is minimized. This non-normative process is called texture padding. For intra mode, a lowpass extrapolation filter increasing the PSNR of boundary blocks by an average of 2dB was developed [4]. For inter mode setting these pels to 0 gives good efficiency.

Vop texture may be encoded using B-frames. The related shape information of the B-frame is coded using the inter mode as described in below using the closest P or I frame as a reference vop.

4. Encoder Architecture

Figure M-4a shows the block diagram of this object-based video coder. This diagram focuses on the object-based mode in order to allow a better understanding of how shape coding influences the encoder and decoder. Image analysis creates the bounding rectangle for the current vop S_k and estimates texture and shape motion of the current vop S_k with respect to the reference vop $S?_{k-1}$. Parameter coding encodes the parameters predictively. The parameters get transmitted, decoded and the new reference vop is stored in the vop memory and also handed to the compositor of the decoder for display. The increased complexity due to the coding of arbitrarily shaped video objects becomes evident in Figure M-4b that shows a detailed view of the parameter coding.

Since this part of ISO/IEC 14496 defines motion vectors for motion compensation of texture and for motion compensation of shape, we refer to them as texture motion vectors and shape motion vectors, respectively. The parameter coder encodes first the shape of each boundary block using shape motion vectors and texture motion vectors for prediction. Lossy shape coding may change a boundary block to a transparent block. If the boundary block is not transparent, its shape motion vector is coded. The shape motion coder knows which motion vectors to code by analyzing the possibly lossily encoded shape parameters.

For texture prediction, the reference vop is padded as described above. The prediction error is then padded using the original shape parameters to determine the area to be padded. Using the original shape as a reference for padding is again an encoder choice. Using the original or the coded shape for defining the texture of a macroblock makes obviously only a difference if the shape is coded lossily. Assuming that the texture between object and background in the original vop differs significantly, using the original shape to define the area to be padded in the macroblock prevents the background texture from influencing the padding process. Texture padding will use the object texture for padding resulting in an efficiently to code block. If on the other hand the coded shape is used to define the area to be padded, background texture may affect the padding result potentially increasing the bitrate for texture coding drastically. Finally, the texture of each macroblock is encoded using DCT.

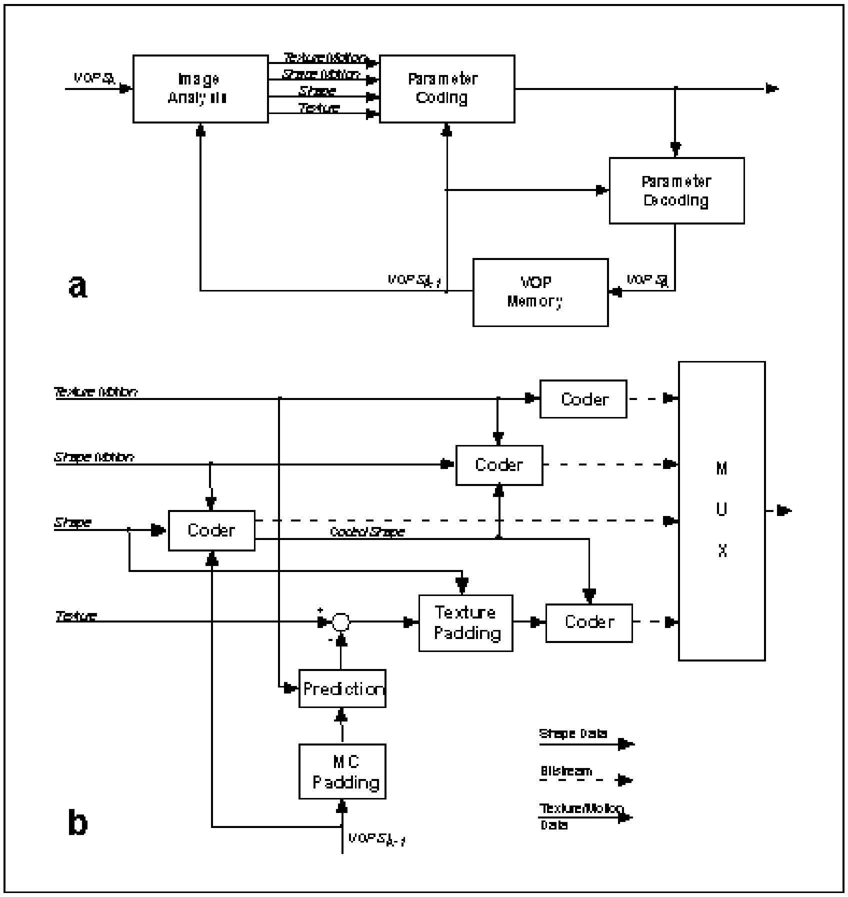


Figure -4 -- Block diagram of the video encoder (a) and the parameter coder (b) for coding of arbitrarily shaped video objects.

5. Encoding Guidelines

When encoding arbitrarily shaped VOs several new encoding options become available when compared to coding of rectangular video sequences. Here we provide some hints on lossy shape coding, shape coding mode selection

1. Lossy Shape Coding

The decoder has no possibility to find out whether the encoder uses lossless or lossy shape coding, what shape coding strategy the encoder uses or texture padding algorithm is used. Therefore, post-processing filter for the binary shape information may not be used. In its reference implementation of a video coder, this part of ISO/IEC 14496 chose to control lossy shape coding by using an alpha-threshold. The alpha threshold defines the maximum number of incorrectly coded pels within a boundary block. A fixed or adaptive alpha-threshold can be used for each bab. Using an alpha threshold may change the topology of the shape. Often, isolated pels at the object boundary are coded as part of the object. Using morphological filters at the encoder to smooth object boundaries or to filter

the prediction error with respect to the original shape provides a much more predictable quality of a lossily encoded shape. Furthermore, the encoder can choose to neglect the prediction error signal at the object boundary if it does not change the topology of the object and to update the prediction error in those parts of the block where the prediction error leads to a change of the object topology.

2. Coding Mode Selection

As far as shape coding is concerned, this part of ISO/IEC 14496 allows for 18 coding modes of each BAB: (Intra/inter/inter with MC)*(horizontal/vertical scanning)*(Subsampling factor 0/1/2). The influence of different shape coding modes on the performance of the coder in terms of coding efficiency but also in terms of computational complexity is of interest, since the shape coder including padding can account for up to 50% of the encoder operations [3]. When encoding a binary shape, checking every mode achieves the highest compression. However, the gain achieved by this checking should be evaluated in relation to the overall bitrate required for coding a vop. Table M-1 shows the main usage for each coding mode.

Table -1 -- Shape coding modes and their main usage

	Mode	**Main Usage**
1	intra	I frames, arbitrarily shaped still texture object, error resilience
2	inter, inter MC	P frames
3	horizontal/vertical scanning	Low-bitrate shape coding
4	subsampling to a block size 8x8 or 4x4	Low-bitrate lossy shape coding

The intra mode is used for I frames and for coding the shape of the arbitrarily shaped still texture object. Similarly, the use of the intra mode as well as the inter mode with and without motion compensation should be investigated for coding of P frames. For low-bitrate applications, the shape usually requires a significant percentage of the overall bitrate. Then it makes sense for the encoder to investigate the gain of adapting the scanning direction of each BAB reducing the shape bit rate by up to 8%. In case of lossy shape coding at low bit rates, the subsampling filter reduces the bit rate for shape coding by up to 20% for an alpha threshold of 16. The use of the subsampling filter is most effective for I babs. In case of lossless shape coding, the coder should signal at the beginning of each vop that the subsampling feature would not be used. This avoids the overhead of signaling the subsampling for each BAB.

6. Conclusions

This part of ISO/IEC 14496 standardizes an object-based decoder. Several choices regarding texture extrapolation at object boundaries and shape encoding are available to the encoder. It was found that for texture extrapolation a simple filter setting the texture outside of the *original* object shape to gray is sufficient. In the case of lossy shape coding, not using the *original* object shape for defining the support region of the texture extrapolation may increase the overall bitrate by more than 100%. Therefore, lossy shape coding makes only sense if the encoder architecture presented in this paper is used.

Fixing the bab scanning to horizontal instead of adaptively switching between horizontal and vertical scanning increases the overall bitrate by just 1% but saves significant compute time since the compute intensive arithmetic coding with context determination has to be used only half as often.

The downsampling of the shape information increases the computational complexity of the encoder. In the case of lossy shape coding in intra mode, the subsampling increases the subjective quality of the coded object shape due to a non-linear shape upsampling filter. For low bitrate video coding using lossily coded shapes, this filter reduces the overall bitrate.

As the experiments show, the ISO/IEC 14496-2 binary shape coder can operate in a very bit efficient manner. In order to save computational complexity for high bit rate applications, the encoder may choose to not explore all of the encoding options and choose a less efficient shape encoding. The increase of the overall bitrate may be less than 1% at high bit rates but the computational complexity of the encoder decreases by 15% to 50%.

7. References

[1] N. Brady, F. Bossen, N. Murphy, "Context-based arithmetic encoding of 2D shape sequences", Special session on shape coding, ICIP 97, Santa Barbara, 1997

[2] A. Katsaggelos, L. Kondi, F. Meier, J. Ostermann, G. Schuster, ‘‘MPEG-4 and rate-distortion based shape coding techniques,’’ Proceedings of the IEEE, pp. 1126-1154, June 1998,

[3] J. Ostermann, ‘‘Coding of arbitrarily shaped video objects with binary and greyscale alpha maps: What can MPEG-4 do for you?’’ ISCAS 98, Monterey, CA, TPA 2-1, June 1998.

[4] A. Kaup, "Adaptive low-pass extrapolation for object-based texture coding of moving video", Proc. Visual Communications and Image Processing, SPIE vol. 3024, pp. 731-741, Feb. 1997.

C.

(normative)

Visual profiles@levels

The table that describes the visual profiles and levels is given below, with the following notes:

1. Enhancement layers are not counted as separate objects.

2. The maximum VMV buffer size is the bound on the memory (in macroblock units) which can be used by the VMV algorithm. This algorithm (see clause D.5) models the pixel memory needed by the entire visual decoding process. This includes memory needed for reference VOPs in the prediction of P- and B-VOPs and the storage of the reconstructed VOPs until such time that they are released by the decoder, plus the memory required to queue B-VOPs until composition occurs. In profiles that contain more than one layer, the memory requirements include all base and enhancement layers. Except for the Simple Visual Profile, some of these macroblocks may overlay on the display, however separate memory is required (prior to composition) in the VMV.

3. The conformance point for the Simple Scalable Visual Profile Levels is the Simple Profile @L1 when spatial scalability is used and Simple Profile @ L2 when temporal scalability is used.

4. The VCV decoder rate is the vcv_decoder_rate (H) referred to in clause D.4; it is the number of macroblocks/second based on the typical spatial and temporal resolutions, as follows:

- 1485 MBs/s corresponds to QCIF at 15Hz.

- 5940 MBs/s corresponds to CIF at 15 Hz and also twice QCIF at 30 Hz.

- 11880 MB/s corresponds to CIF at 30 Hz.
- 7425 MB/s corresponds to 1.25 times CIF at 15 Hz.
- 23760 MB/s corresponds to twice CIF at 30 Hz.
- 97200 MB/s corresponds to twice CCIR 601 at 30 Hz.
- 489600 MB/s corresponds to twice 1920x1088 at 30 Hz.

1. The total (aggregated) vbv_buffer_size is the sum of the individual VBV buffer occupancies at any given time (in units of 16384 bits) for all VOLs of all VOs. This total VBV size is limited according to the profile and level. The VBV buffer size and the maximum total VBV sizes are specified in units of 16384 bits.

2. The maximum video packet length is defined as the maximum number of bits from the start of one slice to the start of the next slice. The constraint applies only when the data-partitioning tool is enabled in the bitstream. When data partitioning is disabled, there is no limit on the size of video packet length.

3. N. A. means Not Applicable.

4. The maximum VCV buffer size (cumulative over all layers of all VOs) is twice the maximum number of macroblocks per VOP in the profile and level combination except for the Simple Visual Profile and Simple Scalable profile (Level 1). For the Simple Visual Profile, this value is maximum number of macroblocks per VOP. For the Simple Scalable profile (Level 1), it is 1.25 times of the maximum number of macroblocks per VOP. The limit applies to both VCV buffer and the boundary MB VCV buffer.

5. The VCV boundary MB decoder rate column bounds the number of macroblocks containing non trivial shape information. The VCV boundary MB decoder rate constrains the total number of boundary MBs in all VOLs concurrently. Note that the boundary macroblocks are added to both the VCV and boundary VCV buffers.

Table N-1 (1/2) to be overlayed

Table N-1 (2/2) to be overlayed

Table -1 -- Definition of Natural Visual Profiles@Levels

Visual Profile	Level	Typical Visual Session Size	Max MBs per VOP	Max ob-jects[1]	Maximum number per type	Max unique Quant Tables	Max. VMV buffer size (MB units)[2]	Max VCV buffer size (MB)[8]	VCV decoder rate (MB/s)[4]	VCV Bound-ary MB decoder rate (MB/s)[9]	Max total VBV buffer size[5]	Max vbv buffer size[5]	Max. video packet length (bits)[6]	Max sprite size (MB units)
N-Bit	L2	CIF	396	16	16 x Core or Simple or N-Bit	4	2376	792	23760	11880	80	40	4096	N. A.[7]
Main	L4	1920 x 1088	8160	32	32 x Main or Core or Simple	4	48960	16320	489600	244800	760	380	16384	65280
Main	L3	CCIR 601	1620	32	32 x Main or Core or Simple	4	9720	3240	97200	48600	320	160	16384	6480
Main	L2	CIF	396	16	16 x Main or Core or Simple	4	2376	1188	23760	11880	80	40	8192	1584
Core	L2	CIF	396	16	16 x Core or Simple	4	2376	792	23760	11880	80	40	8192	N. A.
Core	L1	QCIF	99	4	4 x Core or Simple	4	594	198	5940	2970	16	8	4096	N. A.

Simple [3] Scalable	L2	CIF	396	4	4 x Simple or Simple Scalable	1	3168	792	23760	N.A.	40	20	4096
Simple Scalable	L1	CIF	396	4	4 x Simple or Simple Scalable	1	1782	495	7425	N. A.	40	20	2048
Simple	L3	CIF	396	4	4 x Simple	1	792	396	11880	N. A.	40	20	8192
Simple	L2	CIF	396	4	4 x Simple	1	792	396	5940	N. A.	40	20	4096
Simple	L1	QCIF	99	4	4 x Simple	1	198	99	1485	N.A.	10	5	2048