INTERNATIONAL STANDARD

ISO/IEC 14496-2

Third edition
2004-06-01

Information technology — Coding of audio-visual objects — Part 2: Visual

Technologies de l'information — Codage des objets audiovisuels — Partie 2: Codage visuel

Reference number
ISO/IEC 14496-2:2004(E)

© ISO/IEC 2004

APPLE-1042

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO/IEC 2004

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

1 Scope ... 1

2 Normative references ... 1

3 Terms and definitions ... 1

4 Abbreviations and symbols ... 13
4.1 Arithmetic operators ... 13
4.2 Logical operators ... 14
4.3 Relational operators ... 14
4.4 Bitwise operators ... 15
4.5 Conditional operators ... 15
4.6 Assignment ... 15
4.7 Mnemonics ... 15
4.8 Constants ... 15

5 Conventions ... 16
5.1 Method of describing bitstream syntax ... 16
5.2 Definition of functions ... 17
5.3 Reserved, forbidden and marker_bit ... 18
5.4 Arithmetic precision ... 19

6 Visual bitstream syntax and semantics ... 19
6.1 Structure of coded visual data ... 19
6.2 Visual bitstream syntax ... 38
6.3 Visual bitstream semantics ... 135

7 The visual decoding process ... 236
7.1 Video decoding process ... 237
7.2 Higher syntactic structures ... 238
7.3 VOP reconstruction ... 238
7.4 Texture decoding ... 239
7.5 Shape decoding ... 250
7.6 Motion compensation decoding ... 274
7.7 Interlaced video decoding ... 297
7.8 Sprite decoding ... 306
7.9 Generalized scalable decoding ... 313
7.10 Still texture object decoding ... 323
7.11 Mesh object decoding ... 347
7.12 FBA object decoding ... 352
7.13 3D Mesh Object Decoding ... 358
7.14 NEWPRED mode decoding ... 384
7.15 Output of the decoding process ... 385
7.16 Video object decoding for the studio profile ... 385
7.17 The FGS decoding process ... 427

8 Visual-Systems Composition Issues ... 429
8.1 Temporal Scalability Composition ... 429
8.2 Sprite Composition ... 430
8.3 Mesh Object Composition ... 431
8.4 Spatial Scalability composition ... 432

9 Profiles and Levels ... 432
9.1 Visual Object Types ... 432
9.2 Visual Profiles ... 436
9.3 Visual Profiles@Levels ... 437

© ISO/IEC 2004 – All rights reserved

Annex A (normative) **Coding transforms** **441**
Annex B (normative) **Variable length codes and arithmetic decoding** **451**
Annex C (normative) **Face and body object decoding tables and definitions** **547**
Annex D (normative) **Video buffering verifier** **580**
Annex E (informative) **Features supported by the algorithm** **589**
Annex F (informative) **Preprocessing and postprocessing** **599**
Annex G (normative) **Profile and level indication and restrictions** **625**

© ISO/IEC 2004 – All rights reserved

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 14496-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This third edition cancels and replaces the second edition (ISO/IEC 14496-2:2001), which has been technically revised. It also incorporates the Amendments ISO/IEC 14496-2:2001/Amd. 1:2002, ISO/IEC 14496-2:2001/Amd. 2:2002 and ISO/IEC 14496-2:2001/Amd. 3:2003.

ISO/IEC 14496 consists of the following parts, under the general title *Information technology — Coding of audio-visual objects*:

— *Part 1: Systems*

— *Part 2: Visual*

— *Part 3: Audio*

— *Part 4: Conformance testing*

— *Part 5: Reference software*

— *Part 6: Delivery Multimedia Integration Framework (DMIF)*

— *Part 7: Optimized reference software for coding of audio-visual objects*

— *Part 8: Carriage of ISO/IEC 14496 content over IP networks*

— *Part 9: Reference hardware description*

— *Part 10: Advanced video coding*

— *Part 11: Scene description and application engine*

— *Part 12: ISO base media file format*

— *Part 13: Intellectual Property Management and Protection (IPMP) extentions*

© ISO/IEC 2004 – All rights reserved

— *Part 14: MP4 file format*

— *Part 15: Advanced Video Coding (AVC) file format*

— *Part 16: Animation framework extension (AFX)*

— *Part 17: Streaming text format*

— *Part 18: Font compression and streaming*

— *Part 19: Synthesized texture stream*

© ISO/IEC 2004 – All rights reserved

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

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 13818-2:2000 can be combined to form other decoding algorithms. Considering the practicality of implementing the full syntax of this part of ISO/IEC 14496, 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.

© ISO/IEC 2004 – All rights reserved

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.

FBA object

The FBA object is a collection of nodes in a scene graph which are animated by the FBA (Face and Body Animation) object bitstream. The FBA object is controlled by two separate bitstreams. The first bitstream, called BIFS, contains instances of Body Definition Parameters (BDPs) in addition to Facial Definition Parameters (FDPs), and the second bitstream, FBA bitstream, contains Body Animation Parameters (BAPs) together with Facial Animation Parameters (FAPs).

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 quantised 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 14496-1:2001. 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 14496-1:2001, 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.

© ISO/IEC 2004 – All rights reserved

A body model is a representation of a virtual human or human-like character that allows portraying body movements adequate to achieve nonverbal communication and general actions. A body model is animated by a stream of *body animation parameters* (BAP) encoded for low-bitrate transmission in broadcast and dedicated interactive communications. The BAPs manipulate independent degrees of freedom in the skeleton model of the body to produce animation of the body parts. The BAPs are quantised considering the joint limitations, and prediction errors are calculated and coded arithmetically. Similar to the face, the remote manipulation of a body model in a terminal with BAPs can accomplish lifelike visual scenes of the body in real-time without sending pictorial and video details of the body every frame.

The BAPs, if correctly interpreted, will produce reasonably similar high level results in terms of body posture and animation on different body models, also without the need to initialize or calibrate the model.The BDP set defines the set of parameters to transform the default body to a customized body optionally with its body surface, body dimensions, and texture.

The *body definition parameters (BDP)* allow the encoder to replace the local model of a more capable terminal. BDP parameters include body geometry, calibration of body parts, degrees of freedom, and optionally deformation information.

The FBA Animation specification is defined in ISO/IEC 14496-1:2001 and this part of ISO/IEC 14496. This clause is intended to facilitate finding various parts of specification. As a rule of thumb, FAP and BAP specification is found in the part 2, and FDP and BDP specification in the part 1. However, this is not a strict rule. For an overview of FAPs/BAPs 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. FBA bitstream syntax is found in subclauses “6.2.10 FBA Object”, semantics in “6.3.10 FBA Object”, and subclause “7.12 FBA object decoding” explains in more detail the FAP/BAP decoding process. FAP/BAP masking and interpolation is explained in subclauses “6.3.11.1 FBA Object Plane”, “7.12.1.1 Decoding of FBA”, “7.12.5 FBA masking” . The FIT interpolation scheme is documented in subclause “7.2.5.3.2.4 FIT” of ISO/IEC 14496-1:2001. The FDPs and BDPs and their interpretation are documented in subclause “7.2.5.3.2.6 FDP” of ISO/IEC 14496-1:2001. In particular, the FDP feature points are documented in Figure C-1. Details on body models are documented in Annex C.

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.

3D Mesh Object

The 3D Mesh Object is a 3D polygonal model that can be represented as an IndexedFaceSet or Hierarchical 3D Mesh node in BIFS. It is defined by the position of its vertices (geometry), by the association between each face and its sustaining vertices (connectivity), and optionally by colours, normals, and texture coordinates (properties). Properties do not affect the 3D geometry, but influence the way the model is shaded. 3D mesh coding (3DMC) addresses the efficient coding of 3D mesh object. It comprises a basic method and several options. The basic 3DMC method operates on manifold model and features incremental representation of single resolution 3D model. The model may be triangular or polygonal – the latter are triangulated for coding purposes and are fully recovered in the decoder. Options include: (a) support for computational graceful degradation control; (b) support for non-manifold model; (c) support for error resilience; and (d) quality scalability via hierarchical transmission of levels of detail with implicit support for smooth transition between consecutive levels. The compression of application-specific geometry streams (Face Animation Parameters) and generalized animation parameters (BIFS Anim) are currently addressed elsewhere in this part of ISO/IEC 14496.

In 3DMC, the compression of the connectivity of the 3D mesh (e.g. how edges, faces, and vertices relate) is lossless, whereas the compression of the other attributes (such as vertex coordinates, normals, colours, and texture coordinates) may be lossy.

© ISO/IEC 2004 – All rights reserved

Single Resolution Mode

The incremental representation of a single resolution 3D model is based on the Topological Surgery scheme. For manifold triangular 3D meshes, the Topological Surgery representation decomposes the connectivity of each *connected component* into a *simple polygon* and a *vertex graph*. All the triangular faces of the 3D mesh are connected in the simple polygon forming a *triangle tree*, which is a spanning tree in the dual graph of the 3D mesh. Figure 0-1 shows an example of a triangular 3D mesh, its dual graph, and a triangle tree. The vertex graph identifies which pairs of boundary edges of the simple polygon are associated with each other to reconstruct the connectivity of the 3D mesh. The triangle tree does not fully describe the triangulation of the simple polygon. The missing information is recorded as a *marching edge*.

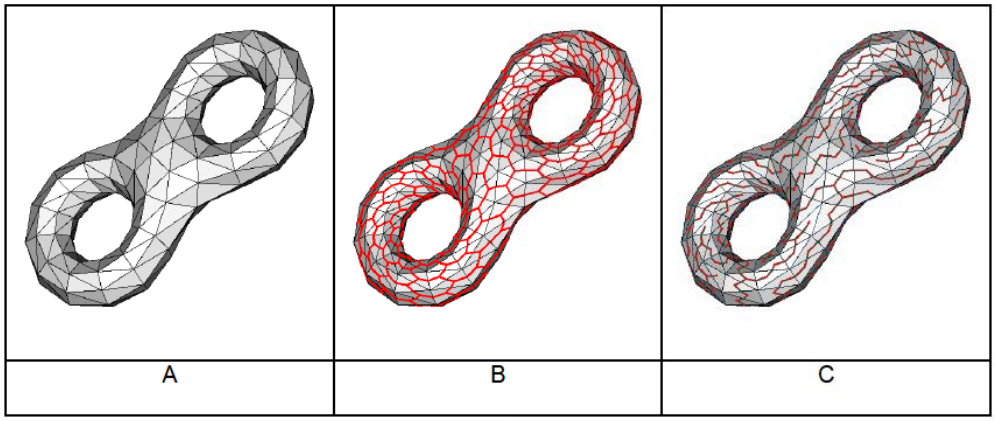


Figure 0-1 — A triangular 3D mesh (A), its dual graph (B), and a triangle tree (C)

For manifold 3D meshes, the connectivity is represented in a similar fashion. The polygonal faces of the 3D mesh are connected in a simple polygon forming a *face tree*. The faces are triangulated, and which edges of the resulting triangular 3D mesh are edges of the original 3D mesh is recorded as a sequence of *polygon_edge* bits. The face tree is also a spanning tree in the dual graph of the 3D mesh, and the vertex graph is always composed of edges of the original 3D mesh.

The vertex coordinates and optional properties of the 3D mesh (normals, colours, and texture coordinates) are quantised, predicted as a function of decoded ancestors with respect to the order of traversal, and the errors are entropy encoded.

Incremental Representation

When a 3D mesh is downloaded over networks with limited bandwidth (e.g. PSTN), it may be desired to begin decoding and rendering the 3D mesh before it has all been received. Moreover, content providers may wish to control such incremental representation to present the most important data first. The basic 3DMC method supports this by interleaving the data such that each triangle may be reconstructed as it is received. Incremental representation is also facilitated by the options of hierarchical transmission for quality scalability and partitioning for error resilience.

Hierarchical Mode

An example of a 3D mesh represented in hierarchical mode is illustrated in Figure 0-2. The hierarchical mode allows the decoder to show progressively better approximations of the model as data are received. The hierarchical 3D mesh decomposition can also be organized in the decoder as layered detail, and view-dependent expansion of this detail can be subsequently accomplished during a viewer's interaction with the 3D model. Downloadable scalability of 3D model allows decoders with widely varied rendering performance to use the content, without necessarily making repeated requests to the encoder for specific versions of the content and without the latencies of updating view-dependent model from the encoder. This quality scalability of 3D meshes is complementary to scaleable still texture and supports bitstream scalability and downloading of quality hierarchies of hybrid imagery to the decoder.

© ISO/IEC 2004 – All rights reserved

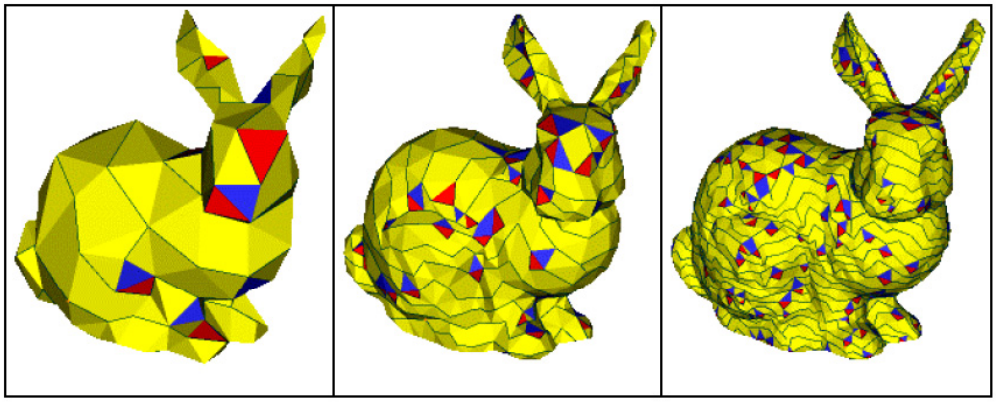


Figure 0-2 — A hierarchy of levels of detail

The hierarchical representation of a 3D model is based on the Progressive Forest Split scheme. In the Progressive Forest Split representation, a manifold 3D mesh is represented as a *base 3D mesh* followed by a sequence of *forest split* operations. The base 3D mesh is encoded as a single resolution 3D mesh using the Topological Surgery scheme. Each forest split operation is composed of a *forest* in the graph of the current 3D mesh, and a *sequence of simple polygons*. Figure 0-3 illustrates the method. To prevent visual artifacts which may occur while switching from a level of detail to the next one, the data structure supports smooth transition between consecutive levels of detail in the form of linear interpolation of vertex positions.

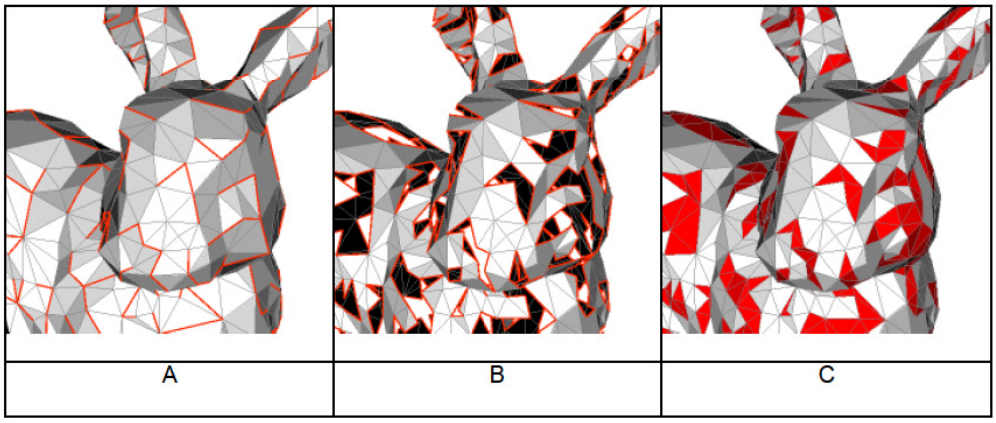


Figure 0-3 — The Forest Split operation. A 3D mesh with a forest of edges marked (A). The tree loops resulting from cutting the 3D mesh through the forest edges (B). Each tree loop is filled by a simple polygon composed of polygonal faces (C).

Error Resilience for 3D mesh object

If the 3D mesh is partitioned into independent parts, it may be possible to perform more efficient data transmission in an error-prone environment, e.g., an IP network or datacasting service in a broadcast TV network. It must be possible to resynchronize after a channel error, and continue data transmission and rendering from that point instead of starting over from scratch. Even with the presence of channel errors, the decoder can start decoding and rendering from the next partition that is received intact from the channel.

Flexible partitioning methods can be used to organize the data, such that it fits the underlying network packet structure more closely, and overhead is reduced to the minimum. To allow flexible partitioning, several connected components may be merged into one partition, where as a large connected component may be divided into several independent partitions. Merging and dividing of connected components using different partition types can be done at any point in the 3D mesh object.

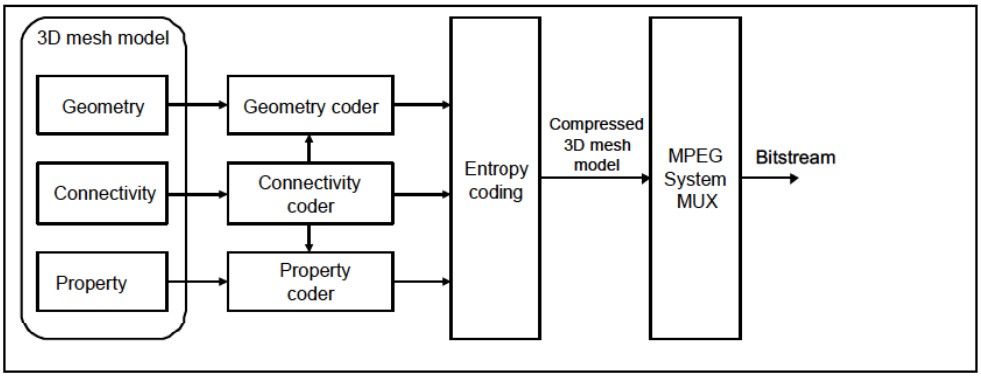
© ISO/IEC 2004 – All rights reserved

Stitching for Non-Manifold and Non-Orientable Meshes

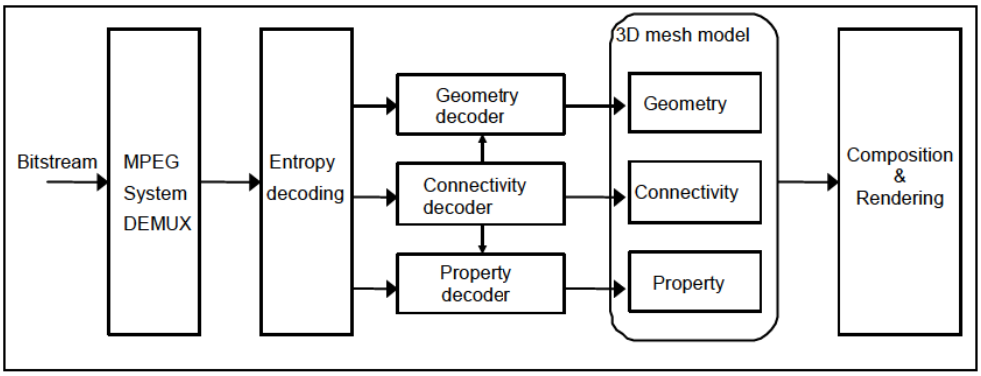
The connectivity of a non-manifold and non-orientable 3D mesh is represented as a manifold 3D mesh and a sequence of *stitches*. Each stitch describes how to identify one or more pairs of vertices along two linear paths of edges, and each one of these paths is contained in one of the vertex graphs that span the connected components of the manifold 3D mesh.

Encoder and Decoder Block Diagrams

High level block diagrams of a general 3D polygonal model encoder and decoder are shown in Figure 0-4. They consist of a 3D mesh connectivity (de)coder, geometry (de)coder, property (de)coder, and entropy (de)coding blocks. Connectivity, vertex position, and property information are extracted from 3D mesh model described in VRML or MPEG-4 BIFS format. The connectivity (de)coder is used for an efficient representation of the association between each face and its sustaining vertices. The geometry (de)coder is used for a lossy or lossless compression of vertex coordinates. The property (de)coder is used for a lossy or lossless compression of colour, normal, and texture coordinate data.



(a) 3D mesh encoder



(b) 3D mesh decoder

Figure 0-4 — General block diagram of the 3D mesh compres

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 in a lossy or lossless fashion.

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

© ISO/IEC 2004 – All rights reserved

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.

In order to preserve the lossless quality, or to restrict the maximum bit count of block data, the block based DPCM coding can be used for Studio Profiles.

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 Studio Profiles particularly, HHC binary alpha block coding is used.

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.

Coding interlaced video

Each frame of interlaced video consists of two fields which are separated by one field-period. Studio Profiles allow either the frame to be encoded as a VOP or the two fields to be encoded as two VOPs. Frame encoding or field encoding can be adaptively selected on a frame-by-frame basis. Frame encoding is typically preferred when the video scene contains significant detail with limited motion. Field encoding, in which the second field can be predicted from the first, works better when there is fast movement.

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. In order to encode the highly active scene with higher VOP rate, a Reduced Resolution VOP tool is provided. When this tool is used , the size of the macroblock used for motion compensation decoding is 32x32 pixels and the size of block is 16x16 pixels.

For Studio Profiles particularly, in frame encoding, the prediction from the previous reference frame can itself be either frame-based or field-based.

© ISO/IEC 2004 – All rights reserved

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. Studio Profiles support the 4:2:2 and 4:4:4 chorominance formats in addition.

RGB color components

Studio Profiles support coding of RGB color components. The resolution of each component shall be identical when input data is treated as RGB color components.

Pixel depth

This part of ISO/IEC 14496 supports pixel depths between 4 and 12 bits in luminance and chrominance planes. IStudio Profiles support 8, 10 and 12 bits in luminance and chrominance or RGB 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 13818-2:2000 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.

© ISO/IEC 2004 – All rights reserved

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.

Object 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.

Object based spatial scalability is composed of texture spatial scalability and shape spatial scalability. They can be used independently or together by its application. Binary shape spatial scalability is used for the applications that have 'binary shape only' mode as well as the applications of general object based spatial scalability.

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.

Fine Granularity Scalability

Two profiles are developed in response to the growing need for a video coding method for Streaming Video on Internet applications. It provides the definition and description of Advanced Simple (AS) Profile and Fine Granularity Scalable (FGS) Profile. AS Profile provides the capability to distribute single-layer frame based video at a wide range of bit rates available for the distribution of video on Internet. FGS Profile uses AS Video Object in the base layer and provides the description of two enhancement layer types - Fine Granularity Scalability (FGS) and FGS Temporal Scalability (FGST). FGS Profile allows the coverage of a wide range of bit rates for the distribution of video on Internet with the flexibility of using multiple layers, where there is a wide range of bandwidth variation.

Fine Granularity Scalability (FGS) provides quality scalability for each VOP. Figure 0-5 shows a basic FGS decoder structure.

© ISO/IEC 2004 – All rights reserved

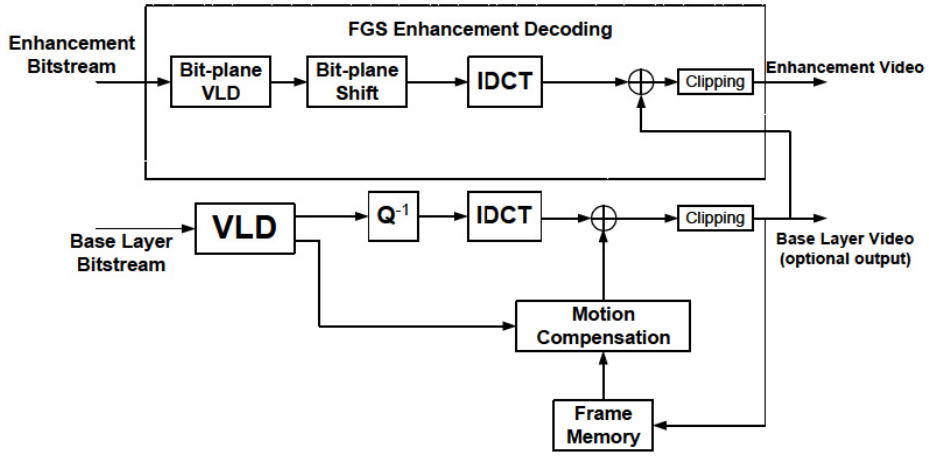


Figure 0-5 — A Basic FGS Decoder Structure

To reconstruct the enhanced VOP, the enhancement bitstream is first decoded using bit-plane VLD. The decoded block-bps are used to reconstruct the DCT coefficients in the DCT domain which are then right-shifted based on the frequency weighting and selective enhancement shifting factors. The output of bit-plane shift is the DCT coefficients of the image domain residues. After the IDCT, the image domain residues are reconstructed. They are added to the reconstructed clipped base-layer pixels to reconstruct the enhanced VOP. The reconstructed enhanced VOP pixels are limited into the value range between 0 and 255 by the clipping unit in the enhancement layer to generate the final enhanced video. The reconstructed base layer video is available as an optional output since each base layer reconstructed VOP needs to be stored in the frame buffer for motion compensation.

The basic FGS enhancement layer consists of FGS VOPs that enhance the quality of the base-layer VOPs as shown in Figure 0-6.

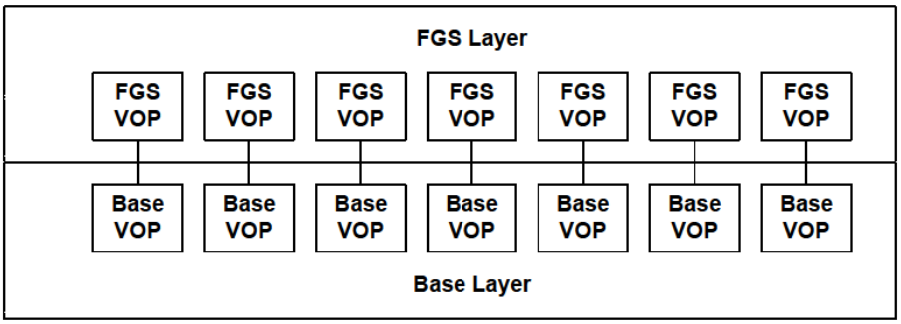


Figure 0-6 — Basic FGS Enhancement Structure

When FGS temporal scalability (FGST) is used, there are two possible enhancement structures. One structure is to have two separate enhancement layers for FGS and FGST as shown in Figure 0-7 and the other structure is to have one combined enhancement layer for FGS and FGST as shown in Figure 0-8.

© ISO/IEC 2004 – All rights reserved

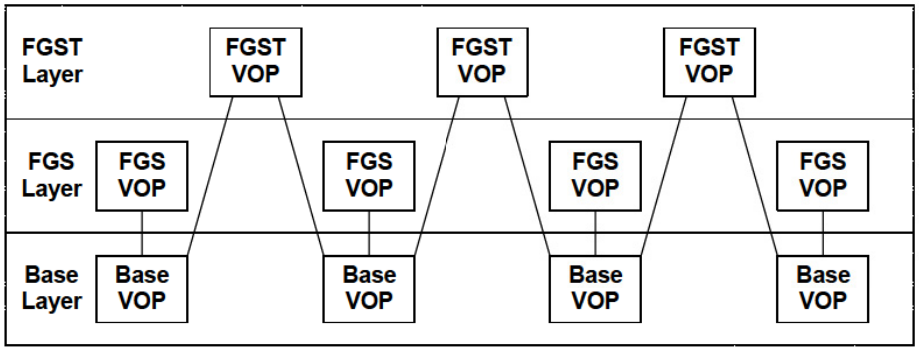


Figure 0-7 — Two Separate Enhancement Layers for FGS and FGST

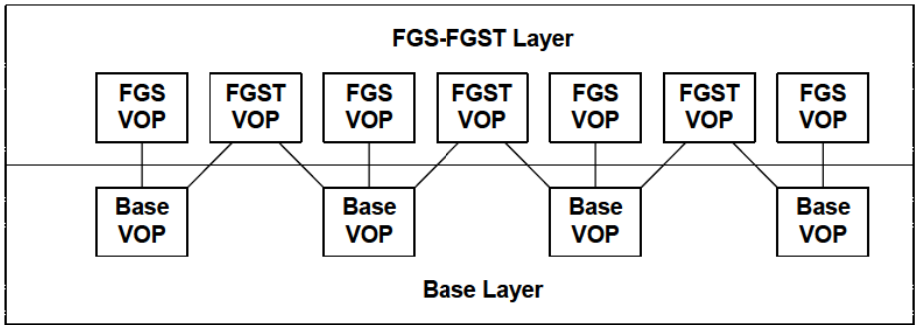


Figure 0-8 — One Combined Enhancement Layer for FGS and FGST

In either one of these two structures that include FGS temporal scalability, the prediction for the FGS temporal scalable VOPs can only be from the base layer. Each FGS temporal scalable VOP has two separate parts. The first part contains motion vector data and the second part contains the DCT texture data. The syntax for the first part is similar to that in the temporal scalability described in subclause 6.2. The DCT texture data in the second part are coded using bit-plane coding in the same way as that in FGS. To distinguish the temporal scalability in subclause 6.2 and FGS temporal scalability, the FGS temporal scalability layer in Figure 0-7 is called "FGST layer". The combined FGS and FGST layer in Figure 0-8 is called "FGS-FGST layer". The "FGS VOP" shown in Figure 0-7 and Figure 0-8 is an fgs vop with **fgs_vop_coding_type** being 'I'. The "FGST VOP" shown in Figure 0-7 and Figure 0-8 is an fgs vop with **fgs_vop_coding_type** being 'P' or 'B'.

The code value of **profile_and_level_indication** in VisualObjectSequence() has been extended to include the profile and level indications for AS Profile and FGS Profile. The identifier for an enhancement layer is the syntax **video_object_type_indication** in VideoObjectLayer(). A unique code is defined for FGS Object Type to indicate that this VOL contains fgs vops. Another unique code is defined for AS Object Type to indicate that this VOL is the base-layer. There is a syntax **fgs_layer_type** in VideoObjectLayer() to indicate whether this VOL is an FGS layer as shown in Figure 0-6 and Figure 0-7, or an FGST layer as shown in Figure 0-7, or an FGS-FGST layer as shown in Figure 0-8. Similar to the syntax structure in subclause 6.2, under each VOL for FGS, there is a hierarchy of fgs vop, fgs macroblock, and fgs block. An fgs vop starts with a unique **fgs_vop_start_code**. Within each fgs vop, there are multiple vop-bps. Each vop-bp in an fgs vop starts with an **fgs_bp_start_code** whose last 5 bits indicate the ID of the vop-bp. In each fgs macroblock, there are 4 block-bps for the luminance component (Y), 2 block-bps for the two chrominance components (U and V) for the 4:2:0 chrominance format. Each block-bp is coded by VLC.

© ISO/IEC 2004 – All rights reserved

Information technology — Coding of audio-visual objects —

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 and body 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 11172-2:1993, ISO/IEC 13818-2:2000 and ITU-T Recommendation H.263.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

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 13818-1:2000, *Information technology — Generic coding of moving pictures and associated audio information: Systems*

ISO/IEC 13818-2:2000, *Information technology — Generic coding of moving pictures and associated audio information: Video*

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

ITU-R Recommendation BT.601-5, *Studio encoding parameters of digital television for studios*

ITU-T Recommendation H.261(1993*), Codec for audiovisual services at px64 kbit/s*

ITU-T Recommendation H.263(2000), *Video Coding for Low Bitrate Communication*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

© ISO/IEC 2004 – All rights reserved

3.1 **3D mesh**: A polygonal mesh. It is defined by the position of its vertices (geometry), by the association between each face and its sustaining vertices (connectivity), and optionally by colors, normals, and texture coordinates (properties). Properties do not affect the 3D geometry, but influence the way the model is shaded.

3.2 **3D mesh connectivity**: The association between each face of a 3D mesh and its sustaining vertices.

3.3 **3D mesh geometry**: The three dimensional coordinates of the vertices of a 3D mesh.

3.4 **3D mesh properties**: The normals, colors, and texture coordinates of a 3D mesh.

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

3.6 **B-field VOP**: A field structure B-VOP.

3.7 **B-frame VOP**: A frame structure B-VOP.

3.8 **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.

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

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

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

3.12 **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.

3.13 **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.

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

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

3.16 **block**: An 8-row by 8-column (or 16-row by 16-column in motion compensation decoding of Reduced Resolution VOP) matrix of samples, or 64 DCT coefficients (source, quantised or inverse quantised).

3.17 **block-bp**: An array of 64 bits, one from each DCT coefficient at the same significant position of accuracy in a zigzag scan order. When frequency weighting is used, block-bps are formed after the weighting is applied to the DCT coefficients in an 8x8 block.

3.18 **body animation parameters, BAP**: Coded streaming animation parameters that manipulate the body joints, and that govern deformations of the body surface with the use of BAT.

3.19 **body definition parameters, BDP**: Downloadable data to customize a baseline body model in the decoder to a particular body, or to download a body model along with the information about how to animate it. The BDPs are normally transmitted once per session, followed by a stream of compressed BAPs. BDPs may include body geometry, body skeleton, and body deformation tables.

3.20 **body deformation table, BodyDefTable**: A downloadable function mapping from incoming BAPs to body surface geometry that provides a combination of BAPs for controlling body surface geometry deformation.

© ISO/IEC 2004 – All rights reserved

3.21 **3.20 bottom field**: One of two fields that comprise a frame. Each line of a bottom field is spatially located immediately below the corresponding line of the top field.

3.22 **boundary (of a 3D mesh)**: The set of boundary edges of a 3D mesh.

3.23 **boundary edge (of a 3D mesh)**: An edge of a 3D mesh which has exactly one incident face.

3.24 **bounding loop:** The boundary of a simple polygon.

3.25 **bounding loop index**: An index of the bounding loop table which contains the list of vertices lying on the bounding loop

3.26 **branching triangle (of a triangle tree)**: A triangle of a triangle tree with three incident triangle tree edges.

3.27 **branching vertex (of a vertex tree or of a vertex graph)**: A vertex of a vertex tree with three or more incident vertex tree edges.

3.28 **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.

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

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

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

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

3.33 **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.

3.34 **coded B-frame**: A B-frame VOP or a pair of B-field VOPs that is coded.

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

3.36 **coded frame**: A coded frame is a coded I-frame, a coded P-frame or a coded B-frame.

3.37 **coded VOP**: A coded VOP is a coded I-VOP, a coded P-VOP, a coded B-VOP, or a coded S(GMC)-VOP.

3.38 **coded I-frame**: An I-frame VOP or a pair of field VOPs that is coded where the first field VOP is an I-VOP and the second field VOP is an I-VOP or a P-VOP.

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

3.40 **coded P-frame**: A P-frame VOP or a pair of field VOPs that is coded.

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

3.42 **coded S(GMC)-VOP**: A S(GMC)-VOP that is coded.

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

© ISO/IEC 2004 – All rights reserved

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

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

3.46 **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.

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

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

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

3.50 **connected component (of a 3D mesh)**: A 3D mesh can be decomposed into one or more disjoint sets called connected components. Every face of the 3D mesh is assigned to an unique connected component, such that any pair of two faces that share an edge belongs to the same connected component.

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

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

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

3.54 **corner (of a 3D mesh)**: A (face,vertex) pair, where the vertex is one of the sustaining vertices of the face.

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

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

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

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

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

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

3.61 **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, or 3D meshes.

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

3.63 **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.

3.64 **3.64 display aspect ratio**: The ratio height/width (in spatial measurement units such as centimeters) of the intended display.

3.65 **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.

© ISO/IEC 2004 – All rights reserved

3.66 **display process**: The (non-normative) process by which reconstructed frames are displayed.

3.67 **dual graph (of a 3D mesh)**: A graph composed of a set of *dual graph nodes* and a set of *dual graph edges*. Each dual graph node corresponds to a face of the 3D mesh. Each dual graph edge corresponds to an internal edge of the 3D mesh and links two dual graph nodes.

3.68 **edge (of a 3D mesh)**: An unordered pair of vertex indices of a 3D mesh, consecutive in one or more faces of the 3D mesh (modulo cyclical permutations of the sequences).

3.69 **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.

3.70 **encoder**: An embodiment of an encoding process.

3.71 **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.

3.72 **end of plane; eop**: A symbol to indicate whether a '1' bit is the last 1' bit of a block-bp.

3.73 **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.

3.74 **face (of a 3D mesh)**: An alternating sequence of vertices and edges, such that each edge in the sequence connects the vertices immediately preceding and following it in the sequence (with the exception of the last edge that connects the vertex preceding it with the first vertex of the sequence). No vertex may appear more than once in the sequence. In practice, a face is often represented by the list of vertices in the sequence.

3.75 **global motion compensation, GMC**: The use of global spatial transformation to improve the efficiency of the prediction of sample values. The prediction uses global spatial transformation to provide offsets into the past reference VOPs containing previously decoded sample values that are used to form the prediction error.

3.76 **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.

3.77 **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.

3.78 **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.

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

3.80 **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.

© ISO/IEC 2004 – All rights reserved

3.81 **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.

3.82 **face forest (of a 3D mesh)**: The set of all the face trees corresponding to the connected components of a 3D mesh.

3.83 **face interpolation transform, FIT**: A downloadable node type defined in ISO/IEC 14496-1:2001 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.

3.84 **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).

3.85 **face tree (of a 3D mesh)**: A spanning tree constructed in the dual graph of a connected component.

3.86 **fast forward playback**: The process of displaying a sequence, or parts of a sequence, of VOPs in display order, faster than real-time.

3.87 **fast reverse playback**: The process of displaying a sequence, or parts of a sequence, of VOPs in the reverse of display order, faster than real-time.

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

3.89 **fgs block**: An 8-row by 8-column matrix of bits, each from one DCT coefficient at the same significant position of accuracy, or its coded representation. The usage is clear from the context.

3.90 **fgs macroblock**: The four block-bps of luminance component (Y) and the two (for 4:2:0 chrominance format) corresponding block-bps of chrominance components (U and V) with the same accuracy significance coming from the DCT coefficients of a macroblock. It may also be used to refer to the coded representation of the six block-bps. The usage is clear from the context.

3.91 **fgs macroblock number**: A number for an fgs macroblock within a vop-bp. The fgs macroblock number of the top-left fgs macroblock in each vop-bp shall be zero. The fgs macroblock number increments from left to right and from top to bottom.

3.92 **fgs run**: The number of '0' bits preceding a '1' bit within a block-bp.

3.93 **fgs temporal scalability; FGST**: A type of scalability where an enhancement layer uses predictions from sample data derived from the base layer using motion vectors. The VOP size in the enhancement layer is the same as that in the base layer. FGST is a specific type of temporal scalability where all DCT coefficients are coded using bit-plane coding as in FGS.

3.94 **fgs vop**: The pixel differences between the original VOP and the reconstructed VOP in the base layer. It may be used to refer to the DCT coefficients of the pixel differences or the original VOP. It may also be used to refer to the coded representation of the DCT coefficients. In the context of FGST, fgs vop refers to the original temporal scalable VOP. The usage is clear from the context.

3.95 **fine granularity scalability; FGS**: A type of scalability where an enhancement layer uses prediction from sample data of reconstructed VOP in the base layer. The encoded bitstream for each fgs vop can be truncated into any number of bits. The truncated bitstream for each fgs vop can be decoded to provide quality enhancement proportional to the amount of bits in the truncated bitstream of the fgs vop. The fgs vop has the same size and VOP rate as those of the base layer.

3.96 **field**: For an interlaced video signal, a "field" is the assembly of alternate lines of a frame. Therefore an interlaced frame is composed of two fields, a top field and a bottom field.

© ISO/IEC 2004 – All rights reserved

3.97 **field-based prediction**: A prediction mode using only one field of the reference frame. The predicted block size is 16x16 luminance samples. Field-based prediction is not used in progressive frames.

3.98 **field period**: The reciprocal of twice the frame rate.

3.99 **field VOP; field structure VOP**: A field structure VOP is a coded VOP with vop_structure is equal to "Top field" or "Bottom field".

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

3.101 **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.

3.102 **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.

3.103 **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.

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

3.105 **forward prediction**: Prediction from the past reference VOP.

3.106 **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. For interlaced video a frame consists of two fields, a top field and a bottom field. One of these fields will commence one field period later than the other.

3.107 **frame-based prediction**: A prediction mode using both fields of the reference frame.

3.108 **frame period**: The reciprocal of the frame rate.

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

3.110 **frame VOP; frame structure VOP**: A frame structure VOP is a coded VOP with vop_structure is equal to "Frame".

3.111 **future reference frame (field)**: A future reference frame (field) is a reference frame (field) that occurs at a later time than the current VOP in display order.

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

3.113 **hierarchical 3D mesh**: A BIFS node representing a 3D mesh.

3.114 **hierarchical representation (of a 3D mesh)**: A 3D mesh representation with two or more levels of detail in increasing order of resolution, composed of a first coarse level of detail (base 3D mesh) followed by one or more refinement operations. Each refinement operation determines how to construct a new level of detail as a refinement of the current level of detail.

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

3.116 **I-field VOP**: A field structure I-VOP.

3.117 **I-frame VOP**: A frame structure I-VOP.

© ISO/IEC 2004 – All rights reserved

3.118 **imaginary edge**: An imaginary edge is inserted to triangulate a polygon.

3.119 **incremental representation (of a 3D mesh)**: A representation that allows the terminal to start rendering a subset of the 3D mesh before the whole bitstream has been completely received and/or decoded.

3.120 **IndexedFaceSet**: A BIFS node representing a 3D mesh.

3.121 **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.

3.122 **internal edge (of a 3D mesh)**: An edge of a 3D mesh which has exactly two incident faces.

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

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

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

3.126 **inter shape coding**: Shape coding that uses temporal prediction.

3.127 **inverse quantisation**: 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.

3.128 **jump edge (of a 3D mesh)**: An edge of the vertex graph which does not belong to any vertex tree.

3.129 **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").

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

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

3.132 **leaf triangle (of a 3D mesh)**: A triangle of a triangle tree with exactly one incident triangle tree edge.

3.133 **leaf vertex (of a 3D mesh)**: A vertex of a vertex tree with exactly one incident vertex tree edge.

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

3.135 **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.

3.136 **Mbit**: 1 000 000 bits.

3.137 **macroblock**: The four 8×8 (or 16x16 in motion compensation decoding of Reduced Resolution VOP) blocks of luminance data and the two (for 4:2:0 chrominance format) or four (for 4:2:2 chrominance format), eight (for 4:4:4 chrominance format) corresponding 8×8 (or 16x16 in motion compensation decoding of Reduced Resolution VOP) blocks of chrominance data coming from a 16×16 (or 32x32 in motion compensation decoding of Reduced Resolution VOP) 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.

© ISO/IEC 2004 – All rights reserved

3.138 **manifold (3D mesh)**: A 3D mesh which has neither singular vertices nor singular edges.

3.139 **marching edge (of a 3D mesh)**: An edge shared by two triangles along a triangle run.

3.140 **mesh (2D)**: 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.

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

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

3.143 **mesh object plane**, **MOP**: The instance of a mesh object at a given time.

3.144 **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.

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

3.146 **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.

3.147 **motion vector for shape**: A motion vector used for motion compensation of shape.

3.148 **multiple auxiliary components**: Auxiliary components are defined for the VOP on a pixel-by-pixel basis, and contain data related to the video object, such as disparity, depth, and additional texture. Up to three auxiliary components (including the grayscale shape) are possible.

3.149 **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.

3.150 **non-manifold (3D mesh)**: A 3D mesh that contains singular vertices or singular edges.

3.151 **non-orientable (3D mesh)**: A manifold 3D mesh which is not orientable.

3.152 **opaque macroblock**: A macroblock with shape mask of all 255's.

3.153 **orientable (3D mesh)**: A manifold 3D mesh for which an orientation can be chosen for each of its faces such that for each internal edge, the two faces incident to the edge induce opposite orientations on the common edge. All non-manifold 3D meshes are non-orientable.

3.154 **orientation (of a face of a 3D mesh)**: One of the two possible cyclical orderings of its sustaining vertices.

3.155 **orientation (of an edge of a 3D mesh)**: One of the two possible orderings of the two vertices of the edge.

3.156 **oriented (manifold 3D mesh)**: A manifold 3D mesh for which an orientation has been chosen for each of its faces such that for each internal edge, the two faces incident to the edge induce opposite orientations on the common edge. Note that a manifold 3D mesh may be orientable but not oriented. Such a mesh can be oriented by choosing a consistent orientation for its faces (which may require inverting the orientation of one or more faces).

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

© ISO/IEC 2004 – All rights reserved

3.158 **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.

3.159 **partition**: a bitstream segment of a 3D mesh that can be decoded independently.

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

3.161 **P-field VOP**: A field structure P-VOP.

3.162 **P-frame VOP**: A frame structure P-VOP.

3.163 **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 or rgb colour signals. A "coded VOP" was defined earlier. For progressive video, a picture is identical to a frame, while for interlaced video, a picture can refer to a frame, or the top field or the bottom field of the frame depending on the context.

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

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

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

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

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

3.169 **quantisation matrix**: A set of sixty-four 8-bit values used by the inverse quantisation process.

3.170 **quantised DCT coefficients**: DCT coefficients before inverse quantisation. A variable length coded representation of quantised DCT coefficients is transmitted as part of the coded video bitstream.

3.171 **quantiser scale**: A scale factor coded in the bitstream and used by the decoding process to scale the inverse quantisation.

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

3.173 **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.

3.174 **reference layer**: A layer to be referenced for prediction in a scalable hierarchy. The video_object_id of the reference layer should be the same value as the video_object_id of the enhancement layer. The ref_layer_id of the enhancement layer is set to the same value as the video_object_layer_id of the reference layer.

3.175 **reference VOP**: A reference VOP is a reconstructed VOP that was coded in the form of a coded I-VOP, a coded P-VOP, or a coded S(GMC)-VOP. Reference VOPs are used for forward and backward prediction when P-VOPs, B-VOPs, and S(GMC)-VOPs are decoded.

3.176 **regular triangle (of a triangle tree)**: A triangle of a triangle tree with exactly two incident triangle tree edges.

© ISO/IEC 2004 – All rights reserved

3.177 **regular vertex (of a 3D mesh)**: A vertex of a 3D mesh is *regular* if the set of incident edges and incident faces can be ordered in an alternating sequence of edges and faces, such that each face in the sequence is bounded by the edges immediately preceeding and following it in the sequence.

3.178 **regular vertex (of a vertex tree)**: A vertex of a vertex tree with exactly two incident vertex tree edges.

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

3.180 **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.

3.181 **RGB component**: A matrix, block or single sample representing one of the three primary colours. The symbols used for the rgb signals are Green, Blue and Red.

3.182 **sample aspect ratio**: (abbreviated to **SAR**). This specifies the relative distance between samples. It is defined (for the purposes of this specification) as the vertical displacement of the lines of luminance samples in a frame divided by the horizontal displacement of the luminance samples. Thus its units are (metres per line) ÷ (metres per sample)

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

3.184 **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.

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

3.186 **skipped macroblock**: A macroblock for which no data is encoded.

3.187 **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".

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

3.189 **S(GMC)-VOP**: A picture that is coded using prediction based on global motion compensation from the past VOP.

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

3.191 **simple polygon (of a 3D mesh)**: A connected component of which the dual graph is a tree. A simple polygon has exactly one boundary, and all its vertices lie on the boundary.

3.192 **singular edge (of a 3D mesh)**: An edge of a 3D mesh which has three or more incident faces.

3.193 **singular vertex (of a 3D mesh)**: A vertex of a 3D mesh which is not a regular vertex.

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

3.195 **spanning forest (of a graph)**: A subgraph of a graph composed of one or more trees, each tree spanning a connected component of the graph.

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

© ISO/IEC 2004 – All rights reserved

3.197 **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.

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

3.199 **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.

3.200 **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.

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

3.202 **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.

3.203 **top field**: One of two fields that comprise a frame. Each line of a top field is spatially located immediately above the corresponding line of the bottom field.

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

3.205 **transparent macroblock**: A macroblock with shape mask of all zeros.

3.206 **traversal order**: The traversal order determines which branch to be traversed first at each branching triangle.

3.207 **tree loop (of a 3D mesh)**: An alternating sequence of vertices and edges bounding the hole created by cutting a 3D mesh through a tree of edges within a forest split operation.

3.208 **tree loop face**: a face incident to a tree loop.

3.209 **tree loop corner**: a corner of a tree loop face incident to a tree loop edge.

3.210 **triangle tree (of a 3D mesh)**: A spanning tree in the dual graph of a connected component of a triangular 3D mesh.

3.211 **triangle run**: A path in a triangle tree connecting a first leaf or branching triangle to a last leaf or branching triangle through zero or more intermediate triangles.

3.212 **triangular 3D mesh**: A 3D mesh with triangular faces. Also, a 3D mesh after all the polygonal faces have been triangulated.

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

3.214 **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.

3.215 **vertex graph (of a 3D mesh)**: The graph formed by the vertices and the edges of the 3D mesh which do not correspond to dual graph edges included in the face forest.

3.216 **vertex run**: A path in a vertex tree connecting a first leaf or branching vertex to a last leaf or branching vertex through zero or more intermediate regular vertices.

© ISO/IEC 2004 – All rights reserved

3.217 **vertex tree**: A rooted tree spanning one connected component of the vertex graph.

3.218 **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.

3.219 **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.

3.220 **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.

3.221 **video object plane**, **VOP**: The instance of a video object at a given time.

3.222 **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.

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

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

3.225 **virtual triangle**: A virtual triangle is a leaf triangle added, if necessary, at the end of a partition to signal the end of the partition to the decoder.

3.226 **vop-bp**: An array of block-bps with the same accuracy significance in an fgs vop. There are three color components (Y, U, and V) in a vop-bp. Each color component in a vop-bp consists of all the block-bps of that color.

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

3.228 **warping**: Processing applied to extract a sprite VOP from a static sprite or a reference VOP. 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.

3.229 **Y-vertex**: The vertex of a branching triangle opposite to the incident marching edge.

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

4 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.

4.1 Arithmetic operators

\+ Addition.

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

© ISO/IEC 2004 – All rights reserved

++ 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*.

4.2 Logical operators

|| Logical OR.

&& Logical AND.

! Logical NOT.

4.3 Relational operators

> Greater than.

>= Greater than or equal to.

≥ Greater than or equal to.

< Less than.

<= Less than or equal to.

≤ Less than or equal to.

© ISO/IEC 2004 – All rights reserved

== Equal to.

!= Not equal to.

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

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

4.4 Bitwise operators

& AND

| OR

>> Shift right with sign extension.

<< Shift left with zero fill.

4.5 Conditional operators

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

4.6 Assignment

= Assignment operator.

4.7 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 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.

4.8 Constants

Π 3,141 592 653 58…

e 2,718 281 828 45…

© ISO/IEC 2004 – All rights reserved

5 Conventions

5.1 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) { **data_element** . . . }	If the condition is true, then the group of data elements occurs next in the data stream. This repeats until the condition is not true.
do { **data_element** . . . } while (condition)	The data element always occurs at least once. The data element is repeated until the condition is not true.
do { . . . continue . . . } while (condition)	Continues execution of the next repetition of the nearest while-do loop.
if (condition) { **data_element** . . . } else { **data_element** . . . }	If the condition is true, then the first group of data elements occurs next in the data stream. If the condition is not true, then the second group of data elements occurs next in the data stream.
for (i = m; i < n; i++) { **data_element** . . . }	The group of data elements occurs (n-m) times. Conditional 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 . . . */	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") . . .

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 6–16 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.

© ISO/IEC 2004 – All rights reserved

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.

5.2 Definition of functions

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

5.2.1 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.

5.2.2 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.

5.2.3 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.

5.2.4 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	Mnemonic
zero_bit	1	'0'
while (!bytealigned())		
one_bit	1	'1'
}		

5.2.5 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	Mnemonic
zero_bit	1	'0'
while (!bytealigned())		
one_bit	1	'1'
}		

© ISO/IEC 2004 – All rights reserved

5.2.6 Definition of transparent_mb() function

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

5.2.7 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 6-8.

5.2.8 Definition of byte_align_for_upstream() function

The function byte_align_for_upstream() removes a string of '1' used for stuffing from the upstream message. When the message is already byte aligned before the byte_align_for_upstream() function, additional byte stuffing is no longer allowed.

byte_align_for_upstream() {	No. of bits	Mnemonic
while (!byte_aligned())		
one_bit	1	'1'
}		

5.2.9 Definition of next_start_code_studio() function

The next_start_code_studio() function removes any zero bit and zero byte stuffing and locates the next start code.

next_start_code_studio() {	No. of bits	Mnemonic
while (!bytealigned())		
zero_bit	1	'0'
while (nextbits() != '0000 0000 0000 0000 0000 0001')		
zero_byte	8	'0000 0000'
}		

This function checks whether the current position is byte aligned. If it is not, zero stuffing bits are present. After that any number of zero stuffing bytes may be present before the start code. Therefore start codes are always byte aligned and may be preceded by any number of zero stuffing bits.

5.2.10 Definition of valid_stuffing_bits() function

If the current position in not on a byte boundary, the function valid_stuffing_bits() returns 1 if the remaining bits in the current byte are valid stuffing bits. If the current position is on a byte boundary, valid_stuffing_bits() returns 1 if the remaining bits in the next byte are valid stuffing bits. Otherwise, valid_stuffing_bits() returns 0.

5.2.11 Definition of start_of_bit_plane() function

The function start_of_bit_plane() returns 1 if the next bit in the bitstream is the first bit of the codes associated with a vop-bp. Otherwise it returns 0.

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.

© ISO/IEC 2004 – All rights reserved

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.

5.4 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.

6 Visual bitstream syntax and semantics

6.1 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 14496-1:2001 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 14496-1:2001 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 specified in ISO/IEC 14496-1:2001. 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. In a typical application of FGS, the bitstream at the input of an FGS decoder is a truncated version of the bitstream at the output of an FGS encoder. It is likely that, at the end of each fgs vop before the next fgs_vop_start_code, only partial bits of the fgs vop are at the input of the decoder due to truncation of the fgs vop bitstream. Decoding of the truncated bitstream is not normative. An example of dealing with the truncated bitstream is described in Annex S. The FGS syntax description in this clause is for a complete bitstream without truncation.

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.

© ISO/IEC 2004 – All rights reserved

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 FBA parameter data consists of one non-scaleable bitstream. It defines the animation of the face and body model of the decoder. Face and body animation data is structured as standard formats for downloadable models and their animation controls, and a single layer of compressed face and body animation parameters used for remote manipulation of the face and body model. The face is a node in a scene graph that includes face geometry ready for rendering. The body is also a node in a scene graph that includes body geometry ready for rendering. The face node belongs to the scene graph that is defined by the body node.The shape, texture and expressions of the face are controlled by the bitstream containing instances of Facial Definition Parameter (FDP) sets and/or Facial Animation Parameter (FAP) sets. The body is also controlled by instances of Body Definition Parameter (BDP) sets and/or Body Animation Parameter (BAP) sets.Upon initial or baseline construction, the FBA object contains a generic face with a neutral expression and a generic body with default position. This object can receive FAPs and BAPs from the bitstream and be subsequently rendered to produce animation of the face and the body. If FDPs and BDPs are transmitted, the generic face and body are transformed into a particular face and body of specific shape and appearance. A downloaded face model via FDPs is a scene graph for insertion in the face node, and the downloaded body model via BDPs is a scene graph for insertion of the body node.

6.1.1 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.

At various points in the visual object sequence, a repeat visual_object_sequence_start_code can be inserted for coded video data. In that case, the repeat visual_object_sequence_start_code shall follow a particular VOP. When profile_and_level_indication indicates a Studio Profile, StudioVisualObject() shall follow it.

6.1.2 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 an FBA object. For Studio Profiles particularly, only video object type is supported.

6.1.3 Video object

A video object commences with a video_object_start_code, and is followed by one or more video object layers. For Studio Profiles particularly, a video object layer commences with video_object_layer_start_code which may optionally be followed by Group_of_StudioVideoObjectPlane() and then by one or more coded VOPs. The order of the coded frames in the coded bitstream is the order in which the decoder processes them, which is not necessarily the display order.

6.1.3.1 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.

For Studio Profiles particularly, the output of the decoding process for interlaced sequences consists of a series of reconstructed fields that are separated in time by a field period. The two fields of a frame may be coded separately (field-VOPs). Alternatively the two fields may be coded together as a frame (frame-VOPs). Both frame VOPs and field VOPs may be used in a single video sequence.

In progressive sequences each VOP in the sequence shall be a frame VOP. The sequence, at the output of the decoding process, consists of a series of reconstructed frames that are separated in time by a frame period.

© ISO/IEC 2004 – All rights reserved

6.1.3.2 Frame

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

The relationship between these Y, Cb and Cr components and the primary (analogue) Red, Green and Blue Signals (E'_R, E'_G and E'_B), the chromaticity of these primaries and the transfer characteristics of the source frame may be specified in the bitstream (or specified by some other means). This information does not affect the decoding process.

For Studio Profiles particularly, the three rectangular matrices can be the primary RGB colour matrices.

6.1.3.3 Field

A field consists of every other line of samples in the three rectangular matrices of integers representing a frame.

A frame is the union of a top field and a bottom field. The top field is the field that contains the top-most line of each of the three matrices. The bottom field is the other one.

6.1.3.4 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.

For Studio Profiles particularly, coded VOP may have been derived from a progressive or interlaced frame or an interlaced field. A reconstructed VOP is either a reconstructed frame (when decoding a frame VOP), or one field of a reconstructed frame (when decoding a field VOP).

An I-frame VOP or a pair of field VOPs, where the first field VOP is an I-picture and the second field VOP is an I-VOP or a P-VOP, is called a coded I-frame.

A P-frame VOP or a pair of P-field VOPs is called a coded P-frame.

A B-frame VOP or a pair of B-field VOPs is called a coded B-frame.

A coded I-frame, a coded P-frame or a coded B-frame is called a coded frame

6.1.3.4.1 Field VOPs

If field VOPs are used, then they shall occur in pairs (one top field followed by one bottom field, or one bottom field followed by one top field) and together constitute a coded frame. The two field VOPs that comprise a coded frame shall be encoded in the bitstream in the order in which they shall occur at the output of the decoding process.

When the first VOP of the coded frame is a P-field VOP, then the second VOP of the coded frame shall also be a P-field VOP. Similarly when the first VOP of the coded frame is a B-field VOP the second VOP of the coded frame shall also be a B-field VOP.

When the first VOP of the coded frame is a I-field VOP, then the second VOP of the frame shall be either an I-field VOP or a P-field VOP. If the second VOP is a P-field VOP, then certain restrictions apply (see 7.16.7.4.5).

6.1.3.4.2 Frame VOPs

When coding interlaced sequences using frame VOPs, the two fields of the frame shall be interleaved with one another and then the entire frame is coded as a single frame-VOP.

© ISO/IEC 2004 – All rights reserved

6.1.3.5 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 or a VOP which is coded using prediction based on global motion compensation from a past reference VOP.

6.1.3.6 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 to be displayed after having decoded a GOV header. For Studio Profiles particularly, SMPTE 12M time code information that is not used by the decoding process.

2) 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.

6.1.3.7 Format

6.1.3.7.1 4:2:0 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.

If the matrices represent RGB colour primary matrices, this 4:2:0 format shall not be applied.

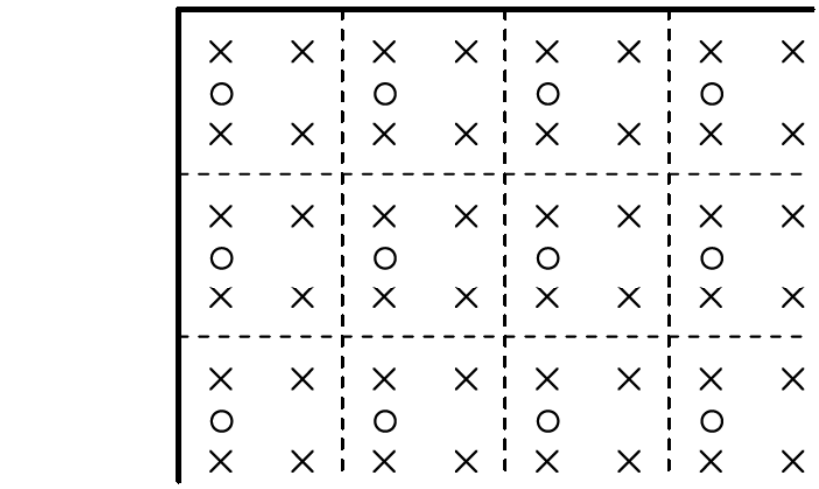
NOTE — When interlaced frames are coded as rectangular field VOPs, the VOP reconstructed from each of these field VOPs shall have a Y-matrix with half the number of lines of the corresponding frame. Thus the total number of lines in the Y-matrix of an entire frame shall be divisible by four.

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.

In each field of an interlaced frame, the chrominance samples do not lie (vertically) mid way between the luminance samples of the field. This is so that the spatial location of the chrominance samples in the frame is the same whether the frame is represented as a single frame-VOP or two field-VOPs.

© ISO/IEC 2004 – All rights reserved



× Represent luminance samples

○ Represent chrominance samples

Figure 6-1 — The position of luminance and chrominance samples in 4:2:0 data

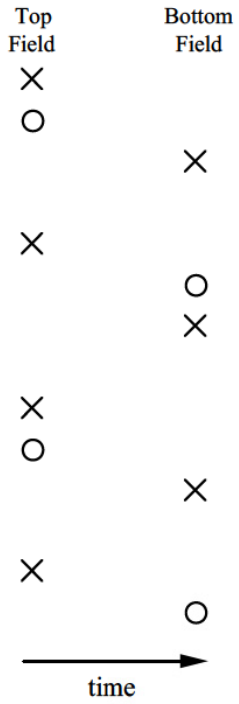


Figure 6-2 — Vertical and temporal positions of samples in an interlaced frame with top_field_first=1

© ISO/IEC 2004 – All rights reserved

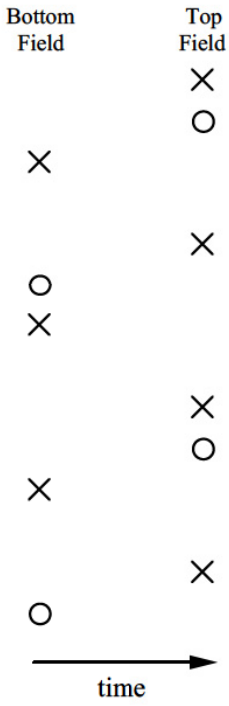


Figure 6-3 — Vertical and temporal position of samples in an interlaced frame with top_field_first=0

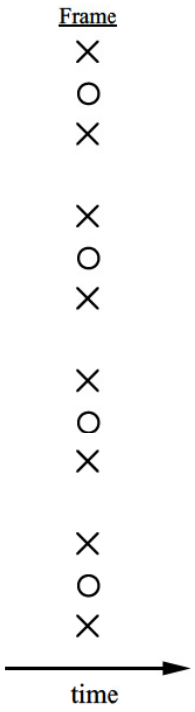


Figure 6-4 — Vertical and temporal positions of samples in a progressive frame

© ISO/IEC 2004 – All rights reserved

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.

6.1.3.7.2 4:2:2 Format

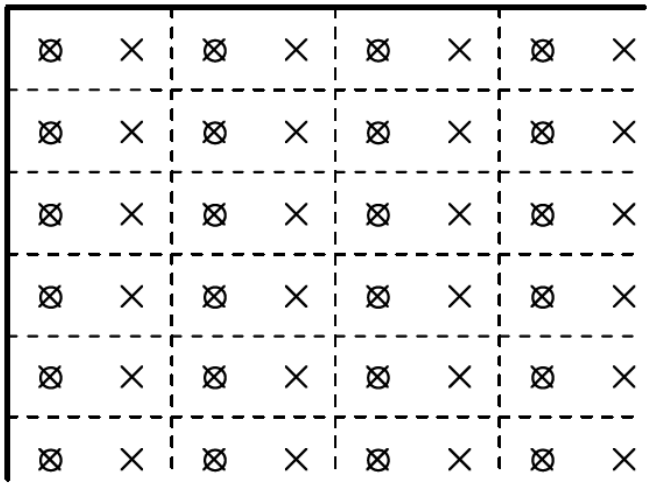
In this format the Cb and Cr matrices shall be one half the size of the Y-matrix in the horizontal dimension and the same size as the Y-matrix in the vertical dimension. The Y-matrix shall have an even number of samples.

If the matrices represent RGB colour primar matrices, this 4:2:2 format shall not be applied.

NOTE — When interlaced frames are coded as rectangular field VOPs, the VOP reconstructed from each of these field VOPs shall have a Y-matrix with half the number of lines of the corresponding frame. Thus the total number of lines in the Y-matrix of an entire frame shall be divisible by two.

The luminance and chrominance samples are positioned as shown in Figure 6-5.

In order to clarify the organisation, Figure 6-6 shows the (vertical) positioning of the samples when the frame is separated into two fields.



× Represent luminance samples

○ Represent chrominance samples

Figure 6-5 — The position of luminance and chrominance samples. 4:2:2 data.

© ISO/IEC 2004 – All rights reserved

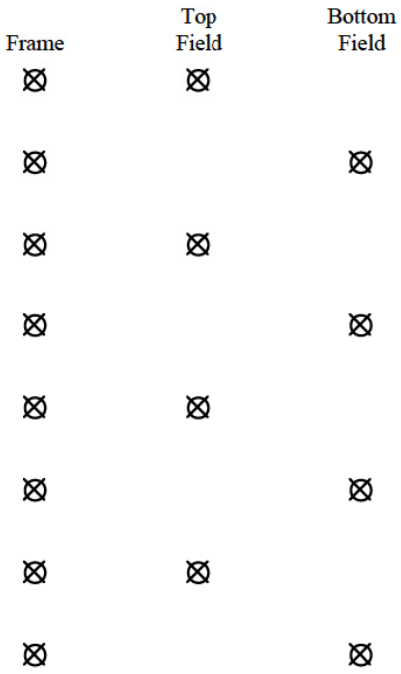


Figure 6-6 — Vertical positions of samples with 4:2:2 and 4:4:4 data

6.1.3.7.3 4:4:4 Format

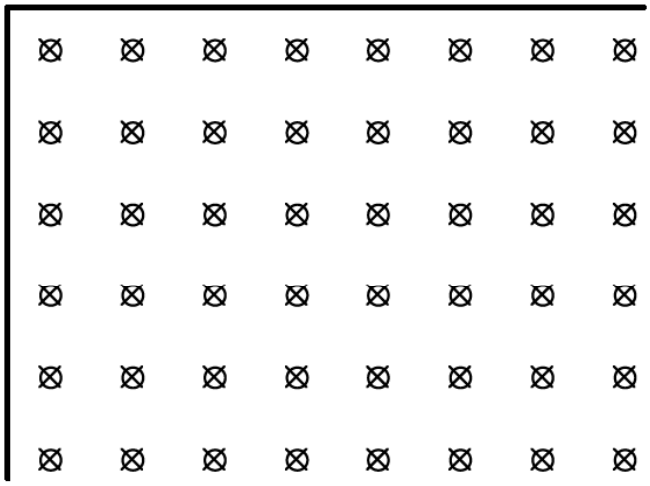
In this format the Cb and Cr matrices shall be the same size as the Y-matrix in the horizontal and the vertical dimensions.

If the matrices are treated as RGB colour primary matrices, the matrices shall follow this format.

NOTE — When interlaced frames are coded as field rectangular VOPs, the VOP reconstructed from each of these field VOPs shall have a Y-matrix with half the number of lines of the corresponding frame. Thus the total number of lines in the Y-matrix of an entire frame shall be divisible by two.

The luminance and chrominance samples are positioned as shown in Figure 6-6 and Figure 6-7.

© ISO/IEC 2004 – All rights reserved



X Represent luminance samples

O Represent chrominance samples

Figure 6-7 — The position of luminance and chrominance samples. 4:4:4 data.

6.1.3.8 VOP reordering

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

A video object layer may contain no coded P-VOPs or no coded S(GMC)-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, P-VOP, or S(GMC)-VOPs the output VOP is the VOP reconstructed from the previous I-VOP or P-VOP, or S(GMC)-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'.

© ISO/IEC 2004 – All rights reserved

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

In terms of VOP reordering, an S(GMC)-VOP can be regarded as a P-VOP.

6.1.3.9 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. There are three chrominance formats for a macroblock, namely, 4:2:0, 4:2:2 and 4:4:4 formats. The order of blocks in a macroblock shall be different for each different chrominance format and are 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 6-8.

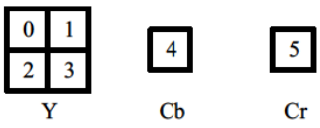


Figure 6-8 — 4:2:0 Macroblock structure

A 4:2:2 Macroblock consists of 8 blocks. This structure holds 4 Y, 2 Cb and 2 Cr Blocks and the block order is depicted in Figure 6-9.

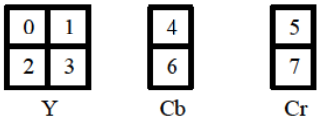


Figure 6-9 — 4:2:2 Macroblock structure

A 4:4:4 Macroblock consists of 12 blocks. This structure holds 4 Y, 4 Cb and 4 Cr (or 4 G, 4 B and 4 R) Blocks and the block order is depicted in Figure 6-10.

© ISO/IEC 2004 – All rights reserved

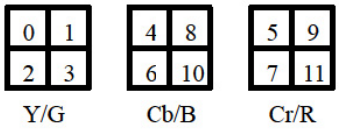


Figure 6-10 — 4:4:4 Macroblock structure

The organisation of VOPs into macroblocks is as follows.

For the case of a progressive VOP (when the interlaced flagin the VOL header is set to "0"), the organisation of lines of luminance VOP into macroblocks is called frame organization and is illustrated in Figure 6-11. In this case, frame DCT coding is employed.

For the case of interlaced VOP (when the interlaced flag is set to "1"), 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 6-11.

- 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 6-12.

In the case of chrominance blocks the structure depends upon the chrominance format that is being used. In the case of 4:2:2 and 4:4:4 formats (where there are two blocks in the vertical dimension of the macroblock) the chrominance blocks are treated in exactly the same manner as the luminance blocks. However, in the 4:2:0 format the chrominance blocks shall always be organised in frame structure for the purposes of DCT coding. It should however be noted that field based predictions may be made for these blocks which will, in the general case, require that predictions for 8x4 regions (after half-sample filtering) must be made.

In field pictures, each picture only contains lines from one of the fields. In this case each block consists of lines taken from successive lines in the picture as illustrated by Figure 6-11.

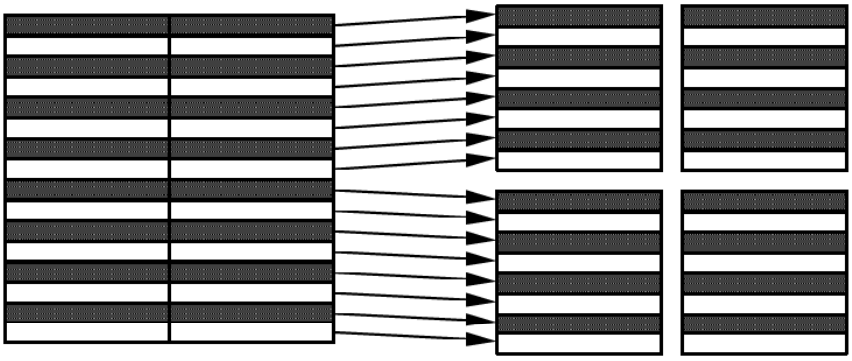


Figure 6-11 — Luminance macroblock structure in frame DCT coding

© ISO/IEC 2004 – All rights reserved

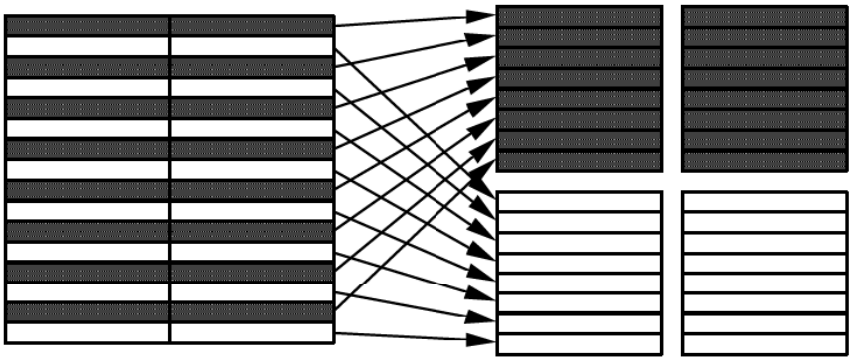


Figure 6-12 — Luminance macroblock structure in field DCT coding

6.1.3.10 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.

6.1.3.11 Field

A field consists of every other line of samples in the three rectangular matrices of integers representing a frame.

A frame is the union of a top field and a bottom field. The top field is the field that contains the top-most line of each of the three matrices. The bottom field is the other one.

Only when profile_and_level_indication indicates the studio profile, a coded VOP may be a frame VOP or a field VOP. A reconstructed VOP is either a reconstructed frame (when decoding a frame VOP), or one field of a reconstructed frame (when decoding a field VOP).

6.1.3.11.1 Field VOPs

If field VOPs are used then they shall occur in pairs (one top field followed by one bottom field, or one bottom field followed by one top field) and together constitute a coded frame. The two field VOPs that comprise a coded frame shall be encoded in the bitstream in the order in which they shall occur at the output of the decoding process.

When the first VOP of the coded frame is a P-field VOP, then the second VOP of the coded frame shall also be a P- field VOP. Similarly when the first VOP of the coded frame is a B-field VOP the second VOP of the coded frame shall also be a B-field VOP.

When the first VOP of the coded frame is a I-field VOP, then the second VOP of the frame shall be either an I-field VOP or a P-field VOP. If the second VOP is a P-field VOP then certain restrictions apply.

6.1.3.11.2 Frame VOPs

When coding interlaced sequences using frame VOPs, the two fields of the frame shall be interleaved with one another and then the entire frame is coded as a single frame-VOP.

© ISO/IEC 2004 – All rights reserved

6.1.3.12 Slice

A slice is a series of an arbitrary number of consecutive macroblocks. The first and last macroblocks of a slice shall not be skipped macroblocks. Every slice shall contain at least one macroblock. Slices shall not overlap. The position of slices may change from picture to picture.

The first and last macroblock of a slice shall be in the same horizontal row of macroblocks.

Slices shall occur in the bitstream in the order in which they are encountered, starting at the upper-left of the picture and proceeding by raster-scan order from left to right and top to bottom (illustrated in the Figures of this clause as alphabetical order).

6.1.3.12.1 The general slice structure

In the most general case it is not necessary for the slices to cover the entire picture. Figure 6-13 shows this case. Those areas that are not enclosed in a slice are not encoded and no information is encoded for such areas (in the specific picture).

If the slices do not cover the entire picture then it is a requirement that if the picture is subsequently used to form predictions then predictions shall only be made from those regions of the picture that were enclosed in slices. It is the responsibility of the encoder to ensure this.

This specification does not define what action a decoder shall take in the regions between the slices.

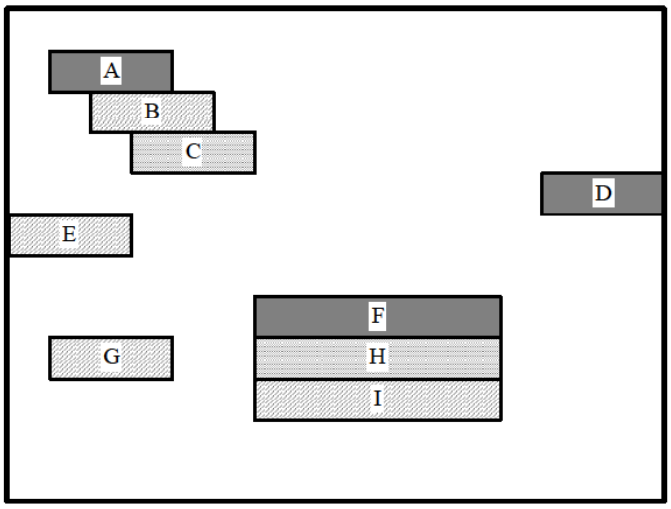


Figure 6-13 — The most general slice structure

6.1.3.12.2 Restricted slice structure

In certain defined levels of defined profiles a restricted slice structure illustrated in Figure 6-14 shall be used. In this case every macroblock in the picture shall be enclosed in a slice.

© ISO/IEC 2004 – All rights reserved

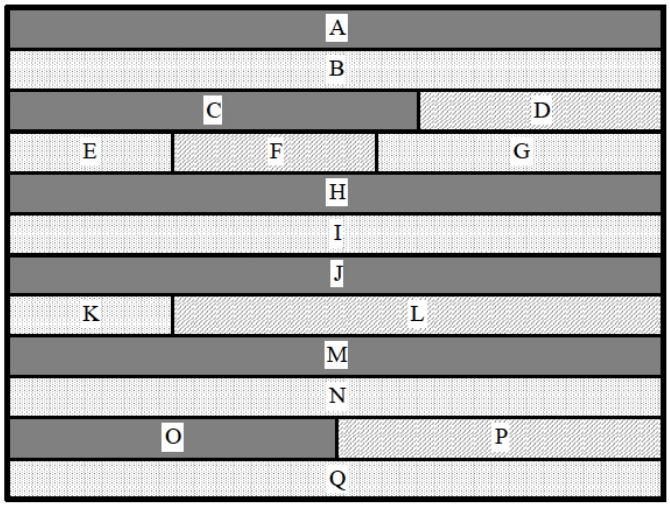


Figure 6-14 — Restricted slice structure

Where a defined level of a defined profile requires that the slice structure obeys the restrictions detailed in this clause, the term "restricted slice structure" may be used.

6.1.4 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 6-15.

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.

© ISO/IEC 2004 – All rights reserved

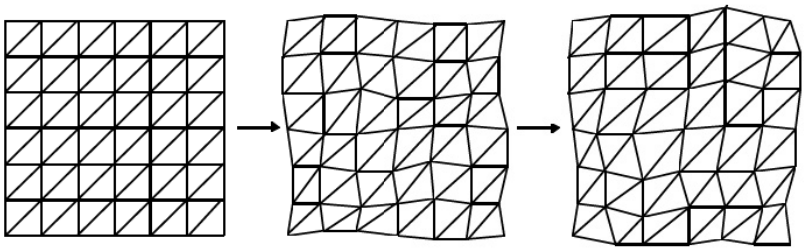


Figure 6-15 — Mesh object with uniform triangular geometry

6.1.4.1 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.

6.1.5 FBA object

Conceptually the FBA object consists of a collection of nodes in a scene graph which are animated by the FBA object bitstream. The shape, texture and expressions of the face are controlled by the bitstream containing instances of Facial Definition Parameter (FDP) sets and/or Facial Animation Parameter (FAP) sets. Upon construction, the FBA 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.

Upon construction, the Body model contains a generic virtual human or human-like body with the default posture. This body can already be rendered. It is also immediately capable of receiving the BAPs from the bitstream, which will produce animation of the body. If BDPs are received, they are used to transform the decoder's generic body into a particular body determined by the parameter contents. Any component can be null. A null component is replaced by the corresponding default component when the body is rendered. Similar to the face, the BAPs can be transmitted also without first downloading BDPs, in which case the decoder animates its local model.

© ISO/IEC 2004 – All rights reserved

No assumption is made and no limitation is imposed on the range of defined mobilities for humanoid animation. In other words the human body model should be capable of supporting various applications, from realistic simulation of human motions to network games using simple human-like models.

6.1.5.1 Structure of the face and body object bitstream

A face and body object is formed by a temporal sequence of face and body object planes. This is depicted as follows in Figure 6-16.

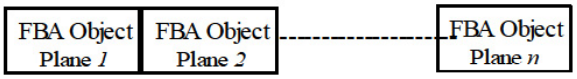
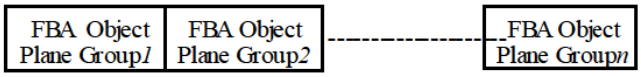


Figure 6-16 — Structure of the FBA object bitstream

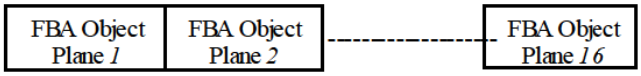
An FBA object represents a node in an ISO/IEC 14496 scene graph. An ISO/IEC 14496 scene is understood as a composition of Audio-Visual objects according to some spatial and temporal relationships. The scene graph is the hierarchical representation of the ISO/IEC 14496 scene structure (see ISO/IEC 14496-1:2001).

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

FBA **object:**



FBA **object plane group:**



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

6.1.5.2 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.

© ISO/IEC 2004 – All rights reserved

6.1.5.3 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 6-17.

Table 6-1 -- Facial Animation Parameter Units

Description		FAPU Value
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 = IRISD0 / 1024
ES0 = 3.5.x – 3.6.x	Eye separation	ES = ES0 / 1024
ENS0 = 3.5.y – 9.15.y	Eye - nose separation	ENS = ENS0 / 1024
MNS0 = 9.15.y – 2.2.y	Mouth - nose separation	MNS = MNS0 / 1024
MW0 = 8.3.x – 8.4.x	Mouth width	MW = MW0 / 1024
AU	Angle Unit	10^{-5} rad

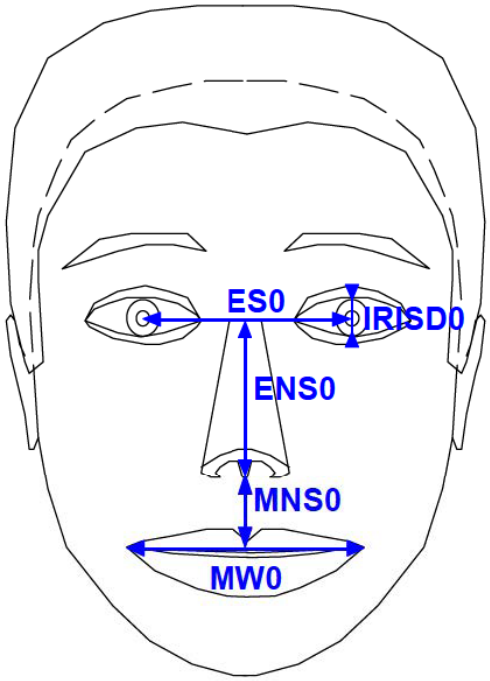


Figure 6-17 — The Facial Animation Parameter Units

© ISO/IEC 2004 – All rights reserved

6.1.5.4 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)

6.1.5.5 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 14496-1:2001). 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.

6.1.5.6 Body animation parameter set

BAP parameters comprise joint angles connecting different body parts. These include: toe, ankle, knee, hip, spine (C1-C7, T1-T12, L1-L5), shoulder, clavicle, elbow, wrist, and the hand fingers. The detailed joint list, with the rotation normals, are given in the following subclause. The rotation angles are assumed to be positive in the counterclockwise rotation direction with respect to the rotation normal. The rotation angles are defined as zero in the default posture, as defined below.

Note that the normals of rotation move with the body, and they are fixed with respect to the parent body part. That is to say, the axes of rotation are not aligned with the body or world coordinate system, but move with the body parts.

The hands are capable of performing complicated motions and are included in the body hierarchy. There are totally 29 degrees of freedom on each hand, assuming that the hand has a standard structure with five fingers.

The unit of rotations (BAPU) is defined as PI/10E-5 radians. The unit of translation BAPs (BAPs HumanoidRoot_tr_vertical, HumanoidRoot_tr_lateral, HumanoidRoot_tr_frontal) is defined in millimeters.

© ISO/IEC 2004 – All rights reserved

6.1.5.7 Description of the default posture of the body

The default posture is defined by standing posture. This posture is defined as follows: the feet should point to the front direction, the two arms should be placed on the side of the body with the palm of the hands facing inward. This posture also implies that all BAPs have default values as 0.

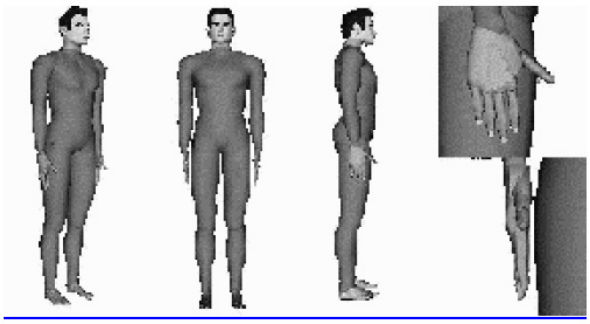


Figure 6-18

6.1.5.8 Body Definition Parameter Set

The BDPs are used to customize the proprietary body model of the decoder to a particular body or to download a body model along with the information about how to animate it. The definition and description of BDP fields is given in Annex C. The BDPs are normally transmitted once per session, followed by a stream of compressed BAPs. However, if the decoder does not receive the BDPs, model-independence of BAPs ensures that it can still interpret the BAP stream. This insures minimal operation in broadcast or teleconferencing applications. The BDP set is specified in BIFS syntax (see ISO/IEC 14496-1:2001). The BDP node defines the body model to be used at the receiver.

6.1.6 3D Mesh Object

The compressed bitstream for a 3D mesh is composed of a header data block with global information, followed by a sequence of connected component data blocks, each one associated with one connected component of the 3D mesh.

3D Mesh Header	CC Data #1	...	CC Data #nCC

If a 3D mesh is coded in error resilience mode, connected component data blocks are grouped or divided into partitions.

Partition #1	Partition #2	...	Partition #nPT

Additionally, if the 3D mesh is represented in hierarchical mode, the last connected component data block is followed by one or more refinement step data blocks, each one of them representing a Forest Split operation. Forest split can be applied to all the components of a bitstream.

Forest Split #1	Forest Split #2	...	Forest Split #nFS

Each connected component data block is composed of three records, the Vertex Graph record, the Triangle Tree record, and the Triangle Data record.

Vertex Graph	Triangle Tree	Triangle Data

© ISO/IEC 2004 – All rights reserved

The triangle tree record contains the structure of a triangle spanning tree which links all the triangles of the corresponding connected component forming a simple polygon. The 3D mesh is represented in a triangulated form in the bitstream, which also contains the information necessary to reconstruct the original faces. The vertex graph record contains the information necessary to stitch pairs of boundary edges of the simple polygon to reconstruct the original connectivity, not only within the current connected component, but also to previously decoded connected components. The connectivity information is categorized as *global* information (per connected component) and *local* information (per triangle). The global information is stored in the Vertex Graph and Triangle Tree records. The local information is stored in the Triangle Data record. The triangle data is arranged on a per triangle basis, where the ordering of the triangles is determined by the traversal of the triangle tree.

Data for triangle #1	Data for triangle #2	...	Data for triangle #nT

The data for a given triangle is organized as follows:

marching edge	td_orientation	polygon_edge	coord	normal	color	texCoord

The *marching edge, td_orientation* and *polygon_edge* constitute the per triangle connectivity information. The other fields contain information to reconstruct the vertex coordinates (*coord*) and optionally, normal, color, and texture coordinate (*texCoord*) information.

If the 3D mesh is encoded in hierarchical mode, each Forest Split data block is composed of an optional pre-smoothing data block, an optional post-smoothing data block, a pre-update data block, an optional smoothing constraints data block, a post-update data block, and an optional other-update data block.

Pre-smoothing	Post-smoothing	Pre-update	Constraints	Post-update	Other-update

The pre-smoothing data block contains the parameters used to apply a smoothing step as a global predictor for vertex coordinates. The post-smoothing data block contains the parameters used to apply a smoothing step to the vertex coordinates to remove quantisation artifacts. The pre update data block contains the information necessary to update the connectivity, and the property data updates for properties bound per-face and per-corner to the new faces created by the connectivity update. The smoothing constraints data block contains information used to perform the smoothing steps with sharp edge discontinuities and fixed vertices. After the connectivity update is applied, the pre-smoothing operation specified by the parameters stored in the pre-smoothing data block is applied as a global predictor for the vertex coordinates. The post update data block contains tree loop vertex coordinate updates, with respect to the vertex coordinates predicted by the pre-smoothing step, if applied; normal, color, and texture coordinate updates for properties bound per-vertex to tree loop vertices; normal, and color updates for properties bound per-face to tree loop faces; and normal, color, and texture coordinate updates for properties bound per-corner to tree loop corners. The other-update data block contains vertex coordinate updates and property updates for all the vertices, faces, and corners not updated by data included in the post-update data block.

6.2 Visual bitstream syntax

6.2.1 Start codes and bitstream structure

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.

© ISO/IEC 2004 – All rights reserved

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 6-2-- 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 6-3 — 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 3F
fgs_bp_start_code	40 through 5F
reserved	60 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
slice_start_code	B7
extension_start_code	B8
fgs_vop_start_code	B9
fba_object_start_code	BA
fba_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
texture_tile_start_code	C1
texture_shape_layer_start_code	C2
stuffing_start_code	C3
reserved	C4-C5
System start codes (see note)	C6 through FF
NOTE System start codes are defined in ISO/IEC 14496-1:2001	

© ISO/IEC 2004 – All rights reserved

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) VideoObjectLayer().

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

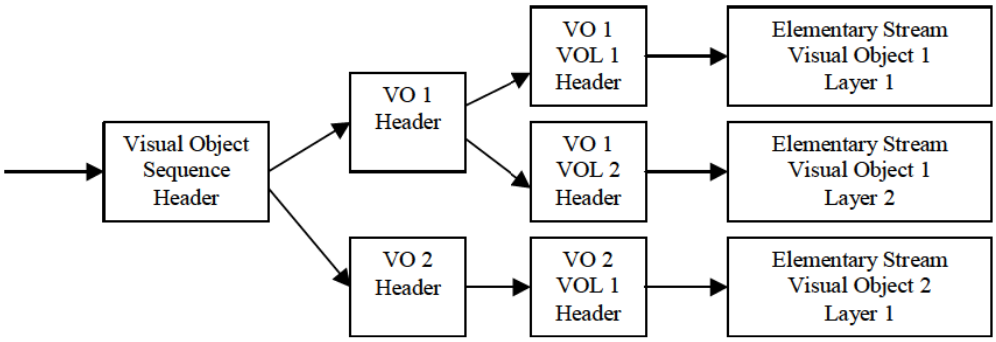


Figure 6-19 — Example Visual Information – Logical Structure

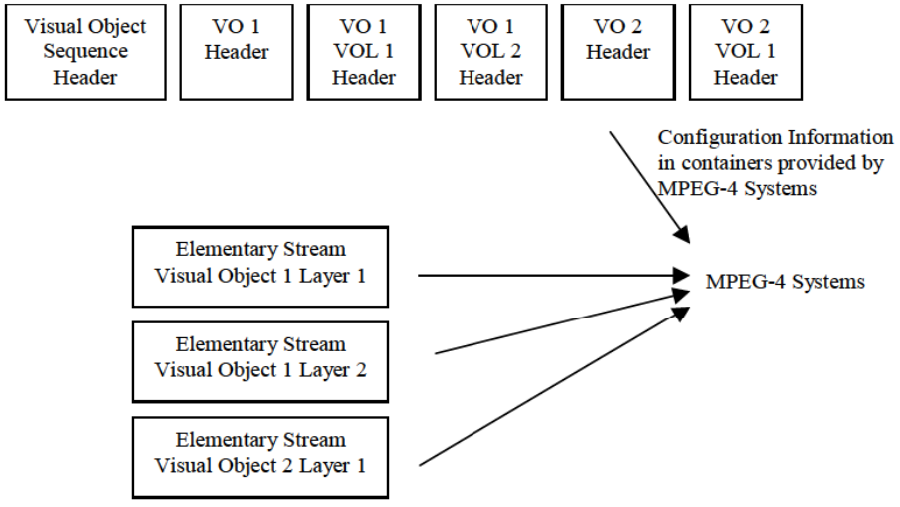


Figure 6-20 — Example Visual Bitstream – Separate Configuration Information / Elementary Stream

© ISO/IEC 2004 – All rights reserved

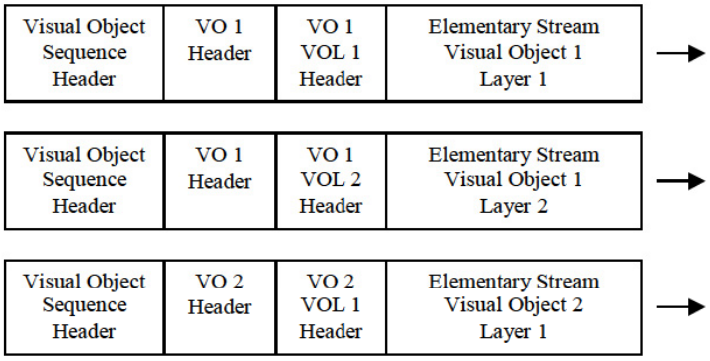


Figure 6-21 — Example Visual Bitstream – Combined Configuration Information / Elementary Stream

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. fba_object().

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().

This part of ISO/IEC 14496 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 14496-1:2001. 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 14496-1:2001, 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 14496-1:2001 defines containers that are used to carry Visual Object Sequence, Visual Object and Video Object Layer configuration information. For video objects one container is used for each layer for each object. This container carries a Visual Object Sequence header, a Visual Object header and a Video Object Layer header. For other types of visual objects, one container per visual object is used. This container carries a Visual Object Sequence header and a Visual Object header. The Visual Object Sequence Header must be identical for all visual streams input simultaneously to a decoder. The Visual Object Headers for each layer of a multilayer object must be identical.

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

© ISO/IEC 2004 – All rights reserved

VisualObjectSequence(), VisualObject() and VideoObjectLayer(), with the exception of repetition of the Visual Object Sequence Header, the Visual Object Header and the Video Object Layer Header as described below. The Visual Object Sequence Header must be identical for all visual streams input simultaneously to a decoder. The Visual Object Headers for each layer of a multilayer object must be identical.

The Visual Object Sequence Header, the Visual Object Header and the Video Object Layer Header may be repeated in a single visual bitstream. Repeating these headers enables random access into the visual bitstream and recovery of these headers when the original headers are corrupted by errors. This header repetition is used only when visual_object_type in the Visual Object Header indicates that visual object type is video. (i.e. visual_object_type=="video ID") All of the data elements in the Visual Object Sequence Header, the Visual Object Header and the Video Object Layer Header repeated in a visual bitstream shall have the same value as in the original headers, except that first_half_vbv_occupancy and latter_half_vbv_occupancy may be changed to specify the VBV occupancy just before the removal of the first VOP following the repeated Video Object Layer Header.

6.2.2 Visual Object Sequence and Visual Object

VisualObjectSequence() {	No. of bits	Mnemonic
do {		
visual_object_sequence_start_code	32	bslbf
profile_and_level_indication	8	uimsbf
if (profile_and_level_indication == 11100001-11101000) {		
next_start_code_studio()		
extension_and_user_data(0)		
StudioVisualObject()		
} else {		
while (next_bits()== user_data_start_code){		
user_data()		
}		
VisualObject()		
}		
} while (next_bits() != visual_object_sequence_end_code)		
visual_object_sequence_end_code	32	bslbf
}		

VisualObject() {	No. of bits	Mnemonic
visual_object_start_code	32	bslbf
is_visual_object_identifier	1	uimsbf
if (is_visual_object_identifier) {		
visual_object_verid	4	uimsbf
visual_object_priority	3	uimsbf
}		
visual_object_type	4	uimsbf
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()		
}		

© ISO/IEC 2004 – All rights reserved

if (visual_object_type == "video ID") {		
video_object_start_code	32	bslbf
VideoObjectLayer()		
}		
else if (visual_object_type == "still texture ID") {		
StillTextureObject()		
}		
else if (visual_object_type == "mesh ID") {		
MeshObject()		
}		
else if (visual_object_type == "FBA ID") {		
FBAObject()		
}		
else if (visual_object_type == "3D mesh ID") {		
3D_Mesh_Object()		
}		
if (next_bits() != "0000 0000 0000 0000 0000 0001")		
next_start_code()		
}		

video_signal_type() {	No. of bits	Mnemonic
video_signal_type	1	bslbf
if (video_signal_type) {		
video_format	3	uimsbf
video_range	1	bslbf
colour_description	1	bslbf
if (colour_description) {		
colour_primaries	8	uimsbf
transfer_characteristics	8	uimsbf
matrix_coefficients	8	uimsbf
}		
}		
}		

6.2.2.1 User data

user_data() {	No. of bits	Mnemonic
user_data_start_code	32	bslbf
while(next_bits() != '0000 0000 0000 0000 0000 0001') {		
user_data	8	uimsbf
}		
}		

© ISO/IEC 2004 – All rights reserved

6.2.3 Video Object Layer

VideoObjectLayer() {	No. of bits	Mnemonic
if(next_bits() == video_object_layer_start_code) {		
short_video_header = 0		
video_object_layer_start_code	32	bslbf
random_accessible_vol	1	bslbf
video_object_type_indication	8	uimsbf
if (video_object_type_indication == "Fine Granularity Scalable") {		
fgs_layer_type	2	uimsbf
video_object_layer_priority	3	uimsbf
aspect_ratio_info	4	uimsbf
if (aspect_ratio_info == "extended_PAR") {		
par_width	8	uimsbf
par_height	8	uimsbf
}		
vol_control_parameters	1	bslbf
if (vol_control_parameters) {		
chroma_format	2	uimsbf
low_delay	1	uimsbf
}		
marker_bit	1	bslbf
vop_time_increment_resolution	16	uimsbf
marker_bit	1	bslbf
fixed_vop_rate	1	bslbf
if (fixed_vop_rate)		
fixed_vop_time_increment	1-16	uimsbf
marker_bit	1	bslbf
video_object_layer_width	13	uimsbf
marker_bit	1	bslbf
video_object_layer_height	13	uimsbf
marker_bit	1	bslbf
interlaced	1	bslbf
if (fgs_layer_type =="FGST" || fgs_layer_type =="FGS_FGST")		
fgs_ref_layer_id	4	uimsbf
if (fgs_layer_type =="FGS" || fgs_layer_type =="FGS_FGST") {		
fgs_frequency_weighting_enable	1	bslbf
if (fgs_frequency_weighting_enable) {		
load_fgs_frequency_weighting_matrix	1	bslbf
if (load_fgs_frequency_weighting_matrix)		
fgs_frequency_weighting_matrix	3*[2-64]	uimsbf
}		
}		
if (fgs_layer_type =="FGST" || fgs_layer_type =="FGS_FGST") {		
fgst_frequency_weighting_enable	1	bslbf

© ISO/IEC 2004 – All rights reserved

if (fgst_frequency_weighting_enable) {		
load_fgst_frequency_weighting_matrix	1	bslbf
if (load_fgst_frequency_weighting_matrix)		
fgst_frequency_weighting_matrix	3*[2-64]	uimsbf
}		
}		
quarter_sample	1	bslbf
fgs_resync_marker_disable	1	bslbf
do {		
if (nextbits_bytealigned() == group_of_vop_start_code)		
Group_of_VideoObjectPlane()		
FGSVideoObjectPlane()		
} while((nextbits_bytealigned()==group_of_vop_start_code)|| (nextbits_bytealigned()==fgs_vop_start_code))		
} else {		
is_object_layer_identifier	1	uimsbf
if (is_object_layer_identifier) {		
video_object_layer_verid	4	uimsbf
video_object_layer_priority	3	uimsbf
}		
aspect_ratio_info	4	uimsbf
if (aspect_ratio_info == “extended_PAR”) {		
par_width	8	uimsbf
par_height	8	uimsbf
}		
vol_control_parameters	1	bslbf
if (vol_control_parameters) {		
chroma_format	2	uimsbf
low_delay	1	uimsbf
vbv_parameters	1	blsbf
if (vbv_parameters) {		
first_half_bit_rate	15	uimsbf
marker_bit	1	bslbf
latter_half_bit_rate	15	uimsbf
marker_bit	1	bslbf
first_half_vbv_buffer_size	15	uimsbf
marker_bit	1	bslbf
latter_half_vbv_buffer_size	3	uimsbf
first_half_vbv_occupancy	11	uimsbf
marker_bit	1	blsbf
latter_half_vbv_occupancy	15	uimsbf
marker_bit	1	blsbf
}		
}		
video_object_layer_shape	2	uimsbf
if (video_object_layer_shape == "grayscale" && video_object_layer_verid != ‘0001’)		
video_object_layer_shape_extension	4	uimsbf

© ISO/IEC 2004 – All rights reserved

marker_bit	1	bslbf
vop_time_increment_resolution	16	uimsbf
marker_bit	1	bslbf
fixed_vop_rate	1	bslbf
if (fixed_vop_rate)		
fixed_vop_time_increment	1-16	uimsbf
if (video_object_layer_shape != "binary only") {		
if (video_object_layer_shape == "rectangular") {		
marker_bit	1	bslbf
video_object_layer_width	13	uimsbf
marker_bit	1	bslbf
video_object_layer_height	13	uimsbf
marker_bit	1	bslbf
}		
interlaced	1	bslbf
obmc_disable	1	bslbf
if (video_object_layer_verid == '0001')		
sprite_enable	1	bslbf
else		
sprite_enable	2	uimsbf
if (sprite_enable== "static" || sprite_enable == "GMC") {		
if (sprite_enable != "GMC") {		
sprite_width	13	uimsbf
marker_bit	1	bslbf
sprite_height	13	uimsbf
marker_bit	1	bslbf
sprite_left_coordinate	13	simsbf
marker_bit	1	bslbf
sprite_top_coordinate	13	simsbf
marker_bit	1	bslbf
}		
no_of_sprite_warping_points	6	uimsbf
sprite_warping_accuracy	2	uimsbf
sprite_brightness_change	1	bslbf
if (sprite_enable != "GMC")		
low_latency_sprite_enable	1	bslbf
}		
if (video_object_layer_verid != '0001' && video_object_layer_shape != "rectangular")		
sadct_disable	1	bslbf
not_8_bit	1	bslbf
if (not_8_ bit) {		
quant_precision	4	uimsbf
bits_per_pixel	4	uimsbf
}		
if (video_object_layer_shape=="grayscale") {		
no_gray_quant_update	1	bslbf
composition_method	1	bslbf

© ISO/IEC 2004 – All rights reserved

linear_composition	1	bslbf
}		
quant_type	1	bslbf
if (quant_type) {		
load_intra_quant_mat	1	bslbf
if (load_intra_quant_mat)		
intra_quant_mat	8*[2-64]	uimsbf
load_nonintra_quant_mat	1	bslbf
if (load_nonintra_quant_mat)		
nonintra_quant_mat	8*[2-64]	uimsbf
if(video_object_layer_shape=="grayscale") {		
for(i=0; i<aux_comp_count; i++) {		
load_intra_quant_mat_grayscale	1	bslbf
if(load_intra_quant_mat_grayscale)		
intra_quant_mat_grayscale[i]	8*[2-64]	uimsbf
load_nonintra_quant_mat_grayscale	1	bslbf
if(load_nonintra_quant_mat_grayscale)		
nonintra_quant_mat_grayscale[i]	8*[2-64]	uimsbf
}		
}		
}		
if (video_object_layer_verid != '0001')		
quarter_sample	1	bslbf
complexity_estimation_disable	1	bslbf
if (!complexity_estimation_disable)		
define_vop_complexity_estimation_header()		
resync_marker_disable	1	bslbf
data_partitioned	1	bslbf
if(data_partitioned)		
reversible_vlc	1	bslbf
if(video_object_layer_verid != '0001') {		
newpred_enable	1	bslbf
if (newpred_enable) {		
requested_upstream_message_type	2	uimsbf
newpred_segment_type	1	bslbf
}		
reduced_resolution_vop_enable	1	bslbf
}		
scalability	1	bslbf
if (scalability) {		
hierarchy_type	1	bslbf
ref_layer_id	4	uimsbf
ref_layer_sampling_direc	1	bslbf
hor_sampling_factor_n	5	uimsbf
hor_sampling_factor_m	5	uimsbf
vert_sampling_factor_n	5	uimsbf
vert_sampling_factor_m	5	uimsbf
enhancement_type	1	bslbf

© ISO/IEC 2004 – All rights reserved

if(video_object_layer_shape == “binary” && hierarchy_type== ‘0’) {		
use_ref_shape	1	bslbf
use_ref_texture	1	bslbf
shape_hor_sampling_factor_n	5	uimsbf
shape_hor_sampling_factor_m	5	uimsbf
shape_vert_sampling_factor_n	5	uimsbf
shape_vert_sampling_factor_m	5	uimsbf
}		
}		
}		
else {		
if(video_object_layer_verid !=”0001”) {		
scalability	1	bslbf
if(scalability) {		
ref_layer_id	4	uimsbf
shape_hor_sampling_factor_n	5	uimsbf
shape_hor_sampling_factor_m	5	uimsbf
shape_vert_sampling_factor_n	5	uimsbf
shape_vert_sampling_factor_m	5	uimsbf
}		
}		
resync_marker_disable	1	bslbf
}		
next_start_code()		
while (next_bits()== user_data_start_code){		
user_data()		
}		
if (sprite_enable == “static” && !low_latency_sprite_enable)		
VideoObjectPlane()		
do {		
if (next_bits() == group_of_vop_start_code)		
Group_of_VideoObjectPlane()		
VideoObjectPlane()		
if ((preceding_vop_coding_type == "B" || preceding_vop_coding_type == "S" || video_object_layer_shape != "rectangular") && next_bits() == stuffing_start_code) {		
stuffing_start_code	32	bslbf
while (next_bits() != ‘0000 0000 0000 0000 0000 0001’)		
stuffing_byte	8	bslbf
}		
} while ((next_bits() == group_of_vop_start_code) || (next_bits() == vop_start_code))		
}		
} else {		
short_video_header = 1		

© ISO/IEC 2004 – All rights reserved

do {		
video_plane_with_short_header()		
} while(next_bits() == short_video_start_marker)		
}		
}		
NOTE — preceding_vop_coding_type has the same value as vop_coding_type in the immediately preceding VideoObjectPlane() in the decoding order.		

define_vop_complexity_estimation_header() {	No. of bits	Mnemonic
estimation_method	2	uimsbf
if (estimation_method ==’00’ || estimation_method == ‘01’) {		
shape_complexity_estimation_disable	1	
if (!shape_complexity_estimation_disable) {		bslbf
opaque	1	bslbf
transparent	1	bslbf
intra_cae	1	bslbf
inter_cae	1	bslbf
no_update	1	bslbf
upsampling	1	bslbf
}		
texture_complexity_estimation_set_1_disable	1	bslbf
if (!texture_complexity_estimation_set_1_disable) {		
intra_blocks	1	bslbf
inter_blocks	1	bslbf
inter4v_blocks	1	bslbf
not_coded_blocks	1	bslbf
}		
marker_bit	1	bslbf
texture_complexity_estimation_set_2_disable	1	bslbf
if (!texture_complexity_estimation_set_2_disable) {		
dct_coefs	1	bslbf
dct_lines	1	bslbf
vlc_symbols	1	bslbf
vlc_bits	1	bslbf
}		
motion_compensation_complexity_disable	1	bslbf
ilf (!motion_compensation_complexity_disable) {		
apm	1	bslbf
npm	1	bslbf
interpolate_mc_q	1	bslbf
forw_back_mc_q	1	bslbf
halfpel2	1	bslbf
halfpel4	1	bslbf
}		
marker_bit	1	bslbf
if(estimation_method == ‘01’) {		
version2_complexity_estimation_disable	1	bslbf

© ISO/IEC 2004 – All rights reserved

if (!version2_complexity_estimation_disable) {		
sadct	1	bslbf
quarterpel	1	bslbf
}		
}		
}		
}		

6.2.4 Group of Video Object Plane

Group_of_VideoObjectPlane() {	No. of bits	Mnemonic
group_of_vop_start_code	32	bslbf
time_code	18	
closed_gov	1	bslbf
broken_link	1	bslbf
next_start_code()		
while (next_bits()== user_data_start_code){		
user_data()		
}		
}		

6.2.5 Video Object Plane and Video Plane with Short Header

VideoObjectPlane() {	No. of bits	Mnemonic
vop_start_code	32	bslbf
vop_coding_type	2	uimsbf
do {		
modulo_time_base	1	bslbf
} while (modulo_time_base != '0')		
marker_bit	1	bslbf
vop_time_increment	1-16	uimsbf
marker_bit	1	bslbf
vop_coded	1	bslbf
if (vop_coded == '0') {		
next_start_code()		
return()		
}		
if (newpred_enable) {		
vop_id	4-15	uimsbf
vop_id_for_prediction_indication	1	bslbf
if (vop_id_for_prediction_indication)		
vop_id_for_prediction	4-15	uimsbf
marker_bit	1	bslbf
}		

© ISO/IEC 2004 – All rights reserved

if ((video_object_layer_shape != "binary only") && (vop_coding_type == "P" || (vop_coding_type == "S" && sprite_enable == "GMC")))		
vop_rounding_type	1	bslbf
if ((reduced_resolution_vop_enable) && (video_object_layer_shape == "rectangular") && ((vop_coding_type == "P") || (vop_coding_type == "I")))		
vop_reduced_resolution	1	bslbf
if (video_object_layer_shape != "rectangular") {		
if(!(sprite_enable == "static" && vop_coding_type == "I")) {		
vop_width	13	uimsbf
marker_bit	1	bslbf
vop_height	13	uimsbf
marker_bit	1	bslbf
vop_horizontal_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
vop_vertical_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
}		
if ((video_object_layer_shape != " binary only") && scalability && enhancement_type)		
background_composition	1	bslbf
change_conv_ratio_disable	1	bslbf
vop_constant_alpha	1	bslbf
if (vop_constant_alpha)		
vop_constant_alpha_value	8	bslbf
}		
if (video_object_layer_shape != ''binary only'')		
if (!complexity_estimation_disable)		
read_vop_complexity_estimation_header()		
if (video_object_layer_shape != "binary only") {		
intra_dc_vlc_thr	3	uimsbf
if (interlaced) {		
top_field_first	1	bslbf
alternate_vertical_scan_flag	1	bslbf
}		
}		
if ((sprite_enable =="static" || sprite_enable=="GMC") && vop_coding_type == "S") {		
if (no_of_sprite_warping_points > 0)		
sprite_trajectory()		
if (sprite_brightness_change)		
brightness_change_factor()		
if(sprite_enable == "static") {		
if (sprite_transmit_mode != "stop" && low_latency_sprite_enable) {		
do {		
sprite_transmit_mode	2	uimsbf

© ISO/IEC 2004 – All rights reserved

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	uimsbf
if(video_object_layer_shape==”grayscale”)		
for(i=0; i<aux_comp_count; i++)		
vop_alpha_quant[i]	6	uimsbf
if (vop_coding_type != “I”)		
vop_fcode_forward	3	uimsbf
if (vop_coding_type == “B”)		
vop_fcode_backward	3	uimsbf
if (!scalability) {		
if (video_object_layer_shape != “rectangular” && vop_coding_type != “I”)		
vop_shape_coding_type	1	bslbf
motion_shape_texture()		
while (nextbits_bytealigned() == resync_marker) {		
video_packet_header()		
motion_shape_texture()		
}		
}		
else {		
if (enhancement_type) {		
load_backward_shape	1	bslbf
if (load_backward_shape) {		
backward_shape_width	13	uimsbf
marker_bit	1	bslbf
backward_shape_ height	13	uimsbf
marker_bit	1	bslbf
backward_shape_horizontal_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
backward_shape_vertical_mc_spatial_ref	13	simsbf
backward_shape()		
load_forward_shape	1	bslbf
if (load_forward_shape) {		
forward_shape_width	13	uimsbf
marker_bit	1	bslbf
forward_shape_height	13	uimsbf
marker_bit	1	bslbf

© ISO/IEC 2004 – All rights reserved

forward_shape_horizontal_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
forward_shape_vertical_mc_spatial_ref	13	simsbf
forward_shape()		
}		
}		
}		
ref_select_code	2	uimsbf
combined_motion_shape_texture()		
}		
}		
else {		
combined_motion_shape_texture()		
while (nextbits_bytealigned() == resync_marker) {		
video_packet_header()		
combined_motion_shape_texture()		
}		
}		
next_start_code()		
}		

6.2.5.1 Complexity Estimation Header

read_vop_complexity_estimation_header() {		No. of bits	Mnemonic
if (estimation_method=='00' || estimation_method=='01') {			
if (vop_coding_type=="I") {			
if (opaque)	**dcecs_opaque**	8	uimsbf
if (transparent)	**dcecs_transparent**	8	uimsbf
if (intra_cae)	**dcecs_intra_cae**	8	uimsbf
if (inter_cae)	**dcecs_inter_cae**	8	uimsbf
if (no_update)	**dcecs_no_update**	8	uimsbf
if (upsampling)	**dcecs_upsampling**	8	uimsbf
if (intra_blocks)	**dcecs_intra_blocks**	8	uimsbf
if (not_coded_blocks)	**dcecs_not_coded_blocks**	8	uimsbf
if (dct_coefs)	**dcecs_dct_coefs**	8	uimsbf
if (dct_lines)	**dcecs_dct_lines**	8	uimsbf
if (vlc_symbols)	**dcecs_vlc_symbols**	8	uimsbf
if (vlc_bits)	**dcecs_vlc_bits**	4	uimsbf
if (sadct)	**dcecs_sadct**	8	uimsbf
}			
if (vop_coding_type=="P") {			
if (opaque)	**dcecs_opaque**	8	uimsbf
if (transparent)	**dcecs_transparent**	8	uimsbf
if (intra_cae)	**dcecs_intra_cae**	8	uimsbf
if (inter_cae)	**dcecs_inter_cae**	8	uimsbf

© ISO/IEC 2004 – All rights reserved

if (no_update)	**dcecs_no_update**	8	uimsbf
if (upsampling)	**dcecs_upsampling**	8	uimsbf
if (intra_blocks)	**dcecs_intra_blocks**	8	uimsbf
if (not_coded)	**dcecs_not_coded_blocks**	8	uimsbf
if (dct_coefs)	**dcecs_dct_coefs**	8	uimsbf
if (dct_lines)	**dcecs_dct_lines**	8	uimsbf
if (vlc_symbols)	**dcecs_vlc_symbols**	8	uimsbf
if (vlc_bits)	**dcecs_vlc_bits**	4	uimsbf
if (inter_blocks)	**dcecs_inter_blocks**	8	uimsbf
if (inter4v_blocks)	**dcecs_inter4v_blocks**	8	uimsbf
if (apm)	**dcecs_apm**	8	uimsbf
if (npm)	**dcecs_npm**	8	uimsbf
if (forw_back_mc_q)	**dcecs_forw_back_mc_q**	8	uimsbf
if (halfpel2)	**dcecs_halfpel2**	8	uimsbf
if (halfpel4)	**dcecs_halfpel4**	8	uimsbf
if (sadct)	**dcecs_sadct**	8	uimsbf
if (quarterpel)	**dcecs_quarterpel**	8	uimsbf
}			
if (vop_coding_type==“B”) {			
if (opaque)	**dcecs_opaque**	8	uimsbf
if (transparent)	**dcecs_transparent**	8	uimsbf
if (intra_cae)	**dcecs_intra_cae**	8	uimsbf
if (inter_cae)	**dcecs_inter_cae**	8	uimsbf
if (no_update)	**dcecs_no_update**	8	uimsbf
if (upsampling)	**dcecs_upsampling**	8	uimsbf
if (intra_blocks)	**dcecs_intra_blocks**	8	uimsbf
if (not_coded_blocks)	**dcecs_not_coded_blocks**	8	uimsbf
if (dct_coefs)	**dcecs_dct_coefs**	8	uimsbf
if (dct_lines)	**dcecs_dct_lines**	8	uimsbf
if (vlc_symbols)	**dcecs_vlc_symbols**	8	uimsbf
if (vlc_bits)	**dcecs_vlc_bits**	4	uimsbf
if (inter_blocks)	**dcecs_inter_blocks**	8	uimsbf
if (inter4v_blocks)	**dcecs_inter4v_blocks**	8	uimsbf
if (apm)	**dcecs_apm**	8	uimsbf
if (npm)	**dcecs_npm**	8	uimsbf
if (forw_back_mc_q)	**dcecs_forw_back_mc_q**	8	uimsbf
if (halfpel2)	**dcecs_halfpel2**	8	uimsbf
if (halfpel4)	**dcecs_halfpel4**	8	uimsbf
if (interpolate_mc_q)	**dcecs_interpolate_mc_q**	8	uimsbf
if (sadct)	**dcecs_sadct**	8	uimsbf
if (quarterpel)	**dcecs_quarterpel**	8	uimsbf
}			
if (vop_coding_type==‘S’&& sprite_enable == “static”) {			
if (intra_blocks)	**dcecs_intra_blocks**	8	uimsbf
if (not_coded_blocks)	**dcecs_not_coded_blocks**	8	uimsbf
if (dct_coefs)	**dcecs_dct_coefs**	8	uimsbf
if (dct_lines)	**dcecs_dct_lines**	8	uimsbf

© ISO/IEC 2004 – All rights reserved

if (vlc_symbols)	**dcecs_vlc_symbols**	8	uimsbf
if (vlc_bits)	**dcecs_vlc_bits**	4	uimsbf
if (inter_blocks)	**dcecs_inter_blocks**	8	uimsbf
if (inter4v_blocks)	**dcecs_inter4v_blocks**	8	uimsbf
if (apm)	**dcecs_apm**	8	uimsbf
if (npm)	**dcecs_npm**	8	uimsbf
if (forw_back_mc_q)	**dcecs_forw_back_mc_q**	8	uimsbf
if (halfpel2)	**dcecs_halfpel2**	8	uimsbf
if (halfpel4)	**dcecs_halfpel4**	8	uimsbf
if (interpolate_mc_q)	**dcecs_interpolate_mc_q**	8	uimsbf
}			
}			
}			

6.2.5.2 Video Plane with Short Header

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

© ISO/IEC 2004 – All rights reserved

gob_layer() {	No. of bits	Mnemonic
gob_header_empty = 1		
if(gob_number != 0) {		
if (next_bits() == gob_resync_marker) {		
gob_header_empty = 0		
gob_resync_marker	17	bslbf
gob_number	5	uimsbf
gob_frame_id	2	bslbf
quant_scale	5	uimsbf
}		
}		
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	bslbf
gob_number++		
}		

video_packet_header() {	No. of bits	Mnemonic
next_resync_marker()		
resync_marker	17-23	uimsbf
if (video_object_layer_shape != “rectangular”) {		
header_extension_code	1	bslbf
if (header_extension_code && !(sprite_enable = “static“ && vop_coding_type == “I”)) {		
vop_width	13	uimsbf
marker_bit	1	bslbf
vop_height	13	uimsbf
marker_bit	1	bslbf
vop_horizontal_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
vop_vertical_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
}		
}		
macroblock_number	1-14	vlclbf
if (video_object_layer_shape != “binary only”)		
quant_scale	3-9	uimsbf
if (video_object_layer_shape == “rectangular”)		
header_extension_code	1	bslbf
if (header_extension_code) {		
do {		
modulo_time_base	1	bslbf
} while (modulo_time_base != ‘0’)		
marker_bit	1	bslbf

© ISO/IEC 2004 – All rights reserved

vop_time_increment	1-16	bslbf
marker_bit	1	bslbf
vop_coding_type	2	uimsbf
if (video_object_layer_shape != "rectangular") {		
change_conv_ratio_disable	1	bslbf
if (vop_coding_type != "I")		
vop_shape_coding_type	1	bslbf
}		
if (video_object_layer_shape != "binary only") {		
intra_dc_vlc_thr	3	uimsbf
if (sprite_enable == "GMC" && vop_coding_type == "S" && no_of_sprite_warping_points > 0)		
sprite_trajectory()		
if ((reduced_resolution_vop_enable) && (video_object_layer_shape == "rectangular") && ((vop_coding_type == "P") || (vop_coding_type == "I")))		
vop_reduced_resolution	1	bslbf
if (vop_coding_type != "I")		
vop_fcode_forward	3	uimsbf
if (vop_coding_type == "B")		
vop_fcode_backward	3	uimsbf
}		
}		
if (newpred_enable) {		
vop_id	4-15	uimsbf
vop_id_for_prediction_indication	1	bslbf
if (vop_id_for_prediction_indication)		
vop_id_for_prediction	4-15	uimsbf
marker_bit	1	bslbf
}		
}		

6.2.5.3 Motion Shape Texture

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

© ISO/IEC 2004 – All rights reserved

combined_motion_shape_texture() {	No. of bits	Mnemonic
do{		
macroblock()		
} while ((nextbits_bytealigned() != resync_marker && nextbits_bytealigned() != '000 0000 0000 0000 0000 0000') || valid_stuffing_bits() == 0)		
}		

data_partitioned_motion_shape_texture() {	No. of bits	Mnemonic
if (vop_coding_type == "I") {		
data_partitioned_i_vop()		
} else if (vop_coding_type == "P" || (vop_coding_type == "S" && sprite_enable == "GMC")) {		
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 bits	Mnemonic
do{		
if (video_object_layer_shape != "rectangular"){		
bab_type	1-3	vlclbf
if (bab_type >= 4) {		
if (!change_conv_rate_disable)		
conv_ratio	1-2	vlclbf
scan_type	1	bslbf
binary_arithmetic_code()		
}		
}		
if (!transparent_mb()) {		
if (video_object_layer_shape != "rectangular") {		
do {		
mcbpc	1-9	vlclbf
} while (derived_mb_type == "stuffing")		
} else {		
mcbpc	1-9	vlclbf
if (derived_mb_type == "stuffing")		
continue		
}		
if (mb_type == 4)		
dquant	2	bslbf
if (use_intra_dc_vlc) {		
for (j = 0; j < 4; j++) {		
if (!transparent_block(j)) {		

© ISO/IEC 2004 – All rights reserved

dct_dc_size_luminance	2-11	vlclbf
if (dct_dc_size_luminance > 0)		
dct_dc_differential	1-12	vlclbf
if (dct_dc_size_luminance > 8)		
marker_bit	1	bslbf
}		
}		
for (j = 0; j < 2; j++) {		
dct_dc_size_chrominance	2-12	vlclbf
if (dct_dc_size_chrominance > 0)		
dct_dc_differential	1-12	vlclbf
if (dct_dc_size_chrominance > 8)		
marker_bit	1	bslbf
}		
}		
}		
} while (next_bits() != dc_marker)		
dc_marker /* 110 1011 0000 0000 0001 */	19	bslbf
for (i = 0; i < mb_in_video_packet; i++) {		
if (!transparent_mb()) {		
ac_pred_flag	1	bslbf
cbpy	1-6	vlclbf
}		
}		
for (i = 0; i < mb_in_video_packet; i++) {		
if (!transparent_mb()) {		
for (j = 0; j < block_count; j++)		
block(j)		
}		
}		
}		

NOTE 1: The value of mb_in_video_packet is the number of macroblocks in a video packet. The count of stuffing macroblocks is not included in this value.

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

NOTE 3: The value of alpha_block_count is 4.

data_partitioned_p_vop() {	**No. of bits**	**Mnemonic**
do{		
if (video_object_layer_shape != “rectangular”){		
bab_type	1-7	vlclbf
if ((bab_type == 1) || (bab_type == 6)) {		
mvds_x	1-18	vlclbf
mvds_y	1-18	vlclbf
}		
if (bab_type >= 4) {		
if (!change_conv_rate_disable)		
conv_ratio	1-2	vlclbf

© ISO/IEC 2004 – All rights reserved

scan_type	1	bslbf
binary_arithmetic_code()		
}		
}		
if (!transparent_mb()) {		
if (video_object_layer_shape != "rectangular") {		
do {		
not_coded	1	bslbf
if (!not_coded)		
mcbpc	1-9	vlclbf
} while (!(not_coded || derived_mb_type != "stuffing"))		
} else {		
not_coded	1	bslbf
if (!not_coded) {		
mcbpc	1-9	vlclbf
if (derived_mb_type == "stuffing")		
continue		
}		
}		
if (!not_coded) {		
if (sprite_enable == "GMC" && vop_coding_type == "S" && derived_mb_type < 2)		
mcsel	1	bslbf
if ((!(sprite_enable == "GMC" && vop_coding_type == "S" && mcsel) && derived_mb_type < 2) || derived_mb_type == 2)		
motion_coding("forward", derived_mb_type)		
}		
}		
} while (next_bits() != motion_marker)		
motion_marker /* 1 1111 0000 0000 0001 */	17	bslbf
for (i = 0; i < mb_in_video_packet; i++) {		
if (!transparent_mb()) {		
if (!not_coded){		
if (derived_mb_type >= 3)		
ac_pred_flag	1	bslbf
cbpy	1-6	vlclbf
if (derived_mb_type == 1 || derived_mb_type == 4)		
dquant	2	bslbf
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	vlclbf
if (dct_dc_size_luminance > 0)		
dct_dc_differential	1-12	vlclbf
if (dct_dc_size_luminance > 8)		
marker_bit	1	bslbf
}		

© ISO/IEC 2004 – All rights reserved

}		
for (j = 0; j < 2; j++) {		
dct_dc_size_chrominance	2-12	vlclbf
if (dct_dc_size_chrominance > 0)		
dct_dc_differential	1-12	vlclbf
if (dct_dc_size_chrominance > 8)		
marker_bit	1	bslbf
}		
}		
}		
}		
}		
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 1: The value of mb_in_video_packet is the number of macroblocks in a video packet. The count of stuffing macroblocks is not included in this value.		
NOTE 2. The value of block_count is 6 in the 4:2:0 format.		
NOTE 3: The value of alpha_block_count is 4.		

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

6.2.5.4 Sprite coding

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

© ISO/IEC 2004 – All rights reserved

sprite_shape_texture() {	No. of bits	Mnemonic
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	Mnemonic
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	Mnemonic
dmv_length	2-12	uimsbf
if (dmv_length != '00')		
dmv_code	1-14	uimsbf
marker_bit	1	bslbf
}		

brightness_change_factor() {	No. of bits	Mnemonic
brightness_change_factor_size	1-4	uimsbf
brightness_change_factor_code	5-10	uimsbf
}		

6.2.6 Macroblock

macroblock() {	No. of bits	Mnemonic
if (vop_coding_type != "B") {		
if (video_object_layer_shape != "rectangular" && !(sprite_enable == "static" && low_latency_sprite_enable && sprite_transmit_mode == "update"))		
mb_binary_shape_coding()		

© ISO/IEC 2004 – All rights reserved

if (video_object_layer_shape != “binary only”) {		
if (!transparent_mb()) {		
if (video_object_layer_shape != “rectangular” && !(sprite_enable = "static" && low_latency_sprite_enable && sprite_transmit_mode == “update”)) {		
do{		
if (vop_coding_type != “I” && !(sprite_enable == "static" && sprite_transmit_mode == “piece”))		
not_coded	1	bslbf
if (!not_coded || vop_coding_type == “I” || (vop_coding_type == "S" && low_latency_sprite_enable && sprite_transmit_mode == "piece"))		
mcbpc	1-9	vlclbf
} while(!(not_coded || derived_mb_type != “stuffing”))		
} else {		
if (vop_coding_type != “I” && !(sprite_enable = "static" && sprite_transmit_mode == “piece”))		
not_coded	1	bslbf
if (!not_coded || vop_coding_type == “I” || (vop_coding_type == "S" && low_latency_sprite_enable && sprite_transmit_mode == "piece"))		
mcbpc	1-9	vlclbf
}		
if (!not_coded || vop_coding_type == “I” || (vop_coding_type == "S" && low_latency_sprite_enable && sprite_transmit_mode == "piece")) {		
if (vop_coding_type == “S” && sprite_enable == “GMC” && (derived_mb_type == 0 || derived_mb_type == 1))		
mcsel	1	bslbf
if (!short_video_header && (derived_mb_type == 3 || derived_mb_type == 4))		
ac_pred_flag	1	bslbf
if (derived_mb_type != “stuffing”)		
cbpy	1-6	vlclbf
else		
return()		
if (derived_mb_type == 1 || derived_mb_type == 4)		
dquant	2	bslbf
if (interlaced)		
interlaced_information()		
if (!(ref_select_code==‘11’ && scalability) && sprite_enable != “static”) {		
if ((derived_mb_type == 0 || derived_mb_type == 1) && (vop_coding_type == “P” || (vop_coding_type == “S” && !mcsel))) {		

© ISO/IEC 2004 – All rights reserved

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)) || (sprite_enable == "GMC" && backward_reference_vop_coding_type == "S")) && video_object_layer_shape != "binary only") {		
if (!transparent_mb()) {		
modb	1-2	vlclbf
if (modb != '1') {		
mb_type	1-4	vlclbf
if (modb == '00')		
cbpb	3-6	vlclbf
if (ref_select_code != '00' || !scalability) {		
if (mb_type != "1" && cbpb!=0)		
dbquant	1-2	vlclbf
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")		

© ISO/IEC 2004 – All rights reserved

}		
if (ref_select_code == '00' && scalability && cbpb !=0) {		
dbquant	1-2	vlclbf
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()) {		
for(j=0; j<aux_comp_count; j++) {		
if(vop_coding_type=="I" || ((vop_coding_type=="P" || (vop_coding_type=="S" && sprite_enable=="GMC")) && !not_coded && (derived_mb_type==3 || derived_mb_type==4))) {		
coda_i	1	bslbf
if(coda_i=="coded") {		
ac_pred_flag_alpha	1	bslbf
cbpa	1-6	vlclbf
for(i=0;i<alpha_block_count;i++)		
if(!transparent_block())		
alpha_block(i)		
}		
} else { /* P, S(GMC) or B macroblock */		
if(vop_coding_type == "P" || (sprite_enable == "GMC" && (vop_coding_type=="S" || backward_reference_vop_coding_type=="S")) || co_located_not_coded != 1) {		
coda_pb	1-2	vlclbf
if(coda_pb=="coded") {		
cbpa	1-6	vlclbf
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. backward_reference_vop_coding_type means the vop_coding_type of the backward reference VOP as described in subclause 7.6.7.

© ISO/IEC 2004 – All rights reserved

6.2.6.1 MB Binary Shape Coding

mb_binary_shape_coding() {	No. of bits	Mnemonic
if(!(scalability && hierarchy_type == '0' && (enhancement_type == '0' || use_ref_shape == '0')) && !(scalability && video_object_layer_shape == "binary only")) {		
bab_type	1-7	vlclbf
if (vop_coding_type == 'P' || vop_coding_type == 'B' || (vop_coding_type == 'S' && sprite_enable == "GMC")) {		
if ((bab_type==1) || (bab_type == 6)) {		
mvds_x	1-18	vlclbf
mvds_y	1-18	vlclbf
}		
}		
if (bab_type >=4) {		
if (!change_conv_ratio_disable)		
conv_ratio	1-2	vlcbf
scan_type	1	bslbf
binary_arithmetic_code()		
}		
} else {		
if (!use_ref_shape || video_object_layer_shape == "binary only") {		
enh_bab_type	1-3	vlclbf
if (enh_bab_type == 3)		
scan_type	1	bslbf
if (enh_bab_type == 1 || enh_bab_type == 3)		
enh_binary_arithmetic_code()		
}		
}		
}		

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

© ISO/IEC 2004 – All rights reserved

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

6.2.6.2 Motion vector

motion_vector (mode) {	No. of bits	Mnemonic
if (mode == „direct“) {		
horizontal_mv_data	1-13	vlclbf
vertical_mv_data	1-13	vlclbf
}		
else if (mode == „forward“) {		
horizontal_mv_data	1-13	vlclbf
if ((vop_fcode_forward != 1) && (horizontal_mv_data != 0))		
horizontal_mv_residual	1-6	uimsbf
vertical_mv_data	1-13	vlclbf
if ((vop_fcode_forward != 1) && (vertical_mv_data != 0))		
vertical_mv_residual	1-6	uimsbf
}		
else if (mode == „backward“) {		
horizontal_mv_data	1-13	vlclbf
if ((vop_fcode_backward != 1) && (horizontal_mv_data != 0))		
horizontal_mv_residual	1-6	uimsbf
vertical_mv_data	1-13	vlclbf
if ((vop_fcode_backward != 1) && (vertical_mv_data != 0))		
vertical_mv_residual	1-6	uimsbf
}		
}		

6.2.6.3 Interlaced Information

interlaced_information( ) {	No. of bits	Mnemonic
if ((derived_mb_type == 3) || (derived_mb_type == 4) || (cbp != 0))		
dct_type	1	bslbf

© ISO/IEC 2004 – All rights reserved

if (((vop_coding_type == "P") && ((derived_mb_type == 0) || (derived_mb_type == 1))) || ((sprite_enable == "GMC") && (vop_coding_type == "S") && (derived_mb_type < 2) && (!mcsel)) || ((vop_coding_type == "B") && (mb_type != "1"))) {		
field_prediction	1	bslbf
if (field_prediction) {		
if (vop_coding_type == "P" || vop_coding_type == "S" || (vop_coding_type == "B" && mb_type != "001")) {		
forward_top_field_reference	1	bslbf
forward_bottom_field_reference	1	bslbf
}		
if ((vop_coding_type == "B") && (mb_type != "0001")) {		
backward_top_field_reference	1	bslbf
backward_bottom_field_reference	1	bslbf
}		
}		
}		
}		

6.2.7 Block

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

block(i) {	No. of bits	Mnemonic
last = 0		
if(!data_partitioned && (derived_mb_type == 3 || derived_mb_type == 4)) {		
if(short_video_header == 1)		
intra_dc_coefficient	8	uimsbf
else if (use_intra_dc_vlc == 1) {		
if (i<4) {		
dct_dc_size_luminance	2-11	vlclbf
if(dct_dc_size_luminance != 0)		
dct_dc_differential	1-12	vlclbf
if (dct_dc_size_luminance > 8)		
marker_bit	1	bslbf
} else {		
dct_dc_size_chrominance	2-12	vlclbf
if(dct_dc_size_chrominance !=0)		
dct_dc_differential	1-12	vlclbf
if (dct_dc_size_chrominance > 8)		
marker_bit	1	bslbf
}		
}		
}		

© ISO/IEC 2004 – All rights reserved

if (pattern_code[i])		
while (! last)		
DCT coefficient	3-24	vlclbf
}		
NOTE "last" is defined to be the LAST flag resulting from reading the most recent DCT coefficient.		

6.2.7.1 Alpha Block

The syntax for DCT coefficient decoding is the same as for block(i) in subclause 6.2.8.

alpha_block(i) {	No. of bits	Mnemonic
last = 0		
if(!data_partitioned && (vop_coding_type == "I" || (vop_coding_type == "P" || (vop_coding_type == "S" && sprite_enable == "GMC")) && !not_coded && (derived_mb_type == 3 || derived_mb_type == 4)))) {		
dct_dc_size_alpha	2-11	vlclbf
if(dct_dc_size_alpha != 0)		
dct_dc_differential	1-12	vlclbf
if (dct_dc_size_alpha > 8)		
marker_bit	1	bslbf
}		
if (pattern_code[i])		
while (! last)		
DCT coefficient	3-24	vlclbf
}		
NOTE "last" is defined to be the LAST flag resulting from reading the most recent DCT coefficient.		

6.2.8 Still Texture Object

StillTextureObject() {	No. of bits	Mnemonic
still_texture_object_start_code	32	bslbf
if (visual_object_verid != 0001) {		
tiling_disable	1	bslbf
texture_error_resilience_disable	1	bslbf
texture_object_id	16	uimsbf
marker_bit	1	bslbf
wavelet_filter_type	1	uimsbf
wavelet_download	1	uimsbf
wavelet_decomposition_levels	4	uimsbf
scan_direction	1	bslbf
start_code_enable	1	bslbf
texture_object_layer_shape	2	uimsbf
quantisation_type	2	uimsbf
if (quantisation_type == 2) {		

© ISO/IEC 2004 – All rights reserved

spatial_scalability_levels	4	uimsbf
if (spatial_scalability_levels != wavelet_decomposition_levels) {		
use_default_spatial_scalability	1	uimsbf
if (use_default_spatial_layer_size == 0)		
for (i=0; i<spatial_scalability_levels – 1; i++)		
wavelet_layer_index	4	uimsbf
}		
}		
if (wavelet_download == “1”){		
uniform_wavelet_filter	1	uimsbf
if (uniform_wavelet_filter == “1”)		
download_wavelet_filters()		
else		
for (i=0; i<wavelet_decomposition_levels; i++)		
download_wavelet_filters( )		
}		
wavelet_stuffing	3	uimsbf
if(!texture_error_resilience_disable) {		
target_segment_length	16	uimsbf
marker_bit	1	bslbf
}		
if(texture_object_layer_shape == “00”) {		
texture_object_layer_width	15	uimsbf
marker_bit	1	bslbf
texture_object_layer_height	15	uimsbf
marker_bit	1	bslbf
}		
else if (texture_object_layer_shape == “01” or “11”) {		
horizontal_ref	15	uimsbf
marker_bit	1	bslbf
vertical_ref	15	uimsbf
marker_bit	1	bslbf
object_width	15	uimsbf
marker_bit	1	bslbf
object_height	15	uimsbf
marker_bit	1	bslbf
/* if tiling_disable == “1” and texture_object_layer_shape == “01” configuration information precedes this point; elementary stream data follows. See Annex K */		
if(tiling_disable == “1”)		
shape_object_decoding()		
}		
if (tiling_disable == “0”){		
tile_width	15	uimsbf
marker_bit	1	bslbf
tile_height	15	uimsbf
marker_bit	1	bslbf
number_of_tiles	16	uimsbf
marker_bit	1	bslbf

© ISO/IEC 2004 – All rights reserved

tiling_jump_table_enable	1	bslbf
if (tiling jump_table_enable == "1") {		
for (i=0; i<number_of_tiles; i++) {		
tile_size_high	16	uimsbf
market_bit	1	bslbf
tile_size_low	16	uimsbf
market_bit	1	bslbf
}		
}		
next_start_code()		
}		
/* if tiling_disable == "0" or texture_object_layer_shape == "00" configuration information precedes this point; elementary stream data follows. See Annex K */		
do {		
if(tiling_disable == "0") {		
texture_tile_start_code	32	bslbf
tile_id	16	uimsbf
if (texture_object_layer_shape == "01") {		
marker_bit	1	bslbf
texture_tile_type	2	uimsbf
marker_bit	1	bslbf
}		
}		
if (!texture_error_resilience_disable) {		
if (texture_object_layer_shape== "01" && tiling_disable=="0") {		
if (texture_tile_type=="boundary tile")		
shape_object_decoding()		
}		
while (nextbit_bytealigned () == texture_marker) {		
TexturePacketHeader ()		
do {		
while (texture_unit_not_completed) {		
DecodeStu()		
if (segment_length >= target_segment_length)		
decode_segment_marker()		
}		
} while (nextbit_bytealigned () != texture_marker)		
}		
}		
else {		
if (texture_object_layer_shape!= "01" && tiling_disable=="0") {		
if (texture_tile_type=="boundary tile")		
shape_object_decoding()		
}		
for (color = "y", "u", "v" ,"α") {		
wavelet_dc_decode()		
}		

© ISO/IEC 2004 – All rights reserved

if (quantisation_type == 1) {		
TextureLayerSQ ( )		
}		
else if (quantisation_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 (quantisation_type == 3) {		
for (color = “y”, “u”, “v”)		
do {		
quant_byte	8	uimsbf
} while (quant_byte >> 7)		
max_bitplanes	5	uimsbf
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)		
}		
}		
} /* error_resi_disable */		
if (tiling_disable == “0”)		
next_start_code()		
} while (nextbits_bytealigned () == texture_tile_start_code)		
}		
else { /* version 1 */		
texture_object_id	16	uimsbf
marker_bit	1	bslbf
wavelet_filter_type	1	uimsbf
wavelet_download	1	uimsbf
wavelet_decomposition_levels	4	uimsbf
scan_direction	1	bslbf
start_code_enable	1	bslbf
texture_object_layer_shape	2	uimsbf
quantisation_type	2	uimsbf
if (quantisation_type == 2) {		
spatial_scalability_levels	4	uimsbf

© ISO/IEC 2004 – All rights reserved

if (spatial_scalability_levels != wavelet_decomposition_levels) {		
use_default_spatial_scalability	1	uimsbf
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	uimsbf
if (uniform_wavelet_filter == “1”)		
download_wavelet_filters()		
else		
for (i=0; i<wavelet_decomposition_levels; i++)		
download_wavelet_filters()		
}		
wavelet_stuffing	3	uimsbf
if(texture_object_layer_shape == “00”){		
texture_object_layer_width	15	uimsbf
marker_bit	1	bslbf
texture_object_layer_height	15	uimsbf
marker_bit	1	bslbf
}		
else {		
horizontal_ref	15	imsbf
marker_bit	1	bslbf
vertical_ref	15	imsbf
marker_bit	1	bslbf
object_width	15	uimsbf
marker_bit	1	bslbf
object_height	15	uimsbf
marker_bit	1	bslbf
shape_object_decoding ( )		
}		
/* configuration information precedes this point; elementary stream data follows. See Annex K */		
for (color = “y”, “u”, “v”)		
wavelet_dc_decode()		
if(quantisation_type == 1)		
TextureLayerSQ()		
else if (quantisation_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()		
}		

© ISO/IEC 2004 – All rights reserved

}		
else if (quantisation_type == 3) {		
for (color = “y”, “u”, “v” ,”α”)		
do{		
quant_byte		
} while(quant_byte >>7)		
max_bitplanes		
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)		
}		
}		
}		
}		

NOTE 1: The value of texture_unit_not_completed is '0' if the decoding of one sub-unit in a texture unit is completed. Otherwise the value is '1'.

NOTE 2: The value of first_packet_decoded is set to ‘0’ if the first packet has not been decoded. The value of first_packet_decoded is set to ‘1’ after the first packet has been decoded.

TexturePacketHeader() {	**No. of bits**	**Mnemonic**
next_texture_marker()		
texture_marker	17	bslbf
do {		
TU_first	8	uimsbf
} while (TU_first >> 7)		
do{		
TU_last	8	uimsbf
} while (TU_last >> 7)		
header_extention_code	1	bslbf
if (header_extention_code) {		
texture_object_id	16	uimsbf
marker_bit	1	bslbf
wavelet_filter_type	1	uimsbf
wavelet_download	1	uimsbf
wavelet_decomposition_levels	4	uimsbf
scan_direction	1	bslbf
start_code_enable	1	bslbf
texture_object_layer_shape	2	uimsbf
quantisation_type	2	uimsbf
if (quantisation_type == 2) {		
spatial_scalability_levels	4	uimsbf
if (spatial_scalability_levels != wavelet_decomposition_levels) {		
use_default_spatial_scalability	1	uimsbf

© ISO/IEC 2004 – All rights reserved

if (use_default_spatial_layer_size == 0)		
for (i=0; i<spatial_scalability_levels – 1; i++)		
wavelet_layer_index	4	uimsbf
}		
}		
if (wavelet_download == “1”){		
uniform_wavelet_filter	1	uimsbf
if (uniform_wavelet_filter == “1”)		
download_wavelet_filters()		
else		
for (i=0; i<wavelet_decomposition_levels; i++)		
download_wavelet_filters( )		
}		
wavelet_stuffing	3	uimsbf
if(texture_object_layer_shape == “00”) {		
texture_object_layer_width	15	uimsbf
marker_bit	1	bslbf
texture_object_layer_height	15	uimsbf
marker_bit	1	bslbf
}		
else {		
if (!first_packet_decoded) {		
horizontal_ref	15	uimsbf
marker_bit	1	bslbf
vertical_ref	15	uimsbf
marker_bit	1	bslbf
object_width	15	uimsbf
marker_bit	1	bslbf
object_height	15	uimsbf
marker_bit	1	bslbf
}		
}		
if (tiling_disable == “0”){		
tile_width	15	uimsbf
marker_bit	1	bslbf
tile_height	15	uimsbf
marker_bit	1	bslbf
}		
target_segment_length	16	uimsbf
marker_bit	1	bslbf
}		
}		

© ISO/IEC 2004 – All rights reserved

DecodeStu() {	No. of bits	Mnemonic
for (color = “y”, “u”, “v” ,”α”)		
wavelet_dc_decode ()		
if(quantisation_type == 1) {		
TextureLayerSQ()		
}		
else if (quantisation_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 (quantisation_type == 3) {		
for (color = “y”, “u”, “v” ,”α”)		
do{		
quant_byte	8	uimsbf
} while(quant_byte >> 7)		
max_bitplanes	5	uimsbf
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)		
}		
}		
}		

© ISO/IEC 2004 – All rights reserved

6.2.8.1 TextureLayerSQ

TextureLayerSQ() {	No. of bits	Mnemonic
if (scan_direction == 0) {		
for ("y", "u", "v","α") {		
do {		
quant_byte	8	uimsbf
} while (quant_byte >> 7)		
for (i=0; i<wavelet_decomposition_levels; i++)		
if (i!=0 || color!= "u","v") {		
max_bitplane[i]	5	uimsbf
if ((i+1)%4==0)		
marker_bit	1	bslbf
}		
}		
for (i = 0; i<tree_blocks; i++)		
for (color = "y", "u", "v" ,"α")		
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()		
}		
}		
}		
NOTE The value of 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.		

6.2.8.2 TextureSpatialLayerSQ

TextureSpatialLayerSQ() {	No. of bits	Mnemonic
texture_spatial_layer_start_code	32	bslbf
texture_spatial_layer_id	5	uimsbf
TextureSpatialLayerSQNSC()		
}		

© ISO/IEC 2004 – All rights reserved

6.2.8.3 TextureSpatialLayerSQNSC

TextureSpatialLayerSQNSC() {	No. of bits	Mnemonic
for (color="y","u","v","α") {		
if ((first_wavelet_layer && color=="y","α") || (second_wavelet_layer && color=="u","v"))		
do {		
quant_byte	8	uimsbf
} while (quant_byte >> 7)		
if (color =="y","α")		
max_bitplanes	5	uimbsf
else if (!first_wavelet_layer)		
max_bitplanes	5	uimbsf
}		
for (color="y","u","v","α")		
if (color="y","α" || !first_wavelet_layer)		
arith_decode_highbands_bb()		
}		
NOTE The value of first_wavelet_layer becomes "true" when the variable 'i' of subclause 6.2.8.1 TextureLayerSQ() equals to zero. Otherwise, it is "false". The value of second_wavelet_layer becomes "true" when the variable 'i' of sublause 6.2.8.1 TextureLayerSQ() equals to one. Otherwise, it is "false".		

6.2.8.4 TextureSpatialLayerMQ

TextureSpatialLayerMQ() {	No. of bits	Mnemonic
texture_spatial_layer_start_code	32	bslbf
texture_spatial_layer_id	5	uimsbf
snr_scalability_levels	5	uimsbf
do {		
TextureSNRLayerMQ( )		
} while (next_bits() == texture_snr_layer_start_code)		
}		

6.2.8.5 TextureSpatialLayerMQNSC

TextureSpatialLayerMQNSC() {	No. of bits	Mnemonic
snr_scalability_levels	5	uimsbf
for (i =0; i<snr_scalability_levels; i++)		
TextureSNRLayerMQNSC ( )		
}		

6.2.8.6 TextureSNRLayerMQ

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

© ISO/IEC 2004 – All rights reserved

6.2.8.7 TextureSNRLayerMQNSC

TextureSNRLayerMQNSC(){	No. of bits	Mnemonic
if (spatial_scalability_levels == wavelet_decomposition_levels && spatial_layer_id == 0) {		
for (color = "y","α") {		
do {		
quant_byte	8	uimsbf
} while (quant_byte >> 7)		
for (i=0; i<spatial_layers; i++) {		
max_bitplane[i]	5	uimsbf
if ((i+1)%4 == 0)		
marker_bit	1	bslbf
}		
}		
}		
else {		
for (color="y", "u", "v" ,"α") {		
do {		
quant_byte	8	uimsbf
} while (quant_byte >> 7)		
for (i=0; i<spatial_layers; i++) {		
max_bitplane[i]	5	uimsbf
if ((i+1)%4 == 0)		
marker_bit	1	bslbf
}		
}		
}		
if (scan_direction == 0) {		
for (i = 0; i<tree_blocks; i++)		
for (color = "y", "u", "v" ,"α")		
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" ,"α") {		
if (wavelet_decomposition_layer_id != 0 || color != "u", "v")		
arith_decode_highbands_bb()		
}		
}		
}		
}		
NOTE The value of spatial_layers is equivalent to the maximum number of the wavelet decomposition layers in that scalability layer.		

© ISO/IEC 2004 – All rights reserved

6.2.8.8 TextureSpatialLayerBQ

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

6.2.8.9 TextureBitPlaneBQ

TextureBitPlaneBQ () {	No. of bits	Mnemonic
for (color = “y”, “u”, “v” ,”α”)		
if (wavelet_decomposition_layer_id == 0){		
all_nonzero[color]	1	bslbf
if (all_nonzero[color] == 0) {		
all_zero[color]	1	bslbf
if (all_zero[color]==0) {		
lh_zero[color]	1	bslbf
hl_zero[color]	1	bslbf
hh_zero[color]	1	bslbf
}		
}		
}		
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()		
}		
}		
}		
}		

6.2.8.10 TextureSNRLayerBQ

TextureSNRLayerBQ() {	No. of bits	Mnemonic
texture_snr_layer_start_code	32	bslbf
texture_snr_layer_id	5	uimsbf
for (i=0; i<wavelet_decomposition_levels; i++) {		
texture_spatial_layer_start_code	32	bslbf

© ISO/IEC 2004 – All rights reserved

texture_spatial_layer_id	5	uimsbf
TextureBitPlaneBQ()		
next_start_code ( )		
}		
}		

6.2.8.11 DownloadWaveletFilters

download_wavelet_filters() {	No. of bits	Mnemonic
lowpass_filter_length	4	uimsbf
highpass_filter_length	4	uimsbf
do{		
if (wavelet_filter_type == 0) {		
filter_tap_integer	16	imsbf
marker_bit	1	bslbf
} else {		
filter_tap_float_high	16	uimsbf
marker_bit	1	bslbf
filter_tap_float_low	16	uimsbf
marker_bit	1	bslbf
}		
} while (lowpass_filter_length--)		
do{		
if (wavelet_filter_type == 0){		
filter_tap_integer	16	imsbf
marker_bit	1	bslbf
} else {		
filter_tap_float_high	16	uimsbf
marker_bit	1	bslbf
filter_tap_float_low	16	uimsbf
marker_bit	1	bslbf
}		
} while (highpass_filter_length--)		
if (wavelet_filter_type == 0) {		
integer_scale	16	uimsbf
marker_bit	1	bslbf
}		
}		

6.2.8.12 Wavelet dc decode

wavelet_dc_decode() {	No. of bits	Mnemonic
mean	8	uimsbf
do{		
quant_dc_byte	8	uimsbf
} while(quant_dc_byte >>7)		

© ISO/IEC 2004 – All rights reserved

do{		
band_offset_byte	8	uimsbf
} while (band_offset_byte >>7)		
do{		
band_max_byte	8	uimsbf
} while (band_max_byte >>7)		
arith_decode_dc()		
}		

6.2.8.13 Wavelet higher bands decode

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

6.2.8.14 Shape Object Decoding

shape_object_decoding() {	**No. of bits**	**Mnemonic**
change_conv_ratio_disable	1	bslbf
sto_constant_alpha	1	bslbf
if (sto_constant_alpha)		
sto_constant_alpha_value	8	bslbf
if (visual_object_verid != 0001) {		
marker_bit	1	bslbf
for(i=0; i<shape_base_layer_height_blocks(); i++) {		
for(j=0; j<shape_base_layer_width_blocks(); j++) {		
bab_type	1-2	vlclbf
if (bab_type == 4) {		
if (!change_conv_ratio_disable)		
conv_ratio	1-2	vlclbf
scan_type	1	bslbf
binary_arithmetic_decode()		
}		
}		
}		
marker_bit	1	bslbf
if (!start_code_enable) {		

© ISO/IEC 2004 – All rights reserved

sto_shape_coded_layers	4	uimsbf
marker_bit	1	bslbf
for(k = 0; k < sto_shape_coded_layers; k++) {		
for(i=0;i<shape_enhanced_layer_height_blocks(); i++)		
for(j=0; j<shape_enhanced_layer_width_blocks(); j++)		
enh_binary_arithmetic_decode()		
marker_bit	1	bslbf
}		
}		
else {		
next_start_code()		
while (nextbits() == texture_shape_layer_start_code) {		
texture_shape_layer_start_code	32	bslbf
texture_shape_layer_id	5	uimsbf
marker_bit	1	bslbf
for(i=0;i<shape_enhanced_layer_height_blocks(); i++)		
for(j=0; j<shape_enhanced_layer_width_blocks(); j++)		
enh_binary_arithmetic_decode()		
marker_bit	1	bslbf
next_start_code()		
}		
texture_spatial_layer_start_code	32	bslbf
texture_spatial_layer_id	5	uimsbf
marker_bit	1	bslbf
}		
} else { /* version 1 */		
for (i=0 ; i<((object_width+15)/16)*(object_height+15)/16) ; i++) {		
bab_type	1-2	vlclbf
if (bab_type==4) {		
if (!change_conv_ratio_disable)		
conv_ratio	1-2	vlclbf
scan_type	1	bslbf
binary_arithmetic_decode()		
}		
}		
}		

6.2.9 Mesh Object

MeshObject() {	No. of bits	Mnemonic
mesh_object_start_code	32	bslbf
do{		
MeshObjectPlane()		
} while (next_bits_bytealigned() == mesh_object_plane_start_code || next_bits_bytealigned() != '0000 0000 0000 0000 0000 0001')		
}		

© ISO/IEC 2004 – All rights reserved

6.2.9.1 Mesh Object Plane

MeshObjectPlane() {	No. of bits	Mnemonic
MeshObjectPlaneHeader()		
MeshObjectPlaneData()		
}		

MeshObjectPlaneHeader() {	No. of bits	Mnemonic
if (next_bits_bytealigned()=='0000 0000 0000 0000 0000 0001'){		
next_start_code()		
mesh_object_plane_start_code	32	bslbf
}		
is_intra	1	bslbf
mesh_mask	1	bslbf
temporal_header()		
}		

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

6.2.9.2 Mesh geometry

mesh_geometry() {	No. of bits	Mnemonic
mesh_type _code	2	bslbf
if (mesh_type_code == '01') {		
nr_of_mesh_nodes_hor	10	uimsbf
nr_of_mesh_nodes_vert	10	uimsbf
marker_bit	1	uimsbf
mesh_rect_size_hor	8	uimsbf
mesh_rect_size_vert	8	uimsbf
triangle_split_code	2	bslbf
}		
else if (mesh_type_code == '10') {		
nr_of_mesh_nodes	16	uimsbf
marker_bit	1	uimsbf
nr_of_boundary_nodes	10	uimsbf
marker_bit	1	uimsbf
node0_x	13	simsbf
marker_bit	1	uimsbf

© ISO/IEC 2004 – All rights reserved

node0_y	13	simsbf
marker_bit	1	uimsbf
for (n=1; n < nr_of_mesh_nodes; n++) {		
delta_x_len_vlc	2-12	vlclbf
if (delta_x_len_vlc)		
delta_x	1-14	vlclbf
delta_y_len_vlc	2-12	vlclbf
if (delta_y_len_vlc)		
delta_y	1-14	vlclbf
}		
}		
}		

6.2.9.3 Mesh motion

mesh_motion() {	No. of bits	Mnemonic
motion_range_code	3	bslbf
for (n=0; n <nr_of_mesh_nodes; n++) {		
node_motion_vector_flag	1	bslbf
if (node_motion_vector_flag == '0') {		
delta_mv_x_vlc	1-13	vlclbf
if ((motion_range_code != 1) && (delta_mv_x_vlc != 0))		
delta_mv_x_res	1-6	uimsbf
delta_mv_y_vlc	1-13	vlclbf
if ((motion_range_code != 1) && (delta_mv_y_vlc != 0))		
delta_mv_y_res	1-6	uimsbf
}		
}		
}		

6.2.10 FBA Object

fba_object() {	No. of bits	Mnemonic
fba_object_start_code	32	bslbf
do {		
fba_object_plane()		
} while (!((nextbits_bytealigned() == '000 0000 0000 0000 0000 0000') && (nextbits_bytealigned() != fba_object_plane_start_code)))		
}		

© ISO/IEC 2004 – All rights reserved

6.2.10.1 FBA Object Plane

fba_object_plane() {	No. of bits	Mnemonic
fba_object_plane_header()		
fba_object_plane_data()		
}		

fba_object_plane_header() {	No. of bits	Mnemonic
If (nextbits_bytealigned() == '0000 0000 0000 0000 0000 0001'){		
next_start_code()		
fba_object_plane_start_code	32	bslbf
}		
is_intra	1	bslbf
fba_object_mask	2	bslbf
temporal_header()		
}		

fba_object_plane_data() {	No. of bits	Mnemonic
if(fba_object_mask &'01') {		
if(is_intra) {		
fap_quant	5	uimsbf
for (group_number = 1; group_number <= 10; group_number++) {		
marker_bit	1	uimsbf
fap_mask_type	2	bslbf
if(fap_mask_type == '01'|| fap_mask_type == '10')		
fap_group_mask[group_number]	2-16	vlcbf
}		
fba_suggested_gender	1	bslbf
fba_object_coding_type	1	bslbf
if(fba_object_coding_type == 0) {		
is_i_new_max	1	bslbf
is_i_new_min	1	bslbf
is_p_new_max	1	bslbf
is_p_new_min	1	bslbf
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()		
}		

© ISO/IEC 2004 – All rights reserved

}		
}		
if(fba_object_mask &'10') {		
if(is_intra) {		
bap_pred_quant_index	5	uimsbf
for (group_number = 1; group_number <= BAP_NUM_GROUPS; group_number++) {		
marker_bit	1	uimsbf
bap_mask_type	2	bslbf
if(bap_mask_type == '01')		
bap_group_mask[group_number]	3-22	vlcbf
else if (bap_mask_type == '00') {		
for(i=0; i<BAPS_IN_GROUP[group_number];i++) {		
bap_group_mask[group_mask][i] = 0		
}		
}		
else if (bap_mask_type == '11') {		
for(i=0; i<BAPS_IN_GROUP[group_number];i++) {		
bap_group_mask[group_mask][i] = 1		
}		
}		
}		
fba_suggested_gender	1	bslbf
fba_object_coding_type	1	bslbf
if (fba_object_coding_type == 0) {		
bap_is_i_new_max	1	bslbf
bap_is_i_new_min	1	bslbf
bap_is_p_new_max	1	bslbf
bap_is_p_new_min	1	bslbf
decode_bap_new_minmax()		
decode_ibap()		
}		
if(fba_object_coding_type == 1)		
decode_bap_i_segment()		
}		
else {		
if (fba_object_coding_type == 0)		
decode_pbap()		
if(fba_object_coding_type == 1)		
decode_bap_p_segment()		
}		
}		

© ISO/IEC 2004 – All rights reserved

temporal_header() {	No. of bits	Mnemonic
if (is_intra) {		
is_frame_rate	1	bslbf
if(is_frame_rate)		
decode_frame_rate()		
is_time_code	1	bslbf
if (is_time_code)		
time_code	18	bslbf
}		
skip_frames	1	bslbf
if(skip_frames)		
decode_skip_frames()		
}		

6.2.10.2 Decode frame rate and skip frames

decode_frame_rate(){	No. of bits	Mnemonic
frame_rate	8	uimsbf
seconds	4	uimsbf
frequency_offset	1	uimsbf
}		

decode_skip_frames(){	No. of bits	Mnemonic
do{		
number_of_frames_to_skip	4	uimsbf
} while (number_of_frames_to_skip = “1111”)		
}		

6.2.10.3 Decode new minmax

decode_new_minmax() {	No. of bits	Mnemonic
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	uimsbf
if (fap_group_mask[group_number] & (1 <<i))		
i_new_max[j]	5	uimsbf
}		
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	uimsbf
if (fap_group_mask[group_number] & (1 <<i))		

© ISO/IEC 2004 – All rights reserved

i_new_min[j]	5	uimsbf
}		
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	uimsbf
if (fap_group_mask[group_number] & (1 <<i))		
p_new_max[j]	5	uimsbf
}		
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	uimsbf
if (fap_group_mask[group_number] & (1 <<i))		
p_new_min[j]	5	uimsbf
}		
}		
}		

6.2.10.4 Decode ifap

decode_ifap(){	**No. of bits**	**Mnemonic**
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])		
}		
}		
}		
}		
}		

© ISO/IEC 2004 – All rights reserved

6.2.10.5 Decode pfap

decode_pfap(){	No. of bits	Mnemonic
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])		
}		
}		
}		
}		
}		

6.2.10.6 Decode viseme and expression

decode_viseme() {	No. of bits	Mnemonic
aa_decode(viseme_select1Q, viseme_select1_cum_freq)		vlclbf
aa_decode(viseme_select2Q, viseme_select2_cum_freq)		vlclbf
aa_decode(viseme_blendQ, viseme_blend_cum_freq)		vlclbf
viseme_def	1	bslbf
}		

decode_expression() {	No. of bits	Mnemonic
aa_decode(expression_select1Q, expression_select1_cum_freq)		vlclbf
aa_decode(expression_intensity1Q, expression_intensity1_cum_freq)		vlclbf
aa_decode(expression_select2Q, expression_select2_cum_freq)		vlclbf
aa_decode(expression_intensity2Q, expression_intensity2_cum_freq)		vlclbf
init_face	1	bslbf
expression_def	1	bslbf
}		

6.2.10.7 Decode i_segment

decode_i_segment(){	No. of bits	Mnemonic
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)		

© ISO/IEC 2004 – All rights reserved

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])		
}		
}		
}		
}		
}		

6.2.10.8 Decode p_segment

decode_p_segment(){	No. of bits	Mnemonic
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])		
}		
}		
}		
}		
}		

6.2.10.9 Decode viseme and expression

decode_i_viseme_segment(){	No. of bits	Mnemonic
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
}		
}		

© ISO/IEC 2004 – All rights reserved

decode_p _viseme_segment(){	No. of bits	Mnemonic
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	Mnemonic
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	Mnemonic
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	Mnemonic
dc_q	16	simsbf
if(dc_q == -256*128)		
dc_q	31	simsbf
}		

© ISO/IEC 2004 – All rights reserved

decode_p_dc(dc_q_diff) {	No. of bits	Mnemonic
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	Mnemonic
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		
}		
}		
}		

6.2.10.10 Decode bap min max

decode_bap_new_minmax() {	No. of bits	Mnemonic
if (bap_is_i_new_max) {		
for (group_number=1;group_number<= BAP_NUM_GROUPS; group_number++)		
for (i=0; i < NBAP_GROUP[group_number]; i++) {		
j=BAPS_IN_GROUP[group_number][i]		
if (!(i & 0x3))		
marker_bit	1	uimsbf
if (bap_group_mask[group_number] & (1 <<i))		
bap_i_new_max[j]	5	uimsbf
}		
if (bap_is_i_new_min) {		
for (group_number = 1; group_number <= BAP_NUM_GROUPS; group_number++)		

© ISO/IEC 2004 – All rights reserved

for (i=0; i < NBAP_GROUP[group_number]; i++) {		
j=BAPS_IN_GROUP[group_number][i]		
if (!(i & 0x3))		
marker_bit	1	uimsbf
if (bap_group_mask[group_number] & (1 <<i))		
bap_i_new_min[j]	5	uimsbf
}		
if (bap_is_p_new_max) {		
for (group_number = 1; group_number <= BAP_NUM_GROUPS; group_number++)		
for (i=0; i < NBAP_GROUP[group_number]; i++) {		
j=BAPS_IN_GROUP[group_number][i]		
if (!(i & 0x3))		
marker_bit	1	uimsbf
if (bap_group_mask[group_number] & (1 <<i))		
bap_p_new_max[j]	5	uimsbf
}		
if (bap_is_p_new_min) {		
for (group_number = 1; group_number <= BAP_NUM_GROUPS; group_number++)		
for (i=0; i < NBAP_GROUP[group_number]; i++) {		
j=BAPS_IN_GROUP[group_number][i]		
if (!(i & 0x3))		
marker_bit	1	uimsbf
if (bap_group_mask[group_number] & (1 <<i))		
bap_p_new_min[j]	5	uimsbf
}		
}		
}		

6.2.10.11 Decode ibap

decode_ibap(){	**No. of bits**	**Mnemonic**
for (group_number = 1; group_number <= BAP_NUM_GROUPS; group_number++) {		
for (i= 0; i<NBAP_GROUP[group_number]; i++) {		
j=BAPS_IN_GROUP[group_number][i]		
if(bap_group_mask[group_number] & (1 << i)) {		
aa_decode(ibap_Q[j],ibap_cum_freq[j])		
}		
}		
}		
}		

© ISO/IEC 2004 – All rights reserved

6.2.10.12 Decode pbap

decode_pbap() {	No. of bits	Mnemonic
for (group_number = 1; group_number <= BAP_NUM_GROUPS; group_number++) {		
for (i= 0; i<NBAP_GROUP[group_number]; i++) {		
j=BAPS_IN_GROUP[group_number][i]		
if(bap_group_mask[group_number] & (1 << i)) {		
aa_decode(pbap_diff[j], pbap_cum_freq[j])		
}		
}		
}		
}		

6.2.10.13 Decode bap i segment

decode_bap_i_segment(){	No. of bits	Mnemonic
for (group_number= 1; group_number<= BAP_NUM_GROUPS; group_number++) {		
for(i=0; i<NBAP_GROUP[group_number]; i++) {		
if(bap_group_mask[group_number] & (1 << i)) {		
j=BAPS_IN_GROUP[group_number][i]		
decode_i_dc(dc_Q[j])		
decode_ac(ac_Q[j])		
}		
}		
}		
}		

6.2.10.14 Decode bap p segment

decode_bap_p_segment(){	No. of bits	Mnemonic
for (group_number = 1; group_number <= BAP_NUM_GROUPS; group_number++) {		
for (i=0; i<NBAP_GROUP[group_number]; i++) {		
If(bap_group_mask[group_number] & (1 << i)) {		
j = BAPS_IN_GROUP[group_number][i]		
decode_p_dc(dc_Q[j])		
decode_ac(ac_Q[j])		
}		
}		
}		
}		

© ISO/IEC 2004 – All rights reserved

6.2.11 3D Mesh Object

6.2.11.1 3D_Mesh_Object

3D_Mesh_Object () {	No. of bits	Mnemonic
3D_MO_start_code	16	uimsbf
3D_Mesh_Object_Header()		
do {		
3D_Mesh_Object_Layer()		
} while (nextbits_bytealigned() == 3D_MOL_start_code)		
}		

6.2.11.2 3D_Mesh_Object_Header

3D_Mesh_Object_Header() {	No. of bits	Mnemonic
ccw	1	bslbf
convex	1	bslbf
solid	1	bslbf
creaseAngle	6	**uimsbf**
coord_header()		
normal_header()		
color_header()		
texCoord_header()		
ce_SNHC	1	**bslbf**
if (ce_SNHC == '1')		
ce_SNHC _header()		
}		

6.2.11.3 3D_Mesh_Object_Layer

3D_Mesh_Object_Layer () {	No. of bits	Mnemonic
3D_MOL_start_code	16	uimsbf
mol_id	8	uimsbf
if (ce_SNHC == '1') {		
ce_SNHC _n_vertices	24	uimsbf
ce_SNHC _n_triangles	24	uimsbf
ce_SNHC _n_edges	24	uimsbf
}		
if (mol_id == 0)		
3DMeshObject_Base_Layer()		
else		
3DMeshObject_Refinement_Layer()		
}		

© ISO/IEC 2004 – All rights reserved

6.2.11.4 3D_Mesh_Object_Base_Layer

3DmeshObject_Base_Layer()	No. of bits	Mnemonic
do {		
3D_MOBL_start_code	16	uimsbf
mobl_id	8	uimsbf
while (!bytealigned())		
one_bit	1	bslbf
qf_start()		
if (3D_MOBL_start_code == "partition_type_0") {		
do {		
connected_component()		
qf_decode(**last_component**, last_component_context)		vlclbf
} while (last_component == '0')		
}		
else if (3D_MOBL_start_code == "partition_type_1") {		
vg_number=0		
do {		
vertex_graph()		
vg_number++		
qf_decode(**has_stitches**, has_stitches_context)		vlclbf
if (has_stitches == '1')		
stitches()		
qf_decode(**codap_last_vg**, codap_last_vg_context)		vlclbf
} while (codap_last_vg == '0')		
}		
else if (3D_MOBL_start_code == "partition_type_2") {		
if(vg_number > 1)		
qf_decode(**codap_vg_id**)		vlclbf
qf_decode(**codap_left_bloop_idx**)		vlclbf
qf_decode(**codap_right_bloop_idx**)		vlclbf
qf_decode(**codap_bdry_pred**)		vlclbf
triangle_tree()		
triangle_data()		
}		
} while (nextbits_bytealigned() == 3D_MOBL_start_code)		
}		

6.2.11.5 coord_header

coord_header() {	No. of bits	Mnemonic
coord_binding	2	uimsbf
coord_bbox	1	bslbf
if (coord_bbox == '1') {		
coord_xmin	32	bslbf
coord_ymin	32	bslbf
coord_zmin	32	bslbf
coord_size	32	bslbf
}		

© ISO/IEC 2004 – All rights reserved

coord_quant	5	uimsbf
coord_pred_type	2	uimsbf
if (coord_pred_type==”tree_prediction” ‖ coord_pred_type==’”parallelogram_prediction”) {		
coord_nlambda	2	uimsbf
for (i=1; i<coord_nlambda; i++)		
coord_lambda	4-27	simsbf
}		
}		

6.2.11.6 normal_header

normal_header() {	No. of bits	Mnemonic
normal_binding	2	uimsbf
if (normal_binding != “not_bound”) {		
normal_bbox	1	bslbf
normal_quant	5	uimsbf
normal_pred_type	2	uimsbf
if (normal_pred_type==”tree_prediction”‖ normal_pred_type==”parallelogram_prediction”) {		
normal_nlambda	2	uimsbf
for (i=1; i<normal_nlambda; i++)		
normal_lambda	3-17	simsbf
}		
}		
}		

6.2.11.7 color_header

color_header() {	No. of bits	Mnemonic
color_binding	2	uimsbf
if (color_binding != “not_bound”) {		
color_bbox	1	bslbf
if (color_bbox == ‘1’) {		
color_rmin	32	bslbf
color_gmin	32	bslbf
color_bmin	32	bslbf
color_size	32	bslbf
}		
color_quant	5	uimsbf
color_pred_type	2	uimsbf
if (color_pred_type==”tree_prediction” ‖ color_pred_type==’”parallelogram_prediction”) {		
color_nlambda	2	uimsbf
for (i=1; i<color_nlambda; i++)		
color_lambda	4-19	simsbf
}		
}		
}		

© ISO/IEC 2004 – All rights reserved

6.2.11.8 texCoord_header

texCoord_header() {	No. of bits	Mnemonic
texCoord_binding	2	uimsbf
if (texCoord_binding != "not_bound") {		
texCoord_bbox	1	bslbf
if (texCoord_bbox == '1') {		
texCoord_umin	32	bslbf
texCoord_vmin	32	bslbf
texCoord_size	32	bslbf
}		
texCoord_quant	5	uimsbf
texCoord_pred_type	2	uimsbf
if (texCoord_pred_type=="tree_prediction" || texCoord_pred_type=="parallelogram_prediction") {		
texCoord_nlambda	2	uimsbf
for (i=1; i<texCoord_nlambda; i++)		
texCoord_lambda	4-19	simsbf
}		
}		
}		

6.2.11.9 ce_SNHC _header

ce_SNHC_header() {	No. of bits	Mnemonic
ce_SNHC_n_proj_surface_spheres	4	uimsbf
if (ce_SNHC_n_proj_surface_spheres != 0) {		
ce_SNHC_x_coord_center_point	32	bslbf
ce_SNHC_y_coord_center_point	32	bslbf
ce_SNHC_z_coord_center_point	32	bslbf
ce_SNHC_normalized_screen_distance_factor	8	uimsbf
for (i=0; i< ce_SNHC_n_proj_surface_spheres; i++) {		
ce_SNHC_radius	32	bslbf
ce_SNHC_min_proj_surface	32	bslbf
ce_SNHC_n_proj_points	8	uimsbf
for (j=0; j< ce_SNHC _n_proj_points; j++) {		
ce_SNHC_sphere_point_coord	11	uimsbf
ce_SNHC_proj_surface	32	bslbf
}		
}		
}		
}		

© ISO/IEC 2004 – All rights reserved

6.2.11.10 connected_component

connected_component() {	No. of bits	Mnemonic
vertex_graph()		
qf_decode(**has_stitches**, has_stitches_context)		vlclbf
if (has_stitches == '1')		
stitches()		
triangle_tree()		
triangle_data()		
}		

6.2.11.11 vertex_graph

vertex_graph() {	No. of bits	Mnemonic
qf_decode(**vg_simple**, vg_simple_context)		vlclbf
depth = 0		
code_last = '1'		
openloops = 0		
do {		
do {		
if (code_last == '1') {		
qf_decode(**vg_last**, vg_last_context)		vlclbf
if (openloops > 0) {		
qf_decode(**vg_forward_run**, vg_forward_run_context)		vlclbf
if (vg_forward_run == '0') {		
openloops--		
if (openloops > 0)		
qf_decode(**vg_loop_index**, vg_loop_index_context)		vlclbf
break		
}		
}		
}		
qf_decode(**vg_run_length**, vg_run_length_context)		vlclbf
qf_decode(**vg_leaf**, vg_leaf_context)		vlclbf
if (vg_leaf == '1' && vg_simple == '0') {		
qf_decode(**vg_loop**, vg_loop_context)		vlclbf
if (vg_loop == '1')		
openloops++		
}		
} while (0)		
if (vg_leaf == '1' && vg_last == '1' && code_last == '1')		
depth--		
if (vg_leaf == '0' && (vg_last == '0' || code_last == '0'))		
depth++		
code_last = vg_leaf		
} while (depth >= 0)		
}		

© ISO/IEC 2004 – All rights reserved

6.2.11.12 stitches

stitches() {	No. of bits	Mnemonic
for each vertex in connected_component {		
qf_decode(**stitch_cmd**, stitch_cmd_context)		vlclbf
if (stitch_cmd) {		
qf_decode(**stitch_pop_or_get**, stitch_pop_or_get_context)		vlclbf
if (stitch_pop_or_get == '1') {		
qf_decode(**stitch_pop**, stitch_pop_context)		vlclbf
qf_decode(**stitch_stack_index,** stitch_stack_index_context)		vlclbf
qf_decode(**stitch_incr_length**, stitch_incr_length_context)		vlclbf
if (stitch_incr_length != 0)		
qf_decode(**stitch_incr_length_sign**, stitch_incr_length_sign_context)		vlclbf
qf_decode(**stitch_push**, stitch_push_context)		vlclbf
if (total length >0)		
qf_decode(**stitch_reverse**, stitch_reverse_context)		vlclbf
}		
else		
qf_decode(**stitch_length**, stitch_length_context)		vlclbf
}		
}		
}		

6.2.11.13 triangle_tree

triangle_tree() {	No. of bits	Mnemonic
depth = 0		
ntriangles = 0		
branch_position = -2		
do {		
qf_decode(**tt_run_length**, tt_run_length_context)		vlclbf
ntriangles += tt_run_length		
qf_decode(**tt_leaf**, tt_leaf_context)		vlclbf
if (tt_leaf == '1') {		
depth--		
}		
else {		
branch_position = ntriangles		
depth++		
}		
} while (depth >= 0)		
if (3D_MOBL_start_code == "partition_type_2")		
if (codap_right_bloop_idx – codap_left_bloop_idx – 1 > ntriangles) {		
if (branch_position == ntriangles – 2) {		
qf_decode(**codap_branch_len**, codap_branch_len_context)		vlclbf

© ISO/IEC 2004 – All rights reserved

ntriangles -= 2		
}		
else		
ntriangles--		
}		
}		

6.2.11.14 triangle_data

triangle_data(i) {	No. of bits	Mnemonic
qf_decode(**triangulated**, triangulated_context)		vlclbf
depth=0		
root_triangle()		
for (i=1; i<ntriangles; i++)		
triangle(i)		
}		

6.2.11.15 root_triangle

root_triangle() {	No. of bits	Mnemonic
if (marching_triangle)		
qf_decode(**marching_pattern**, marching_pattern_context[**marching_pattern**])		vlclbf
else {		
if (3D_MOBL_start_code == "partition_type_2")		
if (tt_leaf == '0' && depth==0)		
qf_decode(**td_orientation**, td_orientation_context)		vlclbf
if (tt_leaf == '0')		
depth++		
else		
depth--		
}		
if (3D_MOBL_start_code == "partition_type_2")		
if (triangulated == '0')		
qf_decode(**polygon_edge**, polygon_edge_context[**polygon_edge**])		vlclbf
root_coord()		
root_normal()		
root_color()		
root_texCoord()		
}		

© ISO/IEC 2004 – All rights reserved

root_coord() {	No. of bits	Mnemonic
if (3D_MOBL_start_code == “partition_type_2”) {		
if (visited[vertex_index] == 0) {		
root_coord_sample()		
if (visited[vertex_index] == 0) {		
coord_sample()		
coord_sample()		
}		
}		
}		
else {		
root_coord_sample()		
coord_sample()		
coord_sample()		
}		
}		

root_normal() {	No. of bits	Mnemonic
if (normal_binding != “not_bound”)		
if (3D_MOBL_start_code == “partition_type_2”) {		
if (normal_binding != “bound_per_vertex” || visited[vertex_index] == 0) {		
root_normal_sample()		
if (normal_binding != “bound_per_face” && (normal_binding != “bound_per_vertex” || visited[vertex_index] == 0)) {		
normal_sample()		
normal_sample()		
}		
}		
else {		
root_normal_sample()		
if (normal_binding != “bound_per_face”) {		
normal_sample()		
normal_sample()		
}		
}		
}		

root_color() {	No. of bits	Mnemonic
if (color_binding != “not_bound”)		
if (3D_MOBL_start_code == “partition_type_2”) {		
if (color_binding != “bound_per_vertex” || visited[vertex_index] == 0) {		
root_color_sample()		

© ISO/IEC 2004 – All rights reserved

if (color_binding != “bound_per_face” && (color_binding != “bound_per_vertex” || visited[vertex_index] == 0)) {		
color_sample()		
color_sample()		
}		
}		
else {		
root_color_sample()		
if (color_binding != “bound_per_face”) {		
color_sample()		
color_sample()		
}		
}		
}		

root_texCoord() {	No. of bits	Mnemonic
if (texCoord_binding != “not_bound”)		
if (3D_MOBL_start_code == “partition_type_2”) {		
if (texCoord_binding != “bound_per_vertex” || visited[vertex_index] == 0) {		
root_texCoord_sample()		
if (texCoord_binding!= “bound_per_vertex” || visited[vertex_index] == 0) {		
texCoord_sample()		
texCoord_sample()		
}		
}		
else {		
root_texCoord_sample()		
texCoord_sample()		
texCoord_sample()		
}		
}		

6.2.11.16 triangle

triangle(i) {	No. of bits	Mnemonic
if (marching_triangle)		
qf_decode(**marching_edge**, marching_edge_context[**marching_edge**])		vlclbf
else {		
if (3D_MOBL_start_code == “partition_type_2”)		
if (tt_leaf == ‘0’ && depth==0)		
qf_decode(**td_orientation**, td_orientation_context)		vlclbf
if (tt_leaf == ‘0’)		

© ISO/IEC 2004 – All rights reserved

depth++		
else		
depth--		
if (triangulated == '0')		
qf_decode(**polygon_edge**, polygon_edge_context[**polygon_edge**])		vlclbf
coord()		
normal()		
color()		
texCoord()		
}		

coord() {	**No. of bits**	**Mnemonic**
if (3D_MOBL_start_code == "partition_type_2") {		
if (visited[vertex_index] == 0)		
if (no_ancestors)		
root_coord_sample()		
else		
coord_sample()		
}		
else {		
if (visited[vertex_index] == 0)		
coord_sample()		
}		
}		

normal() {	**No. of bits**	**Mnemonic**
if (normal_binding == "bound_per_vertex") {		
if (3D_MOBL_start_code == "partition_type_2") {		
if (visited[vertex_index] == 0)		
if (no_ancestors)		
root_normal_sample()		
else		
normal_sample()		
}		
else {		
if (visited[vertex_index] == 0)		
normal_sample()		
}		
} else if (normal_binding == "bound_per_face") {		
if (triangulated == '1' || polygon_edge == '1')		
normal_sample()		
} else if (normal_binding == "bound_per_corner") {		
if (triangulated == '1' || polygon_edge == '1') {		
normal_sample()		
normal_sample()		

© ISO/IEC 2004 – All rights reserved

}		
normal_sample()		
}		
}		

color() {	No. of bits	Mnemonic
if (color_binding == “bound_per_vertex”) {		
if (3D_MOBL_start_code == “partition_type_2”) {		
if (visited[vertex_index] == 0)		
if (no_ancestors)		
root_color_sample()		
else		
color_sample()		
}		
else {		
if (visited[vertex_index] == 0)		
color_sample()		
}		
} else if (color_binding == “bound_per_face”) {		
if (triangulated == ‘1’ || polygon_edge == ‘1’)		
color_sample()		
} else if (color_binding == “bound_per_corner”) {		
if (triangulated == ‘1’ || polygon_edge == ‘1’) {		
color_sample()		
color_sample()		
}		
color_sample()		
}		
}		

texCoord() {	No. of bits	Mnemonic
if (texCoord_binding == “bound_per_vertex”) {		
if (3D_MOBL_start_code == “partition_type_2”) {		
if (visited[vertex_index] == 0)		
if (no_ancestors)		
root_texCoord_sample()		
else		
texCoord_sample()		
}		
else {		
if (visited[vertex_index] == 0)		
texCoord_sample()		
}		
} else if (texCoord_binding == “bound_per_corner”) {		
if (triangulated == ‘1’ || polygon_edge == ‘1’) {		
texCoord_sample()		
texCoord_sample()		

© ISO/IEC 2004 – All rights reserved

}		
texCoord_sample()		
}		
}		

root_coord_sample() {	No. of bits	Mnemonic
for (i=0; i<3; i++)		
for (j=0; j<coord_quant; j++)		
qf_decode(**coord_bit**, zero_context)		vlclbf
}		

root_normal_sample() {	No. of bits	Mnemonic
for (i=0; i<1; i++)		
for (j=0; j<normal_quant; j++)		
qf_decode(**normal_bit**, zero_context)		vlclbf
}		

root_color_sample() {	No. of bits	Mnemonic
for (i=0; i<3; i++)		
for (j=0; j<color_quant; j++)		
qf_decode(**color_bit**, zero_context)		vlclbf
}		

root_texCoord_sample() {	No. of bits	Mnemonic
for (i=0; i<2; i++)		
for (j=0; j<texCoord_quant; j++)		
qf_decode(**texCoord_bit**, zero_context)		vlclbf
}		

coord_sample() {	No. of bits	Mnemonic
for (i=0; i<3; i++) {		
j=0		
do {		
qf_decode(**coord_leading_bit**, coord_leading_bit_context[3*j+i])		vlclbf
j++		
} while (j<coord_quant && coord_leading_bit == '0')		
if (coord_leading_bit == '1') {		
qf_decode(**coord_sign_bit**, zero_context)		vlclbf
do {		
qf_decode(**coord_trailing_bit**, zero_context)		vlclbf
} while (j<coord_quant)		
}		
}		
}		

© ISO/IEC 2004 – All rights reserved

normal_sample() {	No. of bits	Mnemonic
for (i=0; i<1; i++) {		
j=0		
do {		
qf_decode(**normal_leading_bit**, normal_leading_bit_context[j])		vlclbf
j++		
} while (j<normal_quant && normal_leading_bit == '0')		
if (normal_leading_bit == '1') {		
qf_decode(**normal_sign_bit**, zero_context)		vlclbf
do {		
qf_decode(**normal_trailing_bit**, zero_context)		vlclbf
} while (j<normal_quant)		
}		
}		
}		

color_sample() {	No. of bits	Mnemonic
for (i=0; i<3; i++) {		
j=0		
do {		
qf_decode(**color_leading_bit**, color_leading_bit_context[3*j+i])		vlclbf
j++		
} while (j<color_quant && color_leading_bit == '0')		
if (color_leading_bit == '1') {		
qf_decode(**color_sign_bit**, zero_context)		vlclbf
do {		
qf_decode(**color_trailing_bit**, zero_context)		vlclbf
} while (j<color_quant)		
}		
}		
}		

texCoord_sample() {	No. of bits	Mnemonic
for (i=0; i<2; i++) {		
j=0		
do {		
qf_decode(**texCoord_leading_bit**, texCoord_leading_bit_context[2*j+i])		vlclbf
j++		
} while (j<texCoord_quant && texCoord_leading_bit == '0')		

© ISO/IEC 2004 – All rights reserved

if (texCoord_leading_bit == '1') {		
qf_decode(**texCoord_sign_bit**, zero_context)		vlclbf
do {		
qf_decode(**texCoord_trailing_bit**, zero_context)		vlclbf
} while (j<texCoord_quant)		
}		
}		
}		

6.2.11.17 3DMeshObject_Refinement_Layer

3DMeshObject_Refinement_Layer () {	**No. of bits**	**Mnemonic**
do {		
3D_MORL_start_code	16	uimsbf
morl_id	8	uimsbf
connectivity_update	2	uimsbf
pre_smoothing	1	bslbf
if (pre_smoothing == '1')		
pre_smoothing_parameters()		
post_smoothing	1	bslbf
if (post_smoothing == '1')		
post_smoothing_parameters()		
while (!bytealigned())		
one_bit	1	bslbf
qf_start()		
if (connectivity_update == "fs_update")		
fs_pre_update()		
if (pre_smoothing == '1' || post_smoothing == '1')		
smoothing_constraints()		
/* apply pre smoothing step */		
if (connectivity_update == "fs_update")		
fs_post_update()		
if (connectivity_update != "not_updated")		
qf_decode(**other_update**, zero_context)		vlclbf
if (connectivity_update=="not_updated" || other_update == '1')		
other_property_update()		
/* apply post smoothing step */		
} while (nextbits_bytealigned() == 3D_MORL_start_code)		
}		

6.2.11.17.1 pre_smoothing_parameters

pre_smoothing_parameters() {	**No. of bits**	**Mnemonic**
pre_smoothing_n	8	uimsbf
pre_smoothing_lambda	32	bslbf
pre_smoothing_mu	32	bslbf
}		

© ISO/IEC 2004 – All rights reserved

6.2.11.17.2 post_smoothing_parameters

post_smoothing_parameters() {	No. of bits	Mnemonic
post_smoothing_n	8	uimsbf
post_smoothing_lambda	32	bslbf
post_smoothing_mu	32	bslbf
}		

6.2.11.17.3 fs_pre_update

fs_pre_update() {	No. of bits	Mnemonic
for each connected component {		
forest()		
for each tree in forest {		
triangle_tree()		
/* for each tree loop vertex set visited[vertex_index]='1' */		
triangle_data()		
}		
}		
}		

forest () {	No. of bits	Mnemonic
for each edge in connected component		
if (creates no loop in forest)		
qf_decode(**pfs_forest_edge**, pfs_forest_edge_context)		vlclbf
}		

6.2.11.17.4 smoothing_constraints

smoothing_constraints() {	No. of bits	Mnemonic
qf_decode(**smooth_with_sharp_edges**, zero_context)		vlclbf
if (smooth_with_sharp_edges == '1')		
sharp_edge_marks()		
qf_decode(**smooth_with_fixed_vertices**, zero_context)		vlclbf
if (smooth_with_fixed_vertices == '1')		
fixed_vertex_marks()		
}		

sharp_edge_marks () {	No. of bits	Mnemonic
for each edge		
qf_decode(**smooth_sharp_edge**, smooth_sharp_edge_context)		vlclbf
}		

© ISO/IEC 2004 – All rights reserved

fixed_vertex_marks () {	No. of bits	Mnemonic
for each vertex		
qf_decode(**smooth_fixed_vertex**, smooth_fixed_vertex_context)		vlclbf
}		

6.2.11.17.5 fs_post_update

fs_post_update() {	No. of bits	Mnemonic
for each connected component {		
for each tree in forest		
tree_loop_property_update()		
}		

tree_loop_property_update () {	No. of bits	Mnemonic
loop_coord_update()		
loop_normal_update()		
loop_color_update()		
loop_texCoord_update()		
}		

loop_coord_update () {	No. of bits	Mnemonic
for each tree loop vertex		
coord_sample()		
}		

loop_normal_update () {	No. of bits	Mnemonic
if (normal_binding == “bound_per_vertex”) {		
for each tree loop vertex		
normal_sample()		
}		
else if (normal_binding == “bound_per_face”) {		
for each tree loop face		
normal_sample()		
}		
else if (normal_binding == “bound_per_corner”) {		
for each tree loop corner		
normal_sample()		
}		
}		

© ISO/IEC 2004 – All rights reserved

loop_color_update () {	No. of bits	Mnemonic
if (color_binding == “bound_per_vertex”) {		
for each tree loop vertex		
color_sample()		
}		
else if (color_binding == “bound_per_face”) {		
for each tree loop face		
color_sample()		
}		
else if (color_binding == “bound_per_corner”) {		
for each tree loop corner		
color_sample()		
}		
}		

loop_texCoord_update () {	No. of bits	Mnemonic
if (texCoord_binding == “bound_per_vertex”) {		
for each tree loop vertex		
texCoord_sample()		
}		
else if (texCoord_binding == “bound_per_corner”) {		
for each tree loop corner		
texCoord_sample()		
}		
}		

6.2.11.17.6 other_property_update

other_property_update() {	No. of bits	Mnemonic
other_coord_update()		
other_normal_update()		
other_color_update()		
other_texCoord_update()		
}		

other_coord_update () {	No. of bits	Mnemonic
for each vertex in mesh		
if (vertex is not a tree loop vertex)		
coord_sample()		
}		

© ISO/IEC 2004 – All rights reserved

other_normal_update () {	No. of bits	Mnemonic
if (normal_binding == “bound_per_vertex”) {		
for each vertex in mesh		
if (vertex is not a tree loop vertex)		
normal_sample()		
}		
else if (normal_binding == “bound_per_face”) {		
for each face in mesh		
if (face is not a tree loop face)		
normal_sample()		
}		
else if (normal_binding == “bound_per_corner”) {		
for each corner in mesh		
if (corner is not a tree loop corner)		
normal_sample()		
}		
}		

other_color_update () {	No. of bits	Mnemonic
if (color_binding == “bound_per_vertex”) {		
for each vertex in mesh		
if (vertex is not a tree loop vertex)		
color_sample()		
}		
else if (color_binding == “bound_per_face”) {		
for each face in mesh		
if (face is not a tree loop face)		
color_sample()		
}		
else if (color_binding == “bound_per_corner”) {		
for each corner in mesh		
if (corner is not a tree loop corner)		
color_sample()		
}		
}		

other_texCoord_update () {	No. of bits	Mnemonic
if (texCoord_binding == “bound_per_vertex”) {		
for each vertex in tree loop		
if (vertex is not a tree loop vertex)		
texCoord_sample()		
}		
else if (texCoord_binding == “bound_per_corner”) {		
for each corner in mesh		

© ISO/IEC 2004 – All rights reserved

if (corner is not a tree loop corner)		
texCoord_sample()		
}		
}		

6.2.12 Upstream message

6.2.12.1 upstream_message

upstream_message() {	No. of bits	Mnemonic
upstream_message_type	3	bslbf
If (upstream_message_type == Video_NEWPRED)		
upstream_Video_NEWPRED()		
if (upstream_message_type == SNHC_QoS)		
upstream_SNHC_QoS()		
byte_align_for_upstream()		
}		

6.2.12.2 upstream_Video_NEWPRED

upstream_Video_NEWPRED() {	No. of bits	Mnemonic
newpred_upstream_message_type	2	bslbf
if (newpred_upstream_message_type == "NP_NACK")		
unreliable_flag	1	bslbf
vop_id	4-15	uimsbf
macroblock_number	1-14	uimsbf
if (newpred_upstream_message_type == "Intra Refresh Command")		
end_macroblock_number	1-14	uimsbf
if (newpred_upstream_message_type == "NP_NACK")		
requested_vop_id_for_prediction	4-15	uimsbf
}		

6.2.12.3 upstream_SNHC_QoS

upstream_SNHC_QoS() {	No. of bits	Mnemonic
screen_width	12	uimsbf
screen_height	12	uimsbf
n_rendering_modes	4	uimsbf
for (rendering_mode = 0 ; rendering_mode < n_rendering_modes ; rendering_mode++) {		
rendering_mode_type	4	uimsbf
n_curves	8	uimsbf
for (i = 0 ; i < n_curves ; i++) {		
triangle_parameter	24	uimsbf
n_points_on_curve	8	uimsbf

© ISO/IEC 2004 – All rights reserved

for (j = 0 ; j < n_points_on_curve ; j++) {		
screen_coverage_parameter	8	uimsbf
frame_rate_value	12	bslbf
}		
}		
}		
}		

6.2.13 Studio Video Object

6.2.13.1 Studio Visual Object

StudioVisualObject() {	No. of bits	Mnemonic
visual_object_start_code	32	bslbf
visual_object_verid	4	uimsbf
visual_object_type	4	uimsbf
next_start_code_studio()		
extension_and_user_data(1)		
if (visual_object_type == "video ID") {		
video_object_start_code	32	bslbf
StudioVideoObjectLayer()		
} else {		
/* Other visual object types are not supported in StudioVisualObject() */		
}		
}		

6.2.13.2 Extension and user data

extension_and_user_data(i) {	No. of bits	Mnemonic
while ((next_bits() == extension_start_code) ||		
(next_bits() == user_data_start_code)) {		
if ((i==2 || i==4) && (next_bits() == extension_start_code))		
extension_data(i)		
if (next_bits() == user_data_start_code)		
user_data_studio()		
}		
}		

© ISO/IEC 2004 – All rights reserved

6.2.13.2.1 Extension data

extension_data(i) {	No. of bits	Mnemonic
while (next_bits()== extension_start_code) {		
extension_start_code	32	bslbf
/* NOTE - i never takes the value 0 because extension_data() is never called in a VisualObjectSequence() */		
/* NOTE - i never takes the value 1 because extension_data() is never called in a StudioVisualObject() */		
if (i == 2) { /* Called in StudioVideoObjectLayer() */		
if (next_bits()== "Sequence Display Extension ID")		
sequence_display_extension()		
else if (next_bits() == "Quant Matrix Extension ID")		
quant_matrix_extension()		
else if (nextbits() == "VLC Code Extension ID")		
vlc_code_extension()		
}		
/* NOTE - i never takes the value 3 because extension_data() is never called in a Group_of_StudioVideoObjectPlane() */		
if (i == 4) { /* Called in VideoObjectPlane() */		
if (nextbits() == "Quant Matrix Extension ID")		
quant_matrix_extension()		
else if (nextbits() == "Copyright Extension ID")		
copyright_extension()		
else if (nextbits() == "Picture Display Extension ID")		
picture_display_extension()		
else if(nextbits() == "Camera Prameters Extension ID")		
camera_parameters_extension()		
else if (nextbits() == "ITU-T Extension ID")		
ITU-T_extension()		
else if (nextbits() == "VLC Code Extension ID")		
vlc_code_extension()		
}		
}		

6.2.13.2.2 User data Studio

user_data_studio() {	No. of bits	Mnemonic
user_data_start_code	32	bslbf
while(next_bits() != '0000 0000 0000 0000 0000 0001') {		
user_data	8	uimsbf
}		
next_start_code_studio()		
}		

© ISO/IEC 2004 – All rights reserved

6.2.13.2.3 Sequence display extension

sequence_display_extension() {	No. of bits	Mnemonic
extension_start_code_identifier	4	uimsbf
video_format	3	uimsbf
video_range	1	bslbf
colour_description	1	uimsbf
if (colour_description) {		
colour_primaries	8	uimsbf
transfer_characteristics	8	uimsbf
matrix_coefficients	8	uimsbf
}		
display_horizontal_size	14	uimsbf
marker_bit	1	bslbf
display_vertical_size	14	uimsbf
next_start_code_studio()		
}		

6.2.13.2.4 Quant matrix extension

quant_matrix_extension() {	No. of bits	Mnemonic
extension_start_code_identifier	4	uimsbf
load_intra_quantiser_matrix	1	uimsbf
if (load_intra_quantiser_matrix)		
intra_quantiser_matrix[64]	8 * 64	uimsbf
load_non_intra_quantiser_matrix	1	uimsbf
if (load_non_intra_quantiser_matrix)		
non_intra_quantiser_matrix[64]	8 * 64	uimsbf
load_chroma_intra_quantiser_matrix	1	uimsbf
if (load_chroma_intra_quantiser_matrix)		
chroma_intra_quantiser_matrix[64]	8 * 64	uimsbf
load_chroma_non_intra_quantiser_matrix	1	uimsbf
if (load_chroma_non_intra_quantiser_matrix)		
chroma_non_intra_quantiser_matrix[64]	8 * 64	uimsbf
if (video_object_layer_shape == 'grayscale') {		
for(i=0; i<aux_comp_count; i++) {		
load_intra_quantiser_matrix_grayscale[i]	1	uimsbf
if (load_intra_quantiser_matrix_grayscale[i])		
intra_quantiser_matrix_grayscale [i][64]	8 * 64	uimsbf
load_non_intra_quantiser_matrix_grayscale[i]	1	uimsbf
if (load_non_intra_quantiser_matrix_grayscale[i])		
non_intra_quantiser_matrix_grayscale [i][64]	8 * 64	uimsbf
}		
}		
next_start_code_studio()		
}		

© ISO/IEC 2004 – All rights reserved

6.2.13.2.5 Picture display extension

picture_display_extension() {	No. of bits	Mnemonic
extension_start_code_identifier	4	uimsbf
for (i=0; i<number_of_frame_centre_offsets; i++) {		
frame_centre_horizontal_offset	16	simsbf
marker_bit	1	bslbf
frame_centre_vertical_offset	16	simsbf
marker_bit	1	bslbf
}		
next_start_code_studio()		
}		

6.2.13.2.6 Copyright extension

copyright_extension() {	No. of bits	Mnemonic
extension_start_code_identifier	4	uimsbf
copyright_flag	1	uimsbf
copyright_identifier	8	uimsbf
original_or_copy	1	uimsbf
reserved	7	bslbf
marker_bit	1	bslbf
copyright_number_1	20	uimsbf
marker_bit	1	bslbf
copyright_number_2	22	uimsbf
marker_bit	1	bslbf
copyright_number_3	22	uimsbf
next_start_code_studio()		
}		

6.2.13.2.7 Camera Parameters extension

camera_parameters_extension() {	No. of bits	Mnemonic
extension_start_code_identifier	4	uimsbf
reserved	1	uimsbf
camera_id	7	simsbf
marker_bit	1	bslbf
height_of_image_device	22	uimsbf
marker_bit	1	bslbf
focal_length	22	uimsbf
marker_bit	1	bslbf
f_number	22	uimsbf
marker_bit	1	bslbf
vertical_angle_of_view	22	uimsbf
marker_bit	1	bslbf
camera_position_x_upper	16	simsbf
marker_bit	1	bslbf

© ISO/IEC 2004 – All rights reserved

camera_position_x_lower	16	simsbf
marker_bit	1	bslbf
camera_position_y_upper	16	simsbf
marker_bit	1	bslbf
camera_position_y_lower	16	simsbf
marker_bit	1	bslbf
camera_position_z_upper	16	simsbf
marker_bit	1	bslbf
camera_position_z_lower	16	simsbf
marker_bit	1	bslbf
camera_direction_x	22	simsbf
marker_bit	1	bslbf
camera_direction_y	22	simsbf
marker_bit	1	bslbf
camera_direction_z	22	simsbf
marker_bit	1	bslbf
image_plane_vertical_x	22	simsbf
marker_bit	1	bslbf
image_plane_vertical_y	22	simsbf
marker_bit	1	bslbf
image_plane_vertical_z	22	simsbf
marker_bit	1	bslbf
reserved	32	bslbf
next_start_code_studio()		
}		

6.2.13.2.8 ITU-T extension

ITU-T extension () {	**No. of bits**	**Mnemonic**
extension_start_code_identifier	4	uimsbf
while (nextbits() != '0000 0000 0000 0000 0000 0001') {		
ITU-T_data	1	bslbf
}		
next_start_code_studio()		
}		
NOTE – The construct with the while-statement prevents start code emulation.		

6.2.13.2.9 VLC code extension

vlc_code_extension() {	**No. of bits**	**Mnemonic**
extension_start_code_identifier	4	uimsbf
load_vlc_code	4	uimsbf
if (load_intra_vlc_code == 1) {		
for (j=0; j< 12; j++) {		
for (i=0; i< 22; i++) {		
Intra_vlc_length[j][i]	4	uimsbf

© ISO/IEC 2004 – All rights reserved

Intra_vlc_code[j][i]	16	uimsbf
}		
}		
}		
if (load_inter_vlc_code == 1) {		
for (j=0; j< 12; j++) {		
for (i=0; i< 22; i++) {		
inter_vlc_length[j][i]	4	uimsbf
inter_vlc_code[j][i]	16	uimsbf
}		
}		
}		
next_start_code_studio()		
}		

6.2.13.3 Studio Video Object Layer

StudioVideoObjectLayer() {	No. of bits	Mnemonic
video_object_layer_start_code	32	bslbf
random_accessible_vol	1	bslbf
video_object_type_indication	8	uimsbf
video_object_layer_verid	4	uimsbf
video_object_layer_shape	2	uimsbf
video_object_layer_shape_extension	4	uimsbf
progressive_sequence	1	bslbf
if (video_object_layer_shape != “binary only”) {		
rgb_components	1	uimsbf
chroma_format	2	uimsbf
bits_per_pixel	4	uimsbf
}		
if (video_object_layer_shape == “rectangular”) {		
marker_bit	1	bslbf
video_object_layer_width	14	uimsbf
marker_bit	1	bslbf
video_object_layer_height	14	uimsbf
marker_bit	1	bslbf
}		
aspect_ratio_info	4	uimsbf
if (aspect_ratio_info == “extended_PAR”) {		
par_width	8	uimsbf
par_height	8	uimsbf
}		
frame_rate_code	4	uimsbf
first_half_bit_rate	15	uimsbf
marker_bit	1	bslbf
latter_half_bit_rate	15	uimsbf
marker_bit	1	bslbf

© ISO/IEC 2004 – All rights reserved

first_half_vbv_buffer_size	15	uimsbf
marker_bit	1	bslbf
latter_half_vbv_buffer_size	3	uimsbf
first_half_vbv_occupancy	11	uimsbf
marker_bit	1	blsbf
latter_half_vbv_occupancy	15	uimsbf
marker_bit	1	blsbf
low_delay	1	uimsbf
mpeg2_stream	1	uimsbf
if (video_object_layer_shape == "grayscale") {		
for(i=0; i<aux_comp_count; i++) {		
alpha_bits_per_pixel[i]	4	uimsbf
minimum_alpha_level[i]	12	uimsbf
marker_bit	1	blsbf
maximum_alpha_level[i]	12	uimsbf
}		
composition_method	1	bslbf
linear_composition	1	bslbf
}		
if (video_object_layer_shape != "binary only") {		
sprite_enable	1	bslbf
if (sprite_enable) {		
sprite_width	20	uimsbf
marker_bit	1	bslbf
sprite_height	20	uimsbf
marker_bit	1	bslbf
sprite_left_coordinate	20	uimsbf
marker_bit	1	bslbf
sprite_top_coordinate	20	uimsbf
marker_bit	1	bslbf
no_of_sprite_warping_points	6	uimsbf
sprite_warping_accuracy	2	uimsbf
sprite_brightness_change	1	bslbf
sprite_defocusing	1	bslbf
sprite_lens_distortion	1	bslbf
}		
}		
next_start_code_studio()		
extension_and_user_data(2)		
if (sprite_enable)		
StudioVideoObjectPlane()		
do {		
if (next_bits() == group_of_vop_start_code)		
Group_of_StudioVideoObjectPlane()		
StudioVideoObjectPlane()		
} while ((next_bits()== group_of_vop_start_code) || (next_bits() == vop_start_code))		
}		

© ISO/IEC 2004 – All rights reserved

6.2.13.4 Group of Studio Video Object Plane

Group_of_StudioVideoObjectPlane() {	No. of bits	Mnemonic
group_vop_start_codes	32	bslbf
time_code_smpte12m()		
closed_gov	1	bslbf
broken_link	1	bslbf
next_start_code_studio()		
extension_and_user_data(3)		
}		

6.2.13.4.1 Time Code SMPTE12M

time_code_smpte12m(){	Bits	Mnemonic
Time_code[63..48]	16	uimsbf
marker_bit	1	bslbf
Time_code[47..32]	16	uimsbf
marker_bit	1	bslbf
Time_code[31..16]	16	uimsbf
marker_bit	1	bslbf
Time_code[15..0]	16	uimsbf
marker_bit	1	bslbf
reserved_bits	4	bslbf
}		

6.2.13.5 Studio Video Object Plane

StudioVideoObjectPlane() {	No. of bits	Mnemonic
vop_start_code	32	bslbf
time_code_smpte12m()		
temporal_reference	10	bslbf
vop_structure	2	uimsbf
vop_coding_type	2	uimsbf
vop_coded	1	bslbf
if (vop_coded == '0') {		
next_start_code_studio()		
extension_and_user_data(4)		
return()		
}		
if (video_object_layer_shape != "rectangular") {		
if (!(sprite_enable && vop_coding_type == "I")) {		
vop_width	14	uimsbf
marker_bit	1	bslbf
vop_height	14	uimsbf
marker_bit	1	bslbf
vop_horizontal_mc_spatial_ref	14	simsbf
marker_bit	1	bslbf

© ISO/IEC 2004 – All rights reserved

vop_vertical_mc_spatial_ref	14	simsbf
marker_bit	1	bslbf
}		
}		
top_field_first	1	bslbf
repeat_first_field	1	bslbf
progressive_frame	1	bslbf
if (vop_coding_type == "I")		
intra_predictors_reset	1	bslbf
}		
if (video_object_layer_shape != “binary only”) {		
alternate_scan	1	bslbf
frame_pred_frame_dct	1	bslbf
dct_precision	2	uimsbf
intra_dc_precision	2	uimsbf
q_scale_type	1	bslbf
if ((vop_coding_type != "I")		
&& !(sprite_enable && vop_coding_type == "S")) {		
vop_fcode[0][0]	4	uimsbf
vop_fcode[0][1]	4	uimsbf
vop_fcode[1][0]	4	uimsbf
vop_fcode[1][1]	4	uimsbf
dead_zone_disable	1	bslbf
}		
}		
if (video_object_layer_shape == “grayscale”) {		
marker_bit	1	bslbf
for(i=0; i<aux_comp_count; i++) {		
alpha_dct_precision[i]	2	uimsbf
alpha_intra_dc_precision[i]	2	uimsbf
alpha_q_scale_type[i]	1	bslbf
}		
}		
if (sprite_enable && vop_coding_type == “S”) {		
if (no_sprite_points > 0)		
sprite_trajectory()		
if (sprite_brightness_change) {		
vop_sprite_brightness_change	1	bslbf
if (vop_sprite_brightness_change)		
brightness_change_factor()		
}		
if (sprite_defocusing)		
defocusing_control()		
if (sprite_lens_distortion)		
lens_distortion_parameter()		
next_start_code_studio()		
return()		
}		

© ISO/IEC 2004 – All rights reserved

composite_display_flag	1	bslbf
if (composite_display_flag) {		
v_axis	1	bslbf
field_sequence	3	uimsbf
sub_carrier	1	bslbf
burst_amplitude	7	uimsbf
sub_carrier_phase	8	uimsbf
}		
while (nextbits() == '1') {		
extra_bit_picture /* with the value '1' */	1	uimsbf
extra_information_picture	8	uimsbf
}		
extra_bit_picture /* with the value '0' */	1	uimsbf
next_start_code_studio()		
extension_and_user_data(4)		
do {		
StudioSlice()		
} while (nextbits() == slice_start_code)		
}		

6.2.13.6 Studio sprite coding

sprite_trajectory() {	No. of bits	Mnemonic
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	Mnemonic
dmv_length	2-19	uimsbf
if (dmv_length != '00')		
dmv_code	1-21	uimsbf
marker_bit	1	bslbf
}		

brightness_change_factor() {	No. of bits	Mnemonic
brightness_change_factor_size	1-4	uimsbf
brightness_change_factor_code	5-10	uimsbf
}		

defocusing_control() {	No. of bits	Mnemonic
marker_bit	1	bslbf
defocusing_control_parameter	9-12	uimsbf
}		

© ISO/IEC 2004 – All rights reserved

lens_distortion_parameter() {	No. of bits	Mnemonic
marker_bit	1	bslbf
lens_distortion_parameter_1	16	bslbf
marker_bit	1	bslbf
lens_distortion_parameter_2	16	bslbf
marker_bit	1	bslbf
lens_center_horizontal	14	bslbf
marker_bit	1	bslbf
lens_center_vertical	14	bslbf
}		

6.2.13.7 Studio Slice

StudioSlice() {	No. of bits	Mnemonic
slice_start_code	32	uimsbf
macroblock_number	1-14	vlclbf
if (video_object_layer_shape != “binary only”)		
quantiser_scale_code	5	uimsbf
if(video_object_layer_shape==”grayscale”)		
for(i=0; i<aux_comp_count; i++)		
alpha_quantiser_scale_code[i]	5	uimsbf
if (nextbits() == 1) {		
slice_extension_flag	1	bslbf
intra_slice	1	uimsbf
slice_VOP_id_enable	1	uimsbf
slice_VOP_id	6	uimsbf
while (nextbits() == 1) {		
extra_bit_slice	1	uimsbf
extra_information_slice	8	uimsbf
}		
}		
extra_bit_slice	1	uimsbf
do {		
StudioMacroblock()		
} while (next_bits() != ‘000 0000 0000 0000 0000 0000’)		
next_start_code_studio()		
}		

6.2.13.8 Studio Macroblock

StudioMacroblock() {	No. of bits	Mnemonic
if (vop_coding_type != “B”) {		
if (video_object_layer_shape != “rectangular”)		
Studio_mb_binary_shape_coding ()		
if (video_object_layer_shape != “binary only”) {		

© ISO/IEC 2004 – All rights reserved

if (!transparent_mb()) {		
if (vop_coding_type != “I”)		
not_coded	1	bslbf
if (vop_coding_type == “I” || !not_coded) {		
compression_mode	1	bslbf
if (compression_mode == "DCT") { /* DCT */		
StudioMacroblock_modes()		
if (macroblock_quant)		
quantiser_scale_code	5	uimsbf
if (macroblock_motion_forward)		
motion_vectors(0)		
if (macroblock_pattern)		
coded_block_pattern()		
for (i = 0; i < block_count; i++)		
if(!transparent_block(i))		
StudioBlock(i)		
} /* -end- DCT */		
else { /* DPCM */		
marker_bit	1	bslbf
dpcm_scan_order	1	bslbf
StudioDPCMBlock() // Y or G		
StudioDPCMBlock() // U or B		
StudioDPCMBlock() // V or R		
} /* -end- DPCM */		
}		
}		
}		
}		
if(video_object_layer_shape==“grayscale” && !transparent_mb()) {		
for (j=0 ; j<aux_comp_count ; j++) {		
if(macroblock_intra || compression_mode == "DPCM") {		
coda_i[j]	1	bslbf
if(coda_i[j]==”coded”) {		
alpha_compression_mode[j]	1	bslbf
if(alpha_compression_mode[j]== "DCT") {		
alpha_macroblock_quant[j]	1	bslbf
if (alpha_macroblock_quant[j])		
alpha_quantiser_scale_code[j]	5	uimsbf
for(i=0;i<alpha_block_count;i++)		
if(!transparent_block(i))		
StudioAlphaBlock(j,i)		
} /* -end- DCT */		
else { /* DPCM */		
marker_bit	1	bslbf
alpha_dpcm_scan_order[j]	1	bslbf
StudioDPCMBlock() // alpha block		

© ISO/IEC 2004 – All rights reserved

} /* -end- DPCM */		
}		
} else { /* inter macroblock */		
coda_pb[j]	1-2	vlclbf
if(coda_pb[j]=="coded"){		
alpha_compression_mode[j]	1	bslbf
if(alpha_compression_mode[j]== "DCT") {		
cbpa[j]	1-6	vlclbf
if (cbpa[j] != 'no DCT coeff') {		
alpha_macroblock_quant[j]	1	bslbf
if (alpha_macroblock_quant[j])		
alpha_quantiser_scale_code[j]	5	uimsbf
}		
for(i=0 ;i<alpha_block_count ;i++)		
if(!transparent_block(i))		
StudioAlphaBlock(j,i)		
} else {		
marker_bit	1	bslbf
alpha_dpcm_scan_order[j]	1	bslbf
StudioDPCMBlock(j) // Alpha		
}		
}		
} /* -end- inter */		
} /* -roop- aux_comp_count */		
} /* -end- grayscale */		
}		
NOTE The value of block_count is 6 in the 4:2:0 format. The value of alpha_block_count is 4.		

6.2.13.8.1 Studio MB Binary Shape Coding

Studio_mb_binary_shape_coding () {	No. of bits	Mnemonic
bab_type	1-6	vlclbf
if (vop_coding_type == 'P'){		
if (bab_type == 1){		
mvs_x	1-18	vlclbf
mvs_y	1-18	vlclbf
}		
}		
if (bab_type == 4) {		
inferior_symbol_macroblock	1	bslbf
cbbp	3-7	vlclbf
for (i=0: i<4; i++){		
if (coded_binary_block_pattern[i]=='1'){		
inferior_symbol_block	1	bslbf
scan_direction	1	bslbf
backward_load_flag	1	bslbf
clp	3-24	vlclbf

© ISO/IEC 2004 – All rights reserved

marker_bit	1	blsbf
for (j=0; j<8; j++){		
if (coded_line_pattern[j]=='1') {		
lbp	1-23	vlclbf
marker_bit	1	blsbf
}		
}		
}		
}		
}		

6.2.13.8.2 Studio Macroblock modes

StudioMacroblock_modes() {	No. of bits	Mnemonic
macroblock_type	1-9	vlclbf
if (macroblock_motion_forward) {		
if (vop_structure == 'frame') {		
if (frame_pred_frame_dct == 0)		
frame_motion_type	2	uimsbf
} else {		
field_motion_type	2	uimsbf
}		
}		
if ((vop_structure == "Frame picture") &&		
(frame_pred_frame_dct == 0) &&		
(macroblock_intra || macoblock_pattern)){		
dct_type	1	uimsbf
}		
}		

6.2.13.8.3 Motion vectors

motion_vectors (s) {	No. of bits	Mnemonic
if (motion_vector_count == 1) {		
if (mv_format == field)		
motion_vertical_field_select[0][s]	1	uimsbf
motion_vector(0, s)		
} else {		
motion_vertical_field_select[0][s]	1	uimsbf
motion_vector(0, s)		
motion_vertical_field_select[1][s]	1	uimsbf
motion_vector(1, s)		
}		
}		

© ISO/IEC 2004 – All rights reserved

6.2.13.8.4 Motion vector

motion_vector (r, s) {	No. of bits	Mnemonic
motion_code[r][s][0]	1-11	vlclbf
if ((vop_fcode[s][0] != 1) && (motion_code[r][s][0] != 0))		
motion_residual[r][s][0]	1-8	uimsbf
motion_code[r][s][1]	1-11	vlclbf
if ((vop_fcode[s][1] != 1) && (motion_code[r][s][1] != 0))		
motion_residual[r][s][1]	1-8	uimsbf
}		

6.2.13.8.5 Coded block pattern

coded_block_pattern () {	No. of bits	Mnemonic
coded_block_pattern_420	3-9	vlclbf
if (chroma_format == 4:2:2)		
coded_block_pattern_1	2	uimsbf
if (chroma_format == 4:4:4)		
coded_block_pattern_2	6	uimsbf
}		

6.2.13.9 Studio Block

The detailed syntax for the term “DCT coefficient” is fully described in Clause 7.

StudioBlock(i) {	No. of bits	Mnemonic
if (pattern_code[i]) {		
if (macroblock_intra) {		
if (i<4) {		
dct_dc_size_luminance	2-13	vlclbf
if(dct_dc_size_luminance != 0)		
dct_dc_differential	1-18	vlclbf
if (dct_dc_size_luminance > 8)		
marker_bit	1	bslbf
} else {		
dct_dc_size_chrominance	2-13	vlclbf
if(dct_dc_size_chrominance !=0)		
dct_dc_differential	1-18	vlclbf
if (dct_dc_size_chrominance > 8)		
marker_bit	1	bslbf
}		
} else {		
First DCT coefficient	1-32	vlcfbf
}		
while (next_bits() != End of block)		
Subsequent DCT coefficients	1-32	vlclbf
End of block	1-16	vlcfbf
}		
}		

© ISO/IEC 2004 – All rights reserved

6.2.13.9.1 Studio Alpha Block

The syntax for DCT coefficient decoding is the same as for block(i) in 6.2.13.9.

StudioAlphaBlock(j, i) {	No. of bits	Mnemonic
last = 0		
if (alpha_pattern_code[j][i]) {		
if (macroblock_intra || compression_mode == "DPCM") {		
dct_dc_size_alpha	2-13	vlclbf
if(dct_dc_size_alpha != 0)		
dct_dc_differential	1-18	vlclbf
if (dct_dc_size_alpha > 8)		
marker_bit	1	bslbf
} else {		
First DCT coefficient	1-32	vlcfbf
}		
while (next_bits() != End of block)		
Subsequent DCT coefficients	1-32	vlclbf
End of block	1-16	vlcfbf
}		
}		

6.2.13.10 Studio DPCM Block

StudioDPCMBlock() {	No. of bits	Mnemonic
block_mean	8-12	uimsbf
rice_parameter	4	uimsbf
for(i=0 ;i<8 or 16 ; i++) {		
for(j=0 ;j<8 or 16;j++) {		
rice_prefix_code	1-12	vlclbf
if (**rice_prefix_code** == '0000 0000 0001')		
dpcm_residual	4-12	uimsbf
else		
rice_suffix_code	0-11	uimsbf
}		
}		
}		

6.2.14 FGS Video Object

6.2.14.1 FGS Video Object Plane

FGSVideoObjectPlane() {	No. of bits	Mnemonic
fgs_vop_start_code	32	bslbf
fgs_vop_coding_type	2	uimsbf
do {		

© ISO/IEC 2004 – All rights reserved

modulo_time_base	1	bslbf
} while (modulo_time_base != '0')		
marker_bit	1	bslbf
vop_time_increment	1-16	uimsbf
marker_bit	1	bslbf
fgs_vop_max_level_y	5	uimsbf
fgs_vop_max_level_u	5	uimsbf
fgs_vop_max_level_v	5	uimsbf
marker_bit	1	bslbf
fgs_vop_number_of_vop_bp_coded	5	uimsbf
fgs_vop_mc_bit_plane_used	5	uimsbf
fgs_vop_selective_enhancement_enable	1	bslbf
if (fgs_vop_coding_type != "I") {		
if (interlaced)		
top_field_first	1	bslbf
vop_fcode_forward	3	uimsbf
if (fgs_vop_coding_type == "B")		
vop_fcode_backward	3	uimsbf
fgs_ref_select_code	2	uimsbf
do {		
fgs_motion_macroblock()		
} while(nextbits_bytealigned() != '000 0000 0000 0000 0000 0000')		
}		
next_start_code()		
if (nextbits_bytealigned () == fgs_bp_start_code) {		
while(nextbits_bytealigned() != '000 0000 0000 0000 0000 0000' || nextbits_bytealigned () == fgs_bp_start_code) {		
if (start_of_bit_plane())		
fgs_bp_start_code	32	bslbf
else {		
if (! fgs_resync_marker_disable && nextbits_bytealigned () == fgs_resync_marker) {		
next_resync_marker()		
fgs_resync_marker	23	uimsbf
fgs_ macroblock_number	1-14	vlclbf
}		
}		
fgs_macroblock()		
}		
next_start_code()		
}		
}		

© ISO/IEC 2004 – All rights reserved

6.2.14.2 FGS Motion Macroblock

fgs_motion_macroblock() {	No. of bits	Mnemonic
if (fgs_vop_coding_type == "P") {		
fgs_not_coded	1	bslbf
if (!fgs_not_coded) {		
fgs_p_mb_type	1	bslbf
if (interlaced)		
fgs_motion_interlaced_information()		
if (fgs_p_mb_type == 0) {		
fgs_motion_vector("forward")		
if (interlaced && field_prediction)		
fgs_motion_vector("forward")		
}		
if (fgs_p_mb_type == 1) {		
for (j=0; j < 4; j++)		
fgs_motion_vector("forward")		
}		
}		
} else {		
fgs_modb	1	bslbf
if (!fgs_modb) {		
fgs_b_mb_type	1-4	vlclbf
if (interlaced)		
fgs_motion_interlaced_information()		
if (fgs_b_mb_type=='0001' || fgs_b_mb_type=='01') {		
fgs_motion_vector("forward")		
if (interlaced && field_prediction)		
fgs_motion_vector("forward")		
}		
if (fgs_b_mb_type == '01' || fgs_b_mb_type == '001') {		
fgs_motion_vector("backward")		
if (interlaced && field_prediction)		
fgs_motion_vector("backward")		
}		
if (fgs_b_mb_type == '1')		
fgs_motion_vector("direct")		
}		
}		
}		

© ISO/IEC 2004 – All rights reserved

6.2.14.3 FGS Motion Interlaced Information

fgs_motion_interlaced_information( ) {	No. of bits	Mnemonic
fgs_field_prediction	1	bslbf
if (fgs_field_prediction) {		
if (fgs_vop_coding_type == "P" || (fgs_vop_coding_type=="B" && fgs_b_mb_type!="001")){		
fgs_forward_top_field_reference	1	bslbf
fgs_forward_bottom_field_reference	1	bslbf
}		
if ((fgs_vop_coding_type == "B") && (fgs_b_mb_type != "1")) {		
fgs_backward_top_field_reference	1	bslbf
fgs_backward_bottom_field_reference	1	bslbf
}		
}		
}		

6.2.14.4 FGS Motion Vector

fgs_motion_vector (mode) {	No. of bits	Mnemonic
if (mode == „direct") {		
horizontal_mv_data	1-13	vlclbf
vertical_mv_data	1-13	vlclbf
}		
else if (mode == „forward") {		
horizontal_mv_data	1-13	vlclbf
if ((vop_fcode_forward != 1)&&(horizontal_mv_data != 0))		
horizontal_mv_residual	1-6	uimsbf
vertical_mv_data	1-13	vlclbf
if ((vop_fcode_forward != 1)&&(vertical_mv_data != 0))		
vertical_mv_residual	1-6	uimsbf
}		
else if (mode == „backward") {		
horizontal_mv_data	1-13	vlclbf
if ((vop_fcode_backward != 1)&&(horizontal_mv_data != 0))		
horizontal_mv_residual	1-6	uimsbf
vertical_mv_data	1-13	vlclbf
if ((vop_fcode_backward != 1)&&(vertical_mv_data != 0))		
vertical_mv_residual	1-6	uimsbf
}		
}		

© ISO/IEC 2004 – All rights reserved

6.2.14.5 FGS Macroblock

fgs_macroblock() {	No. of bits	Mnemonic
if (fgs_vop_bp_id < 2)		
fgs_cbp	1-9	vlclbf
if (fgs_vop_selective_enhancement_enable==1) {		
if (!mb_shift_factor_received && none_zero_macroblock)		
fgs_shifted_bits	1-5	vlclbf
}		
if (interlaced==1) {		
if (!dct_type_received && non_zero_macroblock)		
fgs_dct_type	1	bslbf
}		
for (i=0; i<6; i++) {		
if (start_decode==1)		
fgs_block()		
}		
}		

NOTE 1 In the syntax of fgs vop, there are three elements: fgs_vop_max_level_y, fgs_vop_max_level_u, and fgs_vop_max_level_v. The maximum value of these three elements is the number of vop-bps in an fgs vop. If any one of the three elements has a smaller value than the number of vop-bps in the fgs vop, the color component corresponding to the smaller value element is absent in one or more vop-bps. If the difference between the number of vop-bps and the value of the element is k, the corresponding color component is absent in the first k vop-bps. There is no need to decode fgs blocks of the color component absent in the vop-bps. start_decode is defined to be a flag to indicate whether decoding should be performed for an fgs block depending on whether the fgs block belongs to an absent color component or not. The value of start_decode is 0 if the fgs block belongs to an absent color component. Otherwise, the value of start_decode is 1 and decoding is performed for the fgs block.

NOTE 2 mb_shift_factor_received is a flag to indicate whether fgs_shifted_bits has been decoded in the previous fgs macroblock with the same fgs macroblock number. It is initialized to 0 before decoding the first vop-bp. It is set to 1 after fgs_shifted_bits is decoded.

NOTE 3 non_zero_macroblock is a flag to indicate whether every block-bp in the fgs macroblock is an all-zero block-bp. If every block-bp in the fgs macroblock is an all-zero block-bp, the value of this flag is 0. Otherwise, its value is 1.

NOTE 4 dct_type_received is a flag to indicate whether fgs_dct_type has been decoded in the previous fgs macroblock with the same fgs macroblock number. It is initialized to 0 before decoding the first vop-bp. It is set to 1 after fgs_dct_type is decoded.

© ISO/IEC 2004 – All rights reserved

6.2.14.6 FGS Block

fgs_block() {	No. of bits	Mnemonic
if (fgs_vop_bp_id>=2 && previous_fgs_msb_not_reached==1)		
fgs_msb_not_reached	1	bslbf
if (fgs_msb_not_reached == 0) {		
while (eop == 0) {		
fgs_run_eop_code	1-16	vlclbf
if (coeff_msb_not_reached ==1)		
fgs_sign_bit	1	bslbf
}		
}		
}		

NOTE 1 previous_fgs_msb_not_reached is defined to be fgs_msb_not_reached decoded in the previous block-bp of the same 8x8 DCT block.

NOTE 2 eop is defined to be the EOP flag resulting from decoding the most recent fgs_run_eop_code. eop is reset to 0 at the beginning of fgs_block().

NOTE 3 coeff_msb_not_reached is defined to be an internal flag indicating, with a value '1', that the MSB of the non-zero DCT coefficient associated with the fgs_run_eop_code above was not reached. The value of this flag is changed to '0' when the MSB of the non-zero DCT coefficient is reached.

6.3 Visual bitstream semantics

6.3.1 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.

Only when profile_and_level_indication indicates a studio profile, extension layers can be inserted in the bitstream. At each point where extensions are allowed in the bitstream any number of the extensions from the defined allowable set may be included. However each type of extension shall not occur more than once.

In the case that a decoder encounters an extension with an extension identification that is described as "reserved" in this specification the decoder shall discard all subsequent data until the next start code. This requirement allows future definition of compatible extensions to this specification.

© ISO/IEC 2004 – All rights reserved

Table 6-4 — extension_start_code_identifier codes

extension_start_code_identifier	Name
0000	reserved
0001	reserved
0010	Sequence Display Extension ID
0011	Quant Matrix Extension ID
0100	Copyright Extension ID
0101	VLC code Extension ID
0110	reserved
0111	Picture Display Extension ID
1000 - 1010	reserved
1011	Camera Parameters Extension ID
1100	ITU-T extension ID
1101	reserved
…	…
1111	reserved

6.3.2 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 6-5. When this field does not exist, the value of visual_object_verid is '0001'.

© ISO/IEC 2004 – All rights reserved

Table 6-5 — Meaning of visual_object_verid

visual_object_verid	Meaning
0000	reserved
0001	object type listed in Table 9-1
0010	object type listed in Table 9-2
0011	reserved
0100	object type listed in Table 9-3
0101	object type listed in Table 9-4
0110 - 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 6-6 which identifies the type of the visual object.

Table 6-6 — Meaning of visual object type

code	visual object type
0000	reserved
0001	video ID
0010	still texture ID
0011	mesh ID
0100	FBA ID
0101	3D mesh ID
01101	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 6-7. If the video_signal_type() is not present in the bitstream then the video format may be assumed to be "Unspecified video format".

© ISO/IEC 2004 – All rights reserved

Table 6-7 — 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 6-8.

Table 6-8 — Colour Primaries

Value	Primaries		
0	(forbidden)		
1	ITU-R Recommendation 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	ITU-R Recommendation 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	ITU-R Recommendation 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

© ISO/IEC 2004 – All rights reserved

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 6-9.

Table 6-9 — Transfer Characteristics

Value	Transfer Characteristic
0	(forbidden)
1	ITU-R Recommendation BT.709 $V = 1{,}099\ L_C^{0{,}45} - 0{,}099$ for $1 \geq L_C \geq 0{,}018$ $V = 4{,}500\ L_C$ for $0{,}018 > L_C \geq 0$
2	Unspecified Video Image characteristics are unknown.
3	reserved
4	ITU-R Recommendation BT.470-2 System M Assumed display gamma 2,2
5	ITU-R Recommendation BT.470-2 System B, G Assumed display gamma 2,8
6	SMPTE 170M $V = 1{,}099\ L_C^{0{,}45} - 0{,}099$ for $1 \geq L_C \geq 0{,}018$ $V = 4{,}500\ L_C$ for $0{,}018 > L_C \geq 0$
7	SMPTE 240M (1987) $V = 1{,}1115\ L_C^{0{,}45} - 0{,}1115$ for $L_C \geq 0{,}0228$ $V = 4{,}0\ L_C$ for $0{,}0228 > L_C$
8	Linear transfer characteristics i.e. $V = L_C$

© ISO/IEC 2004 – All rights reserved

9	Logarithmic transfer characteristic (100:1 range) $V = 1.0-Log_{10}(Lc)/2$ for $1 = L_c = 0.01$ $V= 0.0$ for $0.01 > L_c$
10	Logarithmic transfer characteristic (316.22777:1 range) $V = 1.0-Log_{10}(Lc)/2.5$ for $1 = L_c = 0.0031622777$ $V= 0.0$ for $0.0031622777 > L_c$
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 6-10.

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.

© ISO/IEC 2004 – All rights reserved

Table 6-10 — Matrix Coefficients

Value	Matrix
0	(forbidden)
1	ITU-R Recommendation 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	ITU-R Recommendation 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).

6.3.2.6 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.

6.3.3 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.

© ISO/IEC 2004 – All rights reserved

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 the VOP. 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 6-11.

Table 6-11 — 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
Advanced Real Time Simple	00001010
Core Scalable	00001011
Advanced Coding Efficiency	00001100
Advanced Scalable Texture	00001101
Simple FBA	00001110
Simple Studio	00001111
Core Studio	00010000
Advanced Simple	00010001
Fine Granularity Scalable	00010010
Reserved	00010011 - 11111111

fgs_layer_type – This is a 2-bit code indicating whether this layer is FGS only, FGST only, or a combination of FGS and FGST. Table 6-12 shows the codes and the meanings.

© ISO/IEC 2004 – All rights reserved

Table 6-12 — Code for fgs_layer_type

Code	Meaning
00	reserved
01	FGS
10	FGST
11	FGS-FGST

is_object_layer_identifier: 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 6-13. If both visual_object_verid and video_object_layer_verid exist, the semantics of video_object_layer_verid supersedes the other. When this field does not exist, the value of video_object_layer_verid is substituted by the value of visual_object_verid.

Table 6-13 — Meaning of video_object_layer_verid

video_object_layer_verid	Meaning
0000	Reserved
0001	object type listed in Table 9-1
0010	object type listed in Table 9-2
0011	reserved
0100	object type listed in Table 9-3
0101	object type listed in Table 9-4
0110 - 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. For the transmission of FGS and FGST in two VOLs, the relative transmission priority of an FGS VOL vs. that of an FGST VOL can be specified by setting this parameter in the FGS VOL relative to the same parameter in the FGST VOL.

aspect_ratio_info: This is a four-bit integer which defines the value of pixel aspect ratio. Table 6-14 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 6-14 — 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

© ISO/IEC 2004 – All rights reserved

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 6-15.

Table 6-15 — 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. If this flag is not present in the bitstream, the default value is 0 for visual object types that support B-VOP otherwise it is 1.

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 $R_{vol}(t) <= 400 \times bit\_rate$ 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_half_vbv_occupancy and the latter_half_vbv_occupancy 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 6-16. It identifies the shape type of a video object layer.

© ISO/IEC 2004 – All rights reserved

Table 6-16 — Video Object Layer shape type

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

If video_object_layer_shape == 'rectangular', no shape coding tool (i.e. binary shape or grey scale shape) is used to decode the video object layer. In such a VOL, all macroblocks of each VOP contain the same number of 8x8 blocks, even when (video_object_layer_height % 16) ≤ 8 or (video_object_layer_width % 16) ≤ 8 would make fewer blocks per macroblock possible by using a shape-coding tool for some macroblocks.

video_object_layer_shape_extension: This is a 4-bit integer defined in Table 6-17. It identifies the number (up to 3) and type of auxiliary components that can be used, including the grayscale shape (ALPHA) component. Only a limited number of types and combinations are defined in Table 6-17. More applications are possible by selection of the USER DEFINED type.

Table 6-17 — Semantic meaning of video_object_layer_shape_extension

video_ object_ layer_ shape_ extension	aux_comp_type[0]	aux_comp_type[1]	aux_comp_type[2]	aux_ comp_ count
0000	ALPHA	NO	NO	1
0001	DISPARITY	NO	NO	1
0010	ALPHA	DISPARITY	NO	2
0011	DISPARITY	DISPARITY	NO	2
0100	ALPHA	DISPARITY	DISPARITY	3
0101	DEPTH	NO	NO	1
0110	ALPHA	DEPTH	NO	2
0111	TEXTURE	NO	NO	1
1000	USER DEFINED	NO	NO	1
1001	USER DEFINED	USER DEFINED	NO	2
1010	USER DEFINED	USER DEFINED	USER DEFINED	3
1011	ALPHA	USER DEFINED	NO	2
1100	ALPHA	USER DEFINED	USER DEFINED	3
1101-1111	t.b.d.	t.b.d.	t.b.d.	t.b.d.

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.

© ISO/IEC 2004 – All rights reserved

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 6-18 — 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. A zero value is forbidden.

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. A zero value is forbidden.

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

fgs_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 in the case of fgs_layer_type being FGST or FGS_FGST.

fgs_frequency_weighting_enable: This is a one-bit flag to indicate that frequency weighting is used in this VOL, when set to '1'. Otherwise, when this flag is set to '0', frequency weighting is not used. The default frequency weighting matrix is an all zero matrix when fgs_frequency_weighting_enable is '1'.

load_fgs_frequency_weighting_matrix: This is a one-bit flag which is set to '1' when fgs_frequency_weighting_matrix follows. If it is set to '0' then the default frequency weighting matrix is used.

fgs_frequency_weighting_matrix: This is a list of 2 to 64 three-bit unsigned integers. The integers are in zigzag scan order representing the fgs_frequency_weighting_matrix. A value of 0 indicates that no more values are transmitted and the remaining, non-transmitted values are set to zero.

fgst_frequency_weighting_enable: This is a one-bit flag to indicate that frequency weighting is used in this VOL, when set to '1'. Otherwise, when this flag is set to '0', frequency weighting is not used. The default frequency weighting matrix is an all zero matrix when fgst_frequency_weighting_enable is '1'.

load_fgst_frequency_weighting_matrix: This is a one-bit flag which is set to '1' when fgst_frequency_weighting_matrix follows. If it is set to '0' then the default matrix is used.

© ISO/IEC 2004 – All rights reserved

fgst_frequency_weighting_matrix: This is a list of 2 to 64 three-bit unsigned integers. The integers are in zigzag scan order representing the fgst_frequency_weighting_matrix. A value of 0 indicates that no more values are transmitted and the remaining, non-transmitted values are set to zero.

fgs_resync_marker_disable: This is a one-bit flag which when set to '1' indicates that there is no fgs_resync_marker in coded fgs vops of this VOL. When this flag is set to '0', it indicates that fgs_resync_marker may be used in coded fgs vops of this VOL.

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

sprite_enable: When video_object_layer_verid == '0001', this is a one-bit flag which when set to '1' indicates the usage of static (basic or low latency) sprite coding. When video_object_layer_verid == '0010' or video_object_layer_verid == '0101', this is a two-bit unsigned integer which indicates the usage of static sprite coding or global motion compensation (GMC). Table 6-19 shows the meaning of various codewords. An S-VOP with sprite_enable == "GMC" is referred to as an S (GMC)-VOP in this document.

Table 6-19 — Meaning of sprite_enable codewords

sprite_enable (video_object_layer_ verid == '0001')	sprite_enable (video_object_layer_ verid == '0010') || video_object_layer_verid == '0101')	Sprite Coding Mode
0	00	sprite not used
1	01	static (Basic/Low Latency)
–	10	GMC (Global Motion Compensation)
–	11	Reserved

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_top_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 'static' or 'GMC', 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 6-20 shows the various choices. Note that the value of 4 is disallowed when sprite_enable == 'GMC'.

Table 6-20 — Number of point and implied warping function

Number of points	warping function
0	Stationary
1	Translation
2,3	Affine
4	Perspective
5-63	Reserved

© ISO/IEC 2004 – All rights reserved

sprite_warping_accuracy: This is a 2-bit code which indicates the quantisation accuracy of motion vectors used in the warping process for sprites and GMC. Table 6-21 shows the meaning of various codewords

Table 6-21 — Meaning of sprite warping accuracy codewords

code	sprite_warping_accuracy
00	1/2 pixel
01	1/4 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. This value shall be '0' when sprite_enable=='GMC'.

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 and visual object type is N-bit.

sadct_disable: This is a one-bit flag specifying the inverse transforms to be used for texture decoding. If 'sadct_disable' is set to '1', inverse DCT as described in version 1 is applied to all 8x8-blocks. When set to '0', flag 'sadct_disable' indicates that different types of inverse DCT are used in an adaptive way: inverse DCT is applied to those 8x8-blocks where all 64 pels are opaque, whereas inverse shape-adaptive DCT (SA-DCT) and inverse ΔDC-SA-DCT – an extended version of SA-DCT - are used in inter- and intra-coded 8x8-blocks with at least one transparent and one opaque pel.

quant_precision: This field specifies the number of bits used to represent quantiser scales. 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 scale is used for the decoding of grayscale alpha data. When this flag is set to '0', the grayscale alpha quantiser scale is updated on every macroblock by generating it anew from the luminance quantiser scale 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 quantisation method, two matrices are used, one for intra blocks the other for non-intra blocks.

© ISO/IEC 2004 – All rights reserved

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[i] follows for grayscale alpha or auxiliary component i=0,1,2. If it is set to '0' then there is no change in the quantisation matrix values that shall be used.

© ISO/IEC 2004 – All rights reserved

intra_quant_mat_grayscale[i]: This is a list of 2 to 64 eight-bit unsigned integers defining the grayscale intra alpha quantisation matrix to be used for grayscale alpha or auxiliary component i=0,1,2. The semantics and the default quantisation matrix are identical to those of intra_quant_mat.

nonintra_quant_mat_grayscale: This is a list of 2 to 64 eight-bit unsigned integers defining the grayscale nonintra alpha quantisation matrix[i] to be used for grayscale alpha or auxiliary component i=0,1,2. The semantics and the default quantisation matrix are identical to those of nonintra_quant_mat.

nonintra_quant_mat_grayscale[i]: This is a list of 2 to 64 eight-bit unsigned integers defining the grayscale nonintra alpha quantisation matrix to be used for grayscale alpha or auxiliary component i=0,1,2. The semantics and the default quantisation matrix are identical to those of nonintra_quant_mat.

quarter_sample: This is a one-bit flag which when set to '0' indicates that half sample mode and when set to '1' indicates that quarter sample mode shall be used for motion compensation of the luminance component. For FGST or FGS_FGST enhancement layer, this flag shall be 0.

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 and "01" for version 2.

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).

version2_complexity_estimation_disable: Flag to disable version 2 parameter set.

sadct: Flag enabling transmission of the number of luminance and chrominance blocks coded using SADCT coding mode in % of the total number of blocks (bounding box). When estimation_method == '00' the value of sadct is set to '0'.

quarterpel: Flag enabling transmission of the number of luminance and chrominance block predicted by a quarter-pel vector on one or two dimensions (horizontal and vertical) in % of the total number of blocks (bounding box). When estimation_method == '00' the value of quarterpel is set to '0'.

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).

© ISO/IEC 2004 – All rights reserved

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.

newpred_enable: This is a one-bit flag which, when set to '1', indicates that the NEWPRED mode is enabled. When video_object_layer_verid is equal to '0001', and therefore newpred enable is not transmitted, it takes a default value of zero.

requested_upstream_message_type: This is a two-bits flag which indicates which type of upstream message is needed by the encoder. The syntax and semantics of the upstream message are described in subclause 6.2.12 and 6.3.12.

© ISO/IEC 2004 – All rights reserved

01: need NP_ACK message to be returned for each NEWPRED segment
10: need NP_NACK message to be returned for each NEWPRED segment
11: need both NP_ACK and NP_NACK messages to be returned for each NEWPRED segment
00: reserved

newpred_segment_type: This is a one-bits flag which indicates the unit of selecting reference VOP (NEWPRED segment).

0: Video Packet
1: VOP

reduced_resolution_vop_enable: This is a one-bit flag which indicates that the reduced resolution VOP tool is enabled when set to '1'. When video_object_layer_verid is equal to '0001', and therefore reduced_resolution_vop_enable is not transmitted, it takes a default value of zero.

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'. Additionally, this flag shall be set to '0' for S(GMC)-VOPs.

hierarchy_type: The hierarcical relation between the associated hierarchy layer and its hierarchy embedded layer is defined as shown in Table 6-22.

Table 6-22 — 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 than 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'.

use_ref_shape: This is a one bit flag which indicates the procedure to decode binary shape for spatial scalability. If it is set to '0', scalable shape coding should be used. If it is set to '1' and enhancement_type is set to '0', no shape data is decoded and the up-sampled binary shape of reference_layer should be used for enhancement layer. If enhancement_type is set to '1' and this flag is set to '1', binary shape of the enhancement layer should be decoded as the same non-scalable decoding process. When video_object_layer_verid == '0001', the value of use_ref_shape is set to '1'.

© ISO/IEC 2004 – All rights reserved

use_ref_texture: Reserved flag for future extension. This flag shall be 0 when video_object_layer_ver_id is "0001" or "0010".

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

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

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

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

stuffing_start_code: This is the bit string '000001C3' in hexadecimal. It is used in conjunction with possibly following stuffing_byte(s) for the purpose of stuffing bits to guaranty the VBV buffer regulation.

stuffing_byte: This is the 8-bit string in which the value is '11111111'.

6.3.4 Group of Video Object Plane

group_of_vop_start_code: The group_of_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 6-23. 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 6-23 — 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.

6.3.5 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.

© ISO/IEC 2004 – All rights reserved

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 6-24.

Table 6-24 — 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. 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-, S(GMC)-, 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 non-zero modulo_time_base of the previously decoded I-, S(GMC)-, 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, S(GMC)-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, unless vop_time_increment_resolution is equal to '1', in which case one bit with a value of '0' is used. 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 rules apply: If binary shape or alpha plane does exist for the VOP (i.e. video_object_layer_shape != "rectangular"), it shall be completely transparent. If binary shape or alpha plane does not exist for the VOP (i.e. video_object_layer_shape == "rectangular"), the luminance and chrominance planes of the reconstructed VOP shall be filled with the forward reference VOP as defined in subclause 7.6.7.

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- and S(GMC)- 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_reduced_resolution: This single bit flag signals whether the VOP is encoded in spatialy reduced resolution or not. When vop_reduced_resolution is not transmitted, it takes a default value of zero. When this flag is set to '1', the VOP is encoded spatialy reduced resolution and referred as Reduced Resolution VOP. Reduced Resolution VOP shall be decoded by the texture decoding process including upsampling of IDCT output, and motion compensation decoding process of Reduced Resolution VOP. In this case, the width of the encoded luminance component of the VOP in macroblocks is (video_object_layer_width+31)/32 and the height of the encoded luminance component of the VOP in macroblocks is (video_object_layer_height+31)/32. The displayable part is top and left-aligned in the encoded VOP. This flag shall not be the '1' when the (video_object_layer_width+15)/16 is not the multiple of 2 or the (video_object_layer_width+15)/16 is not the multiple of 2. When this flag is set to "0" or this flag is not present, the VOP is encoded in normal spatial resolution and shall be decoded by the normal decoding process.

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.

© ISO/IEC 2004 – All rights reserved

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_height. 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", hierarchy_type is "1", and this flag is "1", background composition specified in subclause 8.1 is performed. If enhancement type is "1", hierarchy_type is "1" and this flag is "0", any method can be used to make a background for the enhancement layer.

When hierarchy_type is "0", video_object_layer_shape is "binary" (object based spatial scalability), enhancement_type is "1", and background_compositin flag is "1", background composition specified in subclause 8.4 is performed and when 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 that specify a threshold value of quantiser scale used to switch between two VLC's for coding of Intra DC coefficients as per Table 6-25.

Table 6-25 — 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 quantiser scale 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 quantiser scale value for the current macroblock. When usage of Intra DC VLC is specified, the intra DC coefficients are coded by dct_dc_size_luminance, dct_dc_size_chrominance and dct_dc_differential using Table B.13 to Table B.15. When usage of Intra AC VLC is specified, the intra DC coefficients are coded as intra AC coefficients using Table 6-20.

© ISO/IEC 2004 – All rights reserved