
**Information technology — Coded
representation of immersive media —
Part 3:
Versatile video coding**

*Technologies de l'information — Représentation codée de média
immersifs —*

Partie 3: Codage vidéo polyvalent



Reference number
ISO/IEC 23090-3:2021(E)

Documents
AVA Document Sharing Hub
© ISO/IEC 2021



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2021

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier; Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Foreword	vi
Introduction	vii
1 Scope	1
2 Normative references	1
3 Terms and definitions	1
4 Abbreviated terms	13
5 Conventions	16
5.1 General	16
5.2 Arithmetic operators	16
5.3 Logical operators	16
5.4 Relational operators	17
5.5 Bit-wise operators	17
5.6 Assignment operators	17
5.7 Range notation	17
5.8 Mathematical functions	17
5.9 Order of operation precedence	18
5.10 Variables, syntax elements and tables	19
5.11 Text description of logical operations	20
5.12 Processes	21
6 Bitstream and picture formats, partitionings, scanning processes and neighbouring relationships	21
6.1 Bitstream formats	21
6.2 Source, decoded and output picture formats	21
6.3 Partitioning of pictures, subpictures, slices, tiles, and CTUs	23
6.3.1 Partitioning of pictures into subpictures, slices, and tiles	23
6.3.2 Block, quadtree and multi-type tree structures	25
6.3.3 Spatial or component-wise partitionings	26
6.4 Availability processes	27
6.4.1 Allowed quad split process	27
6.4.2 Allowed binary split process	27
6.4.3 Allowed ternary split process	29
6.4.4 Derivation process for neighbouring block availability	30
6.5 Scanning processes	30
6.5.1 CTB raster scanning, tile scanning, and subpicture scanning processes	30
6.5.2 Up-right diagonal scan order array initialization process	34
6.5.3 Horizontal and vertical traverse scan order array initialization process	35
7 Syntax and semantics	35
7.1 Method of specifying syntax in tabular form	35
7.2 Specification of syntax functions and descriptors	36
7.3 Syntax in tabular form	38
7.3.1 NAL unit syntax	38
7.3.2 Raw byte sequence payloads, trailing bits and byte alignment syntax	38
7.3.3 Profile, tier, and level syntax	57
7.3.4 DPB parameters syntax	60
7.3.5 Timing and HRD parameters syntax	60
7.3.6 Supplemental enhancement information message syntax	61
7.3.7 Slice header syntax	62
7.3.8 Weighted prediction parameters syntax	64
7.3.9 Reference picture lists syntax	65
7.3.10 Reference picture list structure syntax	66
7.3.11 Slice data syntax	66
7.4 Semantics	88
7.4.1 General	88
7.4.2 NAL unit semantics	88

7.4.3	Raw byte sequence payloads, trailing bits and byte alignment semantics	95
7.4.4	Profile, tier, and level semantics	141
7.4.5	DPB parameters semantics	146
7.4.6	Timing and HRD parameters semantics	146
7.4.7	Supplemental enhancement information message semantics	150
7.4.8	Slice header semantics	150
7.4.9	Weighted prediction parameters semantics	158
7.4.10	Reference picture lists semantics	159
7.4.11	Reference picture list structure semantics	160
7.4.12	Slice data semantics	161
8	Decoding process	183
8.1	General decoding process	183
8.1.1	General	183
8.1.2	Decoding process for a coded picture	184
8.2	NAL unit decoding process	185
8.3	Slice decoding process	185
8.3.1	Decoding process for picture order count	185
8.3.2	Decoding process for reference picture lists construction	187
8.3.3	Decoding process for reference picture marking	191
8.3.4	Decoding process for generating unavailable reference pictures	192
8.3.5	Decoding process for symmetric motion vector difference reference indices	192
8.3.6	Decoding process for collocated picture and no backward prediction	193
8.4	Decoding process for coding units coded in intra prediction mode	194
8.4.1	General decoding process for coding units coded in intra prediction mode	194
8.4.2	Derivation process for luma intra prediction mode	195
8.4.3	Derivation process for chroma intra prediction mode	198
8.4.4	Cross-component chroma intra prediction mode checking process	199
8.4.5	Decoding process for intra blocks	200
8.5	Decoding process for coding units coded in inter prediction mode	231
8.5.1	General decoding process for coding units coded in inter prediction mode	231
8.5.2	Derivation process for motion vector components and reference indices	235
8.5.3	Decoder-side motion vector refinement process	254
8.5.4	Derivation process for geometric partitioning mode motion vector components and reference indices	259
8.5.5	Derivation process for subblock motion vector components and reference indices	260
8.5.6	Decoding process for inter blocks	285
8.5.7	Decoding process for geometric partitioning mode inter blocks	307
8.5.8	Decoding process for the residual signal of coding blocks coded in inter prediction mode	313
8.5.9	Decoding process for the reconstructed signal of chroma coding blocks coded in inter prediction mode	314
8.6	Decoding process for coding units coded in IBC prediction mode	316
8.6.1	General decoding process for coding units coded in IBC prediction mode	316
8.6.2	Derivation process for block vector components for IBC blocks	317
8.6.3	Decoding process for IBC blocks	321
8.7	Scaling, transformation and array construction process	322
8.7.1	Derivation process for quantization parameters	322
8.7.2	Scaling and transformation process	324
8.7.3	Scaling process for transform coefficients	325
8.7.4	Transformation process for scaled transform coefficients	327
8.7.5	Picture reconstruction process	347
8.8	In-loop filter process	350
8.8.1	General	350
8.8.2	Picture inverse mapping process for luma samples	350
8.8.3	Deblocking filter process	351
8.8.4	Sample adaptive offset process	377
8.8.5	Adaptive loop filter process	379
9	Parsing process	391
9.1	General	391
9.2	Parsing process for k-th order Exp-Golomb codes	391
9.2.1	General	391
9.2.2	Mapping process for signed Exp-Golomb codes	392

9.3	CABAC parsing process for slice data	393
9.3.1	General	393
9.3.2	Initialization process.....	394
9.3.3	Binarization process	418
9.3.4	Decoding process flow	427
Annex A (normative)	Profiles, tiers and levels.....	444
Annex B (normative)	Byte stream format	456
Annex C (normative)	Hypothetical reference decoder	458
Annex D (normative)	Supplemental enhancement information and use of SEI and VUI	480
Bibliography		504

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.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

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. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see <http://patents.iec.ch>).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*, in collaboration with ITU-T. The technically identical text is published as ITU-T H.266 (08/2020).

A list of all parts in the ISO/IEC 23090 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

Purpose

This document specifies a video coding technology known as versatile video coding. It has been designed with two primary goals. The first of these is to specify a video coding technology with a compression capability that is substantially beyond that of the prior generations of such standards, and the second is for this technology to be highly versatile for effective use in a broader range of applications than that addressed by prior standards. Some key application areas for the use of this document particularly include ultra-high-definition video (e.g., with 3840×2160 or 7620×4320 picture resolution and bit depth of 10 bits as specified in Rec. ITU-R BT.2100), video with a high dynamic range and wide colour gamut (e.g., with the perceptual quantization or hybrid log-gamma transfer characteristics specified in Rec. ITU-R BT.2100), and video for immersive media applications such as 360° omnidirectional video projected using a common projection format such as the equirectangular or cubemap projection formats, in addition to the applications that have commonly been addressed by prior video coding standards.

Profiles, tiers, and levels

This document is designed to be versatile in the sense that it serves a wide range of applications, bit rates, resolutions, qualities, and services. Applications include, but are not limited to, video coding for digital storage media, television broadcasting, video streaming services, real-time communication. In the course of creating this document, various requirements from typical applications have been considered, necessary algorithmic elements have been developed, and these have been integrated into a single syntax. Hence, this document is designed to facilitate video data interchange among different applications.

Considering the practicality of implementing the full syntax of this document, however, a limited number of subsets of the syntax are also stipulated by means of "profiles", "tiers", and "levels". These and other related terms are formally defined in Clause 3.

A "profile" is a subset of the entire bitstream syntax that is specified in this document. 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 syntax elements in the bitstream, such as the specified size of the decoded pictures. In many applications, it is currently neither practical nor economical to implement a decoder capable of dealing with all hypothetical uses of the syntax within a particular profile.

In order to deal with this problem, "tiers" and "levels" are specified within each profile. A level of a tier is a specified set of constraints imposed on values of the syntax elements in the bitstream. Some of these constraints are expressed as simple limits on values, while others take the form of constraints on arithmetic combinations of values (e.g. picture width multiplied by picture height multiplied by number of pictures decoded per second). A level specified for a lower tier is more constrained than a level specified for a higher tier.

Coded video content conforming to this document uses a common syntax. In order to achieve a subset of the complete syntax, flags, parameters, and other syntax elements are included in the bitstream that signal the presence or absence of syntactic elements that occur later in the bitstream.

Encoding process, decoding process, and use of VUI parameters and SEI messages

Any encoding process that produces bitstream data that conforms to the specified bitstream syntax format requirements of this document is considered to be in conformance with the requirements of this document. The decoding process is specified such that all decoders that conform to a specified combination of capabilities known as the profile, tier, and level will produce numerically identical cropped decoded output pictures when invoking the decoding process associated with that profile for a bitstream conforming to that profile, tier and level. Any decoding process that produces identical cropped decoded output pictures to those produced by the process described herein (with the correct output order or output timing, as specified) is considered to be in conformance with the requirements of this document.

Rec. ITU-T H.274 | ISO/IEC 23002-7 specifies the syntax and semantics of the video usability information (VUI) parameters and supplemental enhancement information (SEI) messages that do not affect the conformance specifications in Annex C. These VUI parameters and SEI messages may be used together with this document.

Versions of this document

This is the first edition of this document.

Overview of the design characteristics

The coded representation specified in the syntax is designed to enable a high compression capability for a desired image or video quality. The algorithm is typically not mathematically lossless, as the exact source sample values are typically not preserved through the encoding and decoding processes, although some modes are included that provide lossless coding capability. A number of techniques are specified to enable highly efficient compression. Encoding algorithms (not specified within the scope of this document) may select between inter, intra, intra block copy (IBC), and palette coding for block-shaped regions of each picture. Inter coding uses motion vectors for block-based inter-picture prediction to exploit temporal statistical dependencies between different pictures, intra coding uses various spatial prediction modes to exploit spatial statistical dependencies in the source signal within the same picture, and intra block copy coding uses block displacement vectors to reference previously decoded regions of the same picture to exploit statistical similarities among different areas of the same picture. Motion vectors, intra prediction modes, and IBC block vectors are specified for a variety of block sizes in the picture. The prediction residual can then be further compressed using a spatial transform to remove spatial correlation inside a block before it is quantized, producing a possibly irreversible process that typically discards less important visual information while forming a close approximation to the source samples. Finally, the motion vectors, intra prediction modes, and block vectors can also be further compressed using a variety of prediction mechanisms, and, after prediction, are combined with the quantized transform coefficient information and encoded using arithmetic coding.

How to read this document

It is suggested that the reader starts with Clause 1 and moves on to Clause 3. Clause 6 should be read for the geometrical relationship of the source, input, and output of the decoder. Clause 7 specifies the order to parse syntax elements from the bitstream. See subclauses 7.1 to 7.3 for syntactical order and subclause 7.4 for semantics; e.g. the scope, restrictions, and conditions that are imposed on the syntax elements. The actual parsing for most syntax elements is specified in Clause 9. Finally, Clause 8 specifies how the syntax elements are mapped into decoded samples. Throughout reading this document, the reader should refer to Clauses 2, 4, and 5 as needed. Annexes A through D also form an integral part of this document.

Annex A specifies profiles, each being tailored to certain application domains, and defines the so-called tiers and levels of the profiles. Annex B specifies syntax and semantics of a byte stream format for delivery of coded video as an ordered stream of bytes. Annex C specifies the hypothetical reference decoder, bitstream conformance, decoder conformance, and the use of the hypothetical reference decoder to check bitstream and decoder conformance. Annex D specifies syntax and semantics for supplemental enhancement information (SEI) message payloads that affect the conformance specifications in Annex C. Rec. ITU-T H.274 | ISO/IEC 23002-7 specifies the syntax and semantics of the video usability information (VUI) parameters as well as SEI messages that do not affect the conformance specifications in Annex C. These VUI parameters and SEI messages may be used together with this document.

Patent declarations

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of patents.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with ISO and IEC. Information may be obtained from the patent database available at www.iso.org/patents.

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

Information technology — Coded representation of immersive media — Part 3: Versatile video coding

1 Scope

This document specifies a video coding technology known as versatile video coding (VVC), comprising a video coding technology with a compression capability that is substantially beyond that of the prior generations of such standards and with sufficient versatility for effective use in a broad range of applications.

Only the syntax format, semantics, and associated decoding process requirements are specified, while other matters such as pre-processing, the encoding process, system signalling and multiplexing, data loss recovery, post-processing, and video display are considered to be outside the scope of this document. Additionally, the internal processing steps performed within a decoder are also considered to be outside the scope of this document; only the externally observable output behaviour is required to conform to the specifications of this document.

This document is designed to be generic in the sense that it serves a wide range of applications, bit rates, resolutions, qualities and services. Applications include, but are not limited to, video coding for digital storage media, television broadcasting and real-time communication. In the course of creating This document, various requirements from typical applications have been considered, necessary algorithmic elements have been developed, and these have been integrated into a single syntax. Hence, this document is designed to facilitate video data interchange among different applications.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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.

Rec. ITU-T H.274 | ISO/IEC 23002-7, *Versatile supplemental enhancement information messages for coded video bitstreams*

Rec. ITU-T T.35:2000, Procedure for the allocation of ITU-T defined codes for non standard facilities.

3 Terms and definitions

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

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <http://www.electropedia.org/>

3.1

access unit

set of *PUs* that belong to different *layers* and contain *coded pictures* associated with the same time for output from the *DPB*

3.2

adaptive colour transform

cross-component transform applied to the decoded *residual* of a *coding unit* in the 4:4:4 colour format prior to reconstruction and loop filtering

3.3

adaptive loop filter

filtering process that is applied as part of the *decoding process* and is controlled by parameters conveyed in an *APS*

3.4

AC transform coefficient

transform coefficient for which the *frequency index* in at least one of the two dimensions is non-zero

3.5

ALF APS

APS that controls the *ALF* process

3.6

adaptation parameter set

syntax structure containing *syntax elements* that apply to zero or more *slices* as determined by zero or more *syntax elements* found in *slice headers*

3.7

associated GDR picture

previous *GDR picture* (when present) in *decoding order*, for a particular picture with *nuh_layer_id* equal to a particular value *layerId*, that has *nuh_layer_id* equal to *layerId* and between which and the particular *picture* in *decoding order* there is no *IRAP picture* with *nuh_layer_id* equal to *layerId*

3.8

associated GDR subpicture

previous *GDR subpicture* (when present) in *decoding order*, for a particular subpicture with *nuh_layer_id* equal to a particular value *layerId* and subpicture index equal to a particular value *subpicIdx*, that has *nuh_layer_id* equal to *layerId* and subpicture index equal to *subpicIdx* and between which and the particular *subpicture* in *decoding order* there is no *IRAP subpicture* with *nuh_layer_id* equal to *layerId* and subpicture index equal to *subpicIdx*

3.9

associated IRAP picture

previous *IRAP picture* (when present) in *decoding order*, for a particular picture with *nuh_layer_id* equal to a particular value *layerId*, that has *nuh_layer_id* equal to *layerId* and between which and the particular *picture* in *decoding order* there is no *GDR picture* with *nuh_layer_id* equal to *layerId*

3.10

associated IRAP subpicture

previous *IRAP subpicture* (when present) in *decoding order*, for a particular subpicture with *nuh_layer_id* equal to a particular value *layerId* and subpicture index equal to a particular value *subpicIdx*, that has *nuh_layer_id* equal to *layerId* and subpicture index equal to *subpicIdx* and between which and the particular *subpicture* in *decoding order* there is no *GDR subpicture* with *nuh_layer_id* equal to *layerId* and subpicture index equal to *subpicIdx*

3.11

associated non-VCL NAL unit

non-VCL NAL unit (when present) for a *VCL NAL unit* where the *VCL NAL unit* is the *associated VCL NAL unit* of the *non-VCL NAL unit*

3.12

associated VCL NAL unit

preceding *VCL NAL unit* in *decoding order* for a *non-VCL NAL unit* with *nal_unit_type* equal to *EOS_NUT*, *EOB_NUT*, *SUFFIX_APS_NUT*, *SUFFIX_SEI_NUT*, *FD_NUT*, *RSV_NVCL_27*, *UNSPEC_30*, or *UNSPEC_31*; or otherwise the next *VCL NAL unit* in *decoding order*

3.13

bin

bit of a *bin string*

3.14

binarization

set of *bin strings* for all possible values of a *syntax element*

3.15

binarization process

unique mapping process of all possible values of a *syntax element* onto a set of *bin strings*

3.16

binary split

split of a rectangular *MxN block* of samples into two *blocks* where a vertical split results in a first $(M / 2) \times N$ *block* and a second $(M / 2) \times N$ *block*, and a horizontal split results in a first $M \times (N / 2)$ *block* and a second $M \times (N / 2)$ *block*

3.17

bin string

intermediate binary representation of values of *syntax elements* from the *binarization* of the *syntax element*

3.18**bi-predictive slice****B slice**

slice that is decoded using *intra prediction* or using *inter prediction* with at most two *motion vectors* and *reference indices* to *predict* the sample values of each *block*

3.19**bitstream**

sequence of bits, in the form of a *NAL unit stream* or a *byte stream*, that forms the representation of a sequence of *AUs* forming one or more coded video sequences (*CVSs*)

3.20**block**

MxN (M-column by N-row) array of samples, or an MxN array of *transform coefficients*

3.21**block vector**

two-dimensional vector that provides an offset from the coordinates of the current *coding block* to the coordinates of the reference block in the same decoded *slice*

3.22**byte**

sequence of 8 bits, within which, when written or read as a sequence of bit values, the left-most and right-most bits represent the most and least significant bits, respectively

3.23**byte-aligned**

<*bitstream*> positioned an integer multiple of 8 bits from the position of the first bit in the *bitstream*

3.24**byte-aligned**

<bit or *byte* or *syntax element*> position at which it appears in a *bitstream* is byte-aligned

3.25**byte stream**

encapsulation of a *NAL unit stream* into a series of *bytes* containing *start code prefixes* and *NAL units*

3.26**chroma**

sample array or single sample representing one of the two colour difference signals related to the primary colours, represented by the symbols Cb and Cr

Note 1 to entry: The term chroma is used rather than the term chrominance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term chrominance.

3.27**CRA PU**

PU in which the *coded picture* is a *CRA picture*

3.28**CRA picture**

IRAP picture for which each *VCL NAL unit* has *nal_unit_type* equal to *CRA_NUT*

Note 1 to entry: A CRA picture does not use inter prediction in its decoding process, and could be the first picture in the bitstream in decoding order, or could appear later in the bitstream. A CRA picture could have associated RADL or RASL pictures. When a CRA picture has *NoOutputBeforeRecoveryFlag* equal to 1, the associated RASL pictures are not output by the decoder, because they might not be decodable, as they could contain references to pictures that are not present in the bitstream.

3.29**CRA subpicture**

IRAP subpicture for which each *VCL NAL unit* has *nal_unit_type* equal to *CRA_NUT*

3.30**coded layer video sequence:**

sequence of *PU*s with the same value of *nuh_layer_id* that consists, in *decoding order*, of a *CLVSS PU*, followed by zero or more *PU*s that are not *CLVSS PU*s, including all subsequent *PU*s up to but not including any subsequent *PU* that is a *CLVSS PU*

Note 1 to entry: A CLVSS PU could be an IDR PU, a CRA PU, or a GDR PU. The value of *NoOutputBeforeRecoveryFlag* is equal to 1 for each IDR PU, and each CRA PU that has *HandleCraAsClvsStartFlag* equal to 1, and each CRA or GDR PU that is the first PU in

ISO/IEC 23090-3:2021(E)

the layer of the bitstream in decoding order or the first PU in the layer of the bitstream that follows an EOS NAL unit in the layer in decoding order.

3.31

CLVSS PU

PU in which the *coded picture* is a *CLVSS picture*

3.32

CLVSS picture

coded picture that is an *IRAP picture* with *NoOutputBeforeRecoveryFlag* equal to 1 or a *GDR picture* with *NoOutputBeforeRecoveryFlag* equal to 1

3.33

coded picture

coded representation of a *picture* comprising *VCL NAL units* with a particular value of *nuh_layer_id* within an *AU* and containing all *CTUs* of the *picture*

3.34

coded picture buffer

first-in first-out buffer containing *DUs* in *decoding order* specified in the *hypothetical reference decoder*

Note 1 to entry: The hypothetical reference decoder is specified in Annex C.

3.35

coded representation

data element as represented in its coded form

3.36

coded video sequence

sequence of *AUs* that consists, in *decoding order*, of a *CVSS AU*, followed by zero or more *AUs* that are not *CVSS AUs*, including all subsequent *AUs* up to but not including any subsequent *AU* that is a *CVSS AU*

3.37

CVSS AU

IRAP AU or *GDR AU* for which the *coded picture* in each *PU* is a *CLVSS picture*

3.38

coding block

$M \times N$ *block* of samples for some values of *M* and *N* such that the division of a *CTB* into *coding blocks* is a *partitioning*

3.39

coding tree block

$N \times N$ *block* of samples for some value of *N* such that the division of a *component* into *CTBs* is a *partitioning*

3.40

coding tree unit

CTB of *luma* samples, two corresponding *CTBs* of *chroma* samples of a *picture* that has three sample arrays, or a *CTB* of samples of a monochrome *picture*, and *syntax structures* used to code the samples

3.41

coding unit

coding block of *luma* samples, two corresponding *coding blocks* of *chroma* samples of a *picture* that has three sample arrays in the single tree mode, or a *coding block* of *luma* samples of a *picture* that has three sample arrays in the dual tree mode, or two *coding blocks* of *chroma* samples of a *picture* that has three sample arrays in the dual tree mode, or a *coding block* of samples of a monochrome *picture*, and *syntax structures* used to code the samples

3.42

component

array or single sample from one of the three arrays (*luma* and two *chroma*) that compose a *picture* in 4:2:0, 4:2:2, or 4:4:4 colour format or the array or a single sample of the array that compose a *picture* in monochrome format

3.43

context variable

variable specified for the *adaptive binary arithmetic decoding process* of a *bin* by an equation containing recently decoded *bins*

3.44

deblocking filter

filtering process that is applied as part of the *decoding process* in order to minimize the appearance of visual artefacts at the boundaries between *blocks*

3.45**decoded picture**

picture produced by applying the decoding process to a coded picture

3.46**decoded picture buffer**

buffer holding *decoded pictures* for reference, output reordering, or output delay specified for the *hypothetical reference decoder*

3.47**decoder**

embodiment of a *decoding process*

3.48**decoding order**

order in which *syntax elements* are processed by the *decoding process*

3.49**decoding process**

process specified in this document that reads a *bitstream* and derives *decoded pictures* from it

3.50**decoding unit**

AU if *DecodingUnitHrdFlag* is equal to 0 or a subset of an *AU* otherwise, consisting of one or more *VCL NAL units* in an *AU* and the *associated non-VCL NAL units*

3.51**emulation prevention byte**

byte equal to 0x03 that is present within a *NAL unit* when the *syntax elements* of the *bitstream* form certain patterns of *byte* values in a manner that ensures that no sequence of consecutive *byte-aligned bytes* in the *NAL unit* can contain a *start code prefix*

3.52**encoder**

embodiment of an *encoding process*

3.53**encoding process**

process not specified in this document that produces a *bitstream* conforming to this document

3.54**filler data NAL units**

NAL units with *nal_unit_type* equal to *FD_NUT*

3.55**flag**

variable or single-bit *syntax element* that can take one of the two possible values: 0 and 1

3.56**frequency index**

one-dimensional or two-dimensional index associated with a *transform coefficient* prior to the application of a *transform* in the *decoding process*

3.57**GDR AU**

AU in which there is a *PU* for each *layer* present in the *CVS* and the *coded picture* in each present *PU* is a *GDR picture*

3.58**GDR PU**

PU in which the *coded picture* is a *GDR picture*

3.59**GDR picture**

picture for which each *VCL NAL unit* has *nal_unit_type* equal to *GDR_NUT*

Note 1 to entry: The value of *pps_mixed_nalu_types_in_pic_flag* for a *GDR picture* is equal to 0. When *pps_mixed_nalu_types_in_pic_flag* is equal to 0 for a *picture*, and any slice of the *picture* has *nal_unit_type* equal to *GDR_NUT*, all other slices of the *picture* have the same value of *nal_unit_type*, and the *picture* is known to be a *GDR picture* after receiving the first slice.

3.60

GDR subpicture

subpicture for which each VCL NAL unit has nal_unit_type equal to GDR_NUT

3.61

hypothetical reference decoder

hypothetical *decoder* model that specifies constraints on the variability of conforming *NAL unit streams* or conforming *byte streams* that an encoding process may produce

3.62

hypothetical stream scheduler

hypothetical delivery mechanism used for checking the conformance of a *bitstream* or a *decoder* with regards to the timing and data flow of the input of a *bitstream* into the *hypothetical reference decoder*

3.63

IDR PU

PU in which the *coded picture* is an *IDR picture*

3.64

IDR picture

IRAP picture for which each VCL NAL unit has nal_unit_type equal to IDR_W_RADL or IDR_N_LP

Note 1 to entry: An IDR picture does not use inter prediction in its decoding process, and could be the first picture in the bitstream in decoding order, or could appear later in the bitstream. Each IDR picture is the first picture of a CVS in decoding order. When an IDR picture for which each VCL NAL unit has nal_unit_type equal to IDR_W_RADL, it could have associated RADL pictures. When an IDR picture for which each VCL NAL unit has nal_unit_type equal to IDR_N_LP, it does not have any associated leading pictures. An IDR picture does not have associated RASL pictures.

3.65

IDR subpicture

IRAP subpicture for which each VCL NAL unit has nal_unit_type equal to IDR_W_RADL or IDR_N_LP

3.66

inter-layer reference picture

picture in the same *AU* with the current *picture*, with nuh_layer_id less than the nuh_layer_id of the current *picture*, and is marked as "used for long-term reference"

3.67

inter coding

coding of a *coding block*, *slice*, or *picture* that uses *inter prediction*

3.68

inter prediction

prediction derived from blocks of sample values of one or more *reference pictures* as determined by motion vectors

3.69

intra block copy prediction

prediction derived from blocks of sample values of the same decoded *slice* as determined by block vectors

3.70

intra coding

coding of a *coding block*, *slice*, or *picture* that uses *intra prediction*

3.71

intra prediction

prediction derived from neighbouring sample values of the same decoded *slice*

3.72

IRAP AU

AU in which there is a *PU* for each layer present in the CVS and the *coded picture* in each *PU* is an *IRAP picture*

3.73

IRAP PU

PU in which the *coded picture* is an *IRAP picture*

3.74

IRAP picture

coded picture for which all VCL NAL units have the same value of nal_unit_type in the range of IDR_W_RADL to CRA_NUT, inclusive

Note 1 to entry: An IRAP picture could be a CRA picture or an IDR picture. An IRAP picture does not use inter prediction from reference pictures in the same layer in its decoding process. The first picture in the bitstream in decoding order is an IRAP or GDR picture. For a single-layer bitstream, provided the necessary parameter sets are available when they need to be referenced, the IRAP picture and all subsequent non-RASL pictures in the CLVS in decoding order are correctly decodable without performing the decoding process of any pictures that precede the IRAP picture in decoding order.

Note 2 to entry: The value of `pps_mixed_nalu_types_in_pic_flag` for an IRAP picture is equal to 0. When `pps_mixed_nalu_types_in_pic_flag` is equal to 0 for a picture, and any slice of the picture has `nal_unit_type` in the range of `IDR_W_RADL` to `CRA_NUT`, inclusive, all other slices of the picture have the same value of `nal_unit_type`, and the picture is known to be an IRAP picture after receiving the first slice.

3.75

IRAP subpicture

subpicture for which all *VCL NAL units* have the same value of `nal_unit_type` in the range of `IDR_W_RADL` to `CRA_NUT`, inclusive

3.76

intra slice

I slice

slice that is decoded using *intra prediction* only

3.77

layer

set of *VCL NAL units* that all have a particular value of `nuh_layer_id` and the *associated non-VCL NAL units*

3.78

leading picture

picture that precedes the *associated IRAP picture* in *output order*

3.79

leading subpicture

subpicture that precedes the *associated IRAP subpicture* in *output order*

3.80

leaf

terminating node of a tree that is a root node of a tree of depth 0

3.81

level <constraints>

defined set of constraints on the values that may be taken by the *syntax elements* and variables of this document

Note 1 to entry: The same set of levels is defined for all profiles, with most aspects of the definition of each level being in common across different profiles. Individual implementations could, within the specified constraints, support a different level for each supported profile.

3.82

level <transform coefficient value>

value of a *transform coefficient* prior to *scaling*

3.83

list 0 (list 1) motion vector

motion vector associated with a *reference index* pointing into *reference picture list 0 (list 1)*

3.84

list 0 (list 1) prediction

inter prediction of the content of a *slice* using a *reference index* pointing into *reference picture list 0 (list 1)*

3.85

LMCS APS

APS that controls the *LMCS* process

3.86

long-term reference picture

picture with `nuh_layer_id` equal to the `nuh_layer_id` of the current *picture* and marked as "used for long-term reference"

3.87

luma

sample array or single sample representing the monochrome signal related to the primary colours, represented by the symbol or subscript Y or L

Note 1 to entry: The term *luma* is used rather than the term *luminance* in order to avoid implying the use of linear light transfer characteristics that is often associated with the term *luminance*. The symbol *L* is sometimes used instead of the symbol *Y* to avoid confusion with the symbol *y* as used for vertical location.

3.88

luma mapping with chroma scaling

process that is applied as part of the *decoding process* that maps *luma* samples to particular values and in some cases also applies a scaling operation to the values of *chroma* samples

3.89

motion vector

two-dimensional vector used for *inter prediction* that provides an offset from the coordinates in the *decoded picture* to the coordinates in a *reference picture*

3.90

multi-type tree

tree in which a parent node can be split either into two child nodes using a *binary split* or into three child nodes using a *ternary split*, each of which could become the parent node for another split into either two or three child nodes

3.91

NAL unit

syntax structure containing an indication of the type of data to follow and *bytes* containing that data in the form of an *RBSP* interspersed as necessary with *emulation prevention bytes*

3.92

NAL unit stream

sequence of *NAL units*

3.93

operation point

temporal subset of an OLS, identified by an OLS index and a highest value of *TemporalId*

3.94

output layer

layer of an output layer set that is output

3.95

output layer set

set of layers for which one or more layers are specified as the output layers

3.96

OLS layer index

index, of a layer in an OLS, to the list of layers in the OLS

3.97

output order

order of pictures or subpictures within a CLVS indicated by increasing POC values, and for decoded pictures that are output from DPB, this is the order in which the *decoded pictures* are output from the *DPB*

3.98

output time

time when a *decoded picture* is to be output from the *DPB* (for the *decoded pictures* that are to be output from the *DPB*) as specified by the *HRD* according to the output timing *DPB* operation

3.99

palette

set of representative *component* values

3.100

palette prediction

prediction derived from one or more *palettes*

3.101

partitioning

division of a set into subsets such that each element of the set is in exactly one of the subsets

3.102

picture

array of *luma* samples in monochrome format or an array of *luma* samples and two corresponding arrays of *chroma* samples in 4:2:0, 4:2:2, and 4:4:4 colour format

Note 1 to entry: A picture is either a frame or a field. However, in one CVS, either all pictures are frames or all pictures are fields.

3.103

picture header

syntax structure containing *syntax elements* that apply to all *slices* of a coded picture

3.104

picture-level slice index

index, defined when *pps_rect_slice_flag* is equal to 1, of a slice to the list of slices in a picture in the order as the slices are signalled in the PPS when *pps_single_slice_per_subpic_flag* is equal to 0, or in the order of increasing subpicture indices of the subpicture corresponding to the slices when *pps_single_slice_per_subpic_flag* is equal to 1

3.105

picture order count

variable that is associated with each *picture*, uniquely identifies the associated *picture* among all *pictures* in the *CLVS*, and, when the associated *picture* is to be output from the *DPB*, indicates the position of the associated *picture* in *output order* relative to the *output order* positions of the other *pictures* in the same *CLVS* that are to be output from the *DPB*.

3.106

picture parameter set

syntax structure containing *syntax elements* that apply to zero or more entire *coded pictures* as determined by a *syntax element* found in each *picture header*

3.107

picture unit

set of *NAL units* that are associated with each other according to a specified classification rule, are consecutive in *decoding order*, and contain exactly one *coded picture*

3.108

prediction

embodiment of the *prediction process*

3.109

prediction process

use of a *predictor* to provide an estimate of the data element (e.g., sample value or motion vector) currently being decoded

3.110

predictive slice

P slice

slice that is decoded using *intra prediction* or using *inter prediction* with at most one *motion vector* and *reference index* to *predict* the sample values of each *block*

3.111

predictor

combination of specified values or previously decoded data elements (e.g., sample value or motion vector) used in the *decoding process* of subsequent data elements

3.112

profile

specified subset of the syntax of this document

3.113

quadtree

tree in which a parent node can be split into four child nodes, each of which could become the parent node for another split into four child nodes

3.114

quantization parameter

variable used by the *decoding process* for *scaling* of *transform coefficient levels*

3.115

random access

act of starting the decoding process for a *bitstream* at a point other than the beginning of the *bitstream*

3.116

RADL PU

PU in which the *coded picture* is a *RADL picture*

3.117

RADL picture

coded picture for which each *VCL NAL unit* has *nal_unit_type* equal to *RADL_NUT*

Note 1 to entry: All RADL pictures are leading pictures. A RADL picture with *nuh_layer_id* equal to *layerId* is not used as a reference picture for the decoding process of any picture with *nuh_layer_id* equal to *layerId* that follows, in output order, the IRAP picture associated with the RADL picture. When *sps_field_seq_flag* is equal to 0, all RADL pictures, when present, precede, in decoding order, all non-leading pictures of the same associated IRAP picture.

3.118

RADL subpicture

subpicture for which each *VCL NAL unit* has *nal_unit_type* equal to *RADL_NUT*

3.119

RASL PU

PU in which the *coded picture* is a *RASL picture*

3.120

RASL picture

coded picture for which there is at least one *VCL NAL unit* with *nal_unit_type* equal to *RASL_NUT* and other *VCL NAL units* all have *nal_unit_type* equal to *RASL_NUT* or *RADL_NUT*

Note 1 to entry: All RASL pictures are leading pictures of an associated CRA picture. When the associated CRA picture has *NoOutputBeforeRecoveryFlag* equal to 1, the RASL picture is not output and might not be correctly decodable, as the RASL picture could contain references to pictures that are not present in the bitstream. RASL pictures are not used as reference pictures for the decoding process of non-RASL pictures in the same layer, except that a RADL subpicture, when present, in a RASL picture in the same layer could be used for inter prediction of the collocated RADL subpicture in a RADL picture that is associated with the same CRA picture as the RASL picture. When *sps_field_seq_flag* is equal to 0, all RASL pictures, when present, precede, in decoding order, all non-leading pictures of the same associated CRA picture.

3.121

RASL subpicture

subpicture for which each *VCL NAL unit* has *nal_unit_type* equal to *RASL_NUT*

3.122

raster scan

mapping of a rectangular two-dimensional pattern to a one-dimensional pattern such that the first entries in the one-dimensional pattern are from the first top row of the two-dimensional pattern scanned from left to right, followed similarly by the second, third, etc., rows of the pattern (going down) each scanned from left to right

3.123

raw byte sequence payload

syntax structure containing an integer number of *bytes* that is encapsulated in a *NAL unit* and is either empty or has the form of a *string of data bits* containing *syntax elements* followed by an *RBSP stop bit* and zero or more subsequent bits equal to 0

3.124

RBSP stop bit

bit equal to 1 present within a *raw byte sequence payload (RBSP)* after a *string of data bits*, for which the location of the end within an *RBSP* can be identified by searching from the end of the *RBSP* for the *RBSP stop bit*, which is the last non-zero bit in the *RBSP*

3.125

reference index

index into a *reference picture list*

3.126

reference picture

picture that is a *short-term reference picture*, a *long-term reference picture*, or an *inter-layer reference picture*

Note 1 to entry: A reference picture contains samples that could be used for inter prediction in the decoding process of subsequent pictures in decoding order.

3.127

reference picture list

list of *reference pictures* that is used for *inter prediction* of a *P* or *B slice*

Note 1 to entry: Two RPLs, RPL 0 and RPL 1, are generated for each slice of a picture. The set of unique pictures referred to by all entries in the two RPLs associated with a picture consists of all reference pictures that could be used for inter prediction of the associated picture or any picture following the associated picture in decoding order. For the decoding process of a P slice, only RPL 0 is used for inter prediction. For the decoding process of a B slice, both RPL 0 and RPL 1 are used for inter prediction. For decoding the slice data of an I slice, no RPL is used for inter prediction.

3.128**reference picture list 0**

reference picture list used for *inter prediction* of a *P slice* or the first of the two *reference picture lists* used for *inter prediction* of a *B slice*

3.129**reference picture list 1**

second *reference picture list* used for *inter prediction* of a *B slice*

3.130**residual**

decoded difference between a *prediction* of a sample or data element and its decoded value

3.131**scaling**

process of multiplying *transform coefficient levels* by a factor, resulting in *transform coefficients*

3.132**scaling list**

list that associates each *frequency index* with a scale factor for the *scaling* process

3.133**scaling list APS**

APS with syntax elements used to construct the *scaling lists*

3.134**sequence parameter set**

syntax structure containing *syntax elements* that apply to zero or more entire *CLVSs* as determined by the content of a *syntax element* found in the *PPS* referred to by a *syntax element* found in each *picture header*

3.135**short-term reference picture**

picture with *nuh_layer_id* equal to the *nuh_layer_id* of the current *picture* and marked as "used for short-term reference"

3.136**slice**

integer number of complete *tiles* or an integer number of consecutive complete *CTU* rows within a *tile* of a *picture* that are exclusively contained in a single *NAL unit*

3.137**slice header**

part of a coded *slice* containing the data elements pertaining to all *tiles* or *CTU* rows within a *tile* represented in the *slice*

3.138**source**

term used to describe the video material or some of its attributes before encoding

3.139**start code prefix**

unique sequence of three *bytes* equal to 0x000001 embedded in the *byte stream* as a prefix to each *NAL unit*

Note 1 to entry: The location of a start code prefix can be used by a decoder to identify the beginning of a new *NAL unit* and the end of a previous *NAL unit*. Emulation of start code prefixes is prevented within *NAL units* by the inclusion of emulation prevention bytes.

3.140**STSA PU**

PU in which the *coded picture* is an *STSA picture*

3.141**STSA picture**

coded picture for which each *VCL NAL unit* has *nal_unit_type* equal to *STSA_NUT*

Note 1 to entry: An *STSA picture* does not use pictures in the same layer and with the same *TemporalId* as the *STSA picture* for *inter prediction* reference. Pictures following an *STSA picture* in decoding order in the same layer and with the same *TemporalId* as the *STSA picture* do not use pictures prior to the *STSA picture* in decoding order in the same layer and with the same *TemporalId* as the *STSA picture* for *inter prediction* reference. An *STSA picture* enables up-switching, at the *STSA picture*, to the sublayer containing the *STSA picture*, from the immediately lower sublayer of the same layer when the coded picture does not belong to the lowest sublayer. *STSA pictures* in an independent layer always have *TemporalId* greater than 0.

3.142

STSA subpicture

subpicture for which each *VCL NAL unit* has *nal_unit_type* equal to STSA_NUT

3.143

string of data bits

sequence of some number of bits representing *syntax elements* present within a *raw byte sequence payload* prior to the *raw byte sequence payload stop bit*, where the left-most bit is considered to be the first and most significant bit, and the right-most bit is considered to be the last and least significant bit

3.144

sub-bitstream extraction process

specified process by which *NAL units* in a *bitstream* that do not belong to a target set, determined by a target OLS index and a target highest TemporalId, are removed from the *bitstream*, with the output sub-bitstream consisting of the *NAL units* in the *bitstream* that belong to the target set

3.145

sublayer

temporal scalable layer of a temporal scalable *bitstream*, consisting of *VCL NAL units* with a particular value of the TemporalId variable and the associated *non-VCL NAL units*

3.146

sublayer representation

subset of the *bitstream* consisting of *NAL units* of a particular *sublayer* and the lower *sublayers*

3.147

subpicture

rectangular region of one or more *slices* within a *picture*

3.148

subpicture-level slice index

index, defined when *pps_rect_slice_flag* is equal to 1, of a slice to the list of slices in a subpicture in the order as they are signalled in the PPS

3.149

SEI message

syntax structure with specified semantics that conveys a particular type of information that assists in processes related to decoding, display or other purposes but is not needed by the *decoding process* in order to determine the values of the samples in *decoded pictures*

3.150

syntax element

element of data represented in the *bitstream*

3.151

syntax structure

zero or more *syntax elements* present together in the *bitstream* in a specified order

3.152

ternary split

split of a rectangular *MxN block* of samples into three *blocks* where a vertical split results in a first $(M / 4) \times N$ *block*, a second $(M / 2) \times N$ *block*, a third $(M / 4) \times N$ *block*, and a horizontal split results in a first $M \times (N / 4)$ *block*, a second $M \times (N / 2)$ *block*, a third $M \times (N / 4)$ *block*

3.153

tier

specified category of *level* constraints imposed on values of the *syntax elements* in the *bitstream*, where the *level* constraints are nested within a *tier* and a *decoder* conforming to a certain *tier* and *level* would be capable of decoding all *bitstreams* that conform to the same *tier* or the lower *tier* of that *level* or any *level* below it

3.154

tile

rectangular region of *CTUs* within a particular *tile column* and a particular *tile row* in a *picture*

3.155

tile column

rectangular region of *CTUs* having a height equal to the height of the *picture* and a width specified by *syntax elements* in the *picture parameter set*

3.156**tile row**

rectangular region of *CTUs* having a height specified by *syntax elements* in the *picture parameter set* and a width equal to the width of the *picture*

3.157**tile scan**

specific sequential ordering of *CTUs partitioning a picture* in which the *CTUs* are ordered consecutively in *CTU raster scan* in a *tile* whereas *tiles* in a *picture* are ordered consecutively in a *raster scan* of the *tiles* of the *picture*

3.158**trailing picture**

picture for which each *VCL NAL unit* has *nal_unit_type* equal to TRAIL_NUT

Note 1 to entry: Trailing pictures associated with an IRAP or GDR picture also follow the IRAP or GDR picture in decoding order. Pictures that follow the associated IRAP picture in output order and precede the associated IRAP picture in decoding order are not allowed.

3.159**trailing subpicture**

subpicture for which each *VCL NAL unit* has *nal_unit_type* equal to TRAIL_NUT

Note 1 to entry: Trailing subpictures associated with an IRAP or GDR subpicture also follow the IRAP or GDR subpicture in decoding order. Subpictures that follow the associated IRAP subpicture in output order and precede the associated IRAP subpicture in decoding order are not allowed.

3.160**transform**

part of the *decoding process* by which a *block* of *transform coefficients* is converted to a *block* of spatial-domain values

3.161**transform block**

rectangular *MxN block* of samples resulting from a *transform* in the *decoding process*

3.162**transform coefficient**

scalar quantity, considered to be in a frequency domain, that is associated with a particular one-dimensional or two-dimensional *frequency index* in a *transform* in the *decoding process*

3.163**transform coefficient level**

integer quantity representing the value associated with a particular two-dimensional frequency index in the *decoding process* prior to *scaling* for computation of a *transform coefficient* value

3.164**transform unit**

<single tree> *transform block* of *luma* samples and two corresponding *transform blocks* of *chroma* samples of a *picture*, when using a single *coding unit tree* for *luma* and *chroma*, and *syntax structures* used to transform the *transform block* samples

3.165**transform unit**

<dual tree> *transform block* of *luma* samples or two *transform blocks* of *chroma* samples, when using two separate *coding unit trees* for *luma* and *chroma*, and *syntax structures* used to transform the *transform block* samples

3.166**tree**

finite set of nodes with a unique root node

3.167**VCL NAL unit**

collective term for *coded slice NAL units* and the subset of *NAL units* that have *reserved* values of *nal_unit_type* that are classified as VCL NAL units in this document

4 Abbreviated terms

ACT	adaptive colour transform
ALF	adaptive loop filter

ISO/IEC 23090-3:2021(E)

AMVR	adaptive motion vector resolution
APS	adaptation parameter set
AU	access unit
AUD	access unit delimiter
AVC	advanced video coding (Rec. ITU-T H.264 ISO/IEC 14496-10)
B	bi-predictive
BCW	bi-prediction with CU-level weights
BDOF	bi-directional optical flow
BDPCM	block-based delta pulse code modulation
BP	buffering period
CABAC	context-based adaptive binary arithmetic coding
CB	coding block
CBR	constant bit rate
CCALF	cross-component adaptive loop filter
CLVS	coded layer video sequence
CLVSS	coded layer video sequence start
CPB	coded picture buffer
CRA	clean random access
CRC	cyclic redundancy check
CTB	coding tree block
CTU	coding tree unit
CU	coding unit
CVS	coded video sequence
CVSS	coded video sequence start
DPB	decoded picture buffer
DCI	decoding capability information
DRAP	dependent random access point
DU	decoding unit
DUI	decoding unit information
EG	exponential-Golomb
EGk	k-th order exponential-Golomb
EOB	end of bitstream
EOS	end of sequence
FD	filler data
FIFO	first-in, first-out
FL	fixed-length
GBR	green, blue, and red
GCI	general constraints information
GDR	gradual decoding refresh
GPM	geometric partitioning mode
HEVC	high efficiency video coding (Rec. ITU-T H.265 ISO/IEC 23008-2)
HRD	hypothetical reference decoder
HSS	hypothetical stream scheduler
I	intra
IBC	intra block copy
IDR	instantaneous decoding refresh

ILRP	inter-layer reference picture
IRAP	intra random access point
LFNST	low frequency non-separable transform
LPS	least probable symbol
LSB	least significant bit
LTRP	long-term reference picture
LMCS	luma mapping with chroma scaling
MIP	matrix-based intra prediction
MPS	most probable symbol
MSB	most significant bit
MTS	multiple transform selection
MVP	motion vector prediction
NAL	network abstraction layer
OLS	output layer set
OP	operation point
OPI	operating point information
P	predictive
PH	picture header
POC	picture order count
PPS	picture parameter set
PROF	prediction refinement with optical flow
PT	picture timing
PU	picture unit
QP	quantization parameter
RADL	random access decodable leading (picture)
RASL	random access skipped leading (picture)
RBSP	raw byte sequence payload
RGB	red, green, and blue
RPL	reference picture list
SAO	sample adaptive offset
SAR	sample aspect ratio
SEI	supplemental enhancement information
SH	slice header
SLI	subpicture level information
SODB	string of data bits
SPS	sequence parameter set
STRP	short-term reference picture
STSA	step-wise temporal sublayer access
TR	truncated rice
TU	transform unit
VBR	variable bit rate
VCL	video coding layer
VPS	video parameter set
VSEI	versatile supplemental enhancement information (Rec. ITU-T H.274 ISO/IEC 23002-7)
VUI	video usability information
VVC	versatile video coding (this document: Rec. ITU-T H.266 ISO/IEC 23090-3)

5 Conventions

5.1 General

The term "this document" is used to refer to this Recommendation | International Standard.

In this document, the following verbal forms are used:

- "shall" indicates a requirement;
- "should" indicates a recommendation;
- "may" indicates a permission;
- "can" indicates a possibility or a capability.

Information marked as "NOTE" is intended to assist the understanding or use of the document. "Notes to entry" used in Clause 3 provide additional information that supplements the terminological data and can contain provisions relating to the use of a term.

The word "reserved" is used to specify that some values of a particular syntax element are for future use by ITU-T | ISO/IEC and shall not be used in syntax structures conforming to this version of this document, but could potentially be used in syntax structures conforming to future versions of this document by ITU-T | ISO/IEC.

The word "unspecified" is used to describe some values of a particular syntax element to indicate that the values have no specified meaning in this document and are not expected to have a specified meaning in future editions of this document.

NOTE The mathematical operators used in this document are similar to those used in the C programming language. However, the results of integer division and arithmetic shift operations are defined more precisely, and additional operations are defined, such as exponentiation and real-valued division. Numbering and counting conventions generally begin from 0, e.g., "the first" is equivalent to the 0-th, "the second" is equivalent to the 1-th, etc.

5.2 Arithmetic operators

+	addition
−	subtraction (as a two-argument operator) or negation (as a unary prefix operator)
*	multiplication, including matrix multiplication
	exponentiation
x^y	Specifies x to the power of y . In other contexts, such notation is used for superscripting not intended for interpretation as exponentiation.
/	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.
÷	division in mathematical equations where no truncation or rounding is intended
$\frac{x}{y}$	division in mathematical equations where no truncation or rounding is intended
$\sum_{i=x}^y f(i)$	summation of $f(i)$ with i taking all integer values from x up to and including y
	modulus
$x \% y$	Remainder of x divided by y , defined only for integers x and y with $x \geq 0$ and $y > 0$

5.3 Logical operators

$x \ \&\& \ y$	Boolean logical "and" of x and y
$x \ \ y$	Boolean logical "or" of x and y
!	Boolean logical "not"
$x \ ? \ y : z$	if x is TRUE, evaluates to the value of y ; otherwise, evaluates to the value of z

When evaluating a logical expression, the value 0 is interpreted as FALSE and any numerical value not equal to 0 is interpreted as TRUE. The result of any logical expression that evaluates as FALSE is the value 0, and the result of any logical expression that evaluates as TRUE is the value 1.

5.4 Relational operators

>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
=	equal to
!=	not equal to

When a relational operator is applied to a syntax element or variable that has been assigned the value "na" (not applicable), the value "na" is treated as a distinct value for the syntax element or variable. The value "na" is considered not to be equal to any other value.

5.5 Bit-wise operators

&	bit-wise "and"
	When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
	bit-wise "or"
	When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
^	bit-wise "exclusive or"
	When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
x >> y	arithmetic right shift of a two's complement integer representation of x by y binary digits
	This function is defined only for non-negative integer values of y. Bits shifted into the most significant bits (MSBs) as a result of the right shift have a value equal to the MSB of x prior to the shift operation.
x << y	arithmetic left shift of a two's complement integer representation of x by y binary digits
	This function is defined only for non-negative integer values of y. Bits shifted into the least significant bits (LSBs) as a result of the left shift have a value equal to 0.

5.6 Assignment operators

=	assignment operator
++	increment, i.e., $x++$ is equivalent to $x = x + 1$; when used in an array index, evaluates to the value of the variable prior to the increment operation
--	decrement, i.e., $x--$ is equivalent to $x = x - 1$; when used in an array index, evaluates to the value of the variable prior to the decrement operation
+=	increment by amount specified, i.e., $x += 3$ is equivalent to $x = x + 3$, and $x += (-3)$ is equivalent to $x = x + (-3)$
-=	decrement by amount specified, i.e., $x -= 3$ is equivalent to $x = x - 3$, and $x -= (-3)$ is equivalent to $x = x - (-3)$

5.7 Range notation

$x = y..z$ x takes on integer values starting from y to z, inclusive, with x, y, and z being integer numbers and z being greater than or equal to y.

5.8 Mathematical functions

$$\text{Abs}(x) = \begin{cases} x & ; \quad x \geq 0 \\ -x & ; \quad x < 0 \end{cases} \quad (1)$$

$\text{Ceil}(x)$ smallest integer greater than or equal to x . (2)

$\text{Clip1}(x) = \text{Clip3}(0, (1 \ll \text{BitDepth}) - 1, x)$ (3)

$\text{Clip3}(x, y, z) = \begin{cases} x & ; \quad z < x \\ y & ; \quad z > y \\ z & ; \quad \text{otherwise} \end{cases}$ (4)

$\text{ClipH}(o, W, x) = \begin{cases} x + o & ; \quad x < 0 \\ x - o & ; \quad x > W - 1 \\ x & ; \quad \text{otherwise} \end{cases}$ (5)

$\text{Floor}(x)$ largest integer less than or equal to x . (6)

$\text{Log2}(x)$ base-2 logarithm of x . (7)

$\text{Min}(x, y) = \begin{cases} x & ; \quad x \leq y \\ y & ; \quad x > y \end{cases}$ (8)

$\text{Max}(x, y) = \begin{cases} x & ; \quad x \geq y \\ y & ; \quad x < y \end{cases}$ (9)

$\text{Round}(x) = \text{Sign}(x) * \text{Floor}(\text{Abs}(x) + 0.5)$ (10)

$\text{Sign}(x) = \begin{cases} 1 & ; \quad x > 0 \\ 0 & ; \quad x = 0 \\ -1 & ; \quad x < 0 \end{cases}$ (11)

$\text{Sqrt}(x)$ square root of x (12)

$\text{Swap}(x, y) = (y, x)$ (13)

5.9 Order of operation precedence

When order of precedence in an expression is not indicated explicitly by use of parentheses, the following rules apply:

- Operations of a higher precedence are evaluated before any operation of a lower precedence.
- Operations of the same precedence are evaluated sequentially from left to right.

Table 1 specifies the precedence of operations from highest to lowest; a higher position in the table indicates a higher precedence.

NOTE For those operators that are also used in the C programming language, the order of precedence used in this document is the same as used in the C programming language.

Table 1 – Operation precedence from highest (at top of table) to lowest (at bottom of table)

operations (with operands x, y, and z)
"x++", "x--"
"!x", "-x" (as a unary prefix operator)
x^y
"x * y", "x / y", "x ÷ y", " $\frac{x}{y}$ ", "x % y"
"x + y", "x - y" (as a two-argument operator), " $\sum_{i=x}^y f(i)$ "
"x << y", "x >> y"
"x < y", "x <= y", "x > y", "x >= y"
"x == y", "x != y"
"x & y"
"x y"
"x && y"
"x y"
"x ? y : z"
"x.y"
"x = y", "x += y", "x -= y"

5.10 Variables, syntax elements and tables

Syntax elements in the bitstream are represented in **bold** type. Each syntax element is described by its name (all lower case letters with underscore characters), and one descriptor for its method of coded representation. The decoding process behaves according to the value of the syntax element and to the values of previously decoded syntax elements. When a value of a syntax element is used in the syntax tables or the text, it appears in regular (i.e., not bold) type.

In some cases the syntax tables and semantics use the values of other variables derived from the values of syntax elements. Such variables appear in the syntax tables, or text, named by a mixture of lower case and upper case letter and without any underscore characters. Variables starting with an upper case letter are derived for the decoding of the current syntax structure and all depending syntax structures. Variables starting with an upper case letter could, in some cases, be used in the decoding process for later syntax structures without mentioning the originating syntax structure of the variable. Variables starting with a lower case letter are only used within the clause in which they are derived.

In some cases, "mnemonic" names for syntax element values or variable values are used interchangeably with their numerical values. Sometimes "mnemonic" names are used without any associated numerical values. The association of values and names is specified in the text. The names are constructed from one or more groups of letters separated by an underscore character. Each group starts with an upper case letter and could contain more upper case letters.

NOTE The syntax is described in a manner that closely follows the C-language syntactic constructs.

Functions that specify properties of the current position in the bitstream are referred to as syntax functions. These functions are specified in subclause 7.2 and assume the existence of a bitstream pointer with an indication of the position of the next bit to be read by the decoding process from the bitstream. Syntax functions are described by their names, which are constructed as syntax element names and end with left and right round parentheses including zero or more variable names (for definition) or values (for usage), separated by commas (if more than one variable).

Functions that are not syntax functions (including mathematical functions specified in subclause 5.8) are described by their names, which start with an upper case letter, contain a mixture of lower and upper case letters without any underscore character, and end with left and right parentheses including zero or more variable names (for definition) or values (for usage) separated by commas (if more than one variable).

A one-dimensional array is referred to as a list. A two-dimensional array is referred to as a matrix. Arrays can either be syntax elements or variables. Subscripts or square parentheses are used for the indexing of arrays. In reference to a visual depiction of a matrix, the first subscript is used as a row (vertical) index and the second subscript is used as a column

(horizontal) index. The indexing order is reversed when using square parentheses rather than subscripts for indexing. Thus, an element of a matrix s at horizontal position x and vertical position y could be denoted either as $s[x][y]$ or as s_{yx} . A single column of a matrix could be referred to as a list and denoted by omission of the row index. Thus, the column of a matrix s at horizontal position x could be referred to as the list $s[x]$.

A specification of values of the entries in rows and columns of an array could be denoted by $\{ \{ \dots \} \{ \dots \} \}$, where each inner pair of brackets specifies the values of the elements within a row in increasing column order and the rows are ordered in increasing row order. Thus, setting a matrix s equal to $\{ \{ 1 \ 6 \} \{ 4 \ 9 \} \}$ specifies that $s[0][0]$ is set equal to 1, $s[1][0]$ is set equal to 6, $s[0][1]$ is set equal to 4, and $s[1][1]$ is set equal to 9.

Binary notation is indicated by enclosing the string of bit values by single quote marks. For example, '01000001' represents an eight-bit string having only its second and its last bits (counted from the most to the least significant bit) equal to 1.

Hexadecimal notation, indicated by prefixing the hexadecimal number by "0x", is used in some cases instead of binary notation when the number of bits is an integer multiple of 4. For example, 0x41 represents an eight-bit string having only its second and its last bits (counted from the most to the least significant bit) equal to 1.

Numerical values not enclosed in single quotes and not prefixed by "0x" are decimal values.

A value equal to 0 represents a FALSE condition in a test statement. The value TRUE is represented by any value different from zero.

5.11 Text description of logical operations

In the text, a statement of logical operations as would be described mathematically in the following form:

```
if( condition 0 )
    statement 0
else if( condition 1 )
    statement 1
...
else /* informative remark on remaining condition */
    statement n
```

is typically described in the following manner:

- ... as follows / ... the following applies:
- If condition 0, statement 0
- Otherwise, if condition 1, statement 1
- ...
- Otherwise (informative remark on remaining condition), statement n

Each "If ... Otherwise, if ... Otherwise, ..." statement in the text is introduced with "... as follows" or "... the following applies" immediately followed by "If ... ". The last condition of the "If ... Otherwise, if ... Otherwise, ..." is always an "Otherwise, ...". Interleaved "If ... Otherwise, if ... Otherwise, ..." statements can be identified by matching "... as follows" or "... the following applies" with the ending "Otherwise, ...".

In the text, a statement of logical operations as would be described mathematically in the following form:

```
if( condition 0a && condition 0b )
    statement 0
else if( condition 1a || condition 1b )
    statement 1
...
else
    statement n
```

is typically described in the following manner:

- ... as follows / ... the following applies:
- If all of the following conditions are true, statement 0:
 - condition 0a
 - condition 0b
- Otherwise, if one or more of the following conditions are true, statement 1:

- condition 1a
- condition 1b
- ...
- Otherwise, statement n

In the text, a statement of logical operations as would be described mathematically in the following form:

```
if( condition 0 )
  statement 0
if( condition 1 )
  statement 1
```

is typically described in the following manner:

```
When condition 0, statement 0
When condition 1, statement 1
```

5.12 Processes

Processes are used to describe the decoding of syntax elements. A process has a separate specification and invoking. All syntax elements and upper case variables that pertain to the current syntax structure and depending syntax structures are available in the process specification and invoking. A process specification might also have a lower case variable explicitly specified as input. Each process specification has explicitly specified an output. The output is a variable that can either be an upper case variable or a lower case variable.

When invoking a process, the assignment of variables is specified as follows:

- If the variables at the invoking and the process specification do not have the same name, the variables are explicitly assigned to lower case input or output variables of the process specification.
- Otherwise (the variables at the invoking and the process specification have the same name), assignment is implied.

In the specification of a process, a specific coding block is sometimes referred to by the variable name having a value equal to the address of the specific coding block.

6 Bitstream and picture formats, partitionings, scanning processes and neighbouring relationships

6.1 Bitstream formats

This subclause specifies the relationship between the network abstraction layer (NAL) unit stream and byte stream, either of which are referred to as the bitstream.

The bitstream can be in one of two formats: the NAL unit stream format or the byte stream format. The NAL unit stream format is conceptually the more "basic" type. It consists of a sequence of syntax structures called NAL units. This sequence is ordered in decoding order. There are constraints imposed on the decoding order (and contents) of the NAL units in the NAL unit stream.

The byte stream format can be constructed from the NAL unit stream format by ordering the NAL units in decoding order and prefixing each NAL unit with a start code prefix and zero or more zero-valued bytes to form a stream of bytes. The NAL unit stream format can be extracted from the byte stream format by searching for the location of the unique start code prefix pattern within this stream of bytes. Methods of framing the NAL units in a manner other than use of the byte stream format are outside the scope of this document. The byte stream format is specified in Annex B.

6.2 Source, decoded and output picture formats

This subclause specifies the relationship between source and decoded pictures that is given via the bitstream.

The video source that is represented by the bitstream is a sequence of pictures in decoding order.

The source and decoded pictures are each comprised of one or three sample arrays:

- Luma (Y) only (monochrome).
- Luma and two chroma (e.g., YCbCr or YCgCo).

- Green, blue, and red (GBR, also known as RGB).
- Arrays representing other unspecified monochrome or tri-stimulus colour samplings (for example, YZX, also known as XYZ).

For convenience of notation and terminology in this document, the variables and terms associated with these arrays are referred to as luma (or L or Y) and chroma, where the two chroma arrays are referred to as Cb and Cr; regardless of the actual colour representation method in use. The actual colour representation method in use can be indicated in syntax that is specified in VUI parameters as specified in Rec. ITU-T H.274 | ISO/IEC 23002-7.

The variables SubWidthC and SubHeightC are specified in Table 2, depending on the chroma format sampling structure, which is specified through sps_chroma_format_idc.

Table 2 – SubWidthC and SubHeightC values derived from sps_chroma_format_idc

sps_chroma_format_idc	Chroma format	SubWidthC	SubHeightC
0	Monochrome	1	1
1	4:2:0	2	2
2	4:2:2	2	1
3	4:4:4	1	1

In monochrome sampling there is only one sample array, which is nominally considered the luma array.

In 4:2:0 sampling, each of the two chroma arrays has half the height and half the width of the luma array.

In 4:2:2 sampling, each of the two chroma arrays has the same height and half the width of the luma array.

In 4:4:4 sampling, each of the two chroma arrays has the same height and width as the luma array.

The number of bits necessary for the representation of each of the samples in the luma and chroma arrays in a video sequence is in the range of 8 to 16, inclusive.

When the value of sps_chroma_format_idc is equal to 1, the nominal vertical and horizontal relative locations of luma and chroma samples in pictures are shown in Figure 1. Alternative chroma sample relative locations can be indicated in VUI parameters as specified in Rec. ITU-T H.274 | ISO/IEC 23002-7.

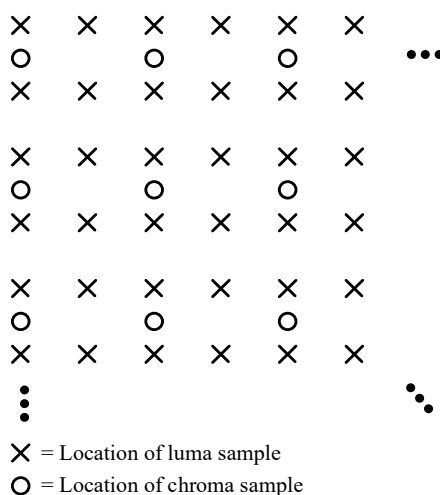


Figure 1 – Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a picture

When the value of sps_chroma_format_idc is equal to 2, the chroma samples are co-sited with the corresponding luma samples and the nominal locations in a picture are as shown in Figure 2.

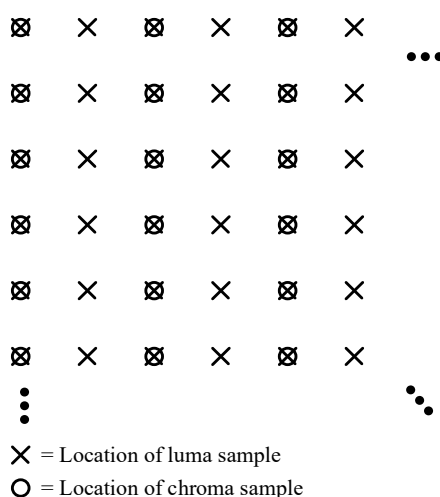


Figure 2 – Nominal vertical and horizontal locations of 4:2:2 luma and chroma samples in a picture

When the value of `sps_chroma_format_idc` is equal to 3, all array samples are co-sited for all cases of pictures and the nominal locations in a picture are as shown in Figure 3.

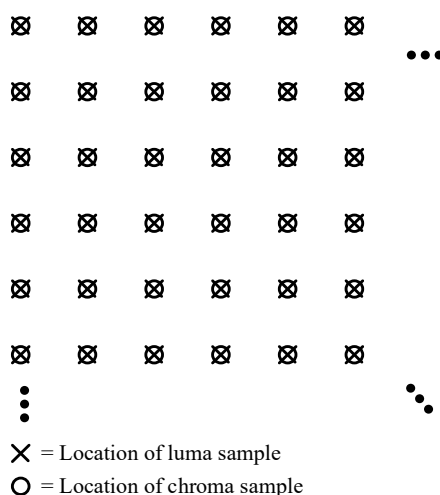


Figure 3 – Nominal vertical and horizontal locations of 4:4:4 luma and chroma samples in a picture

6.3 Partitioning of pictures, subpictures, slices, tiles, and CTUs

6.3.1 Partitioning of pictures into subpictures, slices, and tiles

This subclause specifies how a picture is partitioned into subpictures, slices, and tiles.

A picture is divided into one or more tile rows and one or more tile columns. A tile is a sequence of CTUs that covers a rectangular region of a picture. The CTUs in a tile are scanned in raster scan order within that tile.

A slice consists of an integer number of complete tiles or an integer number of consecutive complete CTU rows within a tile of a picture. Consequently, each vertical slice boundary is always also a vertical tile boundary. It is possible that a horizontal boundary of a slice is not a tile boundary but consists of horizontal CTU boundaries within a tile; this occurs when a tile is split into multiple rectangular slices, each of which consists of an integer number of consecutive complete CTU rows within the tile.

Two modes of slices are supported, namely the raster-scan slice mode and the rectangular slice mode. In the raster-scan slice mode, a slice contains a sequence of complete tiles in a tile raster scan of a picture. In the rectangular slice mode, a slice contains either a number of complete tiles that collectively form a rectangular region of the picture or a number of consecutive complete CTU rows of one tile that collectively form a rectangular region of the picture. Tiles within a rectangular slice are scanned in tile raster scan order within the rectangular region corresponding to that slice.

A subpicture contains one or more slices that collectively cover a rectangular region of a picture. Consequently, each subpicture boundary is also always a slice boundary, and each vertical subpicture boundary is always also a vertical tile boundary.

One or both of the following conditions shall be fulfilled for each subpicture and tile:

- All CTUs in a subpicture belong to the same tile.
- All CTUs in a tile belong to the same subpicture.

Figure 4 shows an example of raster-scan slice partitioning of a picture, where the picture is divided into 12 tiles and 3 raster-scan slices.

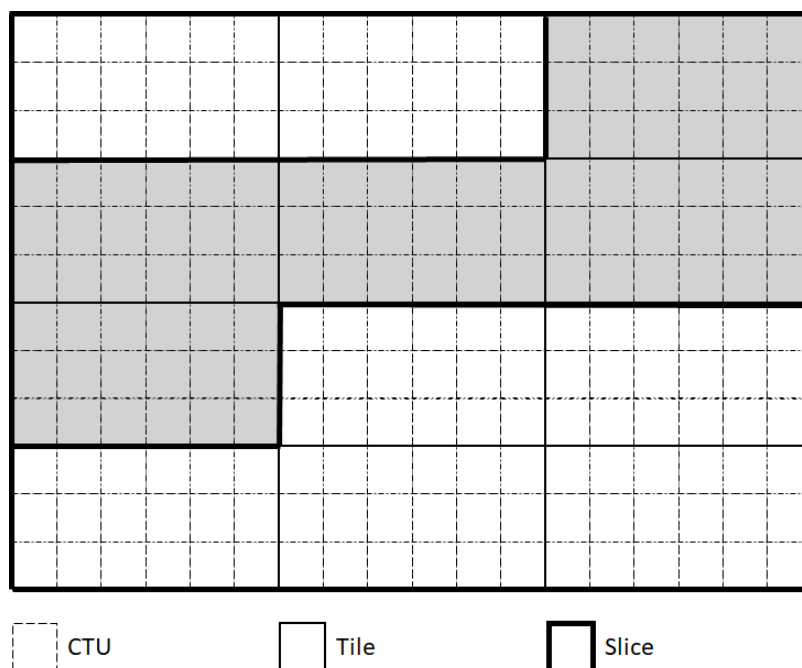


Figure 4 – A picture with 18 by 12 luma CTUs that is partitioned into 12 tiles and 3 raster-scan slices

Figure 5 shows an example of rectangular slice partitioning of a picture, where the picture is divided into 24 tiles (6 tile columns and 4 tile rows) and 9 rectangular slices.

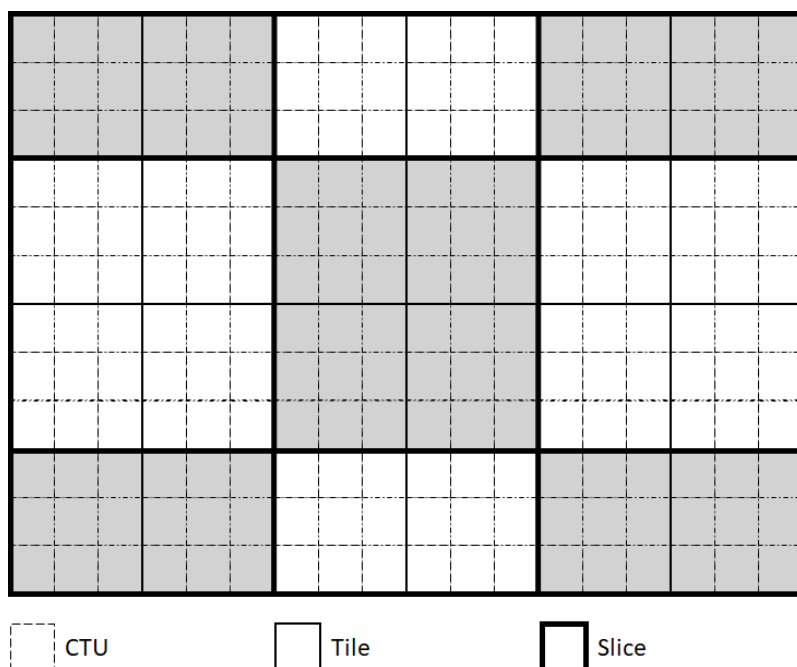


Figure 5 – A picture with 18 by 12 luma CTUs that is partitioned into 24 tiles and 9 rectangular slices

Figure 6 shows an example of a picture partitioned into tiles and rectangular slices, where the picture is divided into 4 tiles (2 tile columns and 2 tile rows) and 4 rectangular slices.

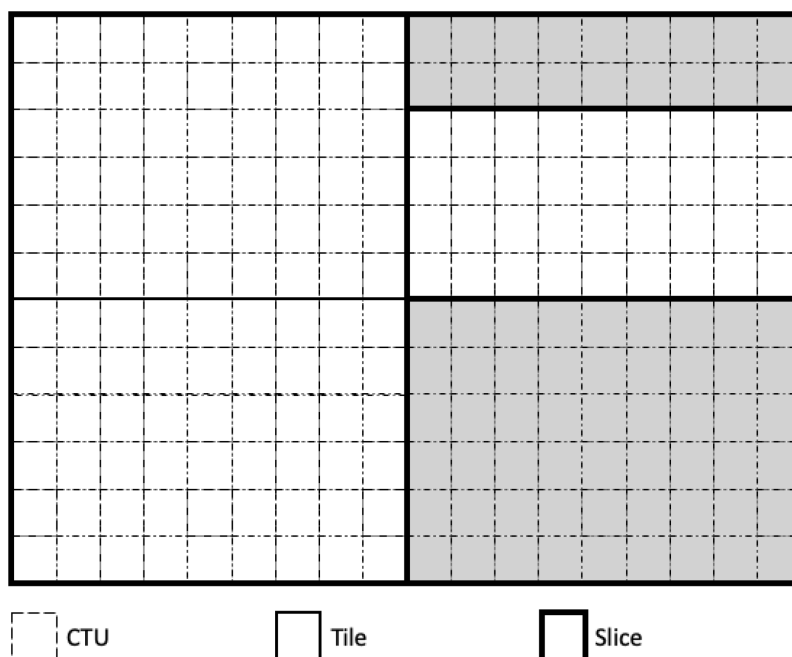


Figure 6 – A picture that is partitioned into 4 tiles and 4 rectangular slices

Figure 7 shows an example of subpicture partitioning of a picture, where a picture is partitioned into 18 tiles, 12 tiles on the left-hand side each covering one slice of 4 by 4 CTUs and 6 tiles on the right-hand side each covering 2 vertically-stacked slices of 2 by 2 CTUs, altogether resulting in 24 slices and 24 subpictures of varying dimensions (each slice is a subpicture).

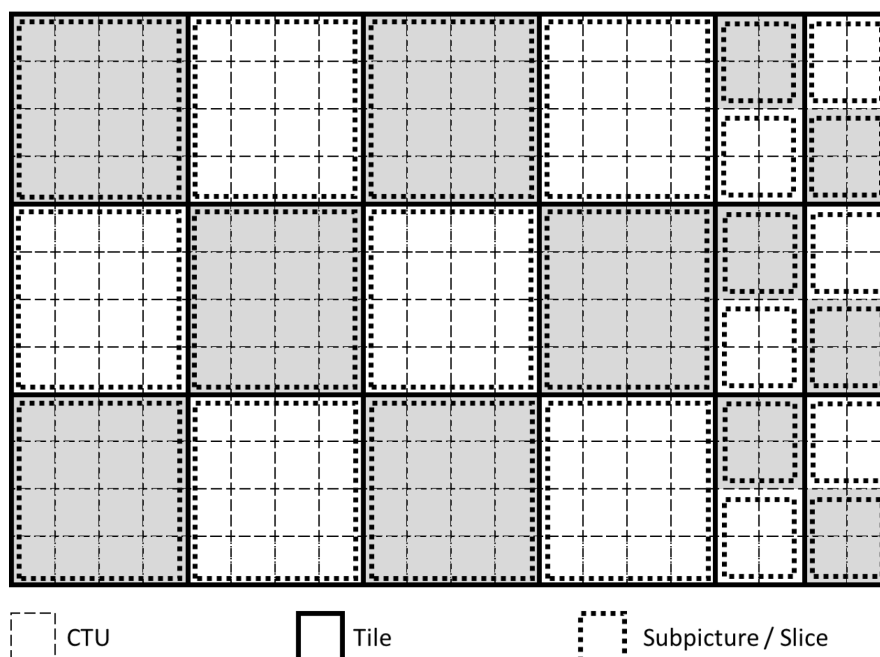


Figure 7 – A picture that is partitioned into 18 tiles, 24 slices and 24 subpictures

6.3.2 Block, quadtree and multi-type tree structures

The samples are processed in units of CTBs. The array size for each luma CTB in both width and height is $CtbSizeY$ in units of samples. The width and height of the array for each chroma CTB are $CtbWidthC$ and $CtbHeightC$, respectively, in units of samples. When the component width is not an integer multiple of the CTB size, the CTBs at the right component boundary are incomplete. When the component height is not an integer multiple of the CTB size, the CTBs at the bottom component boundary are incomplete.

Each CTB is assigned a partition signalling to identify the block sizes for intra or inter prediction and for transform coding. The partitioning is a recursive quadtree partitioning with a nested recursive multi-type tree partitioning. The root of the quadtree is associated with the CTB. The quadtree is split until a leaf is reached, which is referred to as the quadtree leaf.

The root of the multi-type tree is associated with the quadtree leaf. The multi-type tree is split using horizontal or vertical binary splits or horizontal or vertical ternary splits until a leaf is reached, which is associated with the coding block.

The coding block is the root node of the transform tree. The transform tree specifies the position and size of transform blocks. The splitting information for luma and chroma might or might not be identical for the transform tree.

The blocks and associated syntax structures are grouped into "unit" structures as follows:

- One transform block (monochrome picture) or three transform blocks (luma and chroma components of a picture in 4:2:0, 4:2:2 or 4:4:4 colour format) and the associated transform syntax structures units are associated with a transform unit.
- One coding block (monochrome picture) or three coding blocks (luma and chroma), the associated coding syntax structures and the associated transform units are associated with a coding unit.
- One CTB (monochrome picture) or three CTBs (luma and chroma), the associated coding tree syntax structures and the associated coding units are associated with a CTU.

6.3.3 Spatial or component-wise partitionings

The following divisions of processing elements of this document form spatial or component-wise partitioning:

- division of each picture into components;
- division of each component into CTBs;
- division of each picture into subpictures;
- division of each picture into tile columns;
- division of each picture into tile rows;
- division of each tile column into tiles;
- division of each tile row into tiles;
- division of each tile into CTUs;
- division of each picture into slices;
- division of each subpicture into slices;
- division of each slice into CTUs;
- division of each CTU into CTBs;
- division of each CTB into coding blocks, except that the CTBs are incomplete at the right component boundary when the component width is not an integer multiple of the CTB size and the CTBs are incomplete at the bottom component boundary when the component height is not an integer multiple of the CTB size;
- division of each CTU into coding units, except that the CTUs are incomplete at the right picture boundary when the picture width in luma samples is not an integer multiple of the luma CTB size and the CTUs are incomplete at the bottom picture boundary when the picture height in luma samples is not an integer multiple of the luma CTB size;
- division of each coding unit into transform units;
- division of each coding unit into coding blocks;
- division of each coding block into transform blocks;
- division of each transform unit into transform blocks.

For each of these divisions of an entity A into entities B specified as being a partitioning, it is requirement of bitstream conformance that the union of the entities B resulted from the partitioning of the entity A shall cover exactly the entity A with no overlaps, no gaps, and no additions.

For example, corresponding to the division of each picture into subpictures being a partitioning, it is requirement of bitstream conformance that the union of the subpictures resulted from the partitioning of a picture shall cover exactly the picture, with no overlaps, no gaps, and no CTUs in the union that are outside the picture.

6.4 Availability processes

6.4.1 Allowed quad split process

Inputs to this process are:

- a coding block size `cbSize` in luma samples,
- a multi-type tree depth `mttDepth`,
- a variable `treeType` specifying whether a single tree (`SINGLE_TREE`) or a dual tree is used to partition the coding tree node and, when a dual tree is used, whether the luma (`DUAL_TREE_LUMA`) or chroma components (`DUAL_TREE_CHROMA`) are currently processed,
- a variable `modeType` specifying whether intra (`MODE_INTRA`), IBC (`MODE_IBC`), palette (`MODE_PLT`), and inter coding modes can be used (`MODE_TYPE_ALL`), or whether only intra, IBC and palette coding modes can be used (`MODE_TYPE_INTRA`), or whether only inter coding modes can be used (`MODE_TYPE_INTER`) for coding units inside the coding tree node.

Output of this process is the variable `allowSplitQt`.

The variable `allowSplitQt` is derived as follows:

- If one or more of the following conditions are true, `allowSplitQt` is set equal to `FALSE`:
 - `treeType` is equal to `SINGLE_TREE` or `DUAL_TREE_LUMA` and `cbSize` is less than or equal to `MinQtSizeY`;
 - `treeType` is equal to `DUAL_TREE_CHROMA` and `cbSize` is less than or equal to `MinQtSizeC`;
 - `mttDepth` is not equal to 0;
 - `treeType` is equal to `DUAL_TREE_CHROMA` and $(cbSize / SubWidthC)$ is less than or equal to 4;
 - `treeType` is equal to `DUAL_TREE_CHROMA` and `modeType` is equal to `MODE_TYPE_INTRA`;
- Otherwise, `allowSplitQt` is set equal to `TRUE`.

6.4.2 Allowed binary split process

Inputs to this process are:

- a binary split mode `btSplit`,
- a coding block width `cbWidth` in luma samples,
- a coding block height `cbHeight` in luma samples,
- a location (`x0`, `y0`) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture,
- a multi-type tree depth `mttDepth`,
- a maximum multi-type tree depth with offset `maxMttDepth`,
- a maximum binary tree size `maxBtSize`,
- a minimum quadtree size `minQtSize`,
- a partition index `partIdx`,
- a variable `treeType` specifying whether a single tree (`SINGLE_TREE`) or a dual tree is used to partition the coding tree node and, when a dual tree is used, whether the luma (`DUAL_TREE_LUMA`) or chroma components (`DUAL_TREE_CHROMA`) are currently processed,
- a variable `modeType` specifying whether intra (`MODE_INTRA`), IBC (`MODE_IBC`), palette (`MODE_PLT`), and inter coding modes can be used (`MODE_TYPE_ALL`), or whether only intra, IBC and palette coding modes can be used (`MODE_TYPE_INTRA`), or whether only inter coding modes can be used (`MODE_TYPE_INTER`) for coding units inside the coding tree node.

Output of this process is the variable `allowBtSplit`.

Table 3 – Specification of parallelTtSplit and cbSize based on btSplit

	btSplit == SPLIT_BT_VER	btSplit == SPLIT_BT_HOR
parallelTtSplit	SPLIT TT VER	SPLIT TT HOR
cbSize	cbWidth	cbHeight

The variables parallelTtSplit and cbSize are derived as specified in Table 3.

The variable allowBtSplit is derived as follows:

- If one or more of the following conditions are true, allowBtSplit is set equal to FALSE:
 - cbSize is less than or equal to MinBtSizeY;
 - cbWidth is greater than maxBtSize;
 - cbHeight is greater than maxBtSize;
 - mttDepth is greater than or equal to maxMttDepth;
 - treeType is equal to DUAL_TREE_CHROMA and $(cbWidth / SubWidthC) * (cbHeight / SubHeightC)$ is less than or equal to 16;
 - treeType is equal to DUAL_TREE_CHROMA and $(cbWidth / SubWidthC)$ is equal to 4 and btSplit is equal to SPLIT_BT_VER;
 - treeType is equal to DUAL_TREE_CHROMA and modeType is equal to MODE_TYPE_INTRA;
 - $cbWidth * cbHeight$ is equal to 32 and modeType is equal to MODE_TYPE_INTER;
- Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:
 - btSplit is equal to SPLIT_BT_VER;
 - $y0 + cbHeight$ is greater than pps_pic_height_in_luma_samples;
- Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:
 - btSplit is equal to SPLIT_BT_VER;
 - cbHeight is greater than 64;
 - $x0 + cbWidth$ is greater than pps_pic_width_in_luma_samples;
- Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:
 - btSplit is equal to SPLIT_BT_HOR;
 - cbWidth is greater than 64;
 - $y0 + cbHeight$ is greater than pps_pic_height_in_luma_samples;
- Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:
 - $x0 + cbWidth$ is greater than pps_pic_width_in_luma_samples;
 - $y0 + cbHeight$ is greater than pps_pic_height_in_luma_samples;
 - cbWidth is greater than minQtSize;
- Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:
 - btSplit is equal to SPLIT_BT_HOR;
 - $x0 + cbWidth$ is greater than pps_pic_width_in_luma_samples;
 - $y0 + cbHeight$ is less than or equal to pps_pic_height_in_luma_samples;
- Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:
 - mttDepth is greater than 0;

- partIdx is equal to 1;
- MttSplitMode[x0][y0][mttDepth – 1] is equal to parallelTtSplit;
- Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:
 - btSplit is equal to SPLIT_BT_VER;
 - cbWidth is less than or equal to 64;
 - cbHeight is greater than 64;
- Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:
 - btSplit is equal to SPLIT_BT_HOR;
 - cbWidth is greater than 64;
 - cbHeight is less than or equal to 64;
- Otherwise, allowBtSplit is set equal to TRUE.

6.4.3 Allowed ternary split process

Inputs to this process are:

- a ternary split mode ttSplit,
- a coding block width cbWidth in luma samples,
- a coding block height cbHeight in luma samples,
- a location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture,
- a multi-type tree depth mttDepth,
- a maximum multi-type tree depth with offset maxMttDepth,
- a maximum ternary tree size maxTtSize,
- a variable treeType specifying whether a single tree (SINGLE_TREE) or a dual tree is used to partition the coding tree node and, when a dual tree is used, whether the luma (DUAL_TREE_LUMA) or chroma components (DUAL_TREE_CHROMA) are currently processed,
- a variable modeType specifying whether intra (MODE_INTRA), IBC (MODE_IBC), palette (MODE_PLT), and inter coding modes can be used (MODE_TYPE_ALL), or whether only intra, IBC and palette coding modes can be used (MODE_TYPE_INTRA), or whether only inter coding modes can be used (MODE_TYPE_INTER) for coding units inside the coding tree node.

Output of this process is the variable allowTtSplit.

Table 4 – Specification of cbSize based on ttSplit

	ttSplit == SPLIT TT VER	ttSplit == SPLIT TT HOR
cbSize	cbWidth	cbHeight

The variable cbSize is derived as specified in Table 4.

The variable allowTtSplit is derived as follows:

- If one or more of the following conditions are true, allowTtSplit is set equal to FALSE:
 - cbSize is less than or equal to 2 * MinTtSizeY;
 - cbWidth is greater than Min(64, maxTtSize);
 - cbHeight is greater than Min(64, maxTtSize);
 - mttDepth is greater than or equal to maxMttDepth;

- $x0 + cbWidth$ is greater than $pps_pic_width_in_luma_samples$;
- $y0 + cbHeight$ is greater than $pps_pic_height_in_luma_samples$;
- $treeType$ is equal to $DUAL_TREE_CHROMA$ and $(cbWidth / SubWidthC) * (cbHeight / SubHeightC)$ is less than or equal to 32;
- $treeType$ is equal to $DUAL_TREE_CHROMA$ and $(cbWidth / SubWidthC)$ is equal to 8 and $ttSplit$ is equal to $SPLIT_TT_VER$;
- $treeType$ is equal to $DUAL_TREE_CHROMA$ and $modeType$ is equal to $MODE_TYPE_INTRA$;
- $cbWidth * cbHeight$ is equal to 64 and $modeType$ is equal to $MODE_TYPE_INTER$;
- Otherwise, $allowTtSplit$ is set equal to $TRUE$.

6.4.4 Derivation process for neighbouring block availability

Inputs to this process are:

- the luma location $(xCurr, yCurr)$ of the top-left sample of the current block relative to the top-left luma sample of the current picture,
- the luma location $(xNbY, yNbY)$ covered by a neighbouring block relative to the top-left luma sample of the current picture,
- the variable $checkPredModeY$ specifying whether availability depends on the prediction mode,
- the variable $cIdx$ specifying the colour component of the current block.

Output of this process is the availability of the neighbouring block covering the location $(xNbY, yNbY)$, denoted as $availableN$.

The neighbouring block availability $availableN$ is derived as follows:

- If one or more of the following conditions are true, $availableN$ is set equal to $FALSE$:
 - $xNbY$ is less than 0;
 - $yNbY$ is less than 0;
 - $xNbY$ is greater than or equal to $pps_pic_width_in_luma_samples$;
 - $yNbY$ is greater than or equal to $pps_pic_height_in_luma_samples$;
 - $(xNbY \gg CtbLog2SizeY)$ is greater than $(xCurr \gg CtbLog2SizeY)$ and $(yNbY \gg CtbLog2SizeY)$ is greater than or equal to $(yCurr \gg CtbLog2SizeY)$;
 - $(yNbY \gg CtbLog2SizeY)$ is greater than or equal to $(yCurr \gg CtbLog2SizeY) + 1$;
 - $IsAvailable[cIdx][xNbY][yNbY]$ is equal to $FALSE$;
 - The neighbouring block is contained in a different slice than the current block;
 - The neighbouring block is contained in a different tile than the current block;
 - $sps_entropy_coding_sync_enabled_flag$ is equal to 1 and $(xNbY \gg CtbLog2SizeY)$ is greater than or equal to $(xCurr \gg CtbLog2SizeY) + 1$;
- Otherwise, $availableN$ is set equal to $TRUE$.

When all of the following conditions are true, $availableN$ is set equal to $FALSE$:

- $checkPredModeY$ is equal to $TRUE$;
- $CuPredMode[0][xNbY][yNbY]$ is not equal to $CuPredMode[0][xCurr][yCurr]$.

6.5 Scanning processes

6.5.1 CTB raster scanning, tile scanning, and subpicture scanning processes

The variable $NumTileColumns$, specifying the number of tile columns, and the list $ColWidthVal[i]$ for i ranging from 0 to $NumTileColumns - 1$, inclusive, specifying the width of the i -th tile column in units of CTBs, are derived as follows:

```

remainingWidthInCtbsY = PicWidthInCtbsY
for( i = 0; i <= pps_num_exp_tile_columns_minus1; i++ ) {
    ColWidthVal[ i ] = pps_tile_column_width_minus1[ i ] + 1
    remainingWidthInCtbsY -= ColWidthVal[ i ]
}
uniformTileColWidth = pps_tile_column_width_minus1[ pps_num_exp_tile_columns_minus1 ] + 1      (14)
while( remainingWidthInCtbsY >= uniformTileColWidth ) {
    ColWidthVal[ i++ ] = uniformTileColWidth
    remainingWidthInCtbsY -= uniformTileColWidth
}
if( remainingWidthInCtbsY > 0 )
    ColWidthVal[ i++ ] = remainingWidthInCtbsY
NumTileColumns = i

```

The variable NumTileRows, specifying the number of tile rows, and the list RowHeightVal[j] for j ranging from 0 to NumTileRows – 1, inclusive, specifying the height of the j-th tile row in units of CTBs, are derived as follows:

```

remainingHeightInCtbsY = PicHeightInCtbsY
for( j = 0; j <= pps_num_exp_tile_rows_minus1; j++ ) {
    RowHeightVal[ j ] = pps_tile_row_height_minus1[ j ] + 1
    remainingHeightInCtbsY -= RowHeightVal[ j ]
}
uniformTileRowHeight = pps_tile_row_height_minus1[ pps_num_exp_tile_rows_minus1 ] + 1      (15)
while( remainingHeightInCtbsY >= uniformTileRowHeight ) {
    RowHeightVal[ j++ ] = uniformTileRowHeight
    remainingHeightInCtbsY -= uniformTileRowHeight
}
if( remainingHeightInCtbsY > 0 )
    RowHeightVal[ j++ ] = remainingHeightInCtbsY
NumTileRows = j

```

The variable NumTilesInPic is set equal to NumTileColumns * NumTileRows.

The list TileColBdVal[i] for i ranging from 0 to NumTileColumns, inclusive, specifying the location of the i-th tile column boundary in units of CTBs, is derived as follows:

```

for( TileColBdVal[ 0 ] = 0, i = 0; i < NumTileColumns; i++ )
    TileColBdVal[ i + 1 ] = TileColBdVal[ i ] + ColWidthVal[ i ]      (16)

```

NOTE 1 The size of the array TileColBdVal[] in this derivation is one greater than the actual number of tile columns.

The list TileRowBdVal[j] for j ranging from 0 to NumTileRows, inclusive, specifying the location of the j-th tile row boundary in units of CTBs, is derived as follows:

```

for( TileRowBdVal[ 0 ] = 0, j = 0; j < NumTileRows; j++ )
    TileRowBdVal[ j + 1 ] = TileRowBdVal[ j ] + RowHeightVal[ j ]      (17)

```

NOTE 2 The size of the array TileRowBdVal[] in this derivation is one greater than the actual number of tile rows.

The lists CtbToTileColBd[ctbAddrX] and ctbToTileColIdx[ctbAddrX] for ctbAddrX ranging from 0 to PicWidthInCtbsY, inclusive, specifying the conversion from a horizontal CTB address to a left tile column boundary in units of CTBs and to a tile column index, respectively, are derived as follows:

```

tileX = 0
for( ctbAddrX = 0; ctbAddrX <= PicWidthInCtbsY; ctbAddrX++ ) {
    if( ctbAddrX == TileColBdVal[ tileX + 1 ] )
        tileX++
    CtbToTileColBd[ ctbAddrX ] = TileColBdVal[ tileX ]
    ctbToTileColIdx[ ctbAddrX ] = tileX
}      (18)

```

NOTE 3 The sizes of the arrays CtbToTileColBd[] and ctbToTileColIdx[] in this derivation are one greater than the actual picture width in CTBs.

The lists `CtbToTileRowBd[ ctbAddrY ]` and `ctbToTileRowIdx[ ctbAddrY ]` for `ctbAddrY` ranging from 0 to `PicHeightInCtbsY`, inclusive, specifying the conversion from a vertical CTB address to a top tile column boundary in units of CTBs and to a tile row index, respectively, are derived as follows:

```
tileY = 0
for( ctbAddrY = 0; ctbAddrY <= PicHeightInCtbsY; ctbAddrY++ ) {
    if( ctbAddrY == TileRowBdVal[ tileY + 1 ] )
        tileY++
    CtbToTileRowBd[ ctbAddrY ] = TileRowBdVal[ tileY ]
    ctbToTileRowIdx[ ctbAddrY ] = tileY
}
```

(19)

NOTE 4 The sizes of the arrays `CtbToTileRowBd[ ]` and `ctbToTileRowIdx[ ]` in this derivation are one greater than the actual picture height in CTBs.

The lists `SubpicWidthInTiles[ i ]` and `SubpicHeightInTiles[ i ]`, for `i` ranging from 0 to `sps_num_subpics_minus1`, inclusive, specifying the width and the height of the `i`-th subpicture in tile columns and rows, respectively, and the list `subpicHeightLessThanOneTileFlag[ i ]`, for `i` ranging from 0 to `sps_num_subpics_minus1`, inclusive, specifying whether the height of the `i`-th subpicture is less than one tile row, are derived as follows:

```
for( i = 0; i <= sps_num_subpics_minus1; i++ ) {
    leftX = sps_subpic_ctu_top_left_x[ i ]
    rightX = leftX + sps_subpic_width_minus1[ i ]
    SubpicWidthInTiles[ i ] = ctbToTileColIdx[ rightX ] + 1 - ctbToTileColIdx[ leftX ]
    topY = sps_subpic_ctu_top_left_y[ i ]
    bottomY = topY + sps_subpic_height_minus1[ i ]
    SubpicHeightInTiles[ i ] = ctbToTileRowIdx[ bottomY ] + 1 - ctbToTileRowIdx[ topY ]
    if( SubpicHeightInTiles[ i ] == 1 &&
        sps_subpic_height_minus1[ i ] + 1 < RowHeightVal[ ctbToTileRowIdx[ topY ] ] )
        subpicHeightLessThanOneTileFlag[ i ] = 1
    else
        subpicHeightLessThanOneTileFlag[ i ] = 0
}
```

(20)

NOTE 5 When a tile is partitioned into multiple rectangular slices and only a subset of the rectangular slices of the tile is included in the `i`-th subpicture, the tile is counted as one tile in the value of `SubpicHeightInTiles[ i ]`.

When `pps_rect_slice_flag` is equal to 1, the list `NumCtusInSlice[ i ]` for `i` ranging from 0 to `pps_num_slices_in_pic_minus1`, inclusive, specifying the number of CTUs in the `i`-th slice, the list `SliceTopLeftTileIdx[ i ]` for `i` ranging from 0 to `pps_num_slices_in_pic_minus1`, inclusive, specifying the tile index of the tile containing the first CTU in the slice, and the matrix `CtbAddrInSlice[ i ][ j ]` for `i` ranging from 0 to `pps_num_slices_in_pic_minus1`, inclusive, and `j` ranging from 0 to `NumCtusInSlice[ i ] - 1`, inclusive, specifying the picture raster scan address of the `j`-th CTB within the `i`-th slice, and the variable `NumSlicesInTile[ i ]`, specifying the number of slices in the tile containing the `i`-th slice, are derived as follows:

```
if( pps_single_slice_per_subpic_flag ) {
    if( !sps_subpic_info_present_flag ) /* There is no subpicture info and only one slice in a picture. */
        for( j = 0; j < NumTileRows; j++ )
            for( i = 0; i < NumTileColumns; i++ )
                AddCtbsToSlice( 0, TileColBdVal[ i ], TileColBdVal[ i + 1 ], TileRowBdVal[ j ],
                               TileRowBdVal[ j + 1 ] )
} else {
    for( i = 0; i <= sps_num_subpics_minus1; i++ ) {
        NumCtusInSlice[ i ] = 0
        if( subpicHeightLessThanOneTileFlag[ i ] ) /* The slice consists of a set of CTU rows in a tile. */
            AddCtbsToSlice( i, sps_subpic_ctu_top_left_x[ i ],
                           sps_subpic_ctu_top_left_x[ i ] + sps_subpic_width_minus1[ i ] + 1,
                           sps_subpic_ctu_top_left_y[ i ],
                           sps_subpic_ctu_top_left_y[ i ] + sps_subpic_height_minus1[ i ] + 1 )
        else /* The slice consists of a number of complete tiles covering a rectangular region. */
            tileX = ctbToTileColIdx[ sps_subpic_ctu_top_left_x[ i ] ]
            tileY = ctbToTileRowIdx[ sps_subpic_ctu_top_left_y[ i ] ]
            for( j = 0; j < SubpicHeightInTiles[ i ]; j++ )
                for( k = 0; k < SubpicWidthInTiles[ i ]; k++ )
                    AddCtbsToSlice( i, TileColBdVal[ tileX + k ], TileColBdVal[ tileX + k + 1 ],
```

```

    TileRowBdVal[ tileY + j ], TileRowBdVal[ tileY + j + 1 ] )
    }
    }
} else {
    tileIdx = 0
    for( i = 0; i <= pps_num_slices_in_pic_minus1; i++ )
        NumCtusInSlice[ i ] = 0
    for( i = 0; i <= pps_num_slices_in_pic_minus1; i++ ) {
        SliceTopLeftTileIdx[ i ] = tileIdx
        tileX = tileIdx % NumTileColumns
        tileY = tileIdx / NumTileColumns
        if( i < pps_num_slices_in_pic_minus1 ) {
            sliceWidthInTiles[ i ] = pps_slice_width_in_tiles_minus1[ i ] + 1
            sliceHeightInTiles[ i ] = pps_slice_height_in_tiles_minus1[ i ] + 1
        } else {
            sliceWidthInTiles[ i ] = NumTileColumns - tileX
            sliceHeightInTiles[ i ] = NumTileRows - tileY
            NumSlicesInTile[ i ] = 1
        }
        if( sliceWidthInTiles[ i ] == 1 && sliceHeightInTiles[ i ] == 1 ) {
            if( pps_num_exp_slices_in_tile[ i ] == 0 ) {
                NumSlicesInTile[ i ] = 1
                sliceHeightInCtus[ i ] = RowHeightVal[ SliceTopLeftTileIdx[ i ] / NumTileColumns ]
            } else {
                remainingHeightInCtbsY = RowHeightVal[ SliceTopLeftTileIdx[ i ] / NumTileColumns ]
                for( j = 0; j < pps_num_exp_slices_in_tile[ i ]; j++ ) {
                    sliceHeightInCtus[ i + j ] = pps_exp_slice_height_in_ctus_minus1[ i ][ j ] + 1
                    remainingHeightInCtbsY -= sliceHeightInCtus[ i + j ]
                }
                uniformSliceHeight = sliceHeightInCtus[ i + j - 1 ]
                while( remainingHeightInCtbsY >= uniformSliceHeight ) {
                    sliceHeightInCtus[ i + j ] = uniformSliceHeight
                    remainingHeightInCtbsY -= uniformSliceHeight
                    j++
                }
                if( remainingHeightInCtbsY > 0 ) {
                    sliceHeightInCtus[ i + j ] = remainingHeightInCtbsY
                    j++
                }
                NumSlicesInTile[ i ] = j
            }
        }
        ctbY = TileRowBdVal[ tileY ]
        for( j = 0; j < NumSlicesInTile[ i ]; j++ ) {
            AddCtbsToSlice( i + j, TileColBdVal[ tileX ], TileColBdVal[ tileX + 1 ],
                ctbY, ctbY + sliceHeightInCtus[ i + j ] )
            ctbY += sliceHeightInCtus[ i + j ]
            sliceWidthInTiles[ i + j ] = 1
            sliceHeightInTiles[ i + j ] = 1
        }
        i += NumSlicesInTile[ i ] - 1
    } else
        for( j = 0; j < sliceHeightInTiles[ i ]; j++ )
            for( k = 0; k < sliceWidthInTiles[ i ]; k++ )
                AddCtbsToSlice( i, TileColBdVal[ tileX + k ], TileColBdVal[ tileX + k + 1 ],
                    TileRowBdVal[ tileY + j ], TileRowBdVal[ tileY + j + 1 ] )
    if( i < pps_num_slices_in_pic_minus1 ) {
        if( pps_tile_idx_delta_present_flag )
            tileIdx += pps_tile_idx_delta_val[ i ]
        else {
            tileIdx += sliceWidthInTiles[ i ]
            if( tileIdx % NumTileColumns == 0 )
                tileIdx += ( sliceHeightInTiles[ i ] - 1 ) * NumTileColumns
        }
    }
}

```

```

    }
  }
}

```

Where the function `AddCtbsToSlice( sliceIdx, startX, stopX, startY, stopY)` is specified as follows:

```

for( ctbY = startY; ctbY < stopY; ctbY++ )
  for( ctbX = startX; ctbX < stopX; ctbX++ ) {
    CtbAddrInSlice[ sliceIdx ][ NumCtusInSlice[ sliceIdx ] ] = ctbY * PicWidthInCtbsY + ctbX
    NumCtusInSlice[ sliceIdx ]++
  }

```

(22)

It is a requirement of bitstream conformance that the values of `NumCtusInSlice[ i ]` for `i` ranging from 0 to `pps_num_slices_in_pic_minus1`, inclusive, shall be greater than 0. Additionally, it is a requirement of bitstream conformance that the matrix `CtbAddrInSlice[ i ][ j ]` for `i` ranging from 0 to `pps_num_slices_in_pic_minus1`, inclusive, and `j` ranging from 0 to `NumCtusInSlice[ i ] - 1`, inclusive, shall include each of all CTB addresses in the range of 0 to `PicSizeInCtbsY - 1`, inclusive, once and only once.

The lists `NumSlicesInSubpic[ i ]`, `SubpicLevelSliceIdx[ j ]`, and `SubpicIdxForSlice[ j ]`, specifying the number of slices in the `i`-th subpicture, the subpicture-level slice index of the slice with picture-level slice index `j`, and the subpicture index of the slice with picture-level slice index `j`, respectively, are derived as follows:

```

for( i = 0; i <= sps_num_subpics_minus1; i++ ) {
  NumSlicesInSubpic[ i ] = 0
  for( j = 0; j <= pps_num_slices_in_pic_minus1; j++ ) {
    posX = CtbAddrInSlice[ j ][ 0 ] % PicWidthInCtbsY
    posY = CtbAddrInSlice[ j ][ 0 ] / PicWidthInCtbsY
    if( ( posX >= sps_subpic_ctu_top_left_x[ i ] ) &&
        ( posX < sps_subpic_ctu_top_left_x[ i ] + sps_subpic_width_minus1[ i ] + 1 ) &&
        ( posY >= sps_subpic_ctu_top_left_y[ i ] ) &&
        ( posY < sps_subpic_ctu_top_left_y[ i ] + sps_subpic_height_minus1[ i ] + 1 ) ) {
      SubpicIdxForSlice[ j ] = i
      SubpicLevelSliceIdx[ j ] = NumSlicesInSubpic[ i ]
      NumSlicesInSubpic[ i ]++
    }
  }
}

```

(23)

6.5.2 Up-right diagonal scan order array initialization process

Input to this process is a block width `blkWidth` and a block height `blkHeight`.

Output of this process is the array `diagScan[ sPos ][ sComp ]`. The array index `sPos` specify the scan position ranging from 0 to $(\text{blkWidth} * \text{blkHeight}) - 1$. The array index `sComp` equal to 0 specifies the horizontal component and the array index `sComp` equal to 1 specifies the vertical component. Depending on the value of `blkWidth` and `blkHeight`, the array `diagScan` is derived as follows:

```

i = 0
x = 0
y = 0
stopLoop = FALSE
while( !stopLoop ) {
  while( y >= 0 ) {
    if( x < blkWidth && y < blkHeight ) {
      diagScan[ i ][ 0 ] = x
      diagScan[ i ][ 1 ] = y
      i++
    }
    y--
    x++
  }
  y = x
  x = 0
  if( i >= blkWidth * blkHeight )

```

(24)

```

        stopLoop = TRUE
    }

```

6.5.3 Horizontal and vertical traverse scan order array initialization process

Input to this process is a block width blkWidth and a block height blkHeight.

Output of this process are the arrays hTravScan[sPos][sComp] and vTravScan[sPos][sComp]. The array hTravScan represents the horizontal traverse scan order and the array vTravScan represents the vertical traverse scan order. The array index sPos specifies the scan position ranging from 0 to (blkWidth * blkHeight) - 1, inclusive. The array index sComp equal to 0 specifies the horizontal component and the array index sComp equal to 1 specifies the vertical component. Depending on the value of blkWidth and blkHeight, the array hTravScan and vTravScan are derived as follows:

```

i = 0
for( y = 0; y < blkHeight; y++ )
    if( y % 2 == 0 )
        for( x = 0; x < blkWidth; x++ ) {
            hTravScan[ i ][ 0 ] = x
            hTravScan[ i ][ 1 ] = y
            i++
        }
    else
        for( x = blkWidth - 1; x >= 0; x-- ) {
            hTravScan[ i ][ 0 ] = x
            hTravScan[ i ][ 1 ] = y
            i++
        }
}

```

(25)

```

i = 0
for( x = 0; x < blkWidth; x++ )
    if( x % 2 == 0 )
        for( y = 0; y < blkHeight; y++ ) {
            vTravScan[ i ][ 0 ] = x
            vTravScan[ i ][ 1 ] = y
            i++
        }
    else
        for( y = blkHeight - 1; y >= 0; y-- ) {
            vTravScan[ i ][ 0 ] = x
            vTravScan[ i ][ 1 ] = y
            i++
        }
}

```

(26)

7 Syntax and semantics

7.1 Method of specifying syntax in tabular form

The syntax tables specify a superset of the syntax of all allowed bitstreams. Additional constraints on the syntax might be specified, either directly or indirectly, in other clauses.

NOTE An actual decoder is expected to implement some means for identifying entry points into the bitstream and some means to identify and handle non-conforming bitstreams. The methods for identifying and handling errors and other such situations are not specified in this document.

The following table lists examples of the syntax specification format. When **syntax_element** appears, it specifies that a syntax element is parsed from the bitstream and the bitstream pointer is advanced to the next position beyond the syntax element in the bitstream parsing process.

	Descriptor
/* A statement can be a syntax element with an associated descriptor or can be an expression used to specify conditions for the existence, type and quantity of syntax elements, as in the following two examples */	
syntax_element	ue(k)
conditioning statement	

/* A group of statements enclosed in curly brackets is a compound statement and is treated functionally as a single statement. */	
{	
statement	
statement	
...	
}	
/* A "while" structure specifies a test of whether a condition is true, and if true, specifies evaluation of a statement (or compound statement) repeatedly until the condition is no longer true */	
while(condition)	
statement	
/* A "do ... while" structure specifies evaluation of a statement once, followed by a test of whether a condition is true, and if true, specifies repeated evaluation of the statement until the condition is no longer true */	
do	
statement	
while(condition)	
/* An "if ... else" structure specifies a test of whether a condition is true and, if the condition is true, specifies evaluation of a primary statement, otherwise, specifies evaluation of an alternative statement. The "else" part of the structure and the associated alternative statement is omitted if no alternative statement evaluation is needed */	
if(condition)	
primary statement	
else	
alternative statement	
/* A "for" structure specifies evaluation of an initial statement, followed by a test of a condition, and if the condition is true, specifies repeated evaluation of a primary statement followed by a subsequent statement until the condition is no longer true. */	
for(initial statement; condition; subsequent statement)	
primary statement	

7.2 Specification of syntax functions and descriptors

The functions presented in this subclause are used in the specification of the syntax. These functions are expressed in terms of the value of a bitstream pointer that indicates the position of the next bit to be read by the decoding process from the bitstream.

byte_aligned() is specified as follows:

- If the current position in the bitstream is a byte-aligned position, i.e., the current position is an integer multiple of 8 bits from the position of the first bit in the bitstream, the return value of byte_aligned() is equal to TRUE.
- Otherwise, the return value of byte_aligned() is equal to FALSE.

more_data_in_byte_stream(), which is used only in the byte stream NAL unit syntax structure specified in Annex B, is specified as follows:

- If more data follow in the byte stream, the return value of more_data_in_byte_stream() is equal to TRUE.
- Otherwise, the return value of more_data_in_byte_stream() is equal to FALSE.

`more_data_in_payload()` is specified as follows:

- If `byte_aligned()` is equal to TRUE and the current position in the `sei_payload()` or `vui_payload()` syntax structure is $8 * \text{payloadSize}$ bits from the beginning of the syntax structure, the return value of `more_data_in_payload()` is equal to FALSE.
- Otherwise, the return value of `more_data_in_payload()` is equal to TRUE.

`more_rbsp_data()` is specified as follows:

- If there is no more data in the raw byte sequence payload (Rbsp), the return value of `more_rbsp_data()` is equal to FALSE.
- Otherwise, the Rbsp data are searched for the last (least significant, right-most) bit equal to 1 that is present in the Rbsp. Given the position of this bit, which is the first bit (`rbbsp_stop_one_bit`) of the `rbbsp_trailing_bits()` syntax structure, the following applies:
 - If there is more data in an Rbsp before the `rbbsp_trailing_bits()` syntax structure, the return value of `more_rbsp_data()` is equal to TRUE.
 - Otherwise, the return value of `more_rbsp_data()` is equal to FALSE.

The method for enabling determination of whether there is more data in the Rbsp is specified by the application (or in Annex B for applications that use the byte stream format).

`more_rbsp_trailing_data()` is specified as follows:

- If there is more data in an Rbsp, the return value of `more_rbsp_trailing_data()` is equal to TRUE.
- Otherwise, the return value of `more_rbsp_trailing_data()` is equal to FALSE.

`next_bits(n)` provides the next bits in the bitstream for comparison purposes, without advancing the bitstream pointer. Provides a look at the next n bits in the bitstream with n being its argument. When used within the byte stream format as specified in Annex B and fewer than n bits remain within the byte stream, `next_bits(n)` returns a value of 0.

`payload_extension_present()` is specified as follows:

- If the current position in the `sei_payload()` or `vui_payload()` syntax structure is not the position of the last (least significant, right-most) bit that is equal to 1 that is less than $8 * \text{payloadSize}$ bits from the beginning of the syntax structure (i.e., the position of the `sei_payload_bit_equal_to_one` or `vui_payload_bit_equal_to_one` syntax element), the return value of `payload_extension_present()` is equal to TRUE.
- Otherwise, the return value of `payload_extension_present()` is equal to FALSE.

`read_bits(n)` reads the next n bits from the bitstream and advances the bitstream pointer by n bit positions. When n is equal to 0, `read_bits(n)` is specified to return a value equal to 0 and to not advance the bitstream pointer.

The following descriptors specify the parsing process of each syntax element:

- `ae(v)`: context-adaptive arithmetic entropy-coded syntax element. The parsing process for this descriptor is specified in subclause 9.3.
- `b(8)`: byte having any pattern of bit string (8 bits). The parsing process for this descriptor is specified by the return value of the function `read_bits(8)`.
- `f(n)`: fixed-pattern bit string using n bits written (from left to right) with the left bit first. The parsing process for this descriptor is specified by the return value of the function `read_bits(n)`.
- `i(n)`: signed integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function `read_bits(n)` interpreted as a two's complement integer representation with most significant bit written first.
- `se(v)`: signed integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.2 with the order k equal to 0.
- `u(n)`: unsigned integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function `read_bits(n)` interpreted as a binary representation of an unsigned integer with most significant bit written first.
- `ue(v)`: unsigned integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.2 with the order k equal to 0.

7.3 Syntax in tabular form

7.3.1 NAL unit syntax

7.3.1.1 General NAL unit syntax

nal_unit(NumBytesInNalUnit) {	Descriptor
nal_unit_header()	
NumBytesInRbsp = 0	
for(i = 2; i < NumBytesInNalUnit; i++)	
if(i + 2 < NumBytesInNalUnit && next_bits(24) == 0x000003) {	
rbsp_byte [NumBytesInRbsp++]	b(8)
rbsp_byte [NumBytesInRbsp++]	b(8)
i += 2	
emulation_prevention_three_byte /* equal to 0x03 */	f(8)
} else	
rbsp_byte [NumBytesInRbsp++]	b(8)
}	

7.3.1.2 NAL unit header syntax

nal_unit_header() {	Descriptor
forbidden_zero_bit	f(1)
nuh_reserved_zero_bit	u(1)
nuh_layer_id	u(6)
nal_unit_type	u(5)
nuh_temporal_id_plus1	u(3)
}	

7.3.2 Raw byte sequence payloads, trailing bits and byte alignment syntax

7.3.2.1 Decoding capability information RBSP syntax

decoding_capability_information_rbsp() {	Descriptor
dci_reserved_zero_4bits	u(4)
dci_num_ptls_minus1	u(4)
for(i = 0; i <= dci_num_ptls_minus1; i++)	
profile_tier_level(1, 0)	
dci_extension_flag	u(1)
if(dci_extension_flag)	
while(more_rbsp_data())	
dci_extension_data_flag	u(1)
rbsp_trailing_bits()	
}	

7.3.2.2 Operating point information RBSP syntax

operating_point_information_rbsp() {	Descriptor
opi_ols_info_present_flag	u(1)
opi_htid_info_present_flag	u(1)
if(opi_ols_info_present_flag)	
opi_ols_idx	ue(v)
if(opi_htid_info_present_flag)	
opi_htid_plus1	u(3)
opi_extension_flag	u(1)
if(opi_extension_flag)	
while(more_rbsp_data())	
opi_extension_data_flag	u(1)
rbps_trailing_bits()	
}	

7.3.2.3 Video parameter set RBSP syntax

video_parameter_set_rbsp() {	Descriptor
vps_video_parameter_set_id	u(4)
vps_max_layers_minus1	u(6)
vps_max_sublayers_minus1	u(3)
if(vps_max_layers_minus1 > 0 && vps_max_sublayers_minus1 > 0)	
vps_default_ptl_dpb_hrd_max_tid_flag	u(1)
if(vps_max_layers_minus1 > 0)	
vps_all_independent_layers_flag	u(1)
for(i = 0; i <= vps_max_layers_minus1; i++) {	
vps_layer_id[i]	u(6)
if(i > 0 && !vps_all_independent_layers_flag) {	
vps_independent_layer_flag[i]	u(1)
if(!vps_independent_layer_flag[i]) {	
vps_max_tid_ref_present_flag[i]	u(1)
for(j = 0; j < i; j++) {	
vps_direct_ref_layer_flag[i][j]	u(1)
if(vps_max_tid_ref_present_flag[i] && vps_direct_ref_layer_flag[i][j])	
vps_max_tid_il_ref_pics_plus1[i][j]	u(3)
}	
}	
}	
}	
if(vps_max_layers_minus1 > 0) {	
if(vps_all_independent_layers_flag)	
vps_each_layer_is_an_ols_flag	u(1)
if(!vps_each_layer_is_an_ols_flag) {	
if(!vps_all_independent_layers_flag)	
vps_ols_mode_idc	u(2)
if(vps_ols_mode_idc == 2) {	

vps_num_output_layer_sets_minus2	u(8)
for(i = 1; i <= vps_num_output_layer_sets_minus2 + 1; i++)	
for(j = 0; j <= vps_max_layers_minus1; j++)	
vps_ols_output_layer_flag[i][j]	u(1)
}	
}	
vps_num_ptls_minus1	u(8)
}	
for(i = 0; i <= vps_num_ptls_minus1; i++) {	
if(i > 0)	
vps_pt_present_flag[i]	u(1)
if(!vps_default_ptl_dpb_hrd_max_tid_flag)	
vps_ptl_max_tid[i]	u(3)
}	
while(!byte_aligned())	
vps_ptl_alignment_zero_bit /* equal to 0 */	f(1)
for(i = 0; i <= vps_num_ptls_minus1; i++)	
profile_tier_level(vps_pt_present_flag[i], vps_ptl_max_tid[i])	
for(i = 0; i < TotalNumOlss; i++)	
if(vps_num_ptls_minus1 > 0 && vps_num_ptls_minus1 + 1 != TotalNumOlss)	
vps_ols_ptl_idx[i]	u(8)
if(!vps_each_layer_is_an_ols_flag) {	
vps_num_dpb_params_minus1	ue(v)
if(vps_max_sublayers_minus1 > 0)	
vps_sublayer_dpb_params_present_flag	u(1)
for(i = 0; i < VpsNumDpbParams; i++) {	
if(!vps_default_ptl_dpb_hrd_max_tid_flag)	
vps_dpb_max_tid[i]	u(3)
dpb_parameters(vps_dpb_max_tid[i], vps_sublayer_dpb_params_present_flag)	
}	
for(i = 0; i < NumMultiLayerOlss; i++) {	
vps_ols_dpb_pic_width[i]	ue(v)
vps_ols_dpb_pic_height[i]	ue(v)
vps_ols_dpb_chroma_format[i]	u(2)
vps_ols_dpb_bitdepth_minus8[i]	ue(v)
if(VpsNumDpbParams > 1 && VpsNumDpbParams != NumMultiLayerOlss)	
vps_ols_dpb_params_idx[i]	ue(v)
}	
vps_timing_hrd_params_present_flag	u(1)
if(vps_timing_hrd_params_present_flag) {	
general_timing_hrd_parameters()	
if(vps_max_sublayers_minus1 > 0)	
vps_sublayer_cpb_params_present_flag	u(1)
vps_num_ols_timing_hrd_params_minus1	ue(v)
for(i = 0; i <= vps_num_ols_timing_hrd_params_minus1; i++) {	
if(!vps_default_ptl_dpb_hrd_max_tid_flag)	
vps_hrd_max_tid[i]	u(3)
firstSubLayer = vps_sublayer_cpb_params_present_flag ? 0 : vps_hrd_max_tid[i]	

ols_timing_hrd_parameters(firstSubLayer, vps_hrd_max_tid[i])	
}	
if(vps_num_ols_timing_hrd_params_minus1 > 0 && vps_num_ols_timing_hrd_params_minus1 + 1 != NumMultiLayerOlss)	
for(i = 0; i < NumMultiLayerOlss; i++)	
vps_ols_timing_hrd_idx[i]	ue(v)
}	
}	
vps_extension_flag	u(1)
if(vps_extension_flag)	
while(more_rbsp_data())	
vps_extension_data_flag	u(1)
rbp_trailing_bits()	
}	

7.3.2.4 Sequence parameter set RBSP syntax

seq_parameter_set_rbsp() {	Descriptor
sps_seq_parameter_set_id	u(4)
sps_video_parameter_set_id	u(4)
sps_max_sublayers_minus1	u(3)
sps_chroma_format_idc	u(2)
sps_log2_ctu_size_minus5	u(2)
sps_ptl_dpb_hrd_params_present_flag	u(1)
if(sps_ptl_dpb_hrd_params_present_flag)	
profile_tier_level(1, sps_max_sublayers_minus1)	
sps_gdr_enabled_flag	u(1)
sps_ref_pic_resampling_enabled_flag	u(1)
if(sps_ref_pic_resampling_enabled_flag)	
sps_res_change_in_clvs_allowed_flag	u(1)
sps_pic_width_max_in_luma_samples	ue(v)
sps_pic_height_max_in_luma_samples	ue(v)
sps_conformance_window_flag	u(1)
if(sps_conformance_window_flag) {	
sps_conf_win_left_offset	ue(v)
sps_conf_win_right_offset	ue(v)
sps_conf_win_top_offset	ue(v)
sps_conf_win_bottom_offset	ue(v)
}	
sps_subpic_info_present_flag	u(1)
if(sps_subpic_info_present_flag) {	
sps_num_subpics_minus1	ue(v)
if(sps_num_subpics_minus1 > 0) {	
sps_independent_subpics_flag	u(1)
sps_subpic_same_size_flag	u(1)
}	
for(i = 0; sps_num_subpics_minus1 > 0 && i <= sps_num_subpics_minus1; i++) {	
if(!sps_subpic_same_size_flag i == 0) {	

if(i > 0 && sps_pic_width_max_in_luma_samples > CtbSizeY)	
sps_subpic_ctu_top_left_x[i]	u(v)
if(i > 0 && sps_pic_height_max_in_luma_samples > CtbSizeY)	
sps_subpic_ctu_top_left_y[i]	u(v)
if(i < sps_num_subpics_minus1 && sps_pic_width_max_in_luma_samples > CtbSizeY)	
sps_subpic_width_minus1[i]	u(v)
if(i < sps_num_subpics_minus1 && sps_pic_height_max_in_luma_samples > CtbSizeY)	
sps_subpic_height_minus1[i]	u(v)
}	
if(!sps_independent_subpics_flag) {	
sps_subpic_treated_as_pic_flag[i]	u(1)
sps_loop_filter_across_subpic_enabled_flag[i]	u(1)
}	
}	
sps_subpic_id_len_minus1	ue(v)
sps_subpic_id_mapping_explicitly_signalled_flag	u(1)
if(sps_subpic_id_mapping_explicitly_signalled_flag) {	
sps_subpic_id_mapping_present_flag	u(1)
if(sps_subpic_id_mapping_present_flag)	
for(i = 0; i <= sps_num_subpics_minus1; i++)	
sps_subpic_id[i]	u(v)
}	
}	
sps_bitdepth_minus8	ue(v)
sps_entropy_coding_sync_enabled_flag	u(1)
sps_entry_point_offsets_present_flag	u(1)
sps_log2_max_pic_order_cnt_lsb_minus4	u(4)
sps_poc_msb_cycle_flag	u(1)
if(sps_poc_msb_cycle_flag)	
sps_poc_msb_cycle_len_minus1	ue(v)
sps_num_extra_ph_bytes	u(2)
for(i = 0; i < (sps_num_extra_ph_bytes * 8); i++)	
sps_extra_ph_bit_present_flag[i]	u(1)
sps_num_extra_sh_bytes	u(2)
for(i = 0; i < (sps_num_extra_sh_bytes * 8); i++)	
sps_extra_sh_bit_present_flag[i]	u(1)
if(sps_ptl_dpb_hrd_params_present_flag) {	
if(sps_max_sublayers_minus1 > 0)	
sps_sublayer_dpb_params_flag	u(1)
dpb_parameters(sps_max_sublayers_minus1, sps_sublayer_dpb_params_flag)	
}	
sps_log2_min_luma_coding_block_size_minus2	ue(v)
sps_partition_constraints_override_enabled_flag	u(1)
sps_log2_diff_min_qt_min_cb_intra_slice_luma	ue(v)
sps_max_mtt_hierarchy_depth_intra_slice_luma	ue(v)
if(sps_max_mtt_hierarchy_depth_intra_slice_luma != 0) {	
sps_log2_diff_max_bt_min_qt_intra_slice_luma	ue(v)